Collider Ring Design Tutorial using Bmad

August 1, 2024

Georg Hoffstaetter de Torquat
Daria Kuzovkova
David Sagan
Matt Signorelli
Jonathan Unger
Ningdong Wang

Contents

0 Introduction and Overview

This ring design tutorial serves as an introductory guide to the *Bmad* ecosystem for relativistic charged–particle and X-Ray simulations. At the heart of this ecosystem is the open source *Bmad* toolkit (software library) which can be used to construct simulation programs in less time and with fewer bugs than can be done from scratch. But *Bmad* is more than just a toolkit, and the *Bmad* ecosystem contains many programs that use the toolkit as the engine for doing simulations. A few of these programs will be introduced in this tutorial. Principal among these is *Tao* which is a general purpose design and simulation program. The other programs introduced are the *Long Term Tracking* program for particle tracking including spin and radiation, the *Dynamic Aperture* program used for calculating dynamic apertures, and the *SODOM-2* program for calculating the invariant spin field. Not covered is the advanced topic of programming using the *Bmad* toolkit. For a discussion on *Bmad* programming, the reader is referred to the *Bmad* manual.

The EIC will probe the properties of matter via deep inelastic scattering while colliding spin-polarized electrons with protons and nuclei.

In this tutorial you will design, your own collider ring based on the 18 GeV Electron Storage Ring (ESR) of the Electron-Ion Collider (EIC) currently in design at the Brookhaven National Laboratory on Long Island. By the end, you'll gain both an understanding of many important concepts involved in building a real collider ring, as well as experience using *Bmad*.

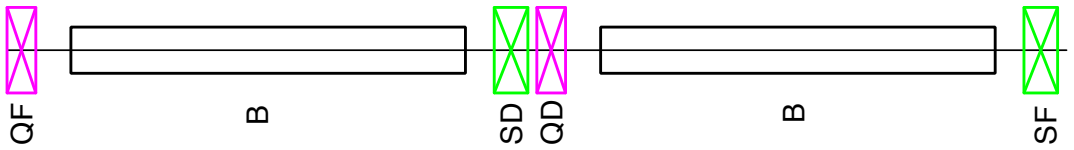
The ring design sections of the tutorial includes a short introduction to what is to be done, followed by an example. The example lattice files include instructions at the top describing how to create the lattice file step-by-step, numbered (1), (2), etc. Each number is also located at the place in the lattice file where that step is performed.

Needed is the *Bmad* manual along with the manuals of programs used for this tutorial. All of this is available at the *Bmad* web site:

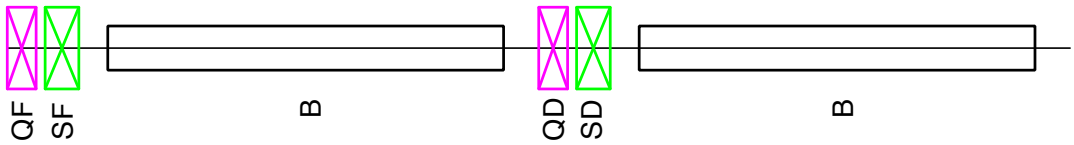
www.classe.cornell.edu/bmad/

Also to be found on the site are the input files for the programs. The input files are in a directory tree whose root directory is **/RingDesign/lattices**.

1 FODO Cells



(a) Forward arc FODO cell.



(b) Reverse arc FODO cell.

Figure 1: QF, QD: Focusing and defocusing quadrupoles respectively. SF, SD: Sextupoles near focusing and defocusing quadrupoles respectively.

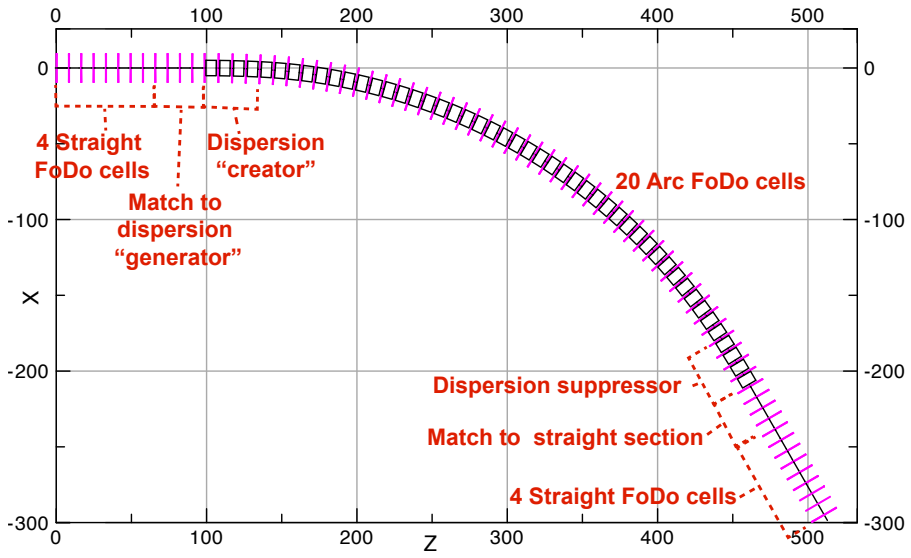


Figure 2: A sextant of the ring.

1.1 Introduction

In the arc FODO cells of the Electron Storage Ring (ESR), a sextupole, a bend, and a corrector coil all need to fit in between each quadrupole. To accommodate the sextupoles, the

quadrupoles cannot be centered between the bends. As a result, there are two types of FODO cells in the ESR: one which has the sextupoles placed before the quadrupoles and one which has the sextupoles placed after the quadrupoles. We'll call these the "forward" and "reverse" cells respectively and are shown in Figs. 1a and 1b. This tutorial will use the convention that all FODO cells start with a full-length horizontally-focusing ($K_1 > 0$) quadrupole.

The ring is made up 6 sextants as discussed in chapter §5. There are two types of sextants: forward sextants with forward FODO cells and reverse sextants with reverse FODO cells. The two types of sextants have mirror symmetry with respect to one another. Each sextant consists of two half-length straight sections with an arc section in between as shown in Fig. 2.

The first straight section, in which the dispersion is zero, is followed by a dispersion "creator" to match to the periodic dispersion in the arc. After the arc, there is a dispersion "suppressor" to match the dispersion free second straight section.

1.2 Example: Forward Arc FODO Cell

Later in the tutorial the sextupoles and corrector coils will be added in.

The phase advance per FODO cell for the lattice to be designed is chosen to be 90° to achieve the emittance requirements. Construction of the forward arc FODO cell without sextupoles will be demonstrated below. Left as an exercise will be the construction of the reverse arc FODO cell without sextupoles along with the forward and reverse straight section FODO cells. Example files can be found in the **/lattices/1_FoDoA/a_Arc/** directory.

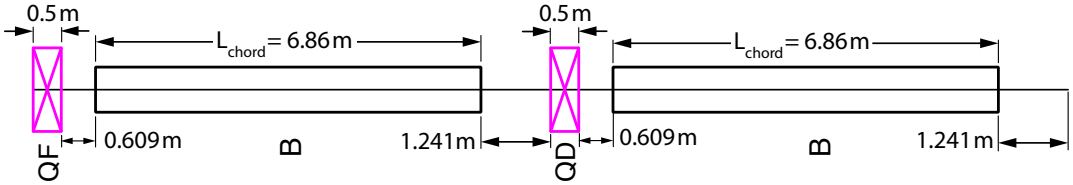


Figure 3: Forward arc FODO cell with lengths specified.

See the Bmad manual for details on the different types of elements.

Figure 3 shows the lengths of each element in the forward arc FODO cell. The ESR uses rectangular bends (rbends) in the arcs. Note: In *Bmad*, following the MAD convention, the length **L** of an rbend in the lattice file specifies the **chord** length, whereas for sector bends (sbends) it specifies the arc length. See the **Bends: Rbend and Sbend** section in the *Bmad* manual for more details. When **rbend**s are read in, *Bmad* will convert them to **sbend**s, will set **L_chord** equal to the input **L**, and will set **L** equal to the true arc length.

Our collider ring with have 132 arc FODO cells, and so each bend must precess the orbit by an angle $2\pi/(2 * 132)$. Choosing initially arbitrary quadrupole strengths **K1**, the *Bmad* file for this lattice is:

```
! Lattice file: fodoAF0.bmad
parameter[p0c]      = 1.7846262612447E10 ! Reference momentum in eV
parameter[particle] = Electron
parameter[geometry] = closed ! Calculates the periodic Twiss parameters

QF: Quadrupole, L = 0.5, K1 =  0.4
```

```
QD: Quadrupole, L = 0.5, K1 = -0.4
B: Rbend, L = 6.86, angle = pi/132
D1: Drift, L = 0.609
D2: Drift, L = 1.241

FODOAF: line = (QF, D1, B, D2, QD, D1, B, D2)
use, FODOAF
```

Make sure you are in the directory **/lattices/1_FoDoA**. On the command line, the lattice can be opened in *Tao* using the command

```
tao -lat fodoAF0.bmad
```

where the "**-lat**" flag is short for **-lattice** and the argument following this flag needs to be the name of the Bmad lattice file being analyzed. Specifying a lattice on the command line will override any default lattice specified in the **tao.init** file. In this case, the **tao.init** file does not specify a lattice so the **-lat** flag is needed on the command line.

When *Tao* is started, a plot window will open showing the beta functions, dispersion, orbit, and lattice layout. If the plot window is too large for the screen, the **-geom** flag followed by a width and height in the format <**width**>**x**<**height**> can be used to adjust the window size. For example:

```
tao -lat fodoAF0.bmad -geom 400x400
```

To view a summary of all the flags, use the command:

```
tao -help
```

After opening the above lattice in *Tao*, the **show lattice** command can be used to view the lattice:

```
Tao> sho lat      ! Abbreviations are acceptable so "sho lat" works.
# Values shown are for the Exit End of each Element:
# Index  name       key                    s       l     beta    phi_a ...
#                                                           a    [2pi] ...
      0  BEGINNING  Beginning_Ele        0.000     ---   43.90    0.000 ...
      1  QF         Quadrupole           0.500   0.500   43.90    0.002 ...
      2  D1         Drift                1.109   0.609   38.69    0.004 ...
      3  B          Sbend                7.969   6.860    3.97    0.098 ...
      4  D2         Drift                9.210   1.241    2.39    0.164 ...
      5  QD         Quadrupole           9.710   0.500    2.39    0.198 ...
      6  D1         Drift               10.319   0.609    2.98    0.234 ...
      7  B          Sbend               17.179   6.860   33.65    0.354 ...
      8  D2         Drift               18.420   1.241   43.90    0.359 ...
      9  END        Marker              18.420   0.000   43.90    0.359 ...
```

Bmad always includes a BEGINNING element at the beginning.

By default, *Bmad* puts an END marker element at the end

By default, *Tao*, will print values evaluated at the downstream **end** of elements (the end where particles exit the element). To print information evaluated at the middle or beginning of each element, include the **-middle** or **-beginning** flags in the **show lat** command respectively.

To see the phase advance at the end of the lattice, use the**show element end** command:

"show ele 9" also works.

```
Tao> show element end
Element # 9
Element Name: END
Key: Marker
S_start, S:      18.420324,      18.420324
```

```
Ref_time_start, Ref_time:  6.144359E-08,  6.144359E-08

Attribute values [Only non-zero values shown]:
    1  L                          =  0.0000000E+00 m
                              ...
Twiss at end of element:
                          A              B        ...
  Beta (m)          43.89736699     2.38487039  |   0.00000000   0.00000000  ...
  Alpha             -4.42108656     0.34433533  |   0.00000000   0.00000000  ...
  Gamma (1/m)        0.46804644     0.46902625  |   Gamma_c =   1.00000000   ...
  Phi (rad)          0.00000000     0.00000000          X              Y
  Eta (m)            0.38412123     0.00000000     0.38412123     0.00000000  ...
  Etap               0.03789803     0.00000000     0.03789803     0.00000000  ...
  dEta/ds            0.03789803     0.00000000     0.03789803     0.00000000  ...
  Sigma              0.00066663     0.00000011     0.00000000     0.00000000

Orbit:  Electron    State: Alive
          Position [mm]  Momentum [mrad]         Spin    | ...
  X:         0.00000000      0.00000000                  | ...
  Y:         0.00000000      0.00000000                  | ...
  Z:         0.00000000      0.00000000                  | ...
```

Phi is the betatron phase.

Tao calculates the Twiss parameters of the eigenmodes of the 1-turn matrix. For an uncoupled ring, the **A** mode corresponds to what other programs label the horizontal (x) Twiss, and the **B** mode corresponds to the vertical (y) Twiss.

1.2.1 Tao.init File

Additionally, *Tao* has an interface with Python allowing use of Python based optimizers.

Given our arbitrary choice of quadrupole strengths, the phase advance at the end of our FODO cell is not 90°. In order to determine what strengths will produce the desired phase advance, the optimizers built into *Tao* can be used. To perform an optimization, **data** and **variables** must be defined in a **tao.init** file. A **tao.init** file is also useful if you'd like *Tao* to run with special settings, e.g. a 400x400 plot page size (so the -**geom** flag does not need to be included when running *Tao*), or even custom plots. In order to determine the correct quadrupoles strengths, for a 90° phase advance, the **tao.init** file might look like:

```
! tao.init file
&tao_plot_page
  plot_page%size = 400, 400     ! Sets the plot page size
  plot_page%text_height = 8     ! Makes the plot page text size smaller
  plot_page%n_curve_pts = 5000 ! Adds more sampled points to the plots
/

&tao_d2_data
  d2_data%name = "phase"
  n_d1_data = 1
/

&tao_d1_data
  ix_d1_data = 1
  datum(1) = "phase.a" "" "" "end" "target" 1.570796326794897 10.0
  datum(2) = "phase.b" "" "" "end" "target" 1.570796326794897 10.0
/
```

```
&tao_var
  v1_var%name = "quads"
  var(1:2)%ele_name = "QF", "QD"
  default_step = 1e-6
  default_attribute = "k1"
/
```

There's a lot going on here, so let's decompose each "**namelist**" in this file.

1.2.2 Plotting

The first namelist, **tao_plot_page**, tells *Tao* to open with a plot window with a size of 400x400, a text height of 8pt, and include 5,000 points for each plot. The last setting will be useful when looking at the betas for the entire ring, as aliasing may be present with the default settings causing plotting artifacts.

1.2.3 Optimization Data Setup

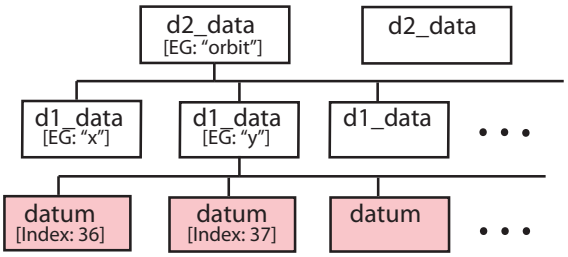


Figure 4: *Tao*'s tree structure for datums. A **d2_data** structure holds an array of **d1_data** structures and a **d1_data** structure holds an array of datums.

Data: The next two namelists, **tao_d2_data** and **tao_d1_data**, define the **data** of the optimization. "**Data**" are the conditions that we wish to satisfy via optimization. In this case, wanted is to make the horizontal and vertical phase advances 90° from start to end. A quick look at the "**Tao Data Types**" section in the "**Data**" chapter of the *Tao* manual shows that the desired data types are called **phase.a** and **phase.b**.

Fig. 4 shows how data is organized in a three level tree-structure which serves as an aid to the user in keeping the data organized. At the top level, **d2_data** structures are defined. A **d2_data** structure is defined with a **tao_d2_data** namelist. In this case, there is a single **d2_data** structure named "**phase**". Every **d2** structure has an array of **d1_data** structures under it. In this example there is only one (**n_d1_data** = 1) and this **d1** structure is defined in the **tao_d1_data** namelist below the **tao_d2_data** namelist. Every **d1_data** structure has an array of **data** structures with each **data** structure representing a single datum. In this example there are two datums, the first for **phase.a** and the second for **phase.b**.

While this example used a single **d2_data** and a single **d1_data** structure, a differing organization could have been used and each datum could have been assigned its own separate **d2_data** and a single **d1_data** structures. Ultimately, the tree organization has no impact on the outcome of the optimization.

The datums for the optimization are defined on the lines

```
datum(1) = "phase.a" "" "" "end" "target" 1.570796326794897 10.0
datum(2) = "phase.b" "" "" "end" "target" 1.570796326794897 10.0
```

The syntax for setting datums is discussed in the “Data and Constraint Initialization" section of the “Initialization" chapter of the *Tao* manual. Basically, the datums will be assigned “**model**” values which are the horizontal and vertical mode phase advances at the **END** element of the FODO cell, and the optimizer will adjust variable settings to try to get these **model** values equal to what is called the “**measured**” values which are set to $\pi/2$ as shown above.

Optimization involves minimization of a merit function $\mathcal{M}$ which is evaluated using the formula

$$\mathcal{M} \equiv \sum_i w_i \left[\delta D_i\right]^2 + \sum_j w_j \left[\delta V_j\right]^2 \tag{1}$$

where the first sum is a sum over the **data** and the second sum is a sum over the **variables**. The w_i and w_j are weights specified by the user and the δD_i and δV_j are functions of the data and variables. The merit function is discussed in depth in the “Optimization" chapter in the *Tao* manual.

The datums defined above both have weights of 10.0. Datum weights can have a large effect on optimizations. Often times, the best way to choose the correct weight is trial and error (is the optimization converging well or not?).

The datums defined above both use a **merit_type** that is set to "**target**". In this case, δD in Eq. (1) is evaluated using the equation

```
δD = model - meas
```

where **model** is the value as calculated from the lattice and **meas** is the “**measured**” value as set on the datum line. In this case, the **meas** value is $\pi/2$ for both datums.

Datum names are in the form

```
<d2-data-name>.<d1-data-name>[datum-index]
```

In this case, the **d2_data** structure has the name (set by **d2_data%name** in the file) of "**phase**". The name of the **d1_data** structure is not specified in the file and so gets the default name of “**1**" (if there were other **d1_data** structs associated with this **d2_data** structure, the default names would be “**2**", “**3**", etc.). The datums have names

```
phase.a datum: phase.1[1]     ! Can be shortened to phase[1]
phase.b datum: phase.1[2]     ! Can be shortened to phase[2]
```

Since there is only one **d1_datum** here the “**.1**” specifier can be omitted. The **show data** command will display a list of defined **d2_data**:

```
Tao> show data
  Name                                  Using for Optimization
  phase[1:2]                                    Using: 1:2
```

To see information on a particular **d2_data** structure, use the command **show data** <**d2**> where <**d2**> is the name of the structure. In this case:

```
Tao> show data phase
Data name: phase.1
                                                                        |    Useit
                        ... Ele         Meas         Model        Design | Opt   Plot
 1  phase.a <target> ... END  1.57079E+0   2.25845E+0   2.25845E+0      T      F
 2  phase.b <target> ... END  1.57079E+0   2.260399+0   2.26039E+0      T      F
```

The **design** value is the value of the "**design**" lattice which is a lattice *Tao* creates representing the lattice that was read in (discussed in depth in chapter §13). The parameters in the design lattice are fixed and cannot change. This is in contrast to the "**model**" lattice where the parameters are changeable and it is the quadrupole strengths of the model lattice which will be varied in the optimization. In this case, the merit function is calculated using the difference **model** - **meas** (see above), the design lattice does not influence the present optimization.

To see information on a particular datum, use the **show data** <**datum**> where <**datum**> is the name of the datum. For example:

```
Tao> show data phase[1]

%ele_name           = "END"
 ... etc ...
%data_type          = "phase.a"
 ... etc ...
%model              =   2.25845820E+00
%design             =   2.25845820E+00
 ... etc ...
%weight             =   1.00000000E+01
%exists             = T
%good_model         = T
%good_design        = T
%good_base          = T
%good_meas          = T
 ... etc ...
%useit_plot         = F
%useit_opt          = T
```

The **useit_opt** logical indicates whether the datum will be used when the merit function is evaluated. For example, if the **meas** value has not been set, *Tao* will set **good_meas** to False and this will cause *Tao* to set **useit_opt** to False.

1.2.4 Optimization Variable Setup

The last namelist, **tao_var**, in the above **tao.init** file defines the **variables** of the optimization

```
&tao_var
  v1_var%name = "quads"
```

```
  var(1:2)%ele_name = "QF", "QD"
  default_step = 1e-6
  default_attribute = "k1"
/
```

variables are the things to vary in the optimization to satisfy the conditions defined by the data. Variables are grouped in a tree structure just like data except for variables there are only two layers instead of the three for data. The top layer is a set of **v1_var** structures. In this case there is a single **v1_var** structure called "**quads**". Every **v1_var** structure has an array of **variables**. In this case, the **quads v1_var** structure has a two variable array with the variables controlling the **k1** attribute of the **QF**, and **QD** quadrupoles respectively. To calculate derivatives for the optimization, *Tao* uses finite differences and so a step size must be specified. The step size can be different for different variables but here a default value of **1e-6** is used that will be applied to all the variables. Variables can have weights which affects the merit function. Weights for variables are useful when there are “degeneracies" or near-degeneracies where, without variable weights, the optimizer can find solutions with small merit function values with variables having unphysically large values. The weights can constrain variables to have reasonable values. In this case there are no degeneracies and variable weights are not needed.

To refer to an individual variable use the syntax:

```
v1-var-name[var-index]
```

where **var-index** is the index of the variable. For example, the first variable in the **quad** structure is **quad[1]**, etc.

When running *Tao*, the **show variable** command can be used to view a list of the **v1_vars** defined

```
Tao> show var
       Name                                          Using for Optimization
    quads[1:2]                                       1:2
```

The **show variable** <**v1**> can be used to show information on a particular **v1_var** named <**v1**>

```
Tao> show var quads
  Variable      Slave Parameters    Meas          Model         Design   Useit_opt
  quads[1]      QF[K1]          0.0000E+00     4.0000E-01     4.0000E-01        T
  quads[2]      QD[K1]          0.0000E+00    -4.0000E-01    -4.0000E-01        T
```

And the **show variable** <**var**> can be used to show information on a particular variable named <**var**>

```
Tao> show var quads[2]
%ele_name         = "QD"
%attrib_name      = "K1"
 ... etc...
%model            =  -4.00000000E-01
 ... etc...
%exists           = T
%good_var         = T
%good_user        = T
%good_opt         = T
%useit_opt        = T
```

1.2.5 Running an Optimization

To see what data and what variables are being used in the optimization, use the **show data** and **show variables** commands as discussed above. To see optimizer parameters, use the **show optimizer** command:

```
Tao> show opti
Data Used:
  phase[1:2]                                    Using: 1:2
Variables Used:
  quads[1:2]                                    Using: 1:2

optimizer:        "lm"
Global optimization parameters (use "set global" to change):
  %de_lm_step_ratio              =   1.00000000E+00
  %de_var_to_population_factor   =   5.00000000E+00
  %lm_opt_deriv_reinit           =  -1.00000000E+00
   ...
```

There are many parameters associated with optimization and it is important to carefully consider what values these parameters have in order to be able to have a successful optimization.

To see how the merit function is being evaluated, use the **show constraints** command:

```
Tao> show cons

Constraints                  ...   Ele/S    Meas-or-Lim     Model        Merit
phase[1]   phase.a <target> ...   END       1.5708E+00   2.2585E+00   4.729E+00
phase[2]   phase.b <target> ...   END       1.5708E+00   2.2604E+00   4.756E+00
quads[1]   QF[K1]           ...   0.50     -1.0000E+30   4.0000E-01   0.000E+00
quads[2]   QD[K1]           ...   9.71     -1.0000E+30  -4.0000E-01   0.000E+00

 figure of merit: 9.484309E+00

! Top 10
Constraints                  ...   Ele/S    Meas-or-Lim     Model        Merit
phase[2]   phase.b <target> ...   END       1.5708E+00   2.2604E+00   4.756E+00
phase[1]   phase.a <target> ...   END       1.5708E+00   2.2585E+00   4.729E+00
quads[2]   QD[K1]           ...   9.71     -1.0000E+30  -4.0000E-01   0.000E+00
quads[1]   QF[K1]           ...   0.50     -1.0000E+30   4.0000E-01   0.000E+00
```

This shows, among other things, that the value of the merit function is 9.48 and that the largest contributor to the merit function by a hair is **phase[2]** which has a contribution of 4.756 which is about 50% of the total merit function. In this optimization, the variables do not contribute to the merit function.

There are several optimizers that can be used as detailed in the *Tao* manual. For global optimization use the **de** (Differential Evolution) optimizer. All other optimizers will only converge to the local optimum but will converge faster than **de**. In this case, the optimum is local so the lm (Levenburg-Marquardt) optimizer will be used which is the default:

```
Tao> run
Optimizing with: lm
Type ``.'' to stop the optimizer before it's finished.
[INFO] tao_dmodel_dvar_calc:
```

```
    Remaking dModel_dVar derivative matrix.
    This may take a while...

  Cycle       Merit     A_lambda
    1     4.3831E-01    1.00E-04
    2     5.7875E-02    1.00E-05
    3     8.5664E-03    1.00E-06
    ... etc...
   20     1.7768E-16    1.00E-23
Written: var1.out
   21     1.7768E-16    0.00E+00
```

The Merit value is now about 10^{-16} showing good convergence which can be seen by examining the data:

```
Tao> show data phase
Data name: phase.1
                                                                        |  Useit
                        ... Ele        Meas          Model         Design | Opt  Plot
 1  phase.a <target> ... END  1.570796E+0   1.570796E+0   2.258458E+0    T     F
 2  phase.b <target> ... END  1.570796E+0   1.570796E+0   2.260399E+0    T     F
```

And the model lattice has the correct phase advance.

Use **write bmad fodoAF.bmad** after optimizing to create a new lattice file with name **fodoAF.bmad** having the optimized quadrupole strengths. Note that **fodoAF.bmad** will contain the beginning periodic Twiss, dispersion, and orbit (only non-zero values are put in the file so the orbit does not explicitly appear in this instance). This is very useful if the **geometry** is switched from **closed** to **open** as will be discussed in the following chapter.

1.3 Exercises

1. **Reverse arc FODO:** Construct the reverse arc FODO cell without sextupoles, and determine the quadrupole strengths for a 90° phase advance. Let the optimized lattice file name be **fodoAR.bmad**.

2. **Forward straight section FODO:** Replace the dipoles in the forward cell by a drift (empty space) of length 5.855m and name the drifts **DB**. This length is chosen to produce the same straight section length as in the RHIC tunnel. Additionally, rename the horizontally-focusing and defocusing quadrupoles **QFSS** and **QDSS** respectively. Which quadrupole strengths lead to 90 degrees phase advance in both planes? Use the name **fodoSSF.bmad** for the optimized lattice file. Note: files are in **1_FODO/b_SS/F**

3. **Reverse straight section FODO:** Replace the dipoles in the reverse cell by a drift of length 5.855m and name the drifts **DB**. Rename the horizontally-focusing and defocusing quadrupoles **QFSS** and **QDSS** respectively. Which quadrupole strengths lead to 90 degrees phase advance in both planes? Use the name **fodoSSR.bmad** for the optimized lattice file. These quadrupole strengths should be exactly the same as in **fodoSSF.bmad**

4. **Analytical FODO cell:** Using a FODO model with the same total length as constructed above but with zero length quadrupoles, and no bends. Calculate by hand the appropriate quad

strengths that lead to a 90 degrees phase advance in x and y. What are the beta functions at the thin quads? How do the quad strengths and beta functions compare to above?

5. **Forward and Reversed Cells:** Check that your forward and reverse arc FODO cells that both start with focusing quads have different periodic beta and alpha functions. Check also that both cells, for the same phase advance of 90 degrees have exactly the same quadrupole strength. Explain why this is so

6. **L, L_arc, and L_chord:** For **rbend** elements, to be consistent with the MAD program, the input "**l**" length in the lattice file is used as the chord length. On input, the **rbends** are converted to **sbends** and after the conversion **l** will be set to the true arc length. For the **rbends** in the above lattice, what is the difference of the values between these two lengths? [Hint: Use the "show ele" command.]

7. **show lattice command:** The **show lattice** command is very powerful (and has a lot of options). Can you figure out what the following command shows?

```
Tao> show lat -attrib k1 quadrupole::*
```

8. **show data <datum> command:** An individual datum is a complicated beast and the output of the **show data <datum>** command reflects this. Look at the full (not abridged output above) output and by reading the *Tao* manual see if you can make sense of the individual components.

9. **The good_user logical:** One component of a datum or variable is **good_user** which is set by the user and controls whether the datum or variable will be used in the optimization. Commands to control **good_user** are **use**, **veto** and **restore**. Play around with these commands to see how changing this affects the optimization.

2 Dispersion Suppressor

2.1 Introduction

To connect an arc to the following straight section in a sextant, the dispersion in the straight section must be suppressed to zero while minimally changing the betas. For a 90° lattice, this can be approximately achieved by using two equivalent arc FODO cells with the bends at half-strength. To get the dispersion and dispersion derivative at the end of the dispersion suppressor exactly zero, the last two quadrupoles can be minimally varied.

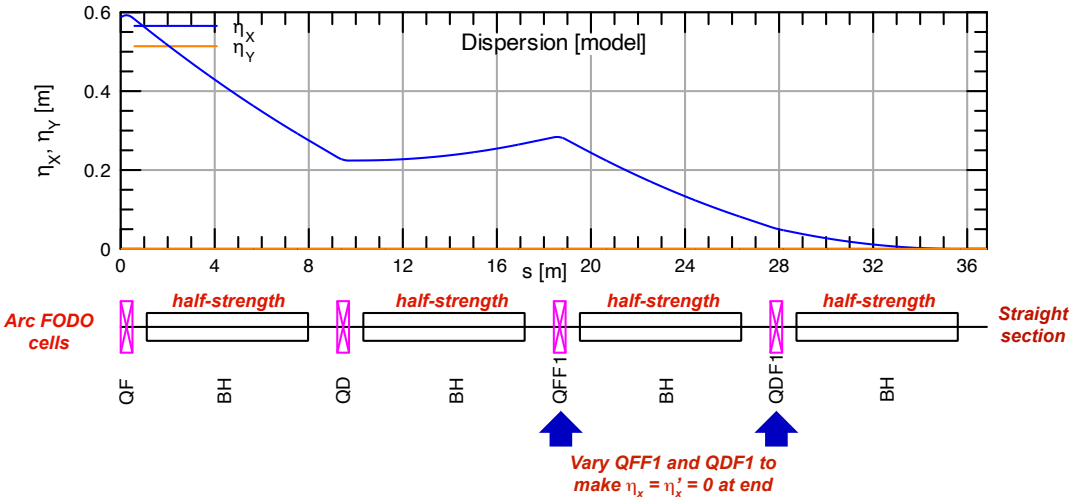


Figure 5: Dispersion suppressor.

2.2 Example: Forward Dispersion Suppressor Creation

The procedure for creating the forward dispersion suppressor is outlined below. The reverse dispersion suppressor construction is left as an exercise. The example files can be found in the **/lattices/2_DispSuppress/F/** directory.

To create the forward dispersion suppressor:

1. Create a copy of the **fodoAF.bmad** file, which contains the optimized quadrupole strengths for the 90° phase advance FODO cell.

2. In this new copy, set **parameter[geometry]** equal to **open**. This tells *Bmad* to *not* calculate the periodic betas, instead propagating the initial betas specified in the lattice file (which in this case, are the beginning periodic betas in the arc cells) through the lattice.

```
parameter[geometry] = open      ! (2): Set geometry to open
```

3. Define a new half-strength bend element and the two special quadrupoles to vary to make $\eta_x = \eta'_x = 0$. The strengths will be calculated via optimization. Because these special quadrupoles are in the forward sextant (F), and are the first "special" quadrupoles (1), use the names **QFF1** and **QDF1** (that is, add an **F1** suffix to the original quadrupole names).

```
! (3): Define half bend and quads for dispersion suppression
BH: rbend, L = 6.86, angle=pi/132/2
QFF1: Quadrupole, L = 0.5, K1 = 0.3
QDF1: Quadrupole, L = 0.5, K1 = -0.3
```

4. Define and use the new line as in Fig. 5 containing the half bends and special quads. Since this is the forward dispersion suppressor, use the name **DISPSUPF** for the line.

```
! (4): Define the new line for half bends
DISPSUPF: line = (QF, D1, BH, D2, QD, D1, BH, D2,
                  QFF1, D1, BH, D2, QDF1, D1, BH, D2)
use, DISPSUPF
```

The completed initial lattice file containing these changes is provided in the example file named **dispsupF0.bmad**.

5. A **tao.init** file is needed to define the optimization parameters. This might look like:

Use the "show opt" to see opt params.

```
&tao_plot_page
  plot_page%size = 400, 400
  plot_page%text_height = 8
  plot_page%n_curve_pts = 5000
/

! Set the number of cycles to 1000 to avoid having to type "run" less
&tao_params
  global%n_opti_cycles = 100
  global%n_opti_loops = 100
/

&tao_d2_data
    d2_data%name = "disp"
    n_d1_data = 1
/

&tao_d1_data
  ix_d1_data = 1
  datum(1) = "eta.x" "" "" "end" "target" 0 10
  datum(2) = "deta_ds.x" "" "" "end" "target" 0 10
/

&tao_var
  v1_var%name = "quads"
  var(1:2)%ele_name = "QFF1", "QDF1"
  default_step = 1e-6
  default_attribute = "k1"
/
```

This **tao.init** file has a namelist **tao_params** in which **n_opti_cycles**, the number of optimization cycles, is set equal to 100 (the default is 20). When the optimizer goes through **n_opti_cycles**, this is called a **loop**. Before each loop, the derivatives of how much the **data**

changes with change in **variable** is computed. Periodic recomputation is needed since lattices are never linear and so the derivatives change as variables are varied. Also in the **tao_params** namelist the number of loops, which is set by **n_opti_loops**, is set to 100 (the default is one). These are two of many parameters that can be set to customize the *Tao* experience. See the *Tao* manual for more options.

The **tao.init** file has two datums, one for the horizontal dispersion **eta.x** and one for its slope **deta_ds.x**. Both are desired to be exactly zero at the end of the lattice and they have equal weight. Additionally, the variables are the k1 strength of the two special quadrupoles **QFF1** and **QDF1**.

6. Starting *Tao* with the new **tao.init** file, the **show constraints** command can be used to verify that the optimization is set up correctly:

```
Tao> show cons

Constraints                  ...   Ele/S  Meas-or-Lim    Model        Merit
disp[1]   eta.x <target>     ...   END      0.0000E+00   5.1814E-02  2.685E-02
disp[2]   deta_ds.x <target> ...   END      0.0000E+00   3.9512E-03  1.561E-04
quads[1]  QFF1[K1]           ...   18.92   -1.0000E+30   3.0000E-01  0.000E+00
quads[2]  QDF1[K1]           ...   28.13   -1.0000E+30  -3.0000E-01  0.000E+00
Constraints                  ...   Ele/S  Meas-or-Lim    Model        Merit

 figure of merit: 2.700305E-02

! Top 10

Constraints                  ...   Ele/S   Meas-or-Lim    Model        Merit
disp[1]   eta.x <target>     ...   END       0.0000E+00   5.1814E-02  2.685E-02
disp[2]   deta_ds.x <target> ...   END       0.0000E+00   3.9512E-03  1.561E-04
quads[2]  QDF1[K1]           ...   28.13    -1.0000E+30  -3.0000E-01  0.000E+00
quads[1]  QFF1[K1]           ...   18.92    -1.0000E+30   3.0000E-01  0.000E+00
Constraints                  ...   Ele/S   Meas-or-Lim    Model        Merit

 figure of merit: 2.700305E-02
```

7. The optimizer can now be run:

```
Tao> run
Optimizing with: lm
Type ''.'' to stop the optimizer before it's finished.
[INFO] tao_dmodel_dvar_calc:
    Remaking dModel_dVar derivative matrix.
    This may take a while...

  Cycle      Merit     A_lambda
    1    7.0232E-05   1.00E-04
    2    4.0222E-05   1.00E-05
    3    1.2014E-05   1.00E-06
    ... etc...
  156    1.9259E-33   1.36E+10
Optimizer at minimum or derivatives need to be recalculated.
Written: var1.out
  157    1.9259E-33   0.00E+00
```

8. The **show data** and **show var** commands can be used to inspect the data and variables:

```
Tao> show data disp
Data name: disp.1
                                                                    |    Useit
                        ... Ele       Meas         Model        Design | Opt   Plot
1  eta.x <target>       ... END  0.0000E+00   1.3877E-17   5.1814E-02    T      F
2  deta_ds.x <target>   ... END  0.0000E+00   0.0000E+00   3.9511E-03    T      F

Tao> show var quads
Variable    Slave Parameters    Meas          Model         Design    Useit_opt
quads[1]    QFF1[K1]         0.0000E+00    3.1279E-01    3.0000E-01          T
quads[2]    QDF1[K1]         0.0000E+00   -3.1244E-01   -3.0000E-01          T
```

The dispersion and its slope are 0 or negligibly small so the optimization has been sucessful.

9. Use **write bmad dispsupF.bmad** to create a new lattice file named **dispsupF.bmad** with the optimized quadrupole strengths. This will be the dispersion suppressor for the forward sextants.

2.3 Exercises

1. **Reverse dispersion suppressor:** Construct the reverse dispersion suppressor, optimizing the last two quadrupole strengths for $\eta_x = \eta'_x = 0$ exactly at the end for a 90° phase advance. How do the two quadrupole values for the reverse dispersion suppressor compare to those obtained in class for the forward suppressor? Name the optimized lattice file name **dispsupR.bmad**.

2. **Forward and Reversed Cells:** Check that your forward and reverse cells that both start with focusing quads have different periodic beta and alpha functions. Check also that both cell, for the same phase advance of 90 degrees have exactly the same quadrupole strength. Explain how this can be the correct solution.

3. **Dispersion derivative:** *Tao* calculates a “dispersion derivative" and a “momentum dispersion" which for the x plane are **deta_ds.x** and **etap.x** respectively. What is the difference between the two? Show that with the lattice being designed they are the same. Hint: See the *Bmad* manual.

4. **Normal mode dispersion:** What is the difference between the horizontal η_x and vertical η_y dispersions and the normal mode dispersions η_a and η_b in the lattice being constructed? Verify your answer by checking the difference for one lattice element (Use the “show ele" command).

5. **Setting optimization parameters:** The **tao.init** file for this chapter sets the global parameter **global%n_opti_cycles** to 100 and this setting happens when Tao is started. Besides being able to set this parameter in the initialization file, this, and all other parameters, may be set from Tao’s command line. What is the command to set **global%n_opti_cycles** from the command line?

6. **Strength of bends:** [Warning! This is a difficult problem!] To a good approximation, a dispersion suppressor between from an arc FODO line to a straight FODO line (where the straight

FODO cells have the same layout as the arc FODO cells except that the bends are replaced by drifts) can be constructed using two arc FODO cells with the first cell having the bend strengths reduced by a factor α and the second cell with the bend strengths reduced by a factor $1 - \alpha$. In the case of 90° FODO cells, α is 0.5. Note: The approximation is only valid in the case of weak bends. Problem: Prove that this is the case and show that for 60° FODO cells the value of α is zero.

3 Dispersion Suppressor to Straight Section Twiss Parameter Matching

3.1 Introduction

The periodic betas in the arcs are different from the periodic betas in the straight sections. After suppressing the dispersion to zero, four quadrupoles after the dispersion suppressor are needed to match α_a, α_b, β_a, and β_b to the periodic betas in the following straight section.

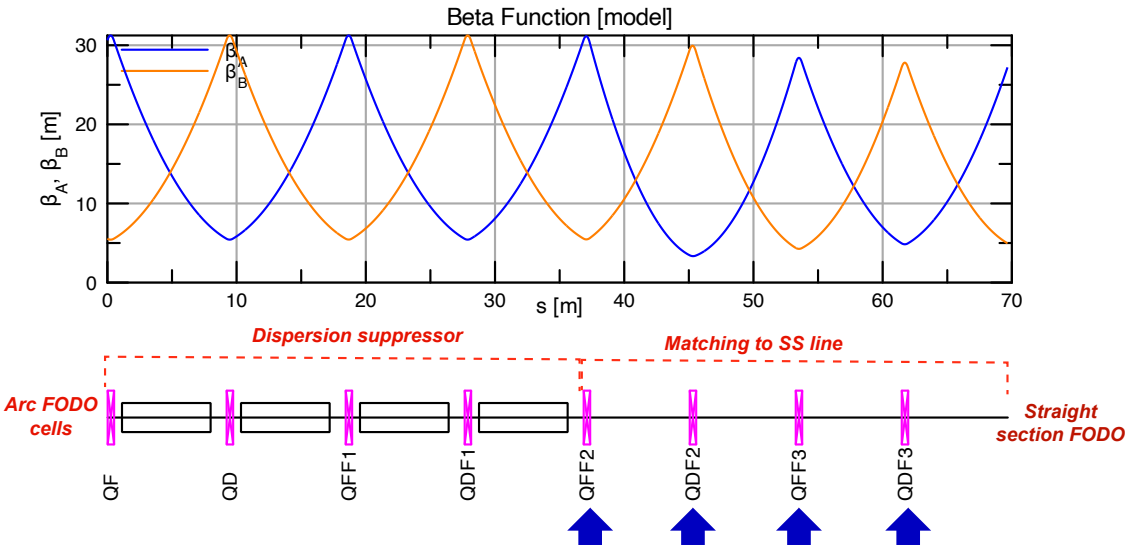


Figure 6: Dispersion suppressor match section.

3.2 Example: Forward Matching Section Creation

The procedure for creating the forward matching section is shown below. Matching the reverse matching section is left as an exercise. The example files can be found in the **/lattices/3_MatchToSS/F/** directory.

To create the forward matching section:

1. Create a copy of the **dispsupF.bmad** file, which contains the optimized dispersion suppressor.
2. In this new copy, add the straight section drift **DB**, which was defined previously in Exercises 2 and 3 of section 1.3.

```
! (2): Add straight section (SS) drift DB
DB: Drift, L = 5.855
```

3. Define four quads for matching to the straight section. Because this is the forward matching section (F), and these are the 2nd and 3rd special QF and QDs, use the names **QFF2**, **QDF2**, **QFF3**, and **QDF3**.

```
! (3): Define quads for matching to SS (two per plane,
!      use SS strengths as starting point)
QFF2: Quadrupole, L = 0.5, K1 = 0.351957452649287
QDF2: Quadrupole, L = 0.5, K1 = -0.351957452649287
QFF3: Quadrupole, L = 0.5, K1 = 0.351957452649287
QDF3: Quadrupole, L = 0.5, K1 = -0.351957452649287
```

4. Define the matching to the straight section (SS) line in Fig. 6 containing the matching quads. Because this is the forward matching to SS, Use the name **MSSF**.

```
! (4): Define the matching to SS line (MSSF)
MSSF: line = ( QFF2, D1, DB, D2, QDF2, D1, DB, D2,
               QFF3, D1, DB, D2, QDF3, D1, DB, D2)
```

"F" stands for Forward.

5. Use the name **ARC_TO_SSF** as the line containing the combined dispersion suppression and matching to SS line:

```
! (5): Define the dispersion suppression and
!       matching to SS line (ARC_TO_SSF)
ARC_TO_SSF: line = (DISPSUPF, MSSF)

use, ARC_TO_SSF
```

The completed initial lattice file containing these changes is provided in the example files, named **mSSF0.bmad**.

6. The last four quads need to be optimized so that at the end of the **ARC_TO_SSF** line the betas are equal to the periodic betas in the straight section FODO cells. Four datums, one for each of α_a, α_b, β_a, and β_b at the end of the line, need to be defined in the **tao.init** file. The "measure" (target) values of these datums can be found from the solution lattice **foDoSSF.bmad** obtained in Exercise 2 of Section 1.3. The variables will be the quadrupole strengths of **QFF2**, **QDF2**, **QFF3**, and **QDF3**. The completed **tao.init** file is provided in the example files.

7. Run the optimizer in *Tao* a couple times until the merit is extremely small. The merit will be of order 10^{-28}. The data should look like:

```
Tao> show data betas
Data name: betas.1
                                                                  |  Useit
                      ... Ele          Meas          Model         Design | Opt  Plot
1  beta.a <target>    ... END    2.7205E+01    2.7205E+01    3.0610E+01     T     F
2  alpha.a <target>   ... END   -2.4024E+00   -2.4024E+00   -2.4011E+00     T     F
3  beta.b <target>    ... END    4.9609E+00    4.9609E+00    5.5667E+00     T     F
4  alpha.b <target>   ... END    4.8460E-01    4.8460E-01    4.7804E-01     T     F
```

You can also verify the Twiss parameters are indeed what you need them to be at the **END** element using command **show element END**.

```
Tao> show ele END
Element # 33
Element Name: END
......
Twiss at end of element:
                          A              B
  Beta (m)        27.20598820     4.96091411   ...
  Alpha           -2.40249037     0.48460556   ...
  Gamma (1/m)      0.24891432     0.24891432   ...
......
```

8. Save the optimized quadrupoles strength by writing a new lattice file **mSSF.bmad**. This will be the matching section in the forward sextants.

3.3 Exercises

1. **Reversed matching section:** Construct the reverse matching section, following the same steps as shown in the example but for the reverse cells. How do your quad values compare to those obtained in class for the forward matching cells? Let the optimized lattice file name be **mSSR.bmad**.

2. **Merit Weight:** Optimization success or failure can depend heavily on the relative weights used for the data and the variables. As an example of this, start again with the unoptimized lattice and this time set the weight for the datum **beta[1]** to 1E10. This can be done in the lattice file or on the Tao command line with:

   ```
   set data betas[1]|weight = 1e10
   ```

   With this, how well does the optimization do?

4 Machine Coordinates

4.1 Coordinate Systems

As explained in the **Coordinates** chapter of the *Bmad* manual, bmad uses three coordinate systems to describe the positioning of lattice elements as shown in Figure 7:

Global Coordinates

The **(X, Y, Z) global** (also called **floor**) coordinate system (notice that capital letters are used for the coordinate axes) is independent of the accelerator machine and is "attached" to the building the accelerator is in. Typically, the **Y**-axis is taken to be pointing vertically up and **(X, Z)** defines the horizontal plane.

Local Coordinates

The **global** coordinate system is not convenient for describing where particles are as they move through the lattice. For this, shown in red, there is the **local** (also called **laboratory**, also called **reference**) curvilinear coordinate system. Laboratory coordinates are also used to describe the nominal (that is, without any "misalignments") position of the lattice elements. The laboratory coordinates start with a curved line often called the "**reference trajectory**". The distance along the reference trajectory is denoted by s. At each point along the reference trajectory, there is a Cartesian **(x, y, z)** coordinate system with the origin at the reference trajectory point. The z-axis is always tangent to, and the x and y-axes always transverse to the reference trajectory.

Element Body Coordinates

Elements can be shifted ("misaligned") from their nominal position. To describe things like electric and magnetic fields or apertures (which can depend upon an element's actual position), **element body** coordinates are used. The **element body** coordinates are the coordinates attached to the physical element. Without any "misalignments", the **element** coordinates correspond to the **laboratory** coordinates.

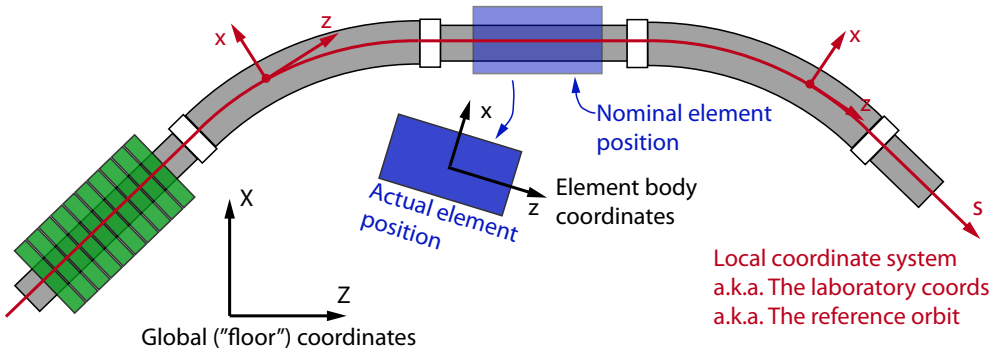


Figure 7: The three coordinate systems used to describe lattice element positioning: **Global**, **reference**, and **element body** coordinates.

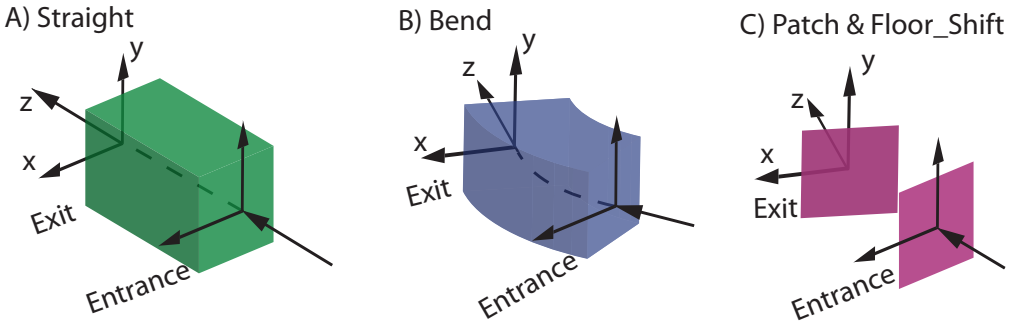


Figure 8: Lattice element geometry types: **Straight**, **bend**, and **patch**. All elements have an **entrance** coordinate system and an **exit** coordinate system.

4.2 Element Geometry Types

All lattice elements have an "**entrance**" end and an "**exit**" end. Normally a particle will enter the element at the **entrance** end and exit at the **exit** end but it is possible to simulate particles going backwards or have lattice elements that are reversed longitudinally.

Lattice elements in *Bmad* have one of four geometry types. Three of them will be discussed here and are shown in Figure 8. [The fourth geometry type, used for X-ray simulations, is used with **mirror**, **multilayer_mirror**, and **crystal** elements.] These three types are called **straight**, **bend** and **patch** based upon how the (x, y, z) laboratory coordinates transform as a function of the longitudinal **s** position from the **entrance** end of the element to the **exit** end.

Straight Geometry

The straight geometry, shown in Figure 8A, is used with elements like **drifts** and **quadrupoles**. Within any one given element, the orientation of the (x, y, z) coordinates is independent of s and the reference trajectory is a straight line so that the **z**-axis at the **exit** end is co-linear with the **entrance z**-axis.

Bend Geometry

The bend geometry shown in Figure 8B is used with **sbend** and **rbend** dipole elements. With this geometry, the reference trajectory is a semi-circle. The (x, y, z) coordinates rotate about an axis which is perpendicular to the z-axis. The default is to have the rotation axis parallel to the y-axis which keeps the orientation of the y-axis independent of s.

Patch Geometry

The patch geometry shown in Figure 8C is used with **patch** and **floor_shift** elements. With the **patch** geometry, the exit coordinates can be arbitrarily positioned with respect to the entrance coordinates. See section §4.6. The reference trajectory within the patch region is undefined since there is no good way to define it.

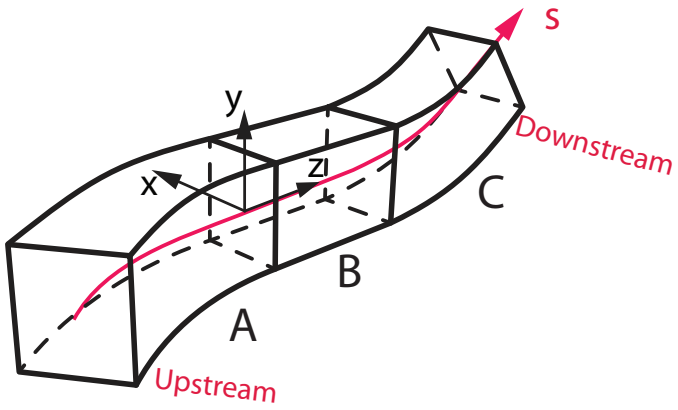


Figure 9: The **local** coordinate system is constructed by taking the ordered list of lattice elements and connecting the **exit** frame of one element to the **entrance** frame of the next.

4.3 Local Coordinate System Construction

The **local** coordinate system is constructed by taking the ordered list of lattice elements and connecting the **(x, y, z) exit** frame of one element to the **(x, y, z) entrance** frame of the next (just like LEGO blocks). Given a line constructed as:

```
lat: line = (A, B, C)
```

The result could look as shown in Figure 9. The reference trajectory is shown in red.

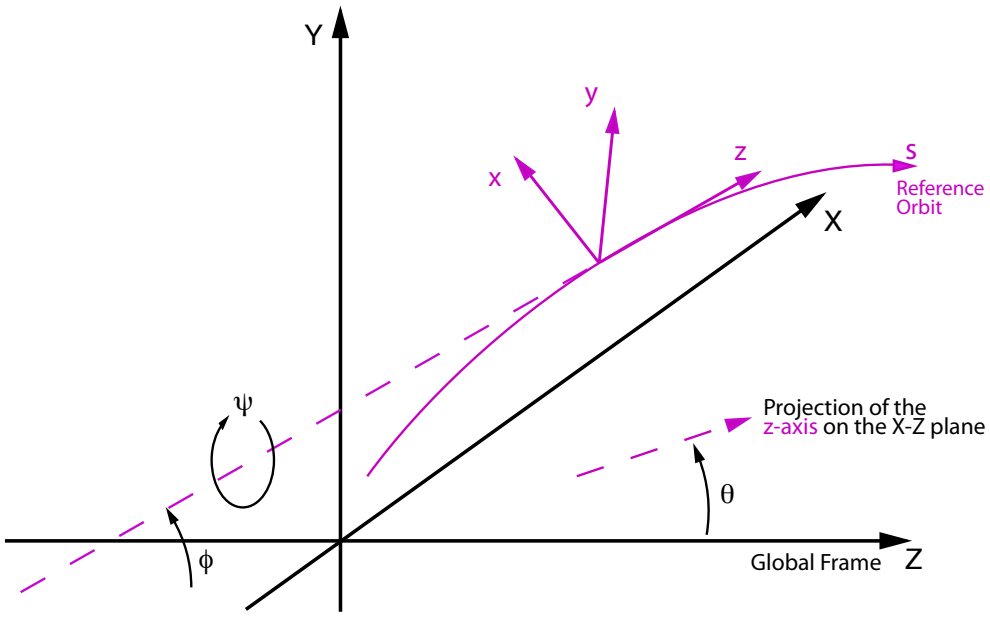


Figure 10: The global coordinate system.

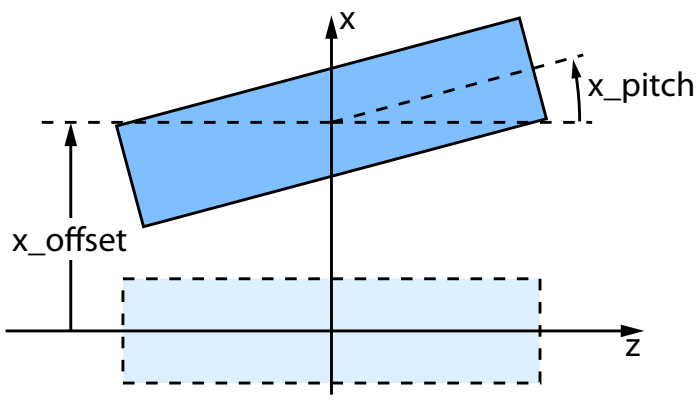
4.4 Laboratory Coordinates Relative to Global Coordinates

For any given **s**-position on the reference orbit, the **local** coordinate system is described with respect to the **global** coordinate system by 6 parameters as shown in Figure 10:

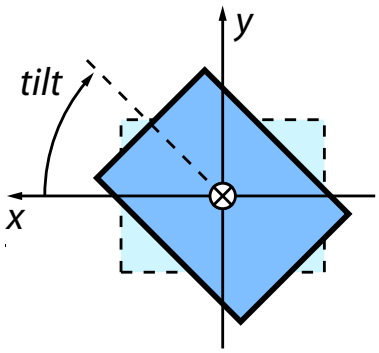
- (**X**, **Y**, **Z**) global position
- θ azimuth angle in the (X, Z) plane.
- ϕ elevation angle
- ψ roll angle.

Notes:

- The default is for the beginning of the lattice (**s** = 0) is to have the local **(x, y, z)** coordinate system aligned with the global **(X, Y, Z)** coordinate system with θ, ϕ and ψ all being zero.
- For a machine that lies in the horizontal plane, the $\phi(s)$ and $\psi(s)$ angles are zero for all **s**.



(a) Effect of x_offset and x_pitch on a straight line element



(b) Effect of a tilt on a straight line element.

Figure 11

4.5 Element Misalignments

Once the **reference** coordinate system is established, the position of any physical element can be shifted ("misaligned"). [Note: **Patch** and **floor_shift** elements cannot be misaligned.] For straight elements, the element attributes that determine any misalignment are:

x_offset, y_offset, z_offset
: The **x_offset**, **y_offset**, and **z_offset** attributes offset the element in the **x**, **y**, and **z** directions respectively. See Figure 11a.

x_pitch, y_pitch
: The **x_pitch** and **y_pitch** attributes rotate the element. A **x_pitch** of $\pi/2$ would rotate the element around the +**y**-axis so that the body +**z**-axis is aligned with the local +**x**-axis. Similarly, a **y_pitch** of $\pi/2$ would rotate the element around the -**x**-axis so that the body +**z**-axis is aligned with the local +**y**-axis. See Figure 11a.

tilt
: A **tilt** rotates the element around the +**z**-axis as shown in Figure 11b

Note: The above only applies to straight elements. Patch like elements are explained below. For a discussion of misalignments for bend type elements see the *Bmad* manual.

```
! Lattice File: misalign.bmad
beginning[beta_a] = 10.    ! m  a-mode beta function
beginning[beta_b] = 10.    ! m  b-mode beta function
beginning[e_tot] = 10e6    ! eV
parameter[geometry] = open  ! or closed
q: quadrupole, L = 1, x_offset = 0.1, x_pitch = 0.04
lat: line = (q)    ! List of lattice elements
use, lat           ! Line used to construct the lattice
```

Start *Tao* with the lattice file **misalign.bmad**. The misalignment can be viewed using the -**floor** option with the **show element** command:

```
Tao> show ele q -floor

 Element #                1
 Element Name: Q
... etc ...

 Attribute values [Only non-zero/non-default values shown]:
    1  L                =  1.0000E+00 m
   13  SPIN_FRINGE_ON   =  T (1)
   31  L_HARD_EDGE      =  1.0000E+00 m
   34  X_PITCH          =  4.0000E-02        55  X_PITCH_TOT    =  4.0000E-02
   36  X_OFFSET         =  1.0000E-01 m      57  X_OFFSET_TOT   =  1.0000E-01 m
... etc ...

Global Floor Coords at End of Element:
                X        Y        Z     Theta
Reference  0.00000  0.00000  1.00000  0.00000 ... ! Without misalignments
Actual     0.11999  0.00000  0.99960  0.04000 ... ! With misalignments
... etc ...
```

In the "**Global Floor Coords**" section, the **Reference** row shows the nominal position of the **exit** end of the element without misalignments. [Due to space constraints the **phi** and **psi** columns are not shown. They are zero in this case.] The **Actual** row shows the position of the physical element at the **exit** end.

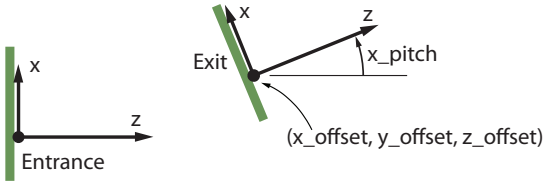
Associated with each misalignment attribute there is a corresponding attribute with a "**_tot**" suffix. The difference is that an attribute like **x_offset** is the misalignment with respect to any **girder** that may be supporting it while the corresponding **x_offset_tot** is the total misalignment of the lattice element with respect to the element's nominal position. Another difference is that misalignments attributes are set by the user while the corresponding **_tot** attributes are calculated by *Bmad*. If there is no **girder** support, the **_tot** attributes will be the same as the misalignment attributes as it is in this case so **x_pitch** is equal to **x_pitch_tot**, etc.

4.6 Patch Elements

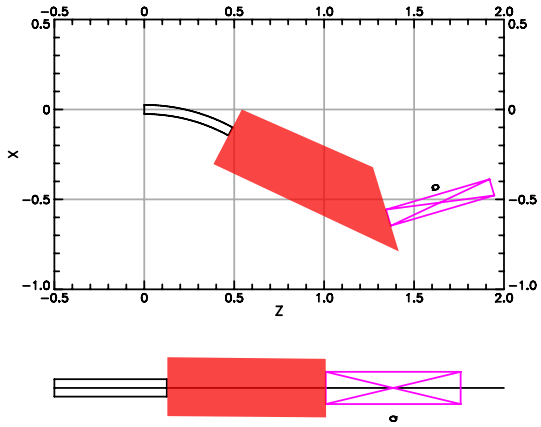
Patch elements are used to shift the reference orbit. As a consequence, the nominal placements of all elements downstream of the patch are affected. This is useful in simulating things like injection or extraction lines where the patch is used to reorient the reference orbit so that it follows the injection or extraction line.

For **patch** elements the same six parameters that are used to misalign straight line elements are, for a **patch**, used to set the placement of the **exit** frame relative to the **entrance** frame. The transformation from entrance coordinates to exit coordinate is:

1. Initially the exit coordinates coincide with the entrance coordinates.
2. The origin of the exit coordinates is translated by (**x_offset**, **y_offset**, **z_offset**)



(a) The body coordinates at the exit end of a **patch** is set by the element attributes **x_offset**, **y_offset**, **z_offset**, **x_pitch**, **y_pitch**, and **tilt**.



(b) Lattice with a patch element. The patch element is the coordinates patch element in a lattice. [Note: The default is not to draw **patch** elements in a **floor_plan** plot.]

Figure 12

3. The **x_pitch** and **y_pitch** rotations (in radians) are applied. The **x_pitch** rotation rotates the +z axis towards the +x axis (rotation around the +y axis). The **y_pitch** rotation rotates the +z axis towards the +y axis (rotation around the -x axis).

4. The **tilt** rotation (in radians) rotates the exit coordinates around the exit coordinate's +z axis.

This transformation is illustrated in Figure 12a. The transformation from **patch entrance** to **exit** coordinates is the same transformation from laboratory coordinates at the center of a straight element to the element body coordinates at the center of the misaligned element.

4.7 Example:

```
! Lattice File: patch.bmad
beginning[beta_a] = 10.    ! m  a-mode beta function
beginning[beta_b] = 10.    ! m  b-mode beta function
beginning[e_tot] = 10e6    ! eV
parameter[geometry] = open  ! or closed

b: sbend, L = 0.5, g = 1     ! g = 1 / bending_radius
p: patch, z_offset = 1, x_pitch = pi/4
q: quadrupole, L = 0.6, k1 = 0.23

lat: line = (b, p, q)    ! List of lattice elements
use, lat                 ! Line used to construct the lattice
```

Start *Tao* with the lattice file **patch.bmad**. Create a **floor_plan** with the command **place r11 floor**. The result is shown in Figure 12b except that, by default, *Tao* does not draw a patch element so, in the figure, the patch has been drawn in by hand. The global coordinates of the nominal positions of the elements can be seen by using the **show lat -floor** command:

```
Tao> show lat -floor

      Values at End of Element:
Ix  name       key              s          X          Y          Z      Theta ...
 0  BEGINNING  Beginning_Ele  0.000     0.0000     0.0000     0.0000     0.0000 ...
 1  B          Sbend          0.500    -0.1224     0.0000     0.4794    -0.5000 ...
 2  P          Patch          1.207    -0.6018     0.0000     1.3570     0.2854 ...
 3  Q          Quadrupole     1.807    -0.4329     0.0000     1.9327     0.2854 ...
 4  END        Marker         1.807    -0.4329     0.0000     1.9327     0.2854 ...
```

A **patch** represents a field free space so a particle traveling through a patch propagate as in a drift. The difference is that in a **patch** there is a coordinate transformation from entrance coordinates to exit coordinates.

4.8 Exercises:

1. Modify the lattice file **simple.bmad** to include a **girder** element supporting elements **B** and **Q**. Misalign the **girder** and verify that a supported element will have **_tot** attributes different from the misalignment attributes.

2. The **show ele -floor** command will show the global coordinates of both the nominal (non-misaligned) position as well as the misaligned (actual) position of an element. For example

   ```
   show ele Q -floor
   ```

   On the *Tao* command line, misalign any element using the **set ele** command and observe what the differences are between the nominal and misaligned positions. Can you predict what the difference will be between the nominal and misaligned positions when you change the **tilt** of **Q**?

3. Using lattice **simple.bmad**, calculate by hand the floor coordinates at the exit end of element Q and compare this with the coordinates as calculated by Bmad.

5 Constructing the Ring

5.1 Ring Construction

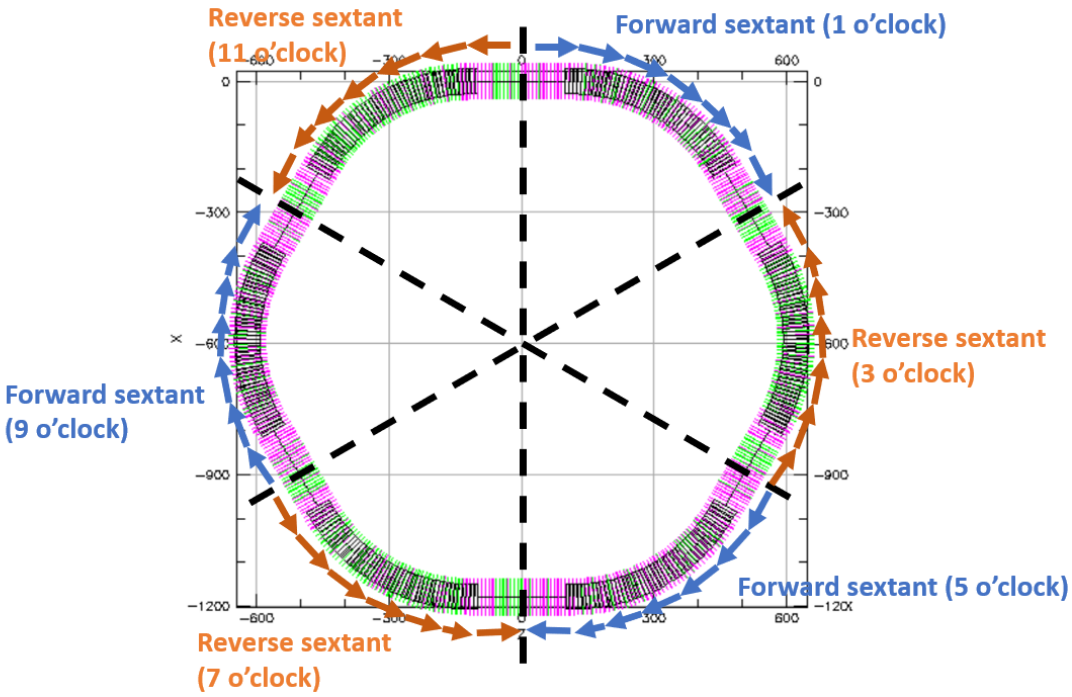


Figure 13: Mirror symmetry about the centers of each straight.

In the previous chapters, forward and reverse arc lines, forward and reverse straight section lines, and forward and reverse lines to connect the arcs to the straight sections (with dispersion suppression and matching the betas) were constructed. In this chapter, the entire ring is created based on Fig. 13. All that's left is a section to match the straight section to the arc, including dispersion "creation" and betas matching, as shown in Fig. 14.

Does this mean a dispersion "creator" for both the forward and reverse sextants needs to be created? Fortunately, no! Considering the mirror symmetry shown in Fig. 13, the quadrupole settings for the forward sextant dispersion suppression and matching are equal to the quadrupole settings for the reverse dispersion *creation* and matching. Likewise, the reverse dispersion suppression and matching quadrupole settings are equal to the forward dispersion creation and matching section quadrupoles. Therefore, no optimization is necessary here. All that is needed is to connect the lines together.

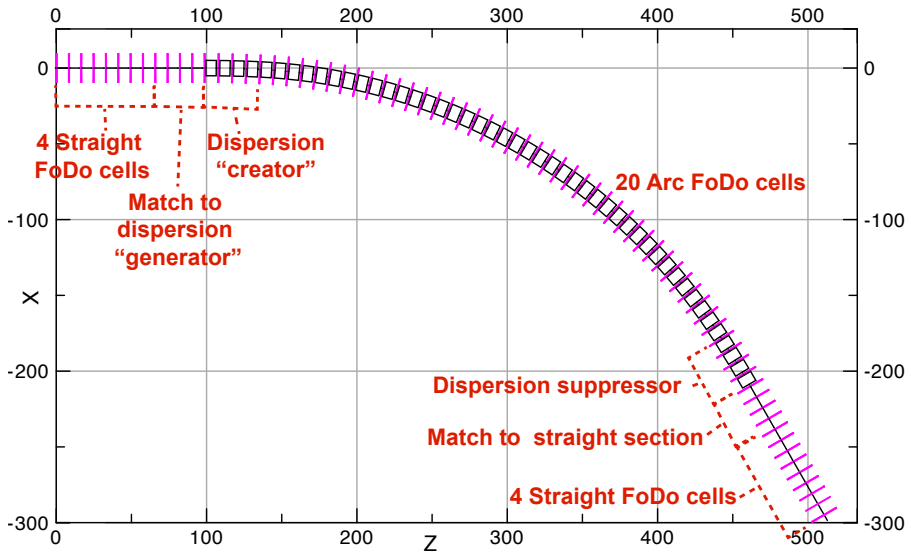


Figure 14: A sextant of the ring.

5.2 Example: Forward Straight Section to Arc

As an example, the forward line to connect the straight section to the arc, **SS_TO_ARCF**, will be constructed. The example lattice file can be found in **/lattices/5_Ring/**. Construction of **SS_TO_ARCR** and the entire ring will be left as an exercise.

Following Fig. 14, both the matching to dispersion creator line (which we'll call **MDCF**), and the dispersion creator line itself (which we'll call **DISPCREF**) need to be construted. Using the mirror symmetry as shown in Fig. 13, it is seen that **SS_TO_ARCF** will use the same quadrupole settings as in **ARC_TO_SSR**. Keeping with our convention of starting each line with a full horizontally-focusing quadrupole, this is

```
! Match forward straight section to dispersion "creator" (use QFR)
MDCF: line = (QFSS, D1, DB, D2, QDR3, D1, DB, D2,
              QFR3, D1, DB, D2, QDR2, D1, DB, D2)

! Forward dispersion "creator" (use QFR)
DISPCREF: line = (QFR2, D1, BH, D2, QDR1, D1, BH, D2,
                  QFR1, D1, BH, D2, QD, D1, BH, D2)

SS_TO_ARCF: line = (MDCF, DISPCREF)
```

After matching the straight sections to the arc, it is now time to contruct the full ring.

5.3 Exercises

1. **Construct the reverse straight section to arc lines:** Construct **SS_TO_ARCR** using the example shown above and Figs. 13 and 14.

2. **Construct the ring**

   2.1. Gather all elements from each *optimized* file

   2.2. Gather the arc and SS FODO cells

```
! (2): Gather arc and SS FoDo cells

! Straight section forward FoDo:
FODOSSF: line = ( QFSS, D1, DB, D2, QDSS, D1, DB, D2)

! Arc forward FoDo:
FODOAF: line = (QF, D1, B, D2, QD, D1, B, D2)

! Arc reverse FoDo:
FODOAR: line = (QF, D2, B, D1, QD, D2, B, D1)

! Straight section reverse FoDo:
FODOSSR: line = ( QFSS, D2, DB, D1, QDSS, D2, DB, D1)
```

   2.3. Gather dispersion suppression and matching to SS lines (ARC_TO_SSF, ARC_TO_SSR)

```
! (3): Gather dispersion suppression
!      and matching to SS lines (ARC_TO_SSF, ARC_TO_SSR)

! Forward dispersion suppressor:
DISPSUPF: line = (QF, D1, BH, D2, QD, D1, BH, D2,
                  QFF1, D1, BH, D2, QDF1, D1, BH, D2)

! Match forward dispersion suppressor to SS:
MSSF: line = ( QFF2, D1, DB, D2, QDF2, D1, DB, D2,
               QFF3, D1, DB, D2, QDF3, D1, DB, D2)

ARC_TO_SSF: line = (DISPSUPF, MSSF)

! Reverse dispersion suppressor:
DISPSUPR: line = ( QF, D2, BH, D1, QD, D2, BH, D1,
                   QFR1, D2, BH, D1, QDR1, D2, BH, D1)

! Match reverse dispersion suppressor to SS:
MSSR: line = ( QFR2, D2, DB, D1, QDR2, D2, DB, D1,
               QFR3, D2, DB, D1, QDR3, D2, DB, D1)

ARC_TO_SSR: line = (DISPSUPR, MSSR)
```

   2.4. Gather matching to dispersion creation and dispersion creation lines (**SS_TO_ARCF**, **SS_TO_ARCR**) from the Example and first exercise in this section

   2.5. Build the ring!

```
! (5): Build the ring!

SEXTANT1:  line = (4*FODOSSF, SS_TO_ARCF, 20*FODOAF, ARC_TO_SSF, 4*FODOSSF)
SEXTANT3:  line = (4*FODOSSR, SS_TO_ARCR, 20*FODOAR, ARC_TO_SSR, 4*FODOSSR)
SEXTANT5:  line = (4*FODOSSF, SS_TO_ARCF, 20*FODOAF, ARC_TO_SSF, 4*FODOSSF)
SEXTANT7:  line = (4*FODOSSR, SS_TO_ARCR, 20*FODOAR, ARC_TO_SSR, 4*FODOSSR)
SEXTANT9:  line = (4*FODOSSF, SS_TO_ARCF, 20*FODOAF, ARC_TO_SSF, 4*FODOSSF)
SEXTANT11: line = (4*FODOSSR, SS_TO_ARCR, 20*FODOAR, ARC_TO_SSR, 4*FODOSSR)

RING: line = (SEXTANT1, SEXTANT3, SEXTANT5, SEXTANT7, SEXTANT9, SEXTANT11)
use, RING
```

3. **Check that the ring closes geometrically** using either **show element end -floor** or **show element end -all**. What is the circumference of the ring?

4. **Use Tao's place command** to display a **floor_plan** drawing (Figs 13 and 14 are **floor_plan** drawings). First, use **place r11 floor_plan** to place the floor template in the plot region (use **show plot** to display the region). Then, try to adjust the drawing boundaries using **scale**, **scale all**, and **x_scale all** commands.

6 Low-Beta Interaction Region Insertion

6.1 Introduction

For IR simulations *Bmad* has a **Beam-Beam** lattice element for simulating the fully-nonlinear weak-strong beam-beam effect. *Bmad* also has a **crab_cavity** element to simulate crabbing with a non-zero crossing angle. Discussion of this is beyond the scope of this tutorial.

In this chapter an Interaction Region (IR) with a low beta Interaction Point (IP) will be constructed by modifying the 6 o'clock straight section quadrupoles as shown in Figs. 15A and 15B.

The IR is divided into two parts which are constructed separately as shown in Fig. 16. IPF is the upstream "forward" section before the IP (labeled IP6) and IPR is the downstream "reverse" section after IP6. At the IP, the betas are to be optimized to the values $\beta_a^* = 0.6$ and $\beta_b^* = 0.06$, and, as standard for many IPs, the alpha values α_a^*, and α_b^* will be optimized to zero. The surrounding quadrupoles will be adjusted to match these IP parameters while still matching to the periodic solution at the ends of the sections furthest from the IP. Because this region is particularly sensitive, this match is achieved with the following steps:

1. Match the forward line from SS to IP
2. Match *backwards* the reverse line from SS to IP
3. Connect the two and match the full line to the periodic solution once more to ensure a perfect match

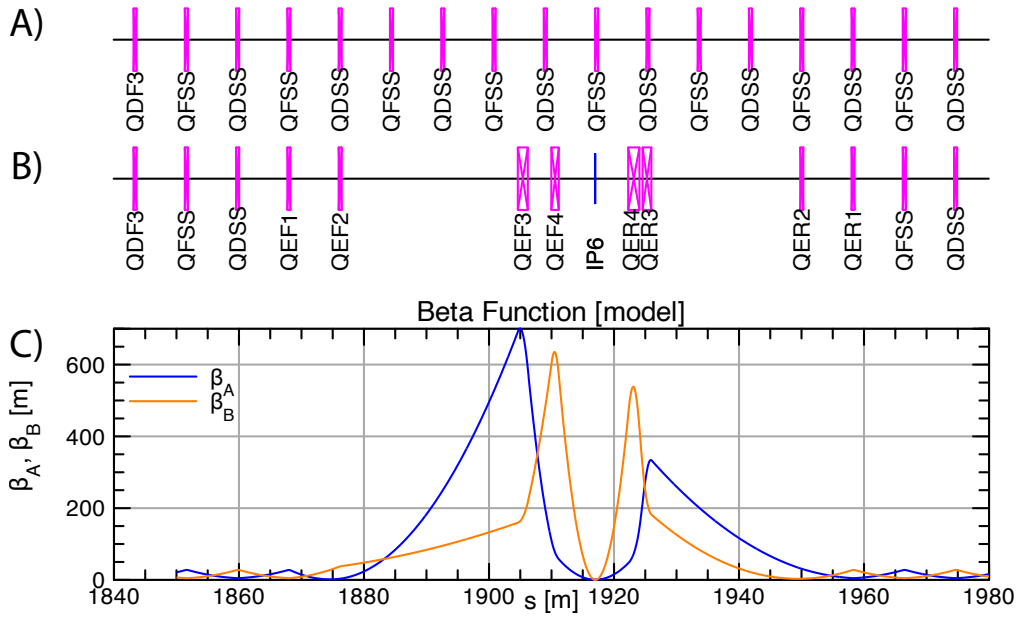


Figure 15: A) Layout before modification. B) Layout with IR constructed. C) IR Beta functions.

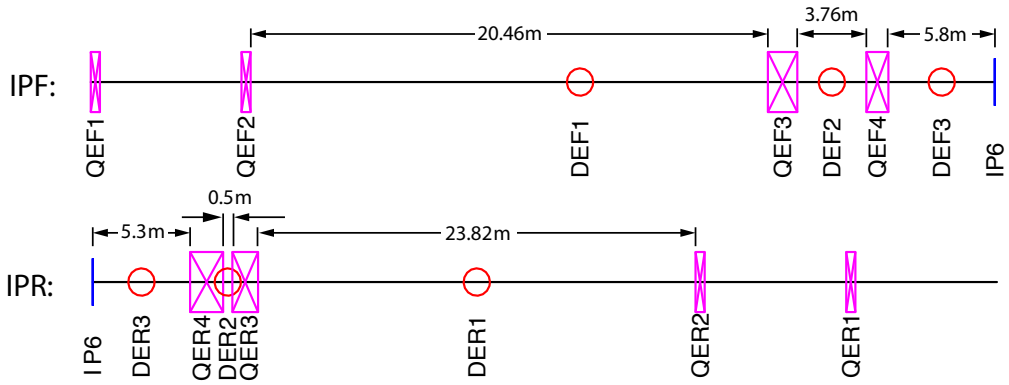


Figure 16: Interaction region dimensions. Red circles mark drifts with non-standard lengths. IPF: Forward section before the IP. IPR: Reverse section after the IP.

6.2 Example: Constructing the Upstream and Downstream IR Lines

The example lattice file can be found in **/lattices/6_IP/a_IP0/**.

Construct the upstream and downstream IR lines, shown on the top and bottom of Fig. 16 respectively, and leave the upstream, downstream, and full IR matching described in the introduction as exercises.

To construct the IR lines:

1. Start with **fodoSSF.bmad**, the optimized forward straight-section lattice file
2. Set geometry to open

```
parameter[geometry] = open    ! (2): Set geometry to open
```

3. Define forward (upstream) interaction region drifts and quads (strengths TBD, start with SS quadrupole strengths)

```
! (3): Define forward (upstream) interaction region drifts and
!       quads (strengths TBD, start with SS)
QEF1: Quadrupole, L = 0.5, K1 = 0.351957452649287
QEF2: Quadrupole, L = 0.5, K1 = -0.351957452649287
DEF1: Drift, L = 20.46
QEF3: Quadrupole, L = 1.6, K1 = 0.351957452649287
DEF2: Drift, L = 3.76
QEF4: Quadrupole, L = 1.2, K1 = -0.351957452649287
DEF3: Drift, L = 5.8
```

4. Define the interaction point element. Use a **Marker** element for this. **Markers** have zero length and have no impact on the orbit or spin transport.

```
! (4): Define IP
IP6: marker
```

5. Define reverse (downstream) interaction region drifts and quads (strengths TBD, start with SS quadrupole strengths)

```
! (5): Define reverse (downstream) interaction region drifts and
!       quads (strengths TBD, start with SS)
DER3: Drift, L = 5.3
QER4: Quadrupole, L = 1.8, K1=-0.351957452649287
DER2: Drift, L = 0.5
QER3: Quadrupole, L = 1.4, K1=0.351957452649287
DER1: Drift, L = 23.82
QER2: Quadrupole, L = 0.5, K1 = 0.351957452649287
QER1: Quadrupole, L = 0.5, K1 =-0.351957452649287
```

6. Define the upstream and downstream lines

```
! (6): Define the upstream and downstream lines
IPF: line = ( QEF1, D1, DB, D2, QEF2, D1, DB,
              D2, DEF1, QEF3, DEF2, QEF4, DEF3, IP6)
IPR: line = (IP6, DER3, QER4, DER2, QER3, DER1,
             QER2, D2, DB, D1, QER1, D2, DB, D1)
```

After defining the two IP lines, the exercises will focus on region matching and optimization of our updated ring.

6.3 Exercises

1. **Forward (upstream) interaction region matching:** Using the **IPF** line defined above, vary **QEF1**, **QEF2**, **QEF3**, and **QEF4** so that at **IP6** $\beta_a = 0.6$, $\beta_b = 0.06$, $\alpha_a = \alpha_b = 0$. After optimizing, instead of **write**-ing the lattice, make a copy of the **var1.out** file and rename it to something like **IPF.out**. This output file is a Bmad lattice file that sets the quadrupoles to their correct strengths.

2. **Reverse (downstream) interaction region matching:** Match *backwards* the downstream interaction region line **IPR** defined above, starting from the end of the line and ending at the interaction point, varying **QER1**, **QER2**, **QER3**, and **QER4** that $\beta_a = 0.6$, $\beta_b = 0.06$, $\alpha_a = \alpha_b = 0$ at **IP6**. Once again, save a copy of the final **var1.out** file containing the optimized quadrupole strengths and rename it to something like **IPR.out**. Hint: the line can be reflected using a negative sign:

```
IPR_reflected: line = (-IPR)
```

3. **Call command:** The optimized quadrupole settings in the **.out** files obtained above can be called directly in the lattice file using the **call** command at the end of the lattice file. *Bmad* will read in the entire lattice file, and then set the quadrupole strengths accordingly. E.g., at the end of the lattice file:

```
! Define and use IP line
IP: line = (IPF,IPR)
use, IP
```

```
! Call the quadrupole settings in IPF.out and IPR.out
call, file = "IPF.out"
call, file = "IPR.out"
```

This should give a very good match, however to ensure a perfect solution, the full line to the periodic solution is matched once more. Using the full IR line (**(IPF, IPR)**) now, vary the downstream quadrupoles **QER1**, **QER2**, **QER3**, and **QER4** to match perfectly to the periodic betas in the straight section at the end of the line. Now you can use **write bmad IP.bmad** with the fully optimized IR quadrupole settings

4. **Add the IR into the ring:** Copy the optimized lattice elements from **IP.bmad** into the ring lattice file, copy the **IPF** and **IPR** lines into the ring lattice file, and finally add the IP into the ring by modifying **SEXTANT5** and **SEXTANT7**:

```
SEXTANT5: line = (4*FODOSSF, SS_TO_ARCF, 20*FODOAF, ARC_TO_SSF, FODOSSF, IPF)
SEXTANT7: line = (IPR, FODOSSR, SS_TO_ARCR, 20*FODOAR, ARC_TO_SSR, 4*FODOSSR)
```

5. **Alpha and Beta Functions:** How should you change the initial betas and alphas for a proper match using the reflected line? Hint: the betas and alphas are the same as calculated in earlier chapters.

6. **Importance of variable step size:** In Exercise 1, you optimized the match the IP Twiss using a **step** size for varying the quadrupole **k1** values of 10^{-6}. How is this step size used with the Levenberg-Marquardt optimization? Verify that if, instead, a step size of 10^{-4} is used the optimization does not converge to a usable solution. Why is this?

7 Tune Cell

7.1 Introduction

In order to adjust the tune of the ring, the phase advance per FODO cell in one of the straight sections can be adjusted, while ensuring it is still matching to the periodic solution in each of the surrounding arcs. We'll put this "tune cell" in the 2 o'clock straight section, and modify the lines/lattice elements by simply appending a **_2** to the names.

Required for the optimization is:

1. Setting the phase advances in the FODOSSF/FODOSSR straight section for desired tunes $Q_x = 54.08$ and $Q_y = 54.14$ (2 constraints)
2. Matching from the dispersion suppressor to the new periodic betas and alphas in the center FODOSSF/FODOSSR section (4 constraints)
3. Matching the center FODOSSF/FODOSSR section to the dispersion creator (would be 4 constraints but these are defined already by the mirror symmetry)

These 6 constraints can be satisfied using the four quadrupoles in the forward matching to SS (**QFF2_2**, **QDF2_2**, **QFF3_2**, and **QDF3_2**), and the two quadrupoles in the straight section (**QFSS_2** and **QDSS_2**).

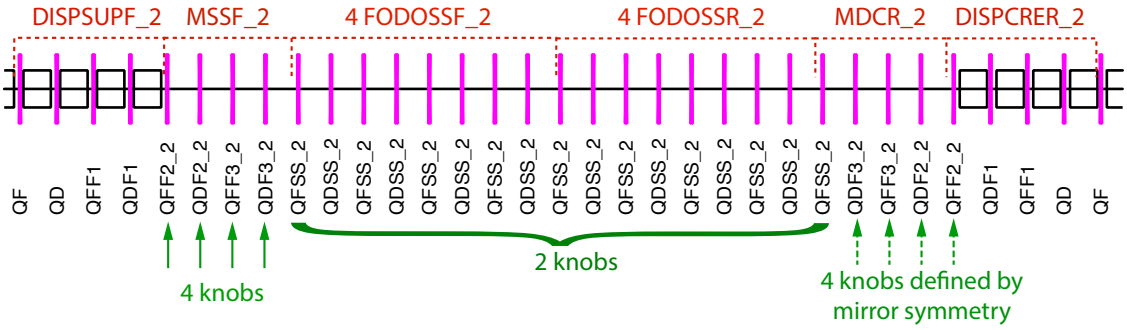


Figure 17: Enter Caption

7.2 Example: tao.init for the Tune Cell

The example file can be found at **/lattices/7_TuneCell/tao.init**.

To define datums that constrain the solution to have periodic betas in the straight section, **expression** datums are used. Expressions are an extremely powerful tool that allows for complex lattice optimizations and matching techniques.

In this case, required is that the betas at one focusing quadrupole in the center FODO cells are equal to the betas at the next focusing quadrupole. Equivalently, required is that the difference of those two are equal to zero. The appropriate datums are

```
! Require periodic betas in center FoDo cells of 2 o'clock tune cell
datum(1) = 'expression: lat::beta.a[QFSS_2##4] - lat::beta.a[QFSS_2##5]'
                                                  '' '' '' 'target' 0 10
datum(2) = 'expression: lat::beta.b[QFSS_2##4] - lat::beta.b[QFSS_2##5]'
                                                  '' '' '' 'target' 0 10
datum(3) = 'expression: lat::alpha.a[QFSS_2##4] - lat::alpha.a[QFSS_2##5]'
                                                   '' '' '' 'target' 0 10
datum(4) = 'expression: lat::alpha.b[QFSS_2##4] - lat::alpha.b[QFSS_2##5]'
                                                   '' '' '' 'target' 0 10
```

The syntax **lat::beta.a[QFSS_2##4]** evaluates to the value of the **a**-mode beta function at the fourth lattice element named **QFSS_2**. The above datums have a **measured** (target) value of zero so that the contribution to the merit function will be zero when the differences of the betas at the fourth **QFSS_2** and at the fifth **QFSS_2** are zero. Thais in, when they are equal. This ensures periodicity in the optics in the straight section.

Besides the above datums, other datums are need to make sure that the outgoing Twiss from the modified straight section correctly match the Twiss at the start of the dispersion creator **DISPCRER_2** (see Fig. 17. The needed datums are:

```
! Match betas to dispersion creator after tune cell
datum(5) = 'beta.a' '' '' 'QFF2_2##2' 'target' 30.6104309489717465  10
datum(6) = 'beta.b' '' '' 'QFF2_2##2' 'target'  5.56679467017438689 10
datum(7) = 'alpha.a' '' '' 'QFF2_2##2' 'target' 2.40115683012980607 100
datum(8) = 'alpha.b' '' '' 'QFF2_2##2' 'target'   -.47804044385806721 100
```

Finally, datums are needed to set the optimized tune to the desired values:

```
! Set phase advance to nearest desired fractional tunes (54.08, 54.14)
datum(9) = 'phase.a' '' '' 'end' 'target' 339.79466141227203 10
datum(10) = 'phase.b' '' '' 'end' 'target' 340.17165253070283 10
```

7.3 Exercises

1. **Implement the tune cell:** Add the new special quadrupoles shown in Fig. 17, and optimize them using the **tao.init** file created in the example so the lattice has the correct tunes $Q_x = 54.08$ and $Q_y = 54.14$.

8 Particle Phase Space Coordinates

8.1 Phase Space

Figure 18a shows the reference orbit (laboratory coordinate system) which was discussed in chapter 4. The reference coordinates (x, y, z) associated with a given point a distance s along the reference point has the coordinate origin at the given point and the z-axis tangent to the reference orbit.

A particle being simulated has its own trajectory as shown in the figure. Given a particle at some point on its trajectory (blue dot in Figure 18a), there is a point at position **s** on the reference orbit such that the z coordinate of the particle is zero in the (**x**, **y**, **z**) coordinate frame associated with the reference orbit point.

With this, the particle's position and momentum **P** can be described using the coordinates:

```
(x(s), y(s), Px(s), Py(s), Pz(s), t(s))
```

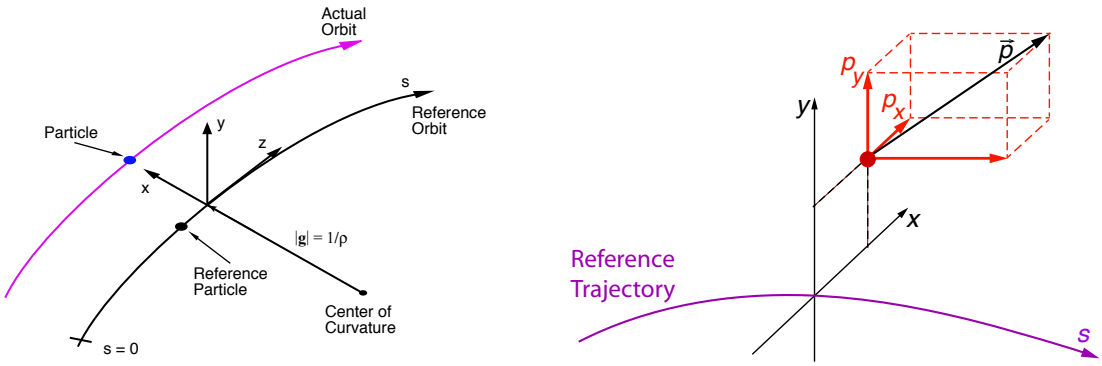
where **t(s)** is the time that the particle is at the given point. From now on, to simplify the notation, the **s** dependence will be dropped.

For tracking purposes, canonical phase space coordinates are used with the convention that upper case **P** denotes (unnormalized) momentum (Figure 18b) and lower case **p** denotes phase space momentum. The phase space coordinates are denoted

```
(x, px, y, py, z, pz)
```

where

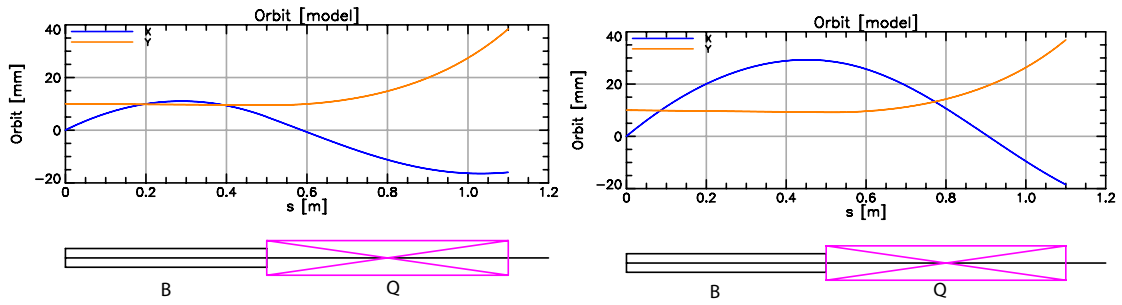
```
px = Px / P0
py = Py / P0
pz = (P - P0) / P0
```



(a) Particle coordinate positions are relative to the reference orbit.

(b) Particle phase space.

Figure 18



(a) Initial orbit.

(b) Orbit after adjusting the starting **px** phase space coordinate.

Figure 19

```
z = c * β * (t_ref - t)
```

with

- **P0** is the reference momentum. See section §9.1.
- **c * β** is the velocity of the particle,
- **t_ref(s)** is the time the reference particle reaches the point **s**. The reference particle is a fictitious particle that can be imagined to be traveling on the reference orbit. Frequently, this reference particle is thought of as describing the center of a bunch of particles.

Notes:

- Do not confuse the canonical **z** coordinate with the **z** coordinate of the particle in the (**x**, **y**, **z**) coordinate frame. By construction, the latter is always zero.
- For a bunch of particles at a given **s** position, in general, the particles will have differing time **t**.
- If the reference particle has the same β value as a particle, canonical **z** will be the longitudinal distance the particle is with respect to the reference particle. Positive **z** indicates that the particle is in front of the reference particle and vice versa.

8.2 Example

Example lattice:

```
! Lattice File: orbit.bmad
beginning[beta_a] = 10.    ! m  a-mode beta function
beginning[beta_b] = 10.    ! m  b-mode beta function
beginning[e_tot] = 10e6    ! eV
parameter[geometry] = open  ! or closed
```

```
bmad_com[spin_tracking_on] = T

particle_start[y] = 0.01
particle_start[px] = 0.06
particle_start[pz] = -0.2
particle_start[spin_x] = 1

b: sbend, L = 0.5, g = 1    ! g = 1 / bending_radius
q: quadrupole, L = 0.6, k1 = 10

lat: line = (b, q)      ! List of lattice elements
use, lat                ! Line used to construct the lattice
```

Start *Tao* with the lattice file **orbit.bmad**. Spin tracking is on (**bmad_com[spin_tracking_on]** set to True) and a non-zero initial orbit is set using **particle_start** parameters. The resulting orbit is shown in Figure 19a.

The initial phase space coordinates can now be varied using the **change** or **set** commands. For example:

```
Tao> change particle_start px 0.04
          Old           New     Old-Design     New-Design          Delta
     0.060000      0.100000       0.000000       0.040000       0.040000
```

The result is shown in Figure 19b.

View Orbits with **show lattice** command

```
Tao> show lat -spin -orbit
      Values at End of Element:
 Index  name       key            ...           orbit   ...           spin
                                  ...               x   ...              x
     0  BEGINNING  Beginning_Ele  ...    0.000000E+00   ...   1.000000E+00
     1  B          Sbend          ...    2.888946E-02   ...   9.886413E-01
     2  Q          Quadrupole     ...   -1.847740E-02   ...   9.768026E-01
     3  END        Marker         ...   -1.847740E-02   ...   9.768026E-01
```

or the **show element** command

```
Tao> show ele 1
 Element #                1
 Element Name: B
... etc ...

Orbit:   Positron    State: Alive
          Position [mm]  Momentum [mrad]        Spin    |
  X:        5.02772161     -44.34051580   -0.14145163   | Particle [sec]:      ...
  Y:        9.52710654      -1.27804602   -0.00171549   | Part-Ref [sec]:      ...
  Z:       -4.78648430    -200.00000000    0.98994368   | (Ref-Part)*Vel [m]:  ...
```

8.3 Exercises:

1. **Phase Space:** Construct a lattice with a single drift of 2 meters length. Set the particle species to be protons with a reference energy of 10^{12} eV (so the approximation that the

particle velocity is speed of light can be made), and Set the initial particle position to have $p_x = 0.2$, $p_z = 1$ with all other coordinates zero. Calculate from first principles what the phase space coordinates will be at the end of the lattice and check that your answer agrees with *Bmad*.

9 RF Cavities

9.1 Reference Energy and the Lcavity and RFcavity Elements

Reference Energy

Every lattice element has a reference particle, a reference energy called **E_tot**, and a reference momentum named **p0c** (in eV). The three are interrelated so knowing the reference particle and either the reference momentum or energy, the third quantity can be calculated. The reference momentum is used for normalization of the phase space momentum as well as normalized strength parameters.

For example, for a quadrupole element, the **b1_gradient** parameter (units: Tesla/m) can be used to specify the linear field gradient and the **k1** parameter (units: $1/m^2$) is the normalized field gradient normalized using the value of the reference momentum as discussed in the "**Magnetostatic Multipole Fields**" section of the "**Electromagnetic Fields**" chapter of the *Bmad* manual.

Lcavity

Lcavity elements are exceptions for lattices with an **open** geometry, where the reference energy/momentum at the beginning of the lattice is what is set in the lattice file. Usually, the reference energy/momentum for a downsteam element inherits the reference energy/momentum of the element just upstream.

Lcavity elements represent an RF cavity where the reference energy/momentum at the exit end of the **lcavity** is set so that a particle entering the cavity with zero phase space coordinates leaves with zero phase space coordinates. In particular, its phase space **pz** will be zero at the exit end. **Lcavity** elements also have **p0c_start** and **E_tot_start** parameters that are set to the reference energy of the upstream element.

Rfcavity

Rfcavity elements also represent an RF cavity, but their reference energy/momentum is the same as the upstream element.

9.2 Example: Cavity Element

```
! Lattice File: cavity.bmad
beginning[beta_a] = 10.    ! m  a-mode beta function
beginning[beta_b] = 10.    ! m  b-mode beta function
beginning[p0c] = 1e8    ! eV

parameter[geometry] = open       ! or closed
parameter[particle] = He+

q1: quad, l = 0.1, k1 = 0.14
q2: quad, l = 0.1, b1_gradient = parameter[p0c] * q1[k1] / c_light
lc: lcavity, l = 1, voltage = 10e8, rf_frequency = 1e9
rf: rfcavity, l = 1, voltage = 10e8, phi0 = 0.25

lat: line = (q1, q2, lc, q1, q2, rf)
use, lat
```

Notes:

- For a **lcavity** phi0 = 0 corresponds to peak acceleration.
- For an **rfcavity** phi0 = 0.25 corresponds to peak acceleration.

Start *Tao* as explained in with the lattice file **cavity.bmad**. Examining the **lcavity** element shows:

```
> tao -lat cavity.bmad

Tao> show ele 3
Element #                  3
Element Name: LC
Key: Lcavity
... etc ...
   51    P0C_START     =  1.000000E+08 eV         BETA_START    =  0.02681151
   52    E_TOT_START   =  3.729740E+09 eV         DELTA_E       =  1.000000E+09 eV
   53    P0C           =  2.910237E+09 eV         BETA          =  0.61530588
   54    E_TOT         =  4.729740E+09 eV         GAMMA         =  1.268571E+00
... etc ...
Orbit:  He+    State:  Alive
          Position [mm]  Momentum [mrad]         Spin    |
  X:        0.00000000       0.00000000                  | Particle [sec]:      ...
  Y:        0.00000000       0.00000000                  | Part-Ref [sec]:      ...
  Z:       -0.00000000       0.00000000                  | (Ref-Part)*Vel [m]:  ...
```

The reference energy at the start of the element, **E_tot_start**, is not the same as the reference energy at the end of the element **E_tot**. The particle orbit, which started out with zero phase space coordinates (there were no **particle_start** statements to give a non-zero starting orbit), still has zero phase space coordinates at the end of the **lcavity** element.

Compare this to the **rfcavity** element:

```
Tao> show ele 6
Element #                  6
Element Name: RF
Key: RFcavity
... etc ...
   53    P0C           =  2.9102374E+09 eV         BETA          =  0.615305883
   54    E_TOT         =  4.7297409E+09 eV         GAMMA         =  1.2685712E+00
... etc ...
 Orbit:  He+    State:  Alive
          Position [mm]  Momentum [mrad]         Spin    |
  X:        0.00000000       0.00000000                  | Particle [sec]:      ...
  Y:        0.00000000       0.00000000                  | Part-Ref [sec]:      ...
  Z:      140.37425587     494.97867675                  | (Ref-Part)*Vel [m]:  ...
```

Here there is no **E_tot_start** parameter since the ending reference energy is always equal to the starting one. Here, the **pz** coordinate at the end of the element is nonzero.

9.3 Adding RF Cavities

The example files for this section can be found at **/lattices/9_RF/AddingRF/**

In a real machine, electrons emit synchrotron radiation as they bend around the ring, so we need to replenish the energy with an RF system. Let's put RF cavities in the 10 o'clock straight section of our ring. We will design a special FODO cell with RF cavities **FODORF** and replace 4 FODO cells in the 10 o'clock straight section.

In the ESR, there are two RF cavities of length 4.017m between each quadrupole with equal drift spaces between each element. We will follow these dimensions as shown in Fig. 20.

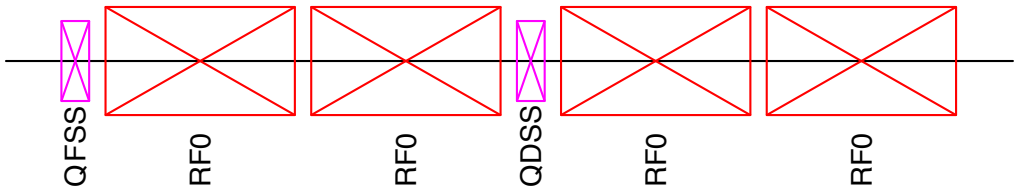


Figure 20: FODO cell with RF cavities.

9.4 Example: Consturcting the FODO cell with RF cavities

1. Define RF cavities and drift elements. The appropriate drift length is calculated using an expression.

```
RF0: RFCavity, L = 4.017, harmon = 7560,
       voltage=68.0/18.0 * 1e6 ! Start with ESR voltage
DRF: Drift, L = ((1.241+5.855+0.609)-2*4.017)/3
```

2. Define the **FODORF** cell.

```
FODORF: line = (QFSS, DRF, RF0, DRF, RF0, DRF, QDSS, DRF, RF0, DRF, RF0, DRF)
```

3. Replace 4 FODO cells in the 10 o'clock section with **FODORF** by modifying **SEXTANT9** and **SEXTANT11**:

```
SEXTANT9:  line = (4*FODOSSF, SS_TO_ARCF, 20*FODOAF,
                                ARC_TO_SSF, 2*FODOSSF, 2*FODORF)
SEXTANT11: line = (2*FODORF, 2*FODOSSR, SS_TO_ARCR,
                          20*FODOAR, ARC_TO_SSR, 4*FODOSSR)
```

9.5 Exercises

1. **Set the synchrotron tune:** The synchrotron tune can be set by using the **set z_tune** command in *Tao*, which varies the RF cavity voltages. Set the synchrotron tune to 0.05 and write down the new RF voltage. Add it to the end of the lattice file as

```
RF0[voltage] = your_voltage
```

2. Modify the **lcavity** in the **cavity.bmad** to have a small length (so the transit time is small), and set the beginning momentum small enough so the relativistic beta is significantly less than one. Starting the particle with a finite z, calculate the ending z after the cavity and verity that the change in z is consistent with what Bmad calculates.

3. **Lcavity** elements have an attribute **phi0_err** which varies the RF phase that a particle sees but does not change the reference energy. Add a finite **phi0_err** to the **cavity** element and verify that the reference energy does not change but that the phase space **pz** of the particle does change. With the **cavity.bmad** lattice, there is a range of values for **phi0_err** where the cavity will decellerate the particle enough so that the particle will turn around and not make it through the cavity. Approximately what is this range?

10 Introduction to Long Term Tracking

Up to this point, only the *Tao* program in the Bmad ecosystem has been used and tracking has implicitly been limited to tracking a single particle once through the lattice (which is needed for computation of the Twiss functions and closed orbit §14.). *Bmad* also supports multi-particle (beam) tracking and a *Bmad* based program, if appropriately configured, can track beams over an arbitrary number of turns. *Tao* is equipped to do beam tracking but *Tao* is not designed for tracking beams over many turns in a ring. Rather, the *Bmad* based program *Long Term Tracking* is designed for this. The *Long Term Tracking* program can simulate such things as misalignments, wake fields, higher order mode cavity resonances, spin polarization, energy ramping, etc. The *Long Term Tracking* program is also capable of simulating beam injection into and extraction out of a storage ring including tracking the beam through the transfer lines.

10.1 Initializing a Beam

Bmad has two ways to define an initial beam distribution. One is providing a file containing individual particle positions. The other is using a **beam_init_struct structure** which holds parameters to initialize a beam. This is discused in detail in the *Bmad* manual.

Bmad accepts two file formats for beam initialization: **ASCII** and **HDF5**. HDF5 is a widely used binary format which is useful when storing beams with a large number of particles. In this tutorial The ASCII format is used. The ASCII beam format describes a single bunch using a **header** section followed by a **table** section. Multiple bunches can be defined using multiple header/table pairs. The header lines start with a "#" sign and contains global parameters and custom parameters. Any line in the header that does not contain a valid parameter is ignored. The last line in the header starts with "#!" and defines the table columns. The table section follows the header section. Each row gives the parameters for one particle in the bunch.

Here is a simple beam initialization file **beam.bmad**. It defines 7 electrons with various z and pz values.

```
# This is the header section
# You can define parameters for all particles in the bunch
# species = electron
# charge  = 1.6e-19
# state   = alive
# Or custom parameters not used by Bmad like
# version = 1.0.0

#! x   px   y   py   z     pz
   0   0    0   0    0     0
   0   0    0   0    0.1   0
   0   0    0   0    0.15  0
   0   0    0   0    0     -0.01
   0   0    0   0    0     0.015
   0   0    0   0    0     0.02
   0   0    0   0    0     0.03
```

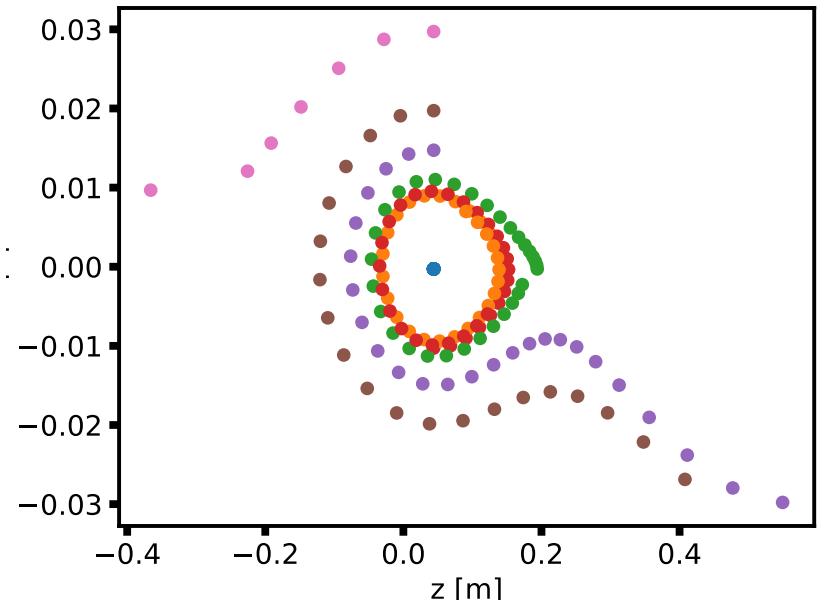


Figure 21: The trajectory of 7 particles in the z-pz plane over 30 turns.

10.2 Initializing long term tracking

Long Term Tracking takes an input file that defines parameters in a namelist, similar to how *Tao* is initialized with a **tao.init** file. A full list of parameters and their descriptions can be found in the *Long Term Tracking* manual. A simple **long_term_tracking.init** file:

```
&params
  ltt%lat_file = "ring.bmad"
  ltt%simulation_mode = "BEAM"
  ltt%tracking_method = "BMAD"
  ltt%n_turns = 30
  ltt%particle_output_every_n_turns = 1
  ltt%phase_space_output_file = "turn"

  beam_init%center = 0, 0, 0, 0, 0, 0
  beam_init%position_file = "beam.bmad"
/
```

The **long_term_tracking.init** file specifies the lattice file name, tracking method, initial beam positions, and program outputs. In this case, *Long Term Tracking* will track the initial beam for 30 turns and output the particle positions at the end of each turn.

You can run *Long Term Tracking* with this initialization file by the command **long_term_tracking** <**ini_file**> where <**ini_file**> is the name of the **master** initialization file (as opposed to other input files like the lattice file). If the **ini_file** is not specified, the default file name is **long_term_tracking.init**.

The output to the terminal will look like:

```
> long_term_tracking
Initialization file: long_term_tracking.init
n_particle: 7
 Cumulative number dead at end of turn 6: 1
 Cumulative number dead at end of turn 19: 2
 Cumulative number dead at end of turn 24: 3
# tracking_time = 0.02552
```

10.3 Particle Track Output

The settings init the init file:

```
  ltt%phase_space_output_file = "turn"
  ltt%particle_output_every_n_turns = 1
```

results in files named **turn##** where ## is the turn number being generated every turn. In these files, particle positions will be recorded with a similar format as the beam initialization file. The files contain a header section with global parameters of the simulation and a table section. The key of each column in the table section is given by the last line of the header section which starts with "##". Each row gives the coordinates and state of one particle.

With some parsing of these files, particle trajectories can be visualized. Python and gnuplot scripts are provided in the **/lattices/10_LongTermTracking/ParticleTrack** directory. Fig. 21 shows the results.

10.4 Averages Output

Besides particle output files, the *Long Term Tracking* program can generate several other types of output. One useful type of output are the averages files. To illustrate these files, run the *Long Term Tracking* program with the input files in the **/lattices/10_LongTermTracking/AveragesOutput** directory. In this case, the master input file is named **averages.init**:

```
&params
  ltt%lat_file = "simple_ring.bmad"          ! Lattice file

  ltt%averages_output_file = "#.dat"
  ltt%averages_output_every_n_turns = 100

  ltt%simulation_mode = "BEAM"
  ltt%tracking_method = "MAP"    !
  ltt%n_turns = 20000                          ! Number of turns to track
  ltt%map_order = 3

  bmad_com%spin_tracking_on = T          ! See Bmad manual for bmad_com parameters.
  bmad_com%radiation_damping_on = T
  bmad_com%radiation_fluctuations_on = T

  beam_init%n_particle = 100
  beam_init%spin = 1, 1, 0               ! See Bmad manual for beam_init_struct parameters.
```

```
  beam_init%a_emit = 1e-9
  beam_init%b_emit = 1e-12
  beam_init%sig_z = 1e-4
  beam_init%sig_pz = 1e-4
/
```

The **ltt%tracking_method** is set to "**MAP**" which means that tracking is done using a one-turn map. Using a map makes tracking fast but the construction of the map at the beginning can take some time. In this case, a very simple lattice is used to speed up the process. If radiation effects are turned on as they are here, the radiation effects are included in the map. The map is partially inverted to make, in the limit of no radiation, the application of the map symplectic.

The setting of **ltt%averages_output_file** to "**#.dat**" means that the averages output files will be produced. There are three averages output files which in this case will be named **emit.dat**, **ave.dat**, and **sigma.dat**. Every 100 turns (set by **ltt%averages_output_every_n_turns**), a line is written to each file recording varias averaged quantities like the beam sigma matrix, emittance, spin polarization, etc.

In the init file, the beam is specified by the **beam_init** settings rather than an beam file and the beam is started with transverse emittances that are smaller than the equilibrium ones. This is illustrated by the the graph in Fig. 22, which shows the approach to equilibrium of the a-mode emittance as a function of turn number. Since the averages files are structured to be easily plotted using **gnuplot**, fig. 22 was drawn with the command

```
gnuplot> plot "emit.dat" using 1:4
```

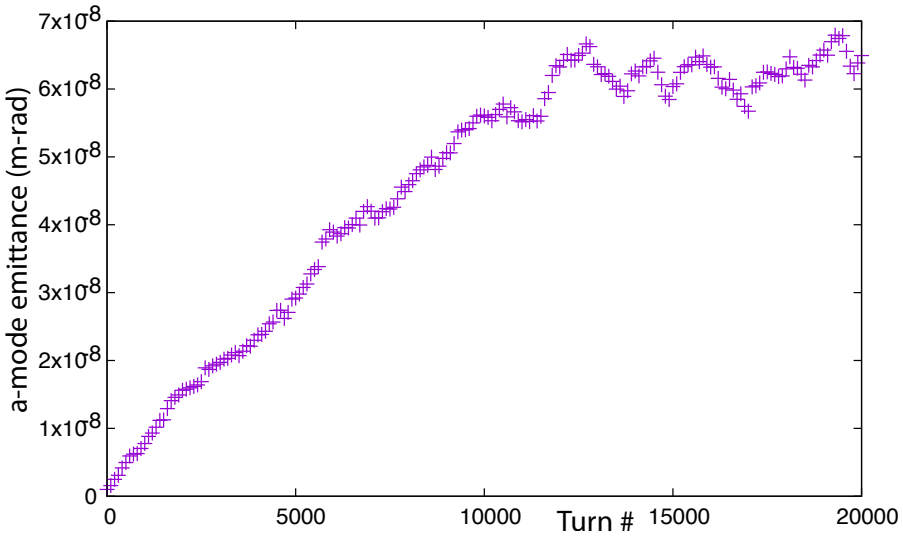


Figure 22: The a-mode emittance as a function of turn number.

10.5 Exercises

1. **Initial parameters:** Experiment with different initial particle parameters and plot their trajectories. How much can you change pz without losing the particle? Track particles and plot their longitudinal phase space. Find stable and unstable fixed points, and invariant ellipses in longitudinal phase space.

2. **Core Emittance:** In the **emit.dat** file there are columns for the "50% Core" emittance. What is this? Hint: Consult the *Long Term Tracking* manual.

11 Control Elements

11.1 Overview

Control elements are elements that control the parameters of other elements. There are four types of control elements: **groups**, **overlays**, **ramper** and **girders**. **Groups** and **overlays** are convenient to do such things as simulate control room "knobs". For example a power supply that powers a chain of magnets. **Girder** elements are used for simulating lattice element support structures. For example, a bend element followed by a quadrupole followed by a sextupole that are all resting on a common table can be simulated with the three elements supported by a girder element. **Ramper** elements are used for simulating machines where parameters can vary turn-by-turn. For example, ramper elements can simulate such things as energy ramping, beta squeeze to achieve colliding beam conditions, or third order resonance slow extraction from a ring to an extraction line.

Discussion of **girder** and **ramper** elements is outside the scope of this tutorial and here **group** and **overlay** elements will be studied. Note: **Group**, **overlay**, **ramper**, and **girder** elements are known as "**minor lords**" since they only control a subset of an element's attributes. The other type of **lord** elements, **multipass** lords and **superposition** lords are called "**major lords**". See the *Bmad* manual for details.

11.2 Example Lattice

The lattice used to illustrate control elements is named **control.bmad** which is situated in the directory **/lattices/11_ControlElements/**:

```
! Lattice File: control.bmad
beginning[beta_a] = 10
beginning[beta_b] = 10

parameter[particle] = muon
parameter[p0c] = 1e9
parameter[geometry] = open

q: quadrupole, l = 1
b: sbend, l = 1
ll: line = (q, b)
use, ll

ov1: overlay = {q[k1]: a+b^2, b[g]: 0.1*a+tan(b)}, var = {a, b}, a = 0.02
ov2: overlay = {q[k1]: 0.7, q[x_offset]: 0.1*hh}, var = {hh}, hh = 0.01
gr1: group = {b[k1]: 0.4*sqrt(z)}, var = {z}
```

Explanation:

- The overlay **ov1** controls two parameters: The **k1** attribute of element **q** (denoted **q[k1]**) and the **g** attribute of element **b** (denoted **b[g]**).
- Overlay **ov1** has two variables called **a** and **b** that are used to control the two attributes.

- The formulas that overlay **ov1** uses to calculate the values of the two controlled attributes are $\mathbf{a+b}^2$ for **q[k1]** and $\mathbf{0.1*a+tan(b)}$ for **b[g]**.
- Since overlay **ov2** also controls **q[k1]**, the value of **q[k1]** is the sum of the contributions of **ov1** and **ov2**.
- The given "formula" for the control of **q[k1]** by **ov2** is just a constant: 0.7. This is a shorthand notation and the actual formula used is **0.7*hh**. Note: When this shorthand notation is used, only one control variable (in this case **hh**) may be used by the overlay.
- The initial values for control variables may be set when defining the control element. For example, **hh** of **ov2** is set to 0.01. Control variables default to a value of zero.

11.3 Control Element Organization in the Lattice

Start *Tao* with the lattice file **control.bmad**. To see the elements in the lattice use the **show lattice** command:

```
Tao> show lat

      Values at End of Element:
 Index  name       key                    s        l     beta      phi ...
                                                            a        a ...
     0  BEGINNING  Beginning_Ele       0.000      ---    10.00    0.000 ...
     1  Q          Quadrupole          1.000    1.000     9.83    0.101 ...
     2  B          Sbend               2.000    1.000     9.60    0.204 ...
     3  END        Marker              2.000    0.000     9.60    0.204 ...
Lord Elements:
     4  OV1        Overlay             2.000      ---     9.60    0.204 ...
     5  OV2        Overlay             1.000      ---     9.83    0.101 ...
     6  GR1        Group               2.000      ---     9.60    0.204 ...
 Index  name       key                    s        l     beta      phi ...
                                                            a        a ...
      Values at End of Element:
```

The list of lattice elements is divided up into two sections:

- The **"tracking"** part of the lattice are the elements to be tracked through. Here the tracking part of the lattice contains elements with index 1 through 3 (the **beginning** element with index 0 is not tracked through and is present to hold Initial parameters like the Twiss parameters).
- The **"lord"** section of the lattice are where the lord elements reside. Here the lord section contains elements with index 4 through 6.
- **Group** and **overlay** elements get assigned a longitudinal **s**-position based upon the **s**-position of the last slave element. This does not affect any calculations and is done since it can be useful information when using the **show lat** and other **show** commands (oftentimes a control element will only control the parameters of one slave element).

11.4 Overlay Control

To see how things are controlled, the **show element** command may be used. Examining the lord **ov1** element shows:

```
Tao> show ele 4     ! Or: show ele ov1

 Element #                 4
 Element Name: OV1
 Key: Overlay
... etc...
Slave_status: Free
Lord_status:  Overlay_Lord
Control Variables:
    1   A                                          =  2.0000000E-02
    2   B                                          =  0.0000000E+00
Slaves: [Attrib_Val = Expression_Val summed over all controlling overlays.]
   Index   Ele_Name   Attribute Attrib_Value  Expression_Val    Expression
       1   Q          K1          2.7000E-02      2.0000E-02    a+b^2
       2   B          G           2.0000E-03      2.0000E-03    0.1*a+tan(b)
... etc...
```

- All lattice elements have a **slave_status** which shows what type of slave the element is and a **lord_status** which shows what type of lord the element is. **overlay** elements automatically have a **lord_status** of **overlay_lord**. In this case, **ov1** has a **slave_status** of **free** since there are no other lord elements that control parameters of **ov1**. In general, **overlay** and **group** lords may control parameters of other lords as well as non-lords.
- When an element parameter is controlled by one or more overlays, the value of that element parameter is the sum of the values for each overlay. Thus in the above example, the contribution to **q[k1]** due to overlay **ov1** is 0.02 (= $a + b^2$) as shown in the "**Expression_Val**" column above. There is also a contribution of 0.007 (= $0.7 \cdot hh$) due to overlay **ov2** making the value of **q[k1]** equal to 0.027 as shown in the "**Attrib_Value**" column above.

Examining the **q** slave element shows that indeed the **k1** attribute has a value of 0.027:

```
Tao> show ele q
 Element #                 1
 Element Name: Q
        ... etc...
    1   L                           =  1.0000000E+00 m
    4   K1                          =  2.7000000E-02 1/m^2
        ... etc...
Slave_status: Minor_Slave
Controller Lord(s):
   Index   Name        Attribute          Lord_Type          Expression
       4   OV1         K1                 Overlay            a+b^2
       5   OV2         K1                 Overlay            0.7*hh
       5   OV2         X_OFFSET           Overlay            0.1*hh

Lord_status:  Not_a_Lord
        ... etc...
```

The **slave_status** of element **q** is set to **minor_slave** to show that it is controlled by one or more minor lords. The **lord_status** of **q** is **not_a_lord** indicating that it does not control anything ("tracking elements", that is elements in the tracking part of the lattice, never control other elements).

Since the value of an attribute that is controlled by overlays depends directly on the overlay variable values, the attribute may not be directly changed. For example, trying to change **q[k1]** directly will result in an error:

```
Tao> set ele q k1 = 0.02
[ERROR | 2017-AUG-26 13:29:26] attribute_free:
    THE ATTRIBUTE: K1
    OF THE ELEMENT: Q
    IS NOT FREE TO VARY SINCE:
    IT IS CONTROLLED BY THE OVERLAY: OV1
```

11.5 Group Control

Overlay elements use what is called "**absolute**" control since the value of a controlled parameter is determined directly by the settings of the overlay variables that the controlled parameter is slaved to. On the other hand, **group** elements use what is called "**relative**" control which is different from absolute control in two respects:

- Only changes in group variable values affect controlled parameters.
- With group control, a controlled parameter may be varied directly.

Looking at an example will make this clear. Starting from the **control.bmad** lattice, consider the effect of changing the **z** variable of the group **gr1** to 0.01.

```
Tao> set ele gr1 z = 0.01

Tao> show ele gr1
 Element #                6
 Element Name: GR1
 Key: Group

... etc ...

Slave_status: Free
Lord_status:  Group_Lord
Control Variables:
    1   Z  =  1.0000000E-02          OLD_Z  =  1.0000000E-02
Slaves:
   Index    Ele_Name  Attribute    Attrib_Value  Expression_Val    Expression
       2    B         K1             4.0000E-02      4.0000E-02    0.4*sqrt(z)
```

For **group** elements, *Bmad* keeps track of what is called the "old" value of a variable. The name of the old value is the variable name with a "**old_**" prefix. In this case the old value of **z** is given the name "**old_z**". Before the **set ele gr1** command was executed, the value of **z** and **old_z** is zero. When the above **set ele gr1** command is executed, the value of **z** becomes 0.01. *Bmad* detects that **z** and **old_z** are different and updates **b[k1]** using the following procedure:

1. Evaluates the formula for **b[k1]** using **z** and **old_z** and takes the difference. In this case the difference is **0.4*sqrt(z)** - **0.4*sqrt(old_z)** = 0.04
2. Changes the value of **b[k1]** by the difference (0.04). Since the old value of **b[k1]** was zero. The new value of **b[k1]** is 0.04.
3. Sets the value of **old_z** equal to **z** indicating that the change was made.

A consequence of the last step in the above procedure, the **show ele** command will always show that **old_z** and **z** are equal.

Now consider the effect of the following commands:

```
Tao> reinit tao
Tao> set ele gr1 z = 0.01
Tao> set ele b k1 = 0.02
Tao> set ele gr1 z = 0.04
```

The result is:

```
Tao> show ele gr1
... etc...
Control Variables:
    1   Z  =  4.0000000E-02           OLD_Z  =  4.0000000E-02
Slaves:
   Index    Ele_Name  Attribute   Attrib_Value  Expression_Val    Expression
       2    B         K1            6.0000E-02      8.0000E-02    0.4*sqrt(z)
```

1. The "**reinit tao**" command resets *Tao* to its initial state.
2. The "**set ele gr1 z = 0.01**" command acts as explained above.
3. The "**set ele b**" command sets the value of **b[k1]** to 0.02. This is independent of the state of element **gr1**.
4. The "**set ele gr1 z = 0.04**" command sets the value of **gr1[z]** to 0.04 which causes the value of **b[k1]** to increase by 0.04 (= 0.08 - 0.04) from 0.02 to a value of 0.06.

11.6 Example:

Consider the situation where you want to control the chromaticity (change in tune with particle energy) of a ring by varying sextupole strengths. To change the chromaticity by 1 unit you want to change the sextupole strengths by amounts that you compute. Here you don't care about the value of the sextupole strengths per se, you only want to vary the sextupole strengths by a certain delta. So the sextupole "knob" can be simulated using a **group** controller which may look like:

```
xqune_1 : group ={SEX_08W:-.6415E-03*k2,...}, var = {k2}
```

Note: In this case, since the parameter to be controlled for the **sex_08w** element was not specified, the parameter is taken to be the same as the variable of the controller. **k2** in this case.

Notes:

- Group and overlay elements can control other group and overlay elements.
- A given element parameter may only be controlled by a set of group elements or a set of overlay elements but may not be controlled by both group and overlay elements since this would create an ambiguous situation as to how to evaluate the parameter.

11.7 Exercises:

1. The function that a controller uses to control a slave attribute may be specified using an arithmetical expression as in the above examples, or may be specified by a list of "**knot**" points with spline interpolation used to evaluate the function in between points. As an exercise, setup a controller in **control.bmad** that uses knot points that mimics the action of **ov2** at least over some limited interval. Hint: Look at the documentation for **overlay** or **group** elements in the **elements** chapter of the *Bmad* manual.
2. **Group** controllers are good for varying the longitudinal position of elements. Starting with the file simple.bmad add a group controller that varies the **s**-position of the upstream edge of element **B** while keeping the length of the entire lattice constant (hint: The lengths of both **B** and **D** must change in tandem). This situation occurs frequently enough that there is a shortcut attribute called **start_edge** that can be used instead of directly varying the lengths of elements. See the documentation on **group** elements in the **Elements** chapter of the *Bmad* manual for more details.
3. Modify the lattice file **simple.bmad** to include a **girder** element supporting elements **B** and **Q**. Use the **show ele** command to verify that indeed the girder is supporting these two elements.

12 Dynamic Aperture

Dynamic aperture (DA) is measure of the stable phase space in an accelerator over a specified number of turns. Particles outside of the DA will be driven to large amplitudes where they will be lost. In the following sections, strategies for increasing a ring's DA will be shown in a sample ring.

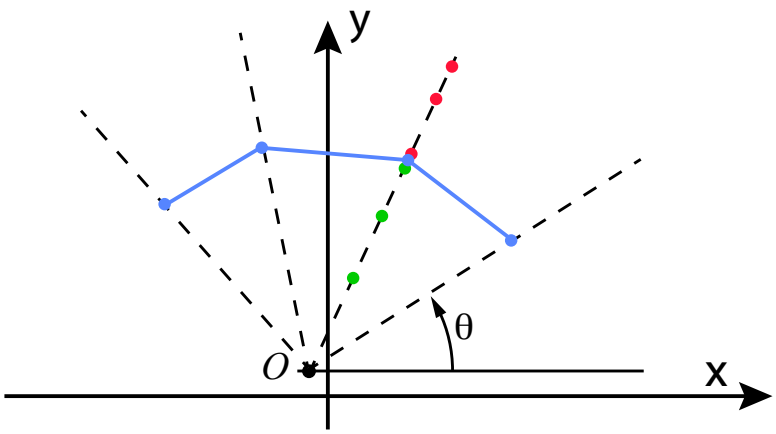


Figure 23: The calculation of a dynamic aperture curve in the x-y plane at a given initial p_z value involves calculating aperture curve points (blue dots) along a set of "rays" (dashed lines). The line segments between points is simply for visualization purposes and does not appear in graphs of real data.

12.1 Calculating Dynamic Aperture

The DA calculation can be done in Tao or the standalone *dynamic_aperture* program. The calculation is done using the same *Bmad* based routines in both programs. Example files for both programs are provided. Points along rays are tested to determine the DA perimeter as shown in Fig. 23. The rays have a common origin point ($\mathcal{O}$) which is taken to be the reference orbit. The calculation of an aperture curve point along a given ray involves iteratively tracking particles with different starting (x, y) position values to find the boundary between stable (green dots) and unstable (red dots) motion.

Files for this example can be found at

```
/lattices/12_DynamicAperture/12.1_CalculatingDA
```

Using the init file **tao_DA.init**, Tao will calculate the DA of out lattice. The DA calculation is defined in the block:

```
&tao_dynamic_aperture
  ix_universe = 1
  ellipse_scale = 10
```

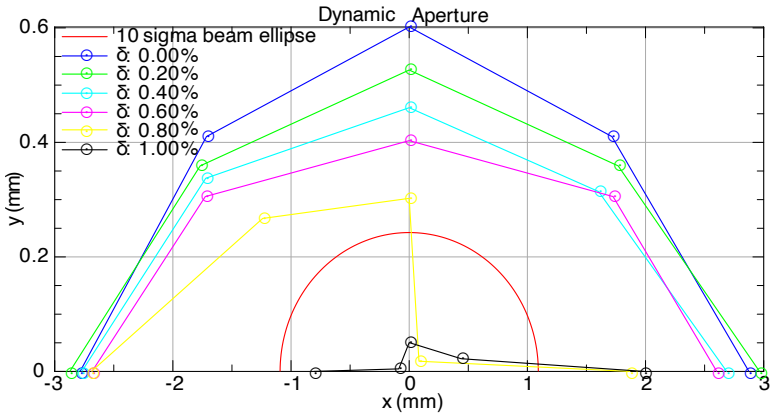


Figure 24: Dynamic aperture results.

```
 pz = 0.00, 0.002, 0.004, 0.006, 0.008, 0.010
 da_param%n_angle = 5
 da_param%n_turn = 1000
 da_param%abs_accuracy = 1e-4
 da_param%min_angle = 0
 da_param%max_angle = 3.14159
 a_emit = 2E-08
 b_emit = 1E-08
/
```

The **pz** list gives each momentum offset the aperture will be calculated for. The **da_param** struct contains the parameters used for a calculation. In this case it is set up to give 5 points along rays arranged from 0 to π. Particle stability is checked over 1000 turns. Run *Tao* with the command

```
tao -init tao_DA.init
```

After the calculation is finished, the results can be plotted using the command

```
place r11 dynamic_aperture
```

The results are shown in Fig. 24

For the standalone program, the file **DA.init** is provided. The parameters are similar to those in the Tao program:

```
&params
 dat_file = "ring_DA.dat"
 ltt%lat_file = "ring.bmad"
 ltt%rfcavity_on = T
 dpz = 0.000, 0.002, 0.004, 0.006, 0.008, 0.010
 da_param%n_angle = 5
 da_param%n_turn = 1000
 da_param%abs_accuracy = 1e-5
 da_param%min_angle = 0
 da_param%max_angle = 3.14159
/
```

The main additions are the output file and lattice file being declared. Using this file, the standalone program can be run by the command:

```
dynamic_aperture DA.init
```

After running, the results will be in the **dat_file** used. The results can the be plotted in *gnuplot* using the command outputted at the end of the DA run or using the plotting software of your choice.

You can run the program using the ring lattice to check the DA. DA calculations can be slow, so it is sometimes advisable to reduce the accuracy, number of turns, or number of angles to get faster results between optimizations, and then come back with a more accurate run to verify. These example parameters are set to take fewer than 5 minutes with the tutorial ring.

12.2 Example: Adding Sextupoles

Files for this example can be found at:

```
/lattices/12_DynamicAperture/12.2_AddingSextupoles
```

A common element used for DA optimization are sextupoles. "**Harmonic**" sextupoles, that is, sextupoles that minimize resonance driving modes, can be used to increase the transverse aperture. Sextupoles can be added to the lattice by following the steps below.

1. Start by defining the chromatic sextupoles (sextupoles that will control the chromaticity and therefore have to be placed in dispersive regions). Two families per plane in each arc will be used. A typical sextupole here looks like:

   ```
   SF1_1: Sextupole, L = 0.584, k2 = 2
   ```

   The sextupole strength will be optimized later. The **ring.bmad** file defines a ring with sextupoles.

2. Harmonic sextupoles should also be defined for the straight sections. We will use one family in each plane

   ```
   SFSS: Sextupole, L = 0.584, k2 = 0.
   SDSS: Sextupole, L = 0.584, k2 = -0.
   ```

3. We need to make room for the sextupoles, so we will split the drift **D2** into two smaller drifts

   ```
   D2S1: Drift, L = 0.501
   D2S2: Drift, L = 0.156
   ```

4. The FODO cells must be altered to include the new sextupoles and drifts. In the straight sections this is simply done:

   ```
   ! Straight section forward FoDo:
   FODOSSF: line = (QFSS, D1, DB, D2S1, SDSS, D2S2,
                    QDSS, D1, DB, D2S1, SFSS, D2S2)
   ```

This also must be done for the reverse line. The arc FODOs are more complicated as we want to families per plane. In order to do this, we will define two FODO cells. For arc 1 this will look like:

```
! 1 o'clock arc forward 2*FoDo:
FODOAF_1: line = (QF, D1, B, D2S1, SD1_1, D2S2,
                  QD, D1, B, D2S1, SF1_1, D2S2,
                  QF, D1, B, D2S1, SD2_1, D2S2,
                  QD, D1, B, D2S1, SF2_1, D2S2)
```

The sample file contains similar FODO declarations for all arcs.

5. Finally the lines that define the ring must be to include the altered FODO cells

```
SEXTANT1:  line = (4*FODOSSF, SS_TO_ARCF,
                   10*FODOAF_1, ARC_TO_SSF_2, 4*FODOSSF_2)
SEXTANT3:  line = (4*FODOSSR_2, SS_TO_ARCR_2,
                   10*FODOAR_3, ARC_TO_SSR, 4*FODOSSR)
SEXTANT5:  line = (4*FODOSSF, SS_TO_ARCF,
                   10*FODOAF_5, ARC_TO_SSF, 1*FODOSSF, IPF)
SEXTANT7:  line = (IPR, 1*FODOSSR, SS_TO_ARCR,
                   10*FODOAR_7, ARC_TO_SSR, 4*FODOSSR)
SEXTANT9:  line = (4*FODOSSF, SS_TO_ARCF,
                   10*FODOAF_9, ARC_TO_SSF, 2*FODOSSF, 2*FODORF)
SEXTANT11: line = (2*FODORF, 2*FODOSSR, SS_TO_ARCR,
                   10*FODOAR_11, ARC_TO_SSR, 4*FODOSSR)
```

12.3 Example: Chromaticity Correction

Files for this example can be found at:

```
/lattices/12_DynamicAperture/12.3_ChromaticityCorrection
```

The first step in chromatic correction is to use the newly placed chromatic sextupoles to correct the linear chromaticity. As there is a chromaticity associated with each plane, we have two things to correct. For optimization, sextupoles can be combined using an **Overlay** element. This allows all sextupoles in the same plane to be optimized together. With one overlay for each plane, this gives two variables for the correction. This optimization can be done with the following steps:

1. First the overlays must be defined

```
OSF: overlay = {SF%_*[k2]: x}, var = {x}
OSD: overlay = {SD%_*[k2]: x}, var = {x}
```

The overlays define an attribute **x** that controls the **k2** attribute of every sextupole that fits a name of the form in each overlay.

2. In the **tao.init** file, a variable block can be used to control the overlays in an optimization

```
&tao_var
    v1_var%name = 'OS'
    var(1:2)%ele_name = "OSF", "OSD"
    default_step = 1e-6
    default_attribute = 'x'
/
```

3. We must also define the data for chromaticity in the **tao.init** file. Instead of correcting setting the chromaticity to zero, it is usually set to a small positive number when above transition or a small negative number when below. This is done to avoid the head-tail instability. In our case we can set it to 1 in both planes.

```
&tao_d2_data
    d2_data%name = 'chrom'
    n_d1_data = 1
/

&tao_d1_data
    ix_d1_data = 1
    datum(1) = 'chrom.a' '' '' '' 'target' 1 100
    datum(2) = 'chrom.b' '' '' '' 'target' 1 100
/
```

4. The optimization can be done using the command **run lmdif**. After the optimizer is finished, the results will be in a output file named **var1.out** by default.

5. After optimizing, make a copy of the **var1.out** file and rename it to something like **correction.out**. Then, call the output file for the sextupole values in your lattice using **call, file = correction.out**.

6. You can then rerun a DA calculation to see how this correction has altered the results.

12.4 Advanced Dynamic Aperture Correction

In addition to linear chromaticity, several other optimizations to improve DA are also available in Bmad. One of these is the W-function:

$$W = \sqrt{A^2 + B^2} \tag{2}$$

where

$$A = \frac{\partial \alpha}{\partial p_z} - \frac{\alpha}{\beta}\frac{\partial \beta}{\partial p_z}, \qquad B = \frac{1}{\beta}\frac{\partial \beta}{\partial p_z} \tag{3}$$

This is a chromatic function that is the amplitude of the linear energy dependent β-beat. similar to the linear chromaticity, this can be corrected with the use of sextupoles. This correction is more complex than before as the phase advance between sextupoles and the quadrupoles creating the W-function is important. Since β-beating goes as twice the betatron phase advance, sextupoles that are 90°apart will work against each other. The solution is to split the sextupoles in each arc into two families per plane. This way we can correct the chromaticity while making sure the sextupoles do not entirely cancel each other for the W-function correction.

12.4.1 Example

Files for this example can be found at:
/lattices/12_DynamicAperture/12.4_AdvancedDA

The sextupole families can again be set up using an overlay, although this time we will center the sextupole strengths around the value for chromaticity correction. This will ensure each arc corrects an equal amount of the total chromaticity.

1. We will begin by altering the overlays. Assume the sextupole setting you found in the last example are SX and SY, then an overlay for sextupoles centered at these values will be

```
OF_1: overlay = {SF1_1[k2]:  SX + x, SF2_1[k2]:  SX - x}, var = {x}
OD_1: overlay = {SD1_1[k2]:  SY + x, SD2_1[k2]:  SY - x}, var = {x}
```

   This must be done for all arcs.

2. With these overlays, the W-function can be optimized, however, without proper phasing between arcs, sextupoles may not be efficient in the correction. In order to fix phases between sextupoles for the optimization, it is often useful to use a phase trombone. Phase trombones are artificial elements that are used to set a phase advance without altering optics. We will need three phase trombones, two surrounding the IP and one to fix the tune. We will fix the tune in the 2 o'clock tune cell.

```
TR_F: match, dphi_a = 0, dphi_b = 0, matrix = phase_trombone
TR_R: match, dphi_a = 0, dphi_b = 0, matrix = phase_trombone
TR_M: match, dphi_a = 0, dphi_b = 0, matrix = phase_trombone
```

3. In order to make sure the tune remains fixed, we can use an overlay to control the trombones

```
TROMBONES: overlay = {TR_F[dphi_a]: x1,
    TR_R[dphi_a]: x2, TR_2[dphi_a]: -x1-x2,
    TR_F[dphi_b]: y1, TR_R[dphi_b]: y2, TR_2[dphi_b]: -y1-y2},
       var = {x1, x2, y1, y2}
```

4. W-function optimization is often done by setting it to zero in specific location in the ring. We will set it to zero at the IP and at the end of arc 7. We will add a marker for the use in the optimization

```
END_7: marker
```

5. The trombones and marker can now be placed in the ring

```
SEXTANT1:  line = (TR_M, 4*FODOSSF, SS_TO_ARCF,
    10*FODOAF_1, ARC_TO_SSF_2, 4*FODOSSF_2)
SEXTANT3:  line = (4*FODOSSR_2, SS_TO_ARCR_2,
    10*FODOAR_3, ARC_TO_SSR, 4*FODOSSR)
SEXTANT5:  line = (4*FODOSSF, SS_TO_ARCF,
    10*FODOAF_5, ARC_TO_SSF, 1*FODOSSF, TR_F, IPF)
SEXTANT7:  line = (IPR, TR_R, 1*FODOSSR, SS_TO_ARCR,
    10*FODOAR_7, END_7 ,ARC_TO_SSR, 4*FODOSSR)
SEXTANT9:  line = (4*FODOSSF, SS_TO_ARCF,
    10*FODOAF_9, ARC_TO_SSF, 2*FODOSSF, 2*FODORF)
SEXTANT11: line = (2*FODORF, 2*FODOSSR, SS_TO_ARCR,
    10*FODOAR_11, ARC_TO_SSR, 4*FODOSSR)
```

6. We must now set up the init file. Define the W-function to be zero at the **IP6** and **END_7** markers

```
    &tao_d2_data
        d2_data%name = 'w'
        n_d1_data = 1
    /
    &tao_d1_data
        ix_d1_data = 1
        datum(1) = 'chrom.w.a' '' '' 'END_7' 'target' 0 100
        datum(2) = 'chrom.w.b' '' '' 'END_7' 'target' 0 100
        datum(3) = 'chrom.w.a' '' '' 'IP6##1'   'target' 0 100
        datum(4) = 'chrom.w.b' '' '' 'IP6##1'   'target' 0 100
    /
```

7. The variables can be set up to control the sextupoles in arcs 5 and 7 and the phase trombones

```
 &tao_var
     v1_var%name = 'sextupoles'
     var(1:4)%ele_name = "OF_5", "OD_5", "OF_7", "OD_7"
     default_step = 1e-6
     default_attribute = 'x'
/


&tao_var
    v1_var%name = 'phases'
    var(1:4)%ele_name="TROMBONES", "TROMBONES", "TROMBONES", "TROMBONES"
    default_step = 1e-6
    var(1:4)%attribute = 'x1', 'x2', 'y1', 'y2'
/
```

8. Run the optimizer in Tao to find the correct sextupole and phase trombone settings.
9. Make a copy of the **var1.out** file and call it in your lattice to use the updated values.

After these steps, you should have a W-function like in Figure 25.

12.5 Exercises

1. **IP and 2 o'clock phase trombones:** This can be split into three parts, the phase on both sides of the IP and the phase in the 2 o'clock phase trombone. These parts can be inserted separately into the init file
   (a) The 2 o'clock phase trombone is the easiest to add the init file, simply copy the data and variables from the previous use of the tune cell.
   (b) The upstream phase trombone can be implemented by fixing $\alpha_{x,y}$ and $\beta_{x,y}$ at the IP. Add a marker at the beginning of arc 5 named **END_5**. Set a datum for **phase_frac.a** and **phase_frac.b** from the marker to the IP. Set the target to get the correct phase. For variables, use the upstream IR quads and the last three quads in the matching section from the dispersion suppressor.

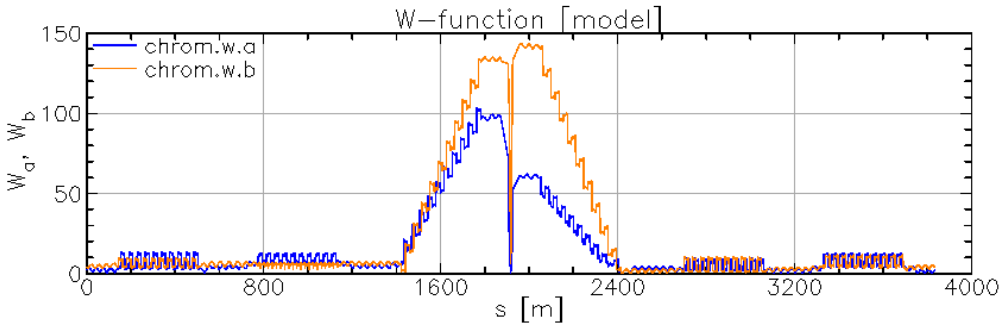


Figure 25: The W-function receives a large contribution form the IP which is corrected over one arc.

(c) The downstream phase can be matched using the same process as the upstream side. Variables can be chosen to be the downstream IR quads and last three quads in the matching section to the dispersion suppressor. We again must fix $\alpha_{x,y}$ and $\beta_{x,y}$, but this time we will fix it at the marker **END_7**. The phase condition is now set from the IP to **END_7**.

After setting up the data and variables, you can run the optimizer. If the lattice may become unstable during the optimization, if this happens, try using the **cut** command (toggles lattice geometry between open and close). Notice that the W-function correction is slightly disrupted by this change. This sometimes requires an iterative process.

13 Model, Design and Base Lattices in Tao

When *Tao* runs, *Tao* creates three lattices (Technically, *Tao* instantiates three lattices per **universe**. See the discussion on universes in the *Tao* manual.):

Design Lattice
The **design** lattice corresponds to the lattice read in from the lattice description file(s). Parameters in this lattice are never varied.

Model Lattice
Initially, when *Tao* is started, The **model** lattice is the same as the **design** lattice. Parameters in the **model** lattice are allowed to vary. That is, all commands to vary lattice parameters vary parameters of the **model** lattice.

Base Lattice
The **base** lattice is a reference lattice used so that changes in the **Model** lattice may be easily viewed. The **Design** lattice can also be used as a reference lattice but since the parameters of the **design** lattice are fixed, this is not always desirable. The parameters of the **base** lattice are set by setting the parameters of the **base** lattice equal to the present state of the **model** lattice.

13.1 Changing Model Parameters

To see the difference between the **model** and **design** lattices, start *Tao* with the lattice file **lattices/13_LatticeTypes/simple.bmad**.

Now issue the following commands:

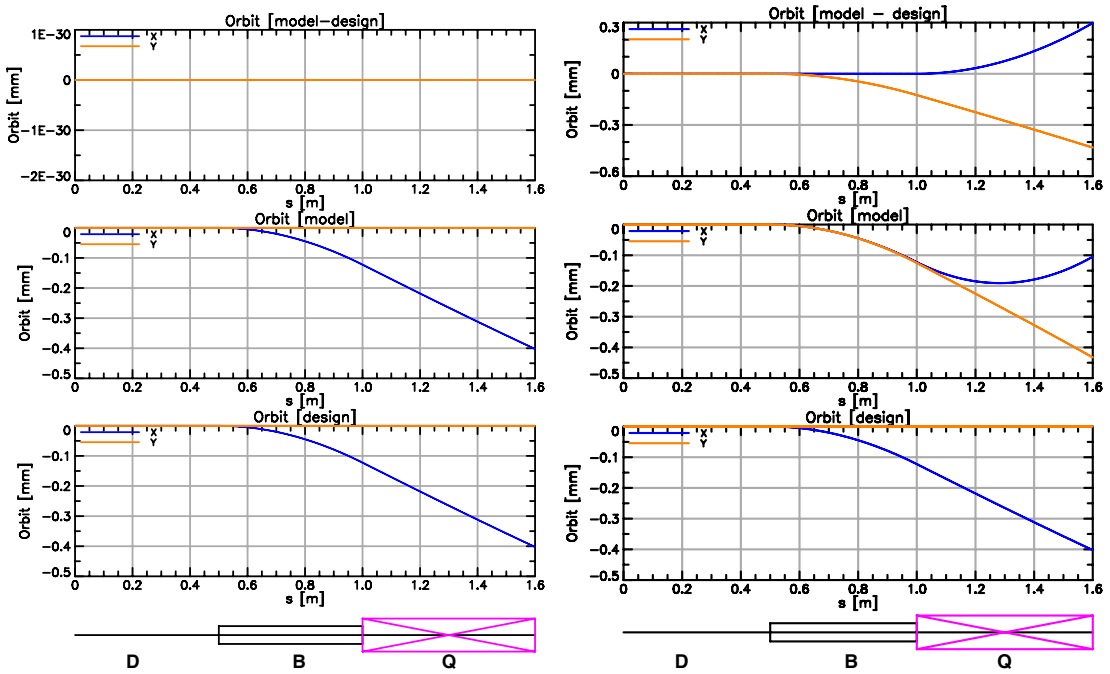
```
Tao> place r23 orbit
Tao> place r13 orbit
Tao> set plot r33 component = design          ! Bottom plot
Tao> set plot r13 component = model - design  ! Plot difference orbit
Tao> scale
```

The "**set plot <plot_name> component = ...**" command sets where the data to be plotted comes from. The result is shown in Figure 26a. The bottom plot shows the **design** lattice orbit, the middle plot shows the **model** lattice orbit and the top plot shows the difference in orbits between **model** and **design**. Since the two lattices are the same when *Tao* is started, the difference orbit is zero.

Now change the **model** lattice using the following commands:

```
Tao> change element b vkick  -0.0005   ! Changes by a given delta
Tao> set element q hkick = 0.001       ! Another way of changing a parameter.
Tao> scale
```

The **change** command changes real numbers by a given delta. The **set** command sets a parameter to a specific value. Unlike the **change** command, the **set** command can also be used with



(a) Initially, the **model** lattice and the **design** lattices are the same.

(b) The **set** and **change** commands will modify **model** lattice parameters.

Figure 26

integer, string and logical parameters. The result is shown in Figure 26b. Since now the **model** lattice is not the same as the **design** lattice, the difference orbit is non-zero.

13.2 Using the Base Lattice

The **base** lattice is used to view changes when the desired reference lattice does not correspond to the **design** lattice.

Continuing from the previous section, issue the following commands:

```
Tao> set lattice base = model  ! Set the Base lattice = Model lattice.
Tao> set plot r33 component = model - base
Tao> set ele q vkick = 5e-4
Tao> scale
```

The **set lattice** command sets the **base** lattice equal to the **model** lattice. The third command varies the **model** lattice. The result is shown in Figure 27. The bottom plot of the orbit difference between **model** and **base** is not the same as the orbit difference between **model** and **design**.

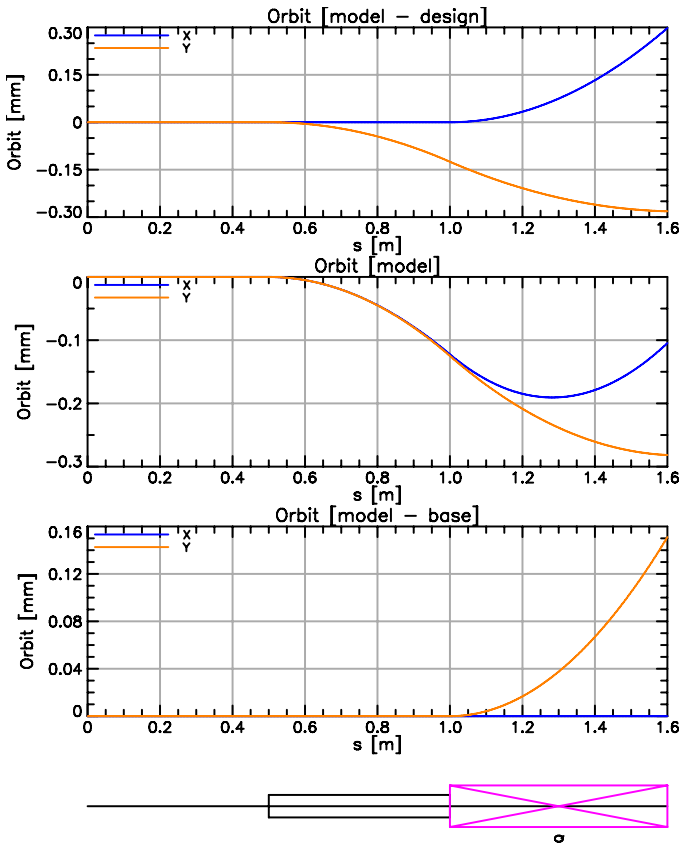


Figure 27: The **base** lattice is used to view changes when the reference lattice configuration does not correspond to the **design** lattice.

13.3 Exercises

1. **Orbit plots:** Modify the middle graph in Fig. 27 so that the vertical orbit curve becomes the horizontal orbit of the **design** lattice.

2. **Alias command:** To save on typing, alias commands may be defined. Define an alias command called **setit** so that typing "**setit 1e-6**" is equivalent to "**set ele q vkick = 1e-6**". Hint: Look in the manual or type "**help alias**" to get more information on setting and using alias commands. Note: Do not confuse *Tao* alias commands with the alias string component of a lattice element.

3. **Adjust the tune using `set tune`:** The tune of the lattice can also be set by using the **`set tune`** command in *Tao*, which scales together all focusing quadrupoles and all defocusing quadrupoles to set the horizontal and vertical tunes. Using a lattice with the ring you have constrcted, use **`set tune -mask *, QF, QD 0.08 0.14`** to set the tune of the ring. Plot the

difference between the model and design lattices to see how much “beta-beating” there is around the ring.

14 Orbit Correction

While operating accelerators, the beam's orbit is measured with Beam Position Monitors (BPMs). The accelerator is designed with ideal BPM readings in mind, which are often 0. After extended operational experience, operators learn good readings for all BPMs that are referred to as golden orbits. It then becomes important to change the beam's trajectory to obtain the desired BPM readings.

14.1 Example: Adding Corrector Coils and BPMs

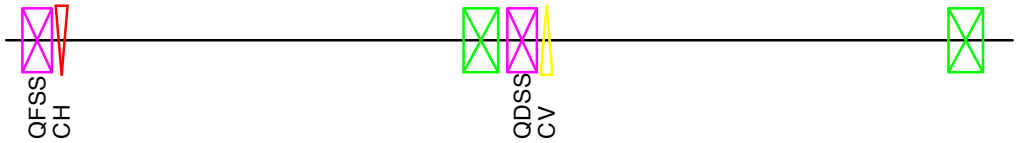


Figure 28: FODO cell with corrector coils.

1. Start with the **ring0.bmad** lattice. Place corrector coils in the middle of D1 drifts 6.4 cm away from quadrupoles. This is downstream from the quadrupoles in the forward arcs and upstream in the reverse arcs. Define horizontal and vertical corrector coils and appropriate drift spaces as follows:

```
! (1): Define horizontal/vertical corrector coils and
       split D1 into D1C1 and D1C2 to fit coils
D1C1: Drift, L = 0.064
D1C2: Drift, L = 0.345 !0.609 - 0.064 - 0.2
CH: hkicker, L = 0.2
CV: vkicker, L = 0.2
```

2. Then, add horizontal kickers near focusing quadrupoles and vertical kickers near defocusing quadrupoles. Correctors are added in the D1 drifts. This means placing the correctors after quadrupoles in the forward geometry and before quadrupoles in the reverse geometry.

   Because we have used D1 drifts repeatedly throughout the ring, adding correctors manually is a tedious task. The easiest approach is to use **regular expressions** in Find and Replace. The recipe is the following:

```
1. In the forward arcs for drifts after focusing quadrupoles:
    Replace (QF[^,\n]*,[^,\n]*)D1 with \1D1C1, CH, D1C2 for
2. In the forward arcs for drifts after defocusing quadrupoles:
    Replace (QD[^,\n]*,[^,\n]*)D1 with \1D1C1, CV, D1C2
3. In the reverse arcs for drifts before focusing quadrupoles:
    Replace (D1)([^,\n]*,[^,\n]*QF), with D1C2, CH, D1C1\2
4. In the reverse arcs for drifts before defocusing quadrupoles:
    Replace (D1)([^,\n]*,[^,\n]*QD), with D1C2, CV, D1C1\2
5. In the reverse arcs for drifts at the end of a cell:
    Replace D1) with D1C2, CH, D1C1)
```

In some text editors, you may need to use $1 and $2 instead of \1 and \2.

3. Define and add BPMs after dipoles (**B**, **HB**, and **DB** in the straight section). Here we use a 0-length marker to represent the BPMs.

```
! (3) Define and add BPMs
BPM: marker
```

   Then insert the BPMs after dipoles by searching and replacing **B,** with **B, BPM,** (including the commas) and repeat for **BH**.

14.2 Example: Radiation-induced Sawtooth Orbit Correction

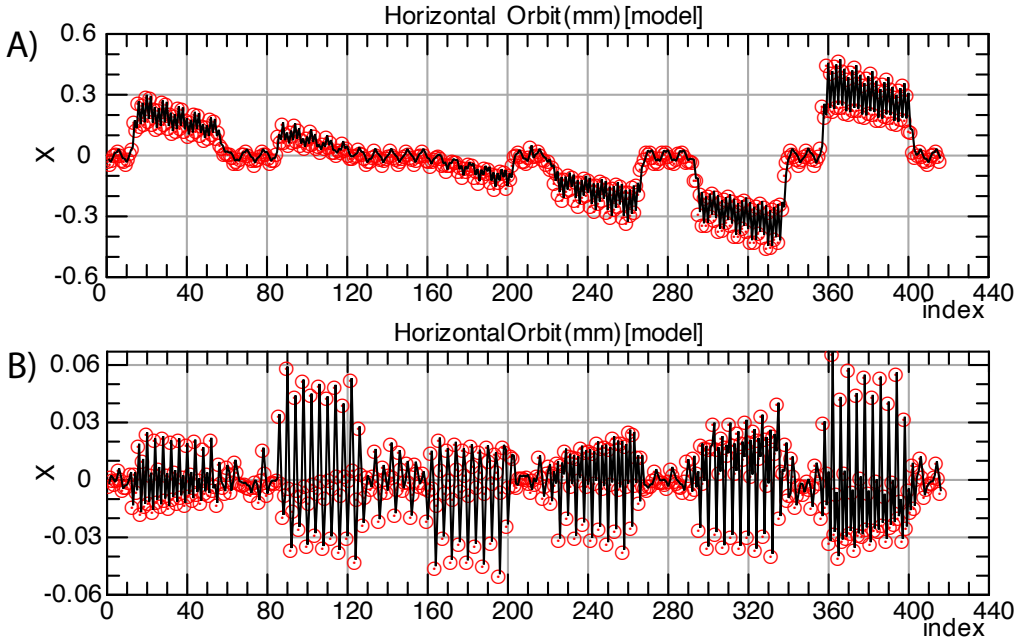


Figure 29: A) The sawtooth orbit due to synchrotron radiation. B) The orbit after correction using kickers and BPMs.

1. Sawtooth orbits are caused by synchrotron radiation energy loss in the ring. This energy loss leads to the particle energy variation along the ring, which causes the horizontal closed orbit to have a sawtooth shape. To produce this shape in Tao, turn on radiation damping with

```
bmad_com[radiation_damping_on] = T
```

   in the **ring0.bmad** file, or equivalently run

```
set bmad_com radiation_damping_on = T
```

in *Tao*. It turns on the deterministic average synchrotron radiation energy loss in the ring. Let's explore two methods to correct the sawtooth orbit.

2. The **taper** command adjusts magnet strengths to eliminate the transverse orbital and Twiss changes due to the radiation damping induced sawtooth effect. The **taper** scales magnet strengths based on local closed orbit momentum deviation and identical elements are changed independently. Tapering is not applied to strengths controlled by an **overlay** element.

   Now run **taper** in *Tao* and check the resulting orbit.

3. Alternatively, we can use kickers to fix the sawtooth orbit. Let's define the orbit readings as data and kicker strengths as variables in **tao.init**.

```
&tao_d2_data
        d2_data%name = 'orbit'
        n_d1_data = 2
        default_merit_type = 'target'
        default_meas = 0
        default_weight = 1
/

&tao_d1_data
        ix_d1_data = 1
        d1_data%name = 'x'
        search_for_lat_eles = 'BPM*'
/

&tao_d1_data
        ix_d1_data = 2
        d1_data%name = 'y'
        search_for_lat_eles = 'BPM*'
/

&tao_var
        v1_var%name = 'hkicker'
        search_for_lat_eles = 'hkicker::*'
        default_step = 1e-6
        default_attribute = 'kick'
/

&tao_var
        v1_var%name = 'vkicker'
        search_for_lat_eles = 'vkicker::*'
        default_step = 1e-6
        default_attribute = 'kick'
/
```

4. Right now all the kickers and BPMs are assumed to be the same, but we want to address them individually. *Tao* has the option to add a unique suffix after duplicate elements. Simply add to your **tao.init**:

```
&tao_design_lattice
        unique_name_suffix = "hkicker::_? vkicker::_? marker::_?"
/
```

5. While *Tao* has a native template for orbit plots, here we provide two custom templates to plot orbit data from BPMs. You can add them to your **tao.init** and show them by running **place r13 bpm_orbit_x** and **place r23 bpm_orbit_y**.

```
&tao_template_plot
  plot%name = 'bpm_orbit_x'
  default_graph%x%major_div_nominal = 10
  default_graph%x%label = ' '
  plot%x_axis_type = 'index'
  plot%n_graph = 1
/

&tao_template_graph
  graph%name = 'x'
  graph_index = 1
  graph%box = 1, 1, 1, 1
  graph%title = 'Horizontal Orbit (mm)'
  graph%y%label = 'X'
  graph%y%major_div = 4
  curve(1)%data_source = 'data'
  curve(1)%data_type   = 'orbit.x'
  curve(1)%y_axis_scale_factor = 1000
  curve(1)%symbol%color = "red"
/

&tao_template_plot
  plot%name = 'bpm_orbit_y'
  default_graph%x%major_div_nominal = 10
  default_graph%x%label = ' '
  plot%x_axis_type = 'index'
  plot%n_graph = 1
/

&tao_template_graph
  graph%name = 'y'
  graph_index = 1
  graph%box = 1, 1, 1, 1
  graph%title = 'Vertical Orbit (mm)'
  graph%y%label = 'Y'
  graph%y%major_div = 4
  curve(1)%data_source = 'data'
  curve(1)%data_type   = 'orbit.y'
  curve(1)%y_axis_scale_factor = 1000
/
```

6. Before we run the optimization, we can choose which variables to change and which data to optimize with **run** and **veto** commands. For example, the sawtooth orbit is entirely in the x-direction, so we can ignore vkickers and orbit.y for this orbit correction with:

```
veto var vkicker
veto data orbit.y
```

   Run the optimizer and check if the orbit is fixed. Write the lattice with optimized kicker strengths into a new file **ring.bmad**.

14.3 Example: Orbit Correction with Quadrupole Misalignments

1. Let's start from **ring.bmad** that we created in chapter §14.2. Add random offsets to all quadrupoles in the lattice. We can first assume that x-offsets of quadrupoles follow a Gaussian distribution of $\sigma = 10\,\mu\mathrm{m}$ with a 3σ cutoff. You can do this with *Tao* command "**set ele quadrupole::* x_offset = 1e-5*ran_gauss(3)**". The **ran_gauss(sig_cut)** command returns an array of Gaussian distributed random variables with unit RMS truncated at sig_cut.

2. Correct the orbit using the same optimization process. You can reuse the **tao.init** we wrote for example §14.2.

3. If the misalignment amplitude is large, the orbit could become unstable and the optimizer will have a hard time finding a solution. Try larger misalignments of $\sigma = 100\,\mu\mathrm{m}$ with "**set ele quadrupole::* x_offset = 1e-4*ran_gauss(3)** and correct the orbit. This will likely fail and you'll see many error messages.

4. One approach to avoid unstable orbits is correcting the orbit with an open lattice first. Reinitialize *Tao* to start fresh from the beginning. Use command **cut** to change the lattice geometry from closed to open. *Tao* now uses the single-turn orbit instead, so it won't complain even if the closed orbit is unstable.

5. We can now run the optimizer and correct the single-turn orbit as close to 0 as possible. After the single-turn orbit is corrected, use command **cut** again to change the lattice geometry back to closed and *Tao* will calculate the periodic orbit. Since the orbit excursions are now much smaller, the periodic orbit should be far away from the unstable region. Run the optimizer again to correct the periodic orbit to 0.

6. **unstable.lattice** is a special datum that can be used in an optimization to avoid unstable solutions. For lattices with a closed geometry, it is 0 if the ring is stable. If the closed orbit is unstable, its value will be positive. This helps the optimizer find a solution in the stable region. You can declare it in **tao.init** as follows:

```
&tao_d2_data
        d2_data%name = 'unstable'
        n_d1_data = 1
/

&tao_d1_data
        ix_d1_data = 1
        d1_data%name = 'lat'
        datum(1) = "unstable.lattice" "" "" "" "target" 0 100
/
```

14.4 Exercises

1. **Zero orbit at IP:** Use 4 horizontal/vertical kickers around the IP (CH/V_104 to CH/V_107, 8 total kickers) to make the orbit exactly 0 at the IP without affecting the rest of the ring.

15 Spin Tracking with Ramping

15.1 Introduction

Bmad is one of the most powerful modern accelerator physics libraries for polarization analyses and spin-tracking simulations. The most important tools of polarization analysis, such as the invariant spin field, the amplitude-dependent spin tune, and spin-orbit resonance strengths, can be computed using *Bmad*. The invariant spin field (ISF) and the amplitude-dependent spin tune (ADST) can be computed in *Bmad* using normal form via *Tao*, K. Yokoya's SODOM-2 algorithm via *SODOM-2*, or stroboscopic averaging via *spin_stroboscope*, but these applications will not be covered in this brief introduction. In this chapter, we will ramp our storage ring energy through a spin-orbit resonance, calculate the polarization transmission, and use it to verify the resonance strength computed by *Tao* using the Froissart-Stora formula.

In a perfectly-flat ring (a ring in which the only magnetic fields encountered on the closed orbit are vertical), the closed-orbit spin tune is $\nu_0 = G\gamma$, where $G = (g-2)/2$ is the anomalous gyromagnetic g-factor and γ is the reference gamma. We will ramp our ring through the $\nu_0 = 60 - Q_y \approx 51.81$ resonance.

Example: Ramping Through a Spin-Orbit Resonance

1. Start with the **spin_lat.bmad** file, located in **lattices/15_SpinTracking**. It is a FODO ring based on the layout of the AGS.

2. Let's define a **ramper** element which ramps $G\gamma$ from 51.71 to 51.82 in 5000 turns. To ramp the **e_tot** attribute for all elements in such a time, we need to obtain the revolution time, which is done by calling **show uni** and looking for the **Lattice branch transit time**. For our ring, this is 2.67024 microseconds. Thus, our **ramper** element is

```
rampe: ramper = {*[e_tot] : 51.71 / anomalous_moment_of(proton) * m_proton +
(.2/5000/anomalous_moment_of(proton) * m_proton) / (2.67024e-6)*time}, var = {time}
```

   We can define this element anywhere in the lattice file to initiate the ramping.

3. Create a **long_term_tracking.init** file, containing the following:

```
&params
  ltt%lat_file = 'spin_lat.bmad'
  ltt%ele_start = '0'
  ltt%ele_stop = ''
  ltt%phase_space_output_file = 'tracking.dat'
  ltt%averages_output_file = 'ramp.dat'
  ltt%averages_output_every_n_turns = 1
  ltt%particle_output_every_n_turns = 1

  ltt%ramping_on = T
  ltt%ramp_update_each_particle = T
  ltt%simulation_mode = 'SINGLE'
  ltt%tracking_method = 'BMAD'
  ltt%n_turns = 5000
```

```
  ltt%rfcavity_on = T
  ltt%split_bends_for_stochastic_rad = F
  ltt%random_seed = 0
  ltt%add_closed_orbit_to_init_position = T

  bmad_com%spin_tracking_on = T
  bmad_com%radiation_damping_on = F
  bmad_com%radiation_fluctuations_on = F

  beam_init%n_particle = 1
  beam_init%spin = 0, 1, 0
  beam_init%center = 0, 0, 1e-5, 0, 2.66, 1.15e-5
/
/
```

4. Plot S_y versus turn number with the following *gnuplot* command:

```
plot 'tracking.dat' u 1:13
```

   Observe the asymptotic value of S_y as the number of turns increases.

5. Load Tao, and set the energy such that $G\gamma = 51.81$ with the command

```
set ele * e_tot = 51.81 / anomalous_moment_of(proton) * mass_of(proton)
```

   Then use the command **show spin -ele 0** to observe the resonance strength. We are working with a B-mode (vertical) resonance, and whether frac$(Q_s + Q)$ or frac$(Q_s - Q)$ is smaller tells you whether to use **Xi_sum** or **Xi_diff**. The correct resonance strength corresponds to whichever frac$(\dots)$ expression is closer to zero. The resonance strength given by *Tao* is normalized, so we have to multiply by $\sqrt{J_y}$ to find the true resonance strength. We can obtain J_y using the equation

$$y(s) = \sqrt{2J_y\beta_y(s)}\cos[\Phi_y(s)], \tag{4}$$

   along with the command **show twiss**.

6. We can find what the asymptotic value of S_y should have been using the Froissart-Stora formula:

$$S_y(\theta \to \infty) = 2\exp\left(-\frac{\pi\epsilon^2}{2\tilde{\alpha}}\right) - 1. \tag{5}$$

   Here, θ is the machine azimuth, ϵ is the resonance strength, and

$$\tilde{\alpha} = \frac{\mathrm{d}}{\mathrm{d}\theta}G\gamma = \frac{1}{2\pi} \times [\text{Change in } G\gamma \text{ per turn}]. \tag{6}$$

   Compare the result of the Froissart-Stora formula to the result obtained from tracking.

15.2 Exercises

1. The AGS has two special spin rotators known as partial Siberian snakes. Add the partial Siberian snakes to **spin_lat.bmad**. They are represented by elements called CSNK (cold snake) and WSNK (warm snake), which are markers in the lattice we have provided to you.

We will represent the ideal (point-like) Siberian snakes using **Taylor** elements. Here is the example syntax:

```
CSNK: Taylor, {S1: q_0|}, {Sx: q_1|}, {Sy: q_2|}, {Sz: q_3|}
```

In this case, the **Taylor** element produces an instantaneous rotation represented by the unit quaternion (q_0, q_1, q_2, q_3). Recall that the unit quaternion $(\cos(\vartheta/2), \sin(\vartheta/2)\vec{e})$ produces a rotation by angle ϑ around the unit vector $\vec{e}$. The cold snake has a rotation angle of $18.3°$, and the warm snake has a rotation angle of $10.57°$. Both snakes rotate the spin around the longitudinal axis.

Play around with the **show spin** command at different energies, and verify that the spin tune is *not* $G\gamma$. Why is this the case?

2. If you have done the previous exercise, comment out the **Taylor** elements and return the snakes to markers for this exercise. Use the script **DepolTao.py** to plot the resonance spectrum in the range $G\gamma = 30$ to $G\gamma = 60$. It is run using the following syntax:

```
python DepolTao.py <lat_file> <output_file> <plot_file> <particle> <SPRINT>
<min Ggamma> <max Ggamma> <J_y>
```

For this case, set <SPRINT> to T. This boolean flag can take the values T or F, and if T, the spin tracking mode is set to SPRINT. SPRINT tracking is fast, but can be inaccurate when magnets are misaligned or there are strong nonlinearities.

What do you observe about the spectrum? Can you explain any of its features? *Hint: look at the structure of the lattice.*

A Python/PyTao Interface

PyTao is a Python wrapper on top of Tao that allows users to access the Tao library via **ctypes** and **pexpect**. More information can be found at https://bmad-sim.github.io/pytao/.

A.1 Installation

The Bmad Distribution must be installed before installing PyTao. If Bmad is installed via a release tarball, the Bmad Distribution must be compiled with the **ACC_ENABLE_SHARED="Y"** flag set in the **bmad_dist/util/dist_prefs** file.

PyTao can be installed in three different ways:

1) Using setuptools

```
python setup.py install
```

2) Using pip

```
# From PyPI distribution
pip install pytao

# or from the source folder
pip install .
```

3) Using conda

```
conda install -c conda-forge pytao
```

A.2 Basic Examples

A.2.1 Initializing Tao

While running Python, To run Tao while running Python, first import the Pytao package:

```
>>> from pytao import Tao
```

Tao is the basic object in PyTao. Anything used to initialize Tao on the command line can be used to initialize a Tao object in Python. For example, a Tao object is created with **-init** to use a specific **tao.init** file or with **-lat** to use a specific lattice file:

```
>>> tao = Tao("-init lattices/1_FoDoA/a_Arc/tao.init -noplot")
```

A.2.2 Send a Command

Anything that you would normally type at a **Tao>** prompt on the command line can be sent as a string using the **tao.cmd** command. The return is a list of output strings. To send a command:

```
>>> tao.cmd("show lat 1:10")
```

User can also send a list of commands using **tao.cmds**. This returns the corresponding list of outputs:

```
>>> tao.cmds(["set lattice model=design", "set ele Q00W x_offset = 1e-6"])
```

A.2.3 Jupyter magic %%tao

There is an alternative way to send commands to Tao directly in a Jupyter notebook, using the **%%tao** command. Multiple lines can be executed in one code block:

```
%%tao
show lat 1:10
show ele 4
```

A.2.4 Interface Commands

Output from the **show** command is designed to be human-readable. To get data suitable for parsing in Python, Tao has a set of special commands under **python**. See the **python** section in the *Tao* manual for more details, or use **help**:

```
%%tao
help python
```

Some commands have **array_out** options. For example, this will return nothing:

```
tao.cmd("python lat_list -array_out 1@0>>Q*|model orbit.floor.x")
```

But calling **tao.cmd_real** on the same command will return the data as an array from an internal pointer:

```
tao.cmd_real("python lat_list -array_out 1@0>>Q*|model orbit.floor.x")
```

For convenience, all of these commands are also available as methods of the Tao class, and the outputs are automatically parsed.

For example, to get the orbit at an **s** position:

```
tao.orbit_at_s(s_offset=1.2)
```

To evaluate and return data values:

```
tao.evaluate("data::cbar.11[1:10]|model")
```

One very useful method to extract array data is **tao.lat_list**. For example, to get the locations and orbits for all elements:

```
s = tao.lat_list("*", "ele.s")
x = tao.lat_list("*", "orbit.vec.1")
y = tao.lat_list("*", "orbit.vec.3")
```

A detailed list of parameters that can be extracted using **tao.lat_list** can be obtained via:

```
?tao.lat_list
```

To see all the available method commands:

```
from pytao import interface_commands

all_cmds = [name for name in dir(Tao) if not name.startswith("_")]
for cmd in all_cmds:
    print(cmd)
```

Each has documentation and an example associated with it:

```
?tao.data_d2
```

Unfortunately there can only be one Tao instance per process, because the internal structures are held in memory and accessed via **ctypes**. So this will replace the current Tao session in memory.

B Answers to Selected Exercises

B.1 Chapter 2

Exercise 5:

The **set** command:

```
set global n_opti_cycles = 100
```

B.2 Chapter 8

Exercise 1:

The phase space momenta p_x, p_y and p_z do not change through a drift and therefore have the same values at the end of the lattice as the beginning. Since p_y is zero, y will be zero at the end. The horizontal momentum P_x (not to be confused with the phase space momentum)

$$P_x = (1 + p_z)\, p_x\, P_0 = 0.4\, P_0 \tag{7}$$

where P_0 is the reference momentum. The longitudinal momentum P_s is

$$P_s = (1 + p_z)\, \sqrt{1 - p_x^2 - p_y^2}\, P_0 = 3.97994974842648\, P0 \tag{8}$$

Given the initial starting position of $x_0 = 0$, the x position at the end is

$$x = x_0 + \frac{P_x}{P_s}\, L = 0.20100756305184242 \text{ meters} \tag{9}$$

where L is the length of the drift.

To calculate z at the end, the particle time t and reference particle time t_{ref} at the end must be computed. The reference time is

$$t_{\text{ref}} = t_{\text{ref0}} + \frac{L}{\beta_{\text{ref}}\, c} = \frac{2}{c} \tag{10}$$

where t_{ref0} is the beginning reference time which is zero and the reference beta β_{ref} is approximated to be unity. The particle time is

$$t = t_0 + \frac{L\sqrt{1 + (P_x/P_s)^2 + (P_y/P_s)^2}}{\beta\, c} = \frac{2.010075630518424}{c} \tag{11}$$

where t_0 is the beginning particle time which is zero and the particle beta is approximated to be unity. the z position at the end, given that the initial position z_0 is zero, is then

$$z = z_0 + \beta\, c\, (t_{\text{ref}} - t) = -0.010075630518424195 \text{ meters} \tag{12}$$

Comparing the computed z here with what *Bmad* calculates shows a tiny difference due to the approximation made that $\beta = \beta_{\text{ref}} = 1$.

B.3 Chapter 9

Exercise 2:

In the **cavity.bmad** lattice file replace the appropriate lines with

```
beginning[p0c] = 2*mass_of(He)
lc: lcavity, l = 1e-6, voltage = mass_of(He), rf_frequency = 1e9
particle_start[z] = 1e-3
```

The command **show ele lc** will show that the beginning beta is 0.740 and the ending beta is 0.916 The ratio of the betas is the same ratio as the ratio of the beginning z (1.00 mm) to ending z (1.24 mm).

Exercise 3:

Approximately, for **phi0_err** values above about 0.2502 or below -0.2502 the particle will not make it through the **lcavity**.

B.4 Chapter 11

Exercise 1:

A knot based overlay to replace **ov2** would be:

```
k2: overlay = {q[k1]:{0, 0.7}, q[x_offset]:{0, 0.1}},
                    var = {hh}, hh = 0.01, x_knot = {0, 1}
```

Exercise 2:

A **group** element to vary the entrance edge of element **B**:

```
gedge: group = {B[start_edge]: ds}, var = {ds}
```

Exercise 3:

In the **simple.bmad** lattice file add:

```
gg: girder = {b, q}
```

B.5 Chapter 13

Exercise 1:

The **show curve r23** command will show that the curves are named **r23.g.x** and **r23.g.y**. [Note: using the alternative name that begins with "**orbit**" will not work here since there are multiple **orbit** plots.] To use the **r23.g.y** curve to draw the **design** horizontal orbit use the commands:

```
set curve r23.g.y data_type = orbit.x
set curve r23.g.y legend_text = "X"
set curve r23.g.y component = design
```

Note: By default, orbit plots have a non-blank **legend_text** which is why the **legend_text** needs to be changed. What happens if, instead of setting to "X", the **legend_text** is set to a blank string?

Exercise 2:

Define a **setit** command by:

```
alias setit set ele q vkick = [[1]]
```