Introduction and Tutorial to Bmad and Tao

David Sagan and Chris Mayes
July 31, 2019

Contents

1 A Guide for the Perplexed

This is a tutorial to introduce the reader to some of the concepts that are used by the *Bmad* toolkit for relativistic charged–particle and X-Ray simulations and as an initial training tutorial for using the *Tao* simulation program.

It is assumed that you know something about accelerator physics. For example, it is assumed that you know about beta functions, dispersion and closed orbits.

2 Overview: What is Bmad? What is Tao?

Bmad

Bmad is an open-source software library (aka toolkit) for simulating charged particles and X-rays. *Bmad* is not a program itself but is used by programs for doing calculations. The advantage of *Bmad* over a stand-alone simulation program is that when new types of simulations need to be developed, *Bmad* can be used to cut down on the time needed to develop such programs with the added benefit that the number of programming errors will be reduced.

Over the years, *Bmad* has been used for a wide range of charged-particle and X-ray simulations. This includes:

```
Lattice design                                  X-ray simulations
Spin tracking                                   Wakefields and HOMs
Beam breakup (BBU) simulations in ERLs          Touschek Simulations
Intra-beam scattering (IBS) simulations         Dark current tracking
Coherent Synchrotron Radiation (CSR)            Frequency map analysis
```

Tao

The disadvantage of *Bmad* is that, as a toolkit, one cannot perform any calculations without first developing a program. To get around this, the *Tao* program was developed. *Tao* is a general purpose simulation program, based upon *Bmad*. *Tao* can be used to view lattices, do Twiss and orbit calculations, nonlinear optimization on lattices, etc., etc. Additionally, *Tao*'s object oriented design makes it relatively easy to extend it. For example, it can be used for orbit flattening in an online machine control system.

2.1 Prerequisites

Bmad and *Tao* are generally used with Unix or Mac OS-X. While *Bmad* has been used with Windows, *Tao*, due to plotting and other issues, is not currently able to run under Windows.

Except when using a Python interface, *Tao* is accessed through the command line. Therefore you will need to run a terminal program to be able to run *Tao*.

3 Orientation

Distributions and Releases

A **Distribution** is a build of *Bmad* and associated libraries and programs (including *Tao*). A **Release** is like a **Distribution** except that it is done on the Linux computer system at CLASSE (Cornell's Laboratory for Accelerator-based Sciences and Education). For the purpose of this tutorial, **Releases** and **Distributions** are considered to be the same.

It is assumed that you already have access to a **Distribution** or a **Release** and that you have setup the requisite environmental variables. If this is not true, and there is no local *Bmad* Guru to guide you, download and setup instructions can be found on the *Bmad* web site (§4).

If everything is setup correctly, the environmental variable **ACC_ROOT_DIR** will point to the root directory of the Distribution or Release you are using. For a Distribution, this directory looks something like:

```
> ls $ACC_ROOT_DIR
PGPLOT                fftw                  lattice              sim_utils
bmad                  fgsl                  open_spacecharge     tao
bsim                  forest                openmpi              util
build_system          gnu_utilities_src     plplot               util_programs
cpp_bmad_interface    gsl                   production            xraylib
debug                 hdf5                  recipes_f-90_LEPP     xsif
examples              lapack                regression_tests
```

For an explanation of what directories contain what, see:

wiki.classe.cornell.edu/ACC/ACL/OffsiteDoc#DistDirs

4 Resources

More information is readily available at the *Bmad* and *Tao* web site:

http://www.classe.cornell.edu/bmad/

Links to the *Bmad* and *Tao* manuals can be found there as well as instructions for downloading and setup:

http://www.classe.cornell.edu/bmad/bmad-manual.pdf
http://www.classe.cornell.edu/bmad/tao-manual.pdf

After you have finished this tutorial, there are lattice file examples in the directory (§3)

```
$ACC_ROOT_DIR/examples
```

[From now on all directory paths are implicitly assumed to be with respect to **$ACC_ROOT_DIR**.] Example *Tao* initialization files are in the directory

```
tao/examples
```

To keep in touch with the latest *Bmad* developments, there are two mailing lists for *Bmad*. The **bmad-l** mailing list is used to send information on Bmad developments. The **bmad-devel-l** mailing list is for programmers. The volume of e-mail is small – Typically less than one a week. Instructions on how to sign up can be obtained from the *Bmad* web page.

4.1 Bmad Directories

A quick introduction to the most important directories in a **Distribution** or a **Release**:

bmad
The **bmad** directory holds the code for the *Bmad* library.

bsim
The **bsim** directory holds some *Bmad* based simulation programs for simulating synchrotron radiation (programs: **synrad** and **synrad3d**), dynamic_aperture (program: **dynamic_aperture**), intra beam scattering (programs: **ibs_linac** and **ibs_ring**), etc.

debug
The **debug** directory is like the **production** directory except that the executables in the **debug/bin** directory have been compiled with the **debug** flag. These executables can be used with a debugger but, since they typically run much slower than their counterparts in **production/bin**, these executables should not be used for normal running.

examples
The **examples** directory holds example programs along with example lattice files.

production
The **production** directory, which is created when the code in a **distribution** or **release** is compiled, contains the libraries, modules, and other files associated with compilation. In particular, the **production/bin** directory contains executable files for *Tao* and other simulation programs.

tao
The **tao** directory holds the code for the *Tao* program as well as example input files.

5 Bmad Based Simulation Programs

Below is a partial list of simulation programs that are based upon *Bmad*. All of these programs are included in any **distribution** or **release** in the **production/bin** directory.

Note: Before running any program, the appropriate environmental variables must be setup. [The setup for running programs and the setup for compiling *Bmad* based programs is one and the same.] Ask your local *Bmad* guru about this or consult the *Bmad* web page (§4) for directions.

bbu
The **bbu** program simulates the beam breakup instability in Energy Recovery Linacs (ERLs). The **bbu** code and documentation is in **bsim/bbu**.

dynamic_aperture
The **dynamic_aperture** program finds the dynamic aperture through tracking. Code and an example can be found in **bsim/dynamic_aperture**.

ibs_linac
The **ibs_linac** program simulates the effect of intra beam scattering (ibs) for beams in a Linac. Code and an example can be found in **bsim/ibs_linac**.

ibs_ring
The **ibs_linac** program simulates the effect of intra beam scattering (ibs) for beams in a ring. Code and an example can be found in **bsim/ibs_linac**.

lux
The **lux** program simulates X-ray beams from generation through to experimental end stations. Code and documentation can be found in the **lux** directory.

moga
The **moga** (multiobjective genetic algorithms) program does multiobjective optimization. The code for this program is in **util_programs/moga**.

synrad
The **synrad** program computes the power deposited on the inside of a vacuum chamber wall due to synchrotron radiation from a particle beam. The calculation is essentially two dimensional but the vertical emittance is used for calculating power densities along the centerline. Crotch geometries can be handled as well as off axis beam orbits. Code and documentation are in **bsim/synrad** and **bsim/code_synrad**.

synrad3d
The **synrad3d** program tracks, in three dimensions, photons generated from a beam within the vacuum chamber. Reflections at the chamber wall is included. Code and documentation are in **bsim/synrad3d** and **bsim/code_synrad3d**.

tao
Tao is a general purpose simulation programs described in §2. Code, documentation, and examples can be found in the **tao** directory. Documentation is also available from the web (§4).

6 Starting Tao

6.1 Preliminaries

To find out if everything is properly setup. Issue the command

```
> which tao
```

The response should be the location of the *Tao* executable which will look something like:

```
/Users/dcs16/bmad/bmad_dist/production/bin/tao
```

If the **which** command does not find an executable, consult your local *Bmad* Guru. One common problem is not initializing the *Bmad* environmental variables (which is the same setup used when programs are to be compiled).

6.2 Tao Startup

To run, *Tao* needs as input a *Bmad* lattice file that describes the machine to be simulated. Lattice files used in this tutorial are available for download from the web. Go to the *Bmad* web site (§4), Follow a link to either the *Bmad* or *Tao* manual page, and there should be a further link to example

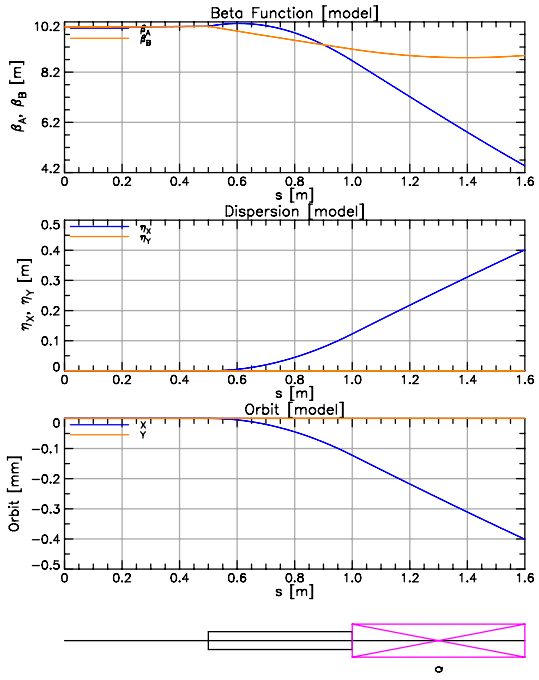


Figure 1: Initial graphics when *Tao* is run with the **simple.bmad** lattice file.

lattice files. Alternatively, the lattice files are available in any **Distribution** or **Release** (§3) in the directory:

```
examples/tutorial_bmad_tao/lattice_files
```

Each lattice shown in this tutorial lists the appropriate file name on the first line.

Note: Other lattice examples can be obtained from the directory:

```
examples/lattice_file_examples
```

As an example of how to run *Tao*, the lattice file shown in section §7 will be used. This lattice file is named **simple.bmad**. Make sure that the directory from which you are running *Tao* does not have a file called **tao.init** since this file will affect things (more on that later). Copy **simple.bmad** to your working directory and run *Tao* with the command:

```
> tao -lat simple.bmad
```

(alternatively just supply the full path name to **simple.bmad**.) *Tao* should open a window for plotting as shown in Figure 1. If this window is too large for your screen, you can adjust the size of the plotting window by using the **-geometry** option at startup. Example:

```
> tao -lat simple.bmad -geom 400x400
```

Consult the *Tao* manual for a list of command line arguments or start *Tao* with:

```
> tao -help
```

After initialization, there should be a "**Tao**>" prompt where you can type *Tao* commands.

6.3 Tao: Online Help

When *Tao* is running, to get a list of Tao commands, use the **help** command:

```
Tao> help

Type ''help <command>'' for help on an individual command
Available commands:
  alias                             read
  call                              restore
  change                            reinitialize
  clip                              run_optimizer
  continue                          scale
  derivative                        set
  end_file                          show
  exit                              single_mode
  flatten                           spawn
... etc...
```

Note: For brevity's sake, "**... etc...**" is used when the output has been truncated. Also the output shown is sometimes modified to fit the printed page.

Tao commands are documented in the "**Tao Commands**" (§10) chapter of the *Tao* manual. When running *Tao*, this same documentation can be displayed using the "**help** <**command**>" command where <**command**> is the name of a command. Example:

```
Tao> help set

The "set" command is used to set values for data, variables, etc.
Subcommands are:
  set beam_init {n@}<component> = <value>
  set particle_start {n@}<coordinate> = <value>
  set bmad_com <component> = <value>
  set csr_param <component> = <value>
  set curve <curve> <component> = <value>
  set data <data_name>|<component> = <value>
  set default <parameter> = <value>
  set element <element_list> <attribute> = <value>
  set floor_plan <component> = <value>
  set geodesic_lm <component> = <value>
... etc...

When running \tao, to see documentation on any of the subcommands, use the
\vn{help set <subcommand>} command. For example, \vn{help set element}
will show information on the \vn{set element} subcommand.
... etc...
```

Two commands, **set** and **show** are complicated enough so that they have "subcommands". For these commands, there is also a second help level for each subcommand. The output of"**help set**" and "**help show**" will give you a list of the **set** and **show** subcommands. Once you have a subcommand list, You can then type, for example, "**help set curve**" for help on the **set curve** subcommand.

6.4 Tao Initialization Files and Tao Command Files

Besides lattice files, *Tao* uses *Tao* specific initialization files for doing such things as configuring data plotting and setting optimization parameters. These initialization files are discussed in the **Tao Initialization** chapter of the *Tao* manual.

Tao also has command files which are files with *Tao* commands that can be executed at startup or while *Tao* is running with the **call** command. If a command file is used at startup, it is also an initialization file. By default, a file called **tao.startup**, if it exists, is used as the initialization command file.

Documentation on *Tao* initialization files is in the *Tao* manual in the "Tao Initialization" chapter.

Besides the initialization files supplied for this tutorial, example *Tao* initialization files can be found in a **Distribution** or **Release** (§3) in the directory:

```
tao/examples
```

Tao initialization can be split among several initialization files but there is always one main initialization file that will reference secondary initialization files if needed. The default name for the main initialization is **tao.init**. The main initialization file can be specified using the -**init** option when starting *Tao*.

Tao initialization files use **namelist** input. The general syntax is:

```
&namelist-name
  parameter1_name = value1
  parameter2_name = value2
  ... etc...
/
```

The namelist starts with **&namelist-name** where **namelist-name** is the name of the namelist. The namelist ends with the slash "/" character. In between, there is a set of lines that set appropriate parameter values. Example:

```
&tao_design_lattice
  n_universes = 1
  design_lattice(1)%file = "lat.bmad"
/
```

A detailed discussion of namelist syntax is given in the **Namelist Syntax** (§9.1) section of the **Tao Initialization** chapter of the *Tao* manual.

At the end of initialization, *Tao* will read in a command file if the appropriate one exists. The default is to read in a command file named **tao.startup**. The name of the startup command file may be specified either in the main initialization file or on the command line using the -**startup** option. After initialization, command files can be called using the **call** command.

See section §19.1 for an example.

6.5 Exercises

6.1 Create an initialization file named **tao.init** (which is the default name for the main initialization file) with a **tao_design_lattice** namelist that specifies **simple.bmad** as the lattice file. With this you should now be able to start *Tao* without the -**lat** option.

6.2 Create a command file that runs the command **show universe**. Use the call command to call this file while *Tao* is running (use the **help call** command to get information on the **call** command if needed).

6.3 In your **tao.init** initialization file from the first exercise, put in a **tao_start** namelist (See §9.2 "Beginning Initialization" in the *Tao* manual) that sets the **startup_file** parameter of this namelist to the name of the command file from the second exercise. Check that now when *Tao* is started the command file is automatically run at startup.

7 Introduction to Bmad Lattices

The basis of any *Bmad* based simulation is a lattice file. The following is a simple example:

```
! Lattice file: simple.bmad
beginning[beta_a] = 10.   ! m  a-mode beta function
beginning[beta_b] = 10.   ! m  b-mode beta function
beginning[e_tot] = 10e6   ! eV   Or can set beginning[p0c]

parameter[geometry] = open          ! Or closed
parameter[particle] = electron      ! Reference particle.

d: drift, L = 0.5
b: sbend, L = 0.5, g = 1, e1 = 0.1, g_err = 0.001   ! g = 1/design_radius
q: quadrupole, L = 0.6, k1 = 0.23

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

Note: Part I of the *Bmad* manual covers lattice syntax so please refer to that for information that is not covered here.

Some comments on the above lattice:

beginning[...], parameter[...]
: Global parameters (parameters of the lattice that are not associated with any one given lattice element) can be set using constructs like **beginning[...]**, **parameter[...]**, **particle_start[...]**, etc. See the **Lattice File Global Parameters** chapter in the *Bmad* manual for a complete list of parameters that can be set.

parameter[geometry]
: The **parameter[geometry]** parameter sets whether the geometry of the lattice is considered **open** (like a 1-pass accelerating linac) or **closed** (like a storage ring). If the lattice is closed, the closed orbit is used as the reference orbit and the computed beta functions correspond to the periodic solution. If the lattice is **open**, the orbit and beta functions are computed using the beginning orbit and beta functions as set in the lattice. The default geometry is **open** if the lattice contains an **lcavity** (linac accelerating RF cavity) element and is **closed** if no **lcavity** is present.

beginning[beta_a], beginning[beta_b]
: The **beginning[beta_a]** and **beginning[beta_b]** parameters set the beta functions at beginning of the lattice. The beginning beta functions will be only used by *Bmad* if the lattice geometry is set to **open**. Note: Some programs will use the labels **beta_x** and **beta_y** for the beta functions. This is inaccurate since the beta functions are associated with the normal modes of oscillation of the beam and, if there is horizontal/vertical coupling, the normal modes will not correspond to purely horizontal and purely vertical motion. In the limit of no coupling, the **a**-mode will correspond to horizontal oscillations and the **b**-mode will correspond to vertical oscillations.

beginning[e_tot]
: The **beginning[e_tot]** parameter sets the reference total energy at the beginning of the lattice. Alternatively, **beginning[p0c]** can be used to set the reference momentum at the beginning of the lattice.

parameter[particle]
: The **parameter[particle]** parameter sets the reference particle. Besides fundamental particles, ions can be used. For example, the reference particle could be set to **#12C+3** which is triply charged carbon-12. The default reference particle is a positron.

d: drift, ..., b: sbend, ..., q: quad, ...
: The lattice element named **d** is a drift element. That is, a field free region. The element named **b** is a dipole bend and the element named **q** is a quadrupole element. See the **Elements** chapter in the *Bmad* manual for more details.

lat: line = (...)
: A lattice consists of an ordered list of elements that the beam goes through. **lines** are used to define this ordered list. In this instance, a line called **lat** contains the elements **d**, **b**, and **q** in that order. See the "**Beam lines and Replacement Lists**" chapter of the *Bmad* manual for more details.

use, lat
: The **use** statement in a lattice file identifies the particular line used to construct the lattice. This is needed since a lattice file may define multiple lines and lines can contain sub-lines, etc.

[Note: The above discussion assumes that the lattice has only one **branch**. Branches will be discussed in more detail later.]

7.1 Exercises

7.1 Element **b** is an **sbend**. What exactly is an **sbend**? And what is the meaning of the **g** and **g_err** parameters of element **b**? [To find the answers look in the **Elements** chapter of the *Bmad* manual.]

7.2 Make a simple FODO lattice with a single cell: [drift, quad1, drift, quad2], and set the quadrupole **k1** values to, say $-1\,\mathrm{m}^{-2}$ and $1\,\mathrm{m}^{-2}$.

7.3 Make a FODO lattice with ten of these cells. [Hint: Make the "lat" line a subline of another line and use a repetition count as explained in section 6.2 "Beam Lines and Lattice Expansion" of the Bmad manual.

8 Tao Show Command

The **show** command of *Tao* is used to display information on anything from the makeup of lattice elements to particle positions within a tracked particle beam. This chapter gives an introduction to using the **show** command.

Start *Tao* as explained in section §6.2 with the lattice file **simple.bmad**. The **show** command has a large set of subcommands, To see the list of subcommands, use the **help show** command:

```
Tao> help show
The "show" command is used to display information.
Format:
  show {-append <file_name>} {-noprint} {-no_err_out} <subcommand>
  show {-write <file_name>} {-noprint} {-no_err_out} <subcommand>

"<subcommand>" may be one of:
  alias                 ! Show aliases .
  beam ...              ! Show beam info .
  branch ...            ! Show lattice branch info .
  building_wall         ! Show building wall info .
  ... etc...
  wall ...              ! Show vacuum chamber wall info .
  wave                  ! Show wave analysis info .

The "show" command has "-append" and "-write" optional arguments which can be
used to write the results to a file.  The "show -append" command will ...
```

Thus, for example, to see information on using the **show branch** subcommand, use the command **help show branch**.

Output from the **show** command may be put in a new file or appended to an existing file using the -**append** or -**write** switches which must appear before <**subcommand**> on the command line. For example:

```
Tao> show -write abc.dat lattice
```

This will dump lattice information to a file named **abc.dat**.

8.1 To Show a List of Elements in the Lattice

To show the elements in the lattice use the **show lattice** subcommand:

```
Tao> show lat
# Values shown are for the Exit End of each Element:
# Index  name      key                       s       l    beta     phi ...
#                                                            a       a ...
     0  BEGINNING Beginning_Ele         0.000     ---   10.00   0.000 ...
     1  D         Drift                 0.500   0.500   10.03   0.050 ...
     2  B         Sbend                 1.000   0.500    7.87   0.104 ...
     3  Q         Quadrupole            1.600   0.600    3.50   0.217 ...
```

```
     4  END       Marker                1.600    0.000     3.50    0.217 ...
# Index  name      key                       s        l     beta      phi ...
#                                                            a        a ...
# Values shown are for the Exit End of each Element:
```

The **s** column shows the longitudinal position from the beginning of the lattice (§12.1) and the **l** column shows the length of the elements.

Comparing the output of the **show lattice** command to the **simple.bmad** lattice file, notice that *Bmad* adds two extra elements to the lattice. A zero length beginning element called **BEGINNING** and a zero length marker element at the end called **END**.

The "**show lat**" command, like many other commands, has optional parameters to customize the table of information printed:

```
Tao> help show lat

Syntax:
  show lattice {-0undef} {-all} {-attribute <attrib>} {-base}
      {-blank_replacement <string>}  {-branch <name_or_index>}
      {-custom <file_name>} {-design} {-floor_coords} {-lords} {-middle}
      {-no_label_lines} {-no_tail_lines} {-no_slaves} {-orbit}
      {-remove_line_if_zero <column #>} {-s <s1>:<s2>} {-tracking_elements}
      {<element_list>}

Show a table of Twiss and orbit data, etc. at the specified element locations.
The default is to show the parameters at the exit end of the elements.
... etc...
```

For example, the -**attribute** switch makes it easy to make a custom table of element attributes:

```
Tao> show lat -no_tail_lines -attrib b1_gradient
 Values shown are for the Exit End of each Element:
 Index  name      key                       s       l          b1
                                                          gradient
      0  BEGINNING Beginning_Ele        0.000     ---          ---
      1  D         Drift                0.500   0.500          ---
      2  B         Sbend                1.000   0.500  0.0000E+00
      3  Q         Quadrupole           1.600   0.600 -7.6620E-03
      4  END       Marker               1.600   0.000          ---
```

When using the -**attribute** switch the first five columns are fixed and additional columns are specified by each instance of -**attribute** appearing on the command line. In this example there is one additional column showing the **b1_gradient** element attribute. The string — is printed for elements that do not have this attribute (The string used when an element does not have a particular attribute may be changed using the appropriate **show lattice** switches).

8.2 To Show the Attributes of a Lattice Element

Use the **show element** command to show the attributes of a particular lattice elements. Example:

```
Tao> show ele b   ! Or "show ele 2" since element B has index 2.

 Element #                2
 Element Name: B
 Key: Sbend
 Sub Key: SBend
 S_start, S:    0.500000,    1.000000
 Ref_time:  3.340005E-09

 Attribute values [Only non-zero/non-default values shown]:
    1   L                           =  5.0000000E-01 m
    6   G                           =  1.0000000E+00 1/m
    7   G_ERR                       =  1.0000000E-03 1/m
    8   RHO                         =  1.0000000E+00 m
   10   FRINGE_TYPE                 =  Basic_Bend (7)
   11   FRINGE_AT                   =  Both_Ends (3)
   12   HIGHER_ORDER_FRINGE_TYPE    =  None (1)
   13   SPIN_FRINGE_ON              =  T (1)
   14   EXACT_MULTIPOLES            =  Off (1)
   19   E1                          =  1.0000000E-01 rad
   29   L_SAGITTA                   =  3.1087578E-02 m
     ... etc...

Twiss at end of element:
                          A              B            Cbar    ...
  Beta (m)          8.65422245     9.11594461  |   0.00000000  ...
  Alpha             3.56155250     0.86569936  |   0.00000000  ...
  Gamma (1/m)       1.58126929     0.19190939  |   Gamma_c =   ...
  Phi (rad)         0.10144612     0.10228316           X     ...
  Eta (m)           0.12252488     0.00000000     0.12252488   ...
  Etap              0.47990496     0.00000000     0.47990496   ...

Orbit:  Electron   State: Alive
         Position[mm] Momentum[mrad]        Spin   |
  X:       -0.12240995    -0.47942554               | t_particle [sec]:   ...
  Y:        0.00000000     0.00000000               | t_part-t_ref [sec]: ...
  Z:        0.02055389     0.00000000               | (t_ref-t_part)*Vel  ...
```

Note: By default, only non-zero attributes are shown. Use the -all option to see all the attributes.

8.3 Using Wild Cards in Element Names

Wild card characters can be used with element names. The wild card characters are:

```
"*" -- Matches to any number of characters.
"%" -- Matches to any single character.
```

The general syntax is:

```
  <name_with_wild_card_characters>                  ! or
  <element_type>::<name_with_wild_card_characters>
```

where <**element_type**> is the type of element (**drift**, **quadrupole**, etc.). [Actually the syntax is a bit more complicated than this. See the "**Matching to Lattice Element Names and Other Attributes**" section (§2.8) in the *Bmad* manual.]

For example, to show all elements whose name begins with "Q" use the **show element** command:

```
Tao> show ele q*

         1  Q                                                    1.000
Number of Matches: 1
```

Or to show all sbend elements the command is:

```
Tao> show ele sbend::*

         2  B                                                    2.000
Number of Matches: 1
```

Element names with wild cards can also be used with the **show lattice** command. For example:

```
Tao> show lat q*

# Values shown are for the Exit End of each Element:
# Index  name      key                       s       l    beta     phi ...
#                                                          a       a ...
     3  Q         Quadrupole             1.600   0.600    3.50   0.217 ...
# Index  name      key                       s       l    beta     phi ...
#                                                          a       a ...
# Values shown are for the Exit End of each Element:
```

Note: When **rbend** (rectangular bend) elements are read in, internally they are converted to **sbend** (sector bend) elements. Thus, a search for **sbend** elements will include all **rbend** elements.

8.4 Exercises

8.1 Lattice elements have string attributes named **type**, **alias** and **descrip**. Modify any lattice so that, say, elements have a non-blank **alias**. Open *Tao* with this lattice and use the **show element** command to, say, search for all elements whose **alias** attribute begins with the letter "**z**". [Hint: See the "**Matching to Lattice Element Names and Other Attributes**" section (§2.8) in the *Bmad* manual.]

8.2 Start *Tao* with a lattice with, say, multiple elements named **q** and determine how to show the second element in the lattice with the name **q**. [Hint: See the "**Matching to Lattice Element Names and Other Attributes**" section (§2.8) in the *Bmad* manual.]

8.3 Some Field strength quantities like the gradient field of a quadrupole can be specified using a reference momentum normalized value (**k1** for a quadrupole) or a unnormalized value (**b1_gradient** for a quadrupole). Use the **show element** command to see the value for the **b1_gradient** of element **q** in the **simple.bmad** lattice. Modify the file **simple.bmad** so that quadrupole field is specified by **b1_gradient** instead of **k1**. Verify that with this lattice, if **parameter[e_tot]** is modified, **k1** will change accordingly but **b1_gradient** will not.

8.4 Explore using other **show** commands. For example, what does **show universe** show?

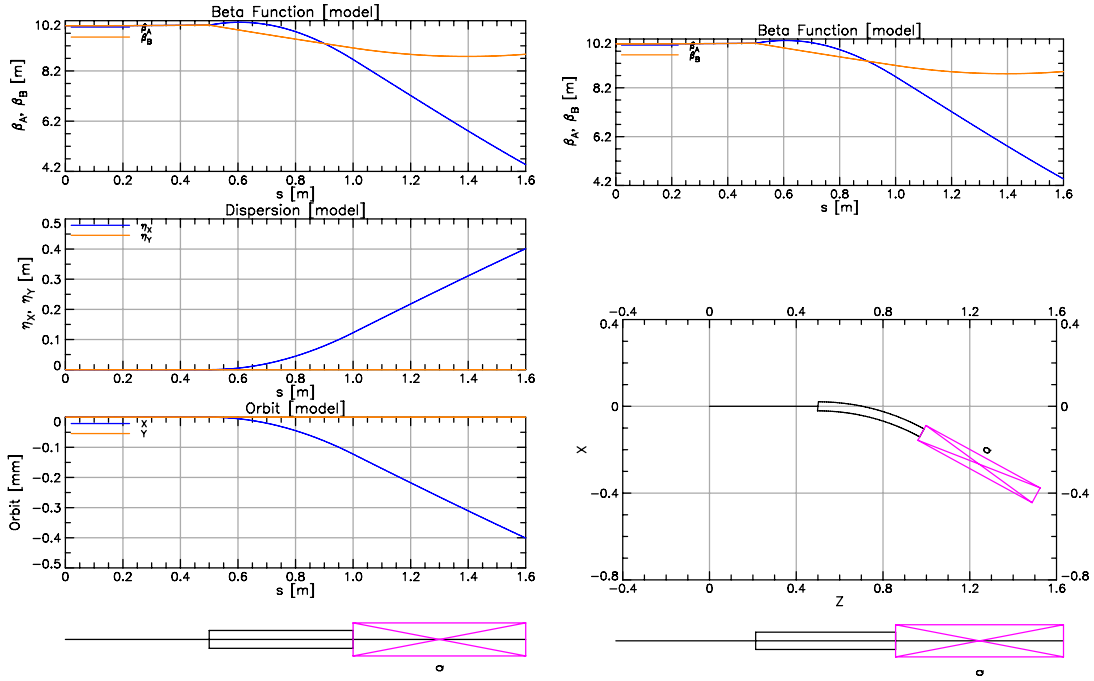
8.5 What is the command to show a list of prior commands that you have typed in?

9 Introduction to Plotting in Tao

First: Start *Tao* as explained in section §6.2 with the lattice file **simple.bmad**.

The default is to have three plots as shown in Figure 2a: beta, dispersion, and orbit, along with what is called a **lat_layout** plot situated at the bottom of the window that graphically shows the lattice elements as a function of longitudinal position. Note: If you do not want *Tao* to display the plot window, use the **-noplot** option on the command line when you start *Tao*.

Plotting is described in the **Plotting** chapter in the *Tao* manual and the setup of custom plots is described in the **Initializing Plotting** (§9.10) section of the **Tao Initialization** chapter.



(a) Initial graphics when *Tao* is run with the **simple.bmad** lattice file.

(b) Graphics after a **place r22 floor_plan** command.

Figure 2: Example *Tao* graphics.

9.1 Nomenclature

A given "**plot**" has a number of "**graphs**" associated with it. A **graph** consists of horizontal and vertical axes (which may or may not be displayed) along with a number of "**curves**". A **curve** is a set of (x, y) points to be plotted. For the curves in Figrue 2a, *Tao* calculates enough points per curve so that the line drawn to connect the points looks smooth. Some graphs do not have any associated curves.

In Figure 2a, all four plots have exactly one associated graph. The **lat_layout** graph does not display its axes and has no associated curves. The other graphs each have two associated curves.

9.2 Displaying a Plot

To change the plots that are being displayed, you have to tell *Tao* what plot to draw and where to draw it. The **show plot -templates** command prints the list of plots that can be displayed:

```
Tao> show plot -templates

Templates:
   Plot                              Description
   ----------------------------      -------------------
   alpha                             Twiss alpha function
   b_div_curl                        Magnetic Field Divergence and Curl.
   b_field                           Magnetic Field Along Orbit
   beta                              Twiss beta function
   bunch_sigma_xy                    Bunch transverse sigmas
   bunch_R1_R2                       Bunch phase space plot.
   cbar                              Cbar coupling matrix
   dbeta                             Chromatic normalized beta beat
... etc...
```

The output of **show plot -templates** shows plot "**templates**". A plot template specifies the parameters needed to draw the plot: what is to be plotted, the number of associated graphs, the x and y-axis scales, colors to be used for the curves, etc. For example, how many graphs are associated with a plot, etc. *Tao* defines a set of default templates and custom ones can be defined by constructing the appropriate initialization file. Directions for constructing templates are given in the **Initializing Plotting** (§9.10) section of the **Tao Initialization** chapter of the *Tao* manual.

To see where in the plot window templates can be placed, use the **show plot** command without any additional arguments:

```
Tao> show plot
                                          Location on Page
Plot Region          <-->  Plot           x1    x2    y1    y2
-----------               ---------------------------------------
layout               <-->  lat_layout    0.00  1.00  0.00  0.15
r11                  <-->                0.00  1.00  0.15  1.00
r12                  <-->                0.00  1.00  0.58  1.00
```

```
r22                  <-->                        0.00   1.00   0.15   0.58
r13                  <-->  beta                  0.00   1.00   0.72   1.00
r23                  <-->  dispersion            0.00   1.00   0.43   0.72
r33                  <-->  orbit                 0.00   1.00   0.15   0.43
r14                  <-->                        0.00   1.00   0.79   1.00
... etc...
```

The output of the command shows a list of plot "**regions**" along with what plot (if any) is associated with a given region. A plot region is a rectangular box within the plot window into which a plot can be placed as illustrated in Figure 3. With the present example, there are four regions that have a plot. For example, The **r13** region has a **beta** plot.

The position of a region within the plotting window is determined by four numbers as shown in Figure 3. **x1** and **x2** determine the horizontal position with a value of 0.0 corresponding to the left border edge (which is a distance **x1b** from the window edge) and 1.0 corresponding to the right border edge (which is a distance **x2b** from the window edge). Similarly, the **y1** and **y2** numbers determine the vertical position with 0.0 corresponding to the bottom border edge and 1.0 corresponds to the top border edge.

Values for **x1**, **x2**, **y1**, and **y2** are given in the right most 4 columns of the output of the **show plot** command. In the present case, for example, the **r13** region has **x1** = 0 and **x2** = 1 so the plot occupies the full horizontal width of the page. See the **Initializing Plotting** (§9.10) section of the **Tao Initialization** chapter of the *Tao* manual for more details.

The **place** command is used to place a **template** plot into a plot **region**. Example:

```
Tao> place r22 floor_plan
```

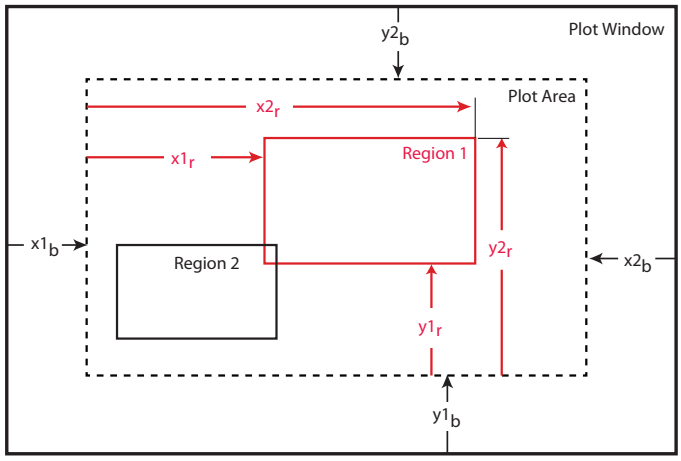


Figure 3: The plot window is divided up into a number of rectangular **regions** into which a plot **template** can be placed. Regions can overlap but if a plot is placed in a given region, plots in any other region that overlap are cleared. The border, within which regions are placed, is displaced from the edge of the window by distances **x1b**, **x2b**, **y1b**, and **y2b**.

This places a **floor_plan** plot in the **r22** region (the **floor_plan** plot draws a bird's eye view of the machine). The result is shown in Figure 2b. Plots associated with regions that overlap the region that is used in the **place** command are erased.

Example place commands:

```
Tao> place r22 none    ! Clear a region
Tao> place * none      ! Clear all regions
```

Note: Plots of the same type can be placed in multiple regions. For example, there could be multiple orbit plots displayed with each plot, say, having differing x-axis scaling.

9.3 Scaling Plots

Plots can be scaled vertically using the **scale** command:

```
Tao> scale beta 0 20  ! Set y_min = 0, y_max = 20.
Tao> scale r13 0 20   ! Same as above if the beta plot is in the r13 region.
Tao> scale beta       ! Tao will calculate nice bounds.
Tao> scale all        ! "all" = all plots.
Tao> scale            ! Same as "scale all".
```

The **x_scale** command can be used to scale the horizontal axis and the **xy_scale** command can be used to simultaneously scale the x and y axis (used for **floor_plan** plots).

```
Tao> x_scale beta 0.8 1.0   ! Scale horizontal axis
Tao> xy_scale floor_plan    ! Combined scale and x_scale.
```

9.4 Getting Information on a Plot

To see the parameters associated with a given plot use the command "**show plot**" with the name or region of the plot. Example:

```
Tao> show plot beta  ! or "show plot r13" is equivalent.

Plot:  beta
Region:  r13
Visible                = T
Location [x1,x2,y1,y2] = .000  1.000  .717  1.000
x_axis_type          = s
... etc...
x%draw_label         = T
x%draw_numbers       = T
autoscale_x          = F
autoscale_y          = F
autoscale_gang_x     = T
autoscale_gang_y     = T
n_curve_pts          = -1
```

```
Graphs:
   g
```

The output shows that the **beta** plot is associated with the region **r13**. Also the output shows that there is one associated graph called "**g**". This graph can be referred to as "**beta.g**" or "**r13.g**".

To display information on a graph use the command **show graph**. Example:

```
Tao> show graph beta

Region.Graph: r13.g
Plot.Graph:   beta.g
type                            = data
title                           = Beta Function
title_suffix                    = [model]
component                       = model
margin                          =   0.15    0.06    0.12    0.12  %BOX
scale_margin                    =   0.00    0.00    0.00    0.00  %GRAPH
... etc...
y%max                           =   1.02000000E+01
y%min                           =   4.20000000E+00
y%major_div                     = 3
y%major_div_nominal             = 4

Curves:
   a
   b
```

Here the "**show graph beta**" command works since there is only one graph associated with the beta plot. The output shows that the graph has two associated curves called **a** and **b**. The **a** curve can be referred to as "**beta.g.a**" or **r13.g.a**" with similar names for the **b** curve.

To display information on curves use the "**show curve**" command. using the "-**line**" option with this command will display the set of points that are used to draw the curve:

```
Tao> show curve -line beta
Region.Graph.Curve: r13.g.a
                    r13.g.b
Plot.Graph.Curve:   beta.g.a
                    beta.g.b
data_source          = lat
... etc...
draw_symbol_index    = F
smooth_line_calc     = T
line%width           = 2
line%color           = 4  Blue
line%pattern         = 1  solid
... etc...

# Smooth line points:
# index        x-axis             a             b
      1  0.000000E+00  1.000000E+01  1.000000E+01
```

```
      2   4.000000E-03   1.000000E+01   1.000000E+01
      3   8.000000E-03   1.000001E+01   1.000001E+01
      4   1.200000E-02   1.000001E+01   1.000001E+01
... etc...
```

9.5 Custom Plotting

The default template plots that *Tao* defines are sufficient for many purposes but at times you may want to define your own. Custom plotting is out of the scope of this tutorial but the reader is referred to the section on **Initializing Plotting** (§9.10) in the **Initializing Tao** chapter of the *Tao* manual. Figure 4 shows an example of what can be done.

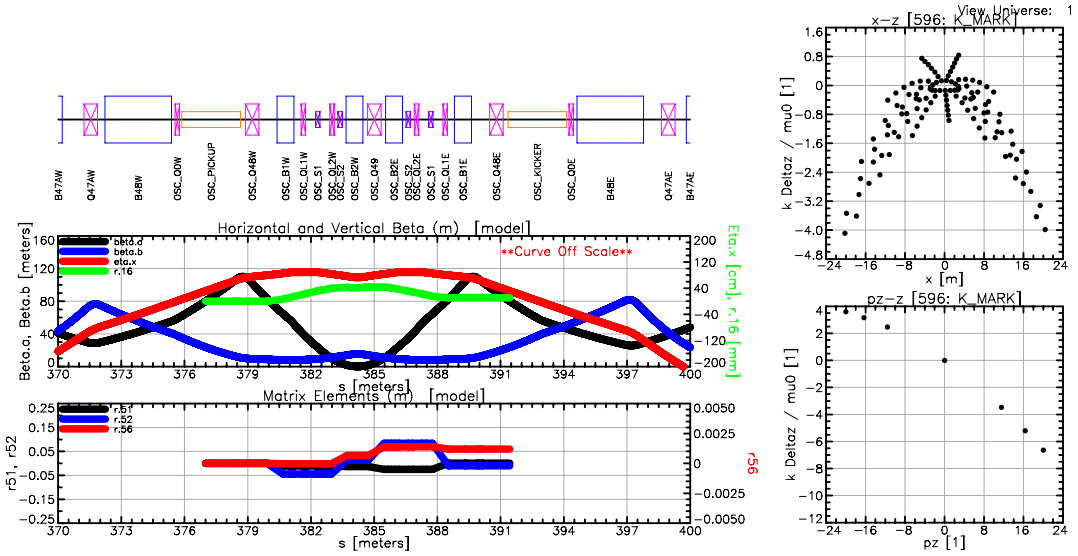


Figure 4: Example of what can be done with custom plotting.

9.6 Exercises

9.1 Try placing other plot templates onto the plot page, such as **b_field**.

9.2 Use the **set** command to set the **draw_symbols** curve logical for the curves in the **beta** plot to True. What does the **beta** plot look like now? [Note: The **set** command is covered in more detail in Section §10.1.]

10 Model Design and Base Lattices in Tao

When *Tao* runs, *Tao* instantiates three lattices (Technically, *Tao* instantiates three lattices per **universe**. See §10.3):

Design Lattice

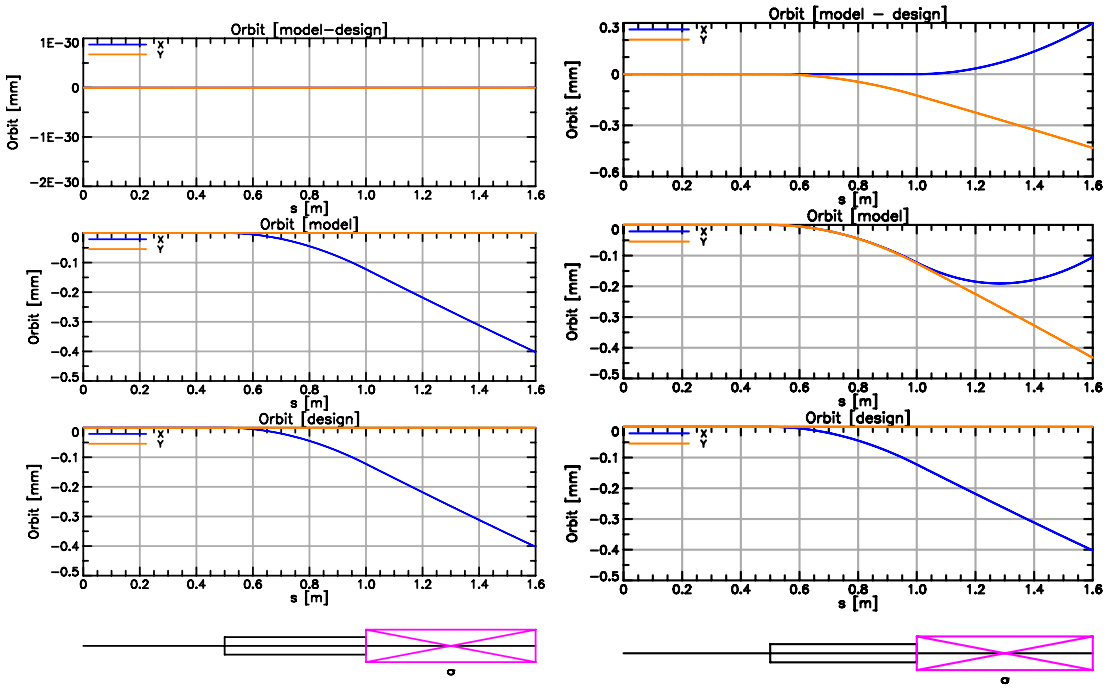
The **design** lattice corresponds to the lattice read in from the lattice description file(s). Parameters in this lattice are never varied.

Model Lattice

Initially, when Tao is started, The **model** lattice has the same parameters as the **design** lattice. Essentially, 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.



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

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

Figure 5

10.1 Changing Model Parameters

To see the difference between the **model** and **design** lattices, start *Tao* as explained in section §6.2 with the lattice file **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 5a. 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 integer, string and logical parameters. The result is shown in Figure 5b. Since now the **model** lattice is not the same as the **design** lattice, the difference orbit is non-zero.

10.2 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 subsection, 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 6. The bottom plot of the orbit difference between **model** and **base** is not the same as the orbit difference between **model** and **design**.

10.3 Multiple Universes

Tao has a concept called a **universe**. A **universe** consists of **model**, **design**, and **base** lattice along with "**data**" (§19.2) which can, say, represent something like an orbit measurement. Multiple

universes may be defined. This is useful in a number of situations. For example, if multiple orbit measurements have been made with steering magnets changing between measurements, each measurement could be associated with a different universe and the entire collection of measurements could be analyzed simultaneously.

The discussion of multiple universes is beyond this tutorial and the interested reader is referred to the *Tao* manual. in the chapter on “Overall Organization and Structure”.

10.4 Exercises

10.1 Show a plot of the orbit as calculated from the **base** lattice.

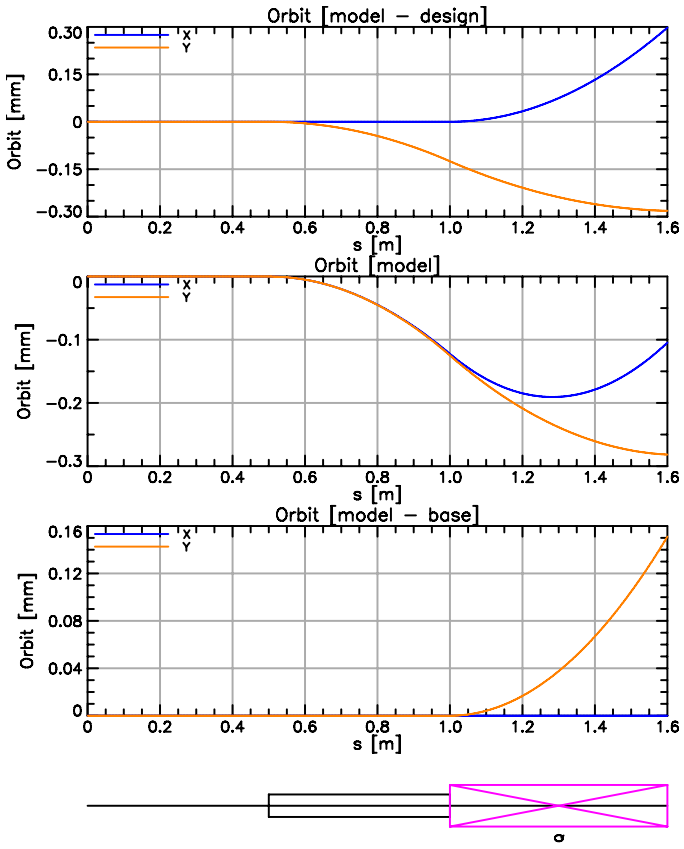


Figure 6: The **base** lattice is used to view changes when the reference lattice configuration does not correspond to the **design** lattice.

11 Control Elements

11.1 Overview

Control elements are elements that control the parameters of other elements. There are three types of control elements: **groups**, **overlays**, 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 (§11.6) are used for simulating support structures.

Note: **Group**, **overlay**, 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 (§15) and **superposition** lords (§14) are called "**major lords**".

11.2 Example Lattice

The lattice used to illustrate control elements is named **control.bmad**:

```
! 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[e1]: 0.4*sqrt(z)}, var = {z}
```

Notes:

- 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 for overlay **ov1** that are used to calculate the values of the two controlled attributes are $\mathbf{a+b^2}$ for **q[k1]** and **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 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 0.2. Control variables default to a value of zero.

11.3 Control Element Organization in the Lattice

Start *Tao* as explained in section §6.2 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 first 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.

11.4 Overlay Control

To see how things are controlled, use the **show element** command. Examining lord **ov1** 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 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 a 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)
```

A **group** element not only has associated variables (in this case a single variable **z**) but *Bmad* also keeps a record of what is called the "old" value (**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**.

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.

What is a group element useful for? 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 some amount that you compute. Here you don't care about the value of the sextupole strengths, you only want to vary the sextupole strengths by a delta. So the sextupole "knob" can be simulated using a **group** controller which may look like:

```
raw_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.

11.6 Girders

A third type of controller is the **girder** element which can be used to simulate support structures like an I-beam that supports a number of magnets or an optical table supporting an optical setup. This is discussed further in Section §12.5. Also see the *Bmad* manual for more details.

11.7 Exercises

11.1 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 that uses knot points that mimics the action of **ov1** at least over some limited interval.

11.2 **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 (§3.21) in the **Elements** chapter of the *Bmad* manual for more details.

12 Machine Coordinates and Patch Elements

12.1 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 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)** is 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, the **(x, y, s) local** (also called **laboratory**, also called **reference**) curvilinear coordinate system is used to describe particle positions and also to describe the nominal (that is, without any "misalignments") position of the lattice elements. **s** is the longitudinal coordinate and **(x, y)** are the transverse coordinates. The curve defined by **x** = **y** = 0

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 the elements 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.

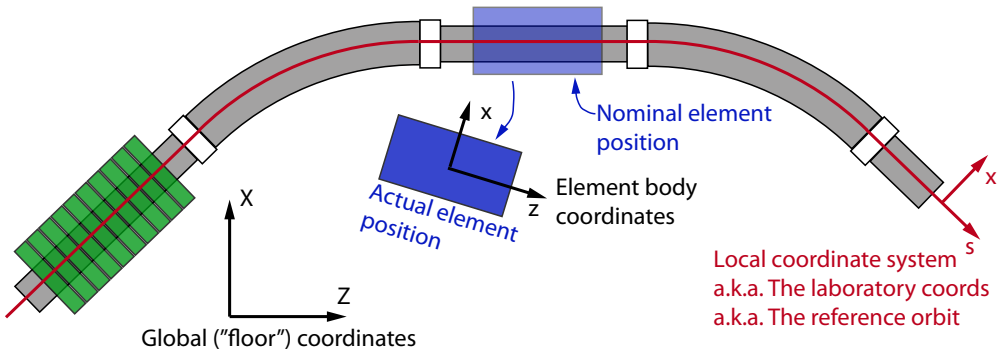


Figure 7: The three coordinate systems used to describe lattice element positioning: **Global**, **reference**, and **element body** coordinates.

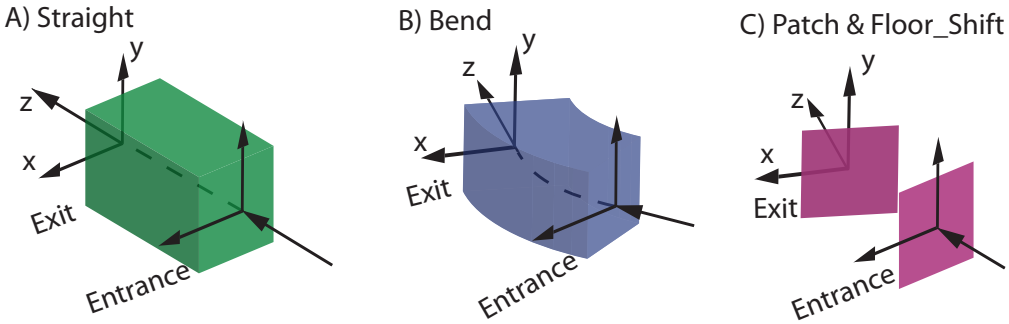


Figure 8: Lattice element geometry types: **Straight**, **bend**, and **patch**.

12.2 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. These three types are called **straight**, **bend** and **patch** based upon how the element body or laboratory coordinates transform as a function of the longitudinal **s** position from the **entrance** end of the element to the **exit** end.

Straight Geometry

With **straight** elements like **drift**s and **quadrupole**s, the coordinates transform as a translation along the **z**-axis so that the **z**-axis at the **exit** end is co-linear with the **entrance** **z**-axis (Figure 8A).

Bend Geometry

Sbend and **rbend** dipole elements have a **bend** geometry where the coordinates rotate about an axis which is in the **x**-**y** plane of the **entrance** (and **exit**) coordinates (Figure 8B).

Patch Geometry

Patch and **floor_shift** elements have a **patch** geometry where the exit coordinates can be arbitrarily positioned with respect to the entrance coordinates (Figure 8C). See §12.6.

Note: The fourth geometry type, used for X-ray simulations, is used with **mirror**, **multilayer_mirror**, and **crystal** elements,

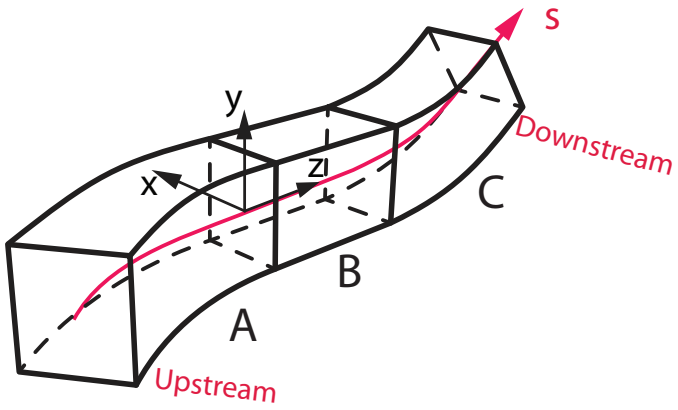


Figure 9: The **local** coordinate system is constructed by taking the ordered list of lattice elements and connecting the **exit** frame of one element to the **entrance** frame of the next.

12.3 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.

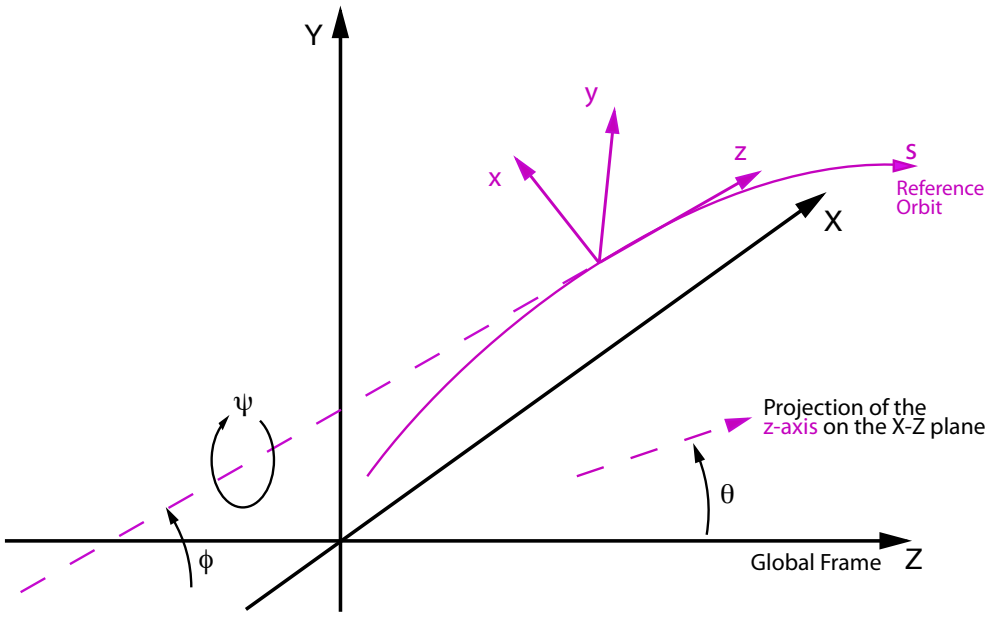


Figure 10: The global coordinate system.

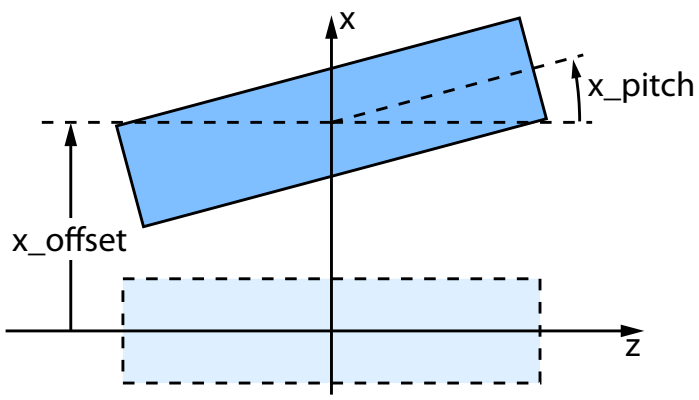
12.4 Laboratory Coordinates Relative to Global Coordinates

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:

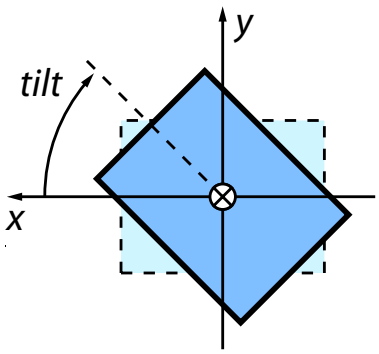
- (**X**, **Y**, **Z**) global position
- θ azimuth angle in the (X, Z) plane.
- ϕ elevation angle
- ψ roll angle.

Notes:

- The default is for the beginning of the lattice (**s** = 0) is to have the local **(x, y, z)** aligned with the global **(X, Y, Z)** 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

12.5 Element Misalignments

Once the **reference** coordinate system is established, the position of any physical element with 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.

Example:

```
! 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
```

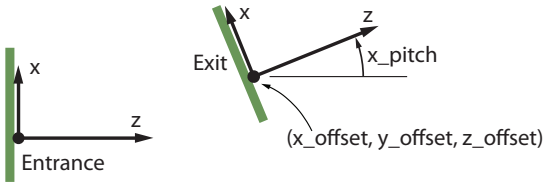
```
use, lat          ! Line used to construct the lattice
```

Start *Tao* as explained in section §6.2 with the lattice file **misalign.bmad**. The misalignment can be viewed using the **-floor** option with the **show element** command:

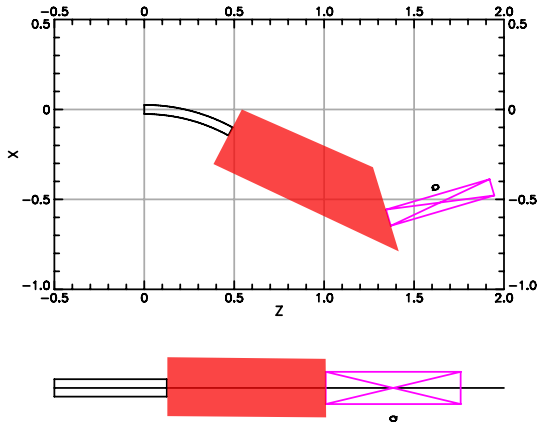
```
Tao> show ele q -floor

 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** (§11.6) 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.



(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

12.6 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**)
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.

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* as explained in section §6.2 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.

12.7 Exercises

12.1 Setup a lattice with a **girder** element and see that when the **girder** is misaligned that a supported element will have **_tot** attributes different from the misalignment attributes.

12.2 Create a lattice with elements **drift**, followed by a **mirror**, followed by a **drift**. Give the **mirror** a finite **graze_angle** and verify that the laboratory coordinate after the **mirror** are rotated by twice the **graze_angle** with respect to the coordinates before the **mirror** so that a photon traveling on the zero-orbit before the **mirror** will stay on the zero-orbit after the **mirror**.

13 Particle Phase Space Coordinates

The previous chapter showed how to describe the placement of lattice elements. This chapter covers how to describe particle trajectories.

13.1 Particle Phase Space Coordinates

The "**reference orbit**" of the local coordinate system is the curve defined by **x** = **y** = 0. At any point on the reference orbit the local coordinate system defines **x** and **y** axes and the **z** axis is defined to be tangent to the **s** axis as shown in Figure 13a.

Given a particle at some point on its trajectory (blue dot in Figure 13a), there is a point **s** on the reference orbit such that in the (**x**, **y**, **z**) coordinate frame with origin at **s** the particle's position is in the x-y plane with z = 0 as shown in Figure 13a. 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))
```

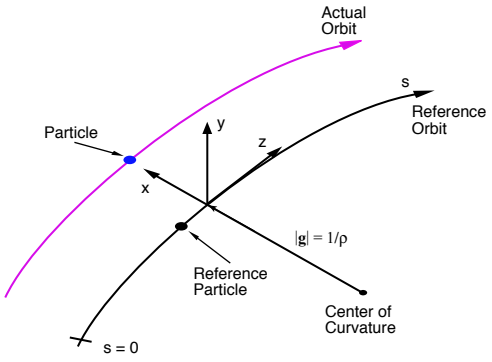
where **t(s)** is the time of the particle. From now on, to simplify the notation, the **s** dependence will be dropped.

For tracking purposes, canonical phase space coordinates are used with the convention that upper case **P** denotes (unnormalized) momentum (Figure 13b) and lower case **p** denotes phase space momentum. The phase space coordinates are denoted

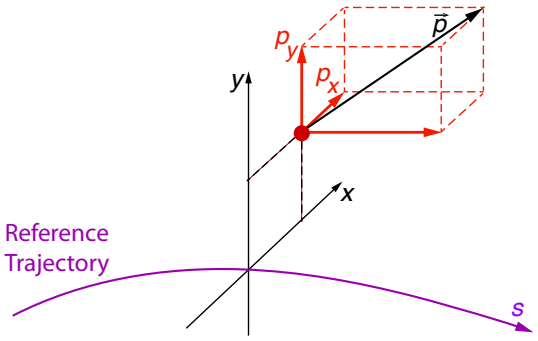
```
(x, px, y, py, z, pz)
```

where

```
px = Px / P0
py = Py / P0
```

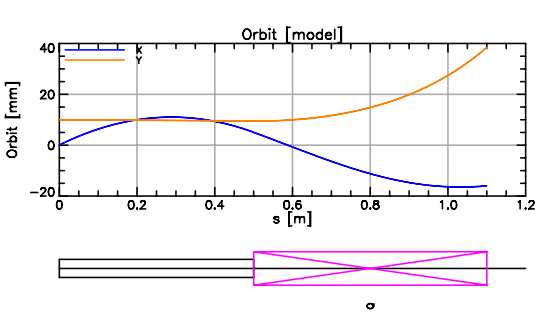


(a) Particle coordinate positions are relative to the reference orbit.

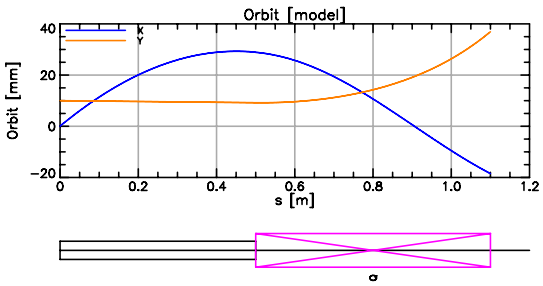


(b) Particle phase space.

Figure 13



(a) Initial orbit.



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

Figure 14

```
pz = (P - P0) / P0
z  = c * β * (t_ref - t)
```

with

- **P0** is the reference momentum which is set in the lattice file.
- **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. 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.

13.2 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* as explained in section §6.2 with the lattice file **orbit.bmad**. Here spin tracking is turned on (**bmad_com[spin_tracking_on] = T**) and a non-zero initial orbit is set using **particle_start** parameters. The resulting orbit is shown in Figure 14a.

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

```
> tao -lat phase_space.bmad

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 14b.

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  ...  0.000000E+00
     1  B         Sbend         ...    5.027722E-03  ... -1.414516E-01
     2  Q         Quadrupole    ...   -1.599068E-02  ... -7.350543E-02
     3  END       Marker        ...   -1.599068E-02  ... -7.350543E-02
```

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]: ...
```

13.3 Reference Energy and the Lcavity and RFcavity Elements

Lcavity and **RFcavity** elements both represent RF cavities. The difference is that the reference energy at the exit end of an **lcavity** is set so that a particle entering the cavity with zero phase space coordinates leaves with zero phase space coordinates and in particular phase space **pz** will be zero at the exit end. On the other hand, **rfcavity** elements do not affect the reference energy. Example:

```
! 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 section §6.2 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.

Questions:

- What are the k1 and b1_gradient values for the first q1 and the second q1? Do you understand this?
- What are the k1 and b1_gradient values for the first q2 and the second q2?

14 Superposition

Superposition is used when elements overlap spatially. In such a case, *Bmad* creates "**slave**" elements that will be tracked through and "**lord**" elements that represent the individual elements. Some examples will make this clear. Note: Superposition is discussed in the "Superposition and Multipass" chapter in the Bmad manual.

14.1 Example 1

Superposition works by defining a **line** as done in any lattice file and then defining an element that will be "superimposed" on top of the line. To superimpose an element you need to specify where the element is placed. To do this, a reference position is specified and the superimposed element is placed at that position shifted by a specified offset as illustrated in Figure 15. The lattice file **superimpose1.bmad** illustrates how superposition is done.

```
! Lattice File: superimpose1.bmad

beginning[beta_a] = 10.    ! m  a-mode beta function
beginning[beta_b] = 10.    ! m  b-mode beta function
beginning[e_tot] = 10e6    ! eV   Or can set p0c
parameter[geometry] = open      ! or closed

q: quadrupole, L = 1, k1 = 0.2
d: drift, L = 1

m1: marker, superimpose, ref = q, ref_origin = beginning, offset = 0.3
m2: marker, superimpose, ref = q, ref_origin = end, offset = 0.4

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

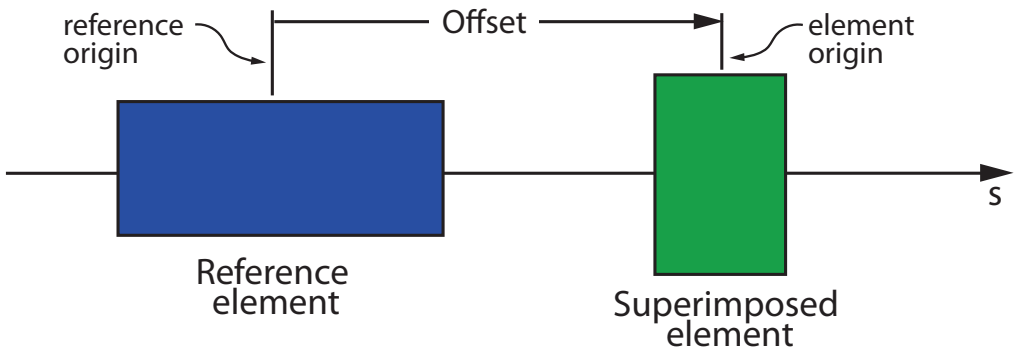


Figure 15: The placement of superimposed elements is determined by an offset from a reference element.

Here two marker elements named **m1** and **m2** are superimposed on the lattice. The offsets for **m1** and **m2** are 0.3 meters and 0.4 meters respectively. The reference position is determined by the reference element, which is specified by the **ref** attribute, and the **ref_origin** attribute which specifies where on the reference element the reference position is. In this example, the reference point for **m1** is the **beginning** (upstream) end of element **q** and the reference point for **m2** is the (downstream) **end** of element **q**. The element origin is similarly defined using the **ele_origin** attribute. In this case, since marker elements have zero length, the setting of **ele_origin** is immaterial.

Start *Tao* as explained in section §6.2 with the lattice file **superimpose1.bmad**. The lattice looks like:

```
Tao> show lat
      Values at End of Element:
 Index  name      key                      s       l    beta     phi     eta ...
                                                           a       a       a ...
     0  BEGINNING Beginning_Ele        0.000     ---   10.00   0.000   0.00 ...
     1  Q#1       Quadrupole           0.300   0.300    9.83   0.030   0.00 ...
     2  M1        Marker               0.300   0.000    9.83   0.030   0.00 ...
     3  Q#2       Quadrupole           1.000   0.700    8.22   0.107   0.00 ...
     4  D#1       Drift                1.400   0.400    6.97   0.160   0.00 ...
     5  M2        Marker               1.400   0.000    6.97   0.160   0.00 ...
     6  D#2       Drift                2.000   0.600    5.37   0.258   0.00 ...
     7  END       Marker               2.000   0.000    5.37   0.258   0.00 ...
 Lord Elements:
     8  Q         Quadrupole           1.000   1.000    8.22   0.107   0.00 ...
 Index  name      key                      s       l    beta     phi     eta ...
                                                           a       a       a ...
      Values at End of Element:
```

The quadrupole **Q** has been split by marker **M1** and so has become a **super_lord** element:

```
Tao> show ele Q

 Element #                8
 Element Name: Q
 Key: Quadrupole
... etc...

Slave_status: Free
Lord_status:  Super_Lord
Slaves:
   Index   Name        Type
       1   Q#1         Quadrupole
       3   Q#2         Quadrupole
```

The two **super_slaves** of **Q**, elements **Q#1** and **Q#2**, will be used when tracking a particle through the lattice.

If parameters of element **Q** are modified, Bmad bookkeeping routines will automatically update the super_slaves. Thus if the **k1** attribute of **Q** is modified:

```
Tao> change ele Q k1 0.11
          Old           New     Old-Design     New-Design          Delta
     0.200000      0.310000       0.110000       0.110000       0.110000    Q
```

The change will be reflected in the slave elements:

```
Tao> show ele 1
 Element #                1
 Element Name: Q#1
 Key: Quadrupole

 Attribute values [Only non-zero/non-default values shown]:
    1   L                             =  3.0000000E-01 m
    4   K1                            =  3.1000000E-01 1/m^2
... etc...

Slave_status: Super_Slave
Associated Super_Lord(s):
   Index   Name                              Type
       8   Q                                 Quadrupole
Lord_status:  Not_a_Lord
```

Parameter values of the super_slave elements are determined by the super_lord and may not be directly set:

```
Tao> change ele Q#1 k1 0.01

[ERROR | 2017-AUG-28 22:38:36] tao_change_ele:
    ATTRIBUTE NOT FREE TO VARY. NOTHING DONE
```

Notes:

- The default value for **ref_origin** and **ele_origin** if not present is **center** — the center of the element.
- The default reference element if **ref** is not present is the zero length **beginning** element at the beginning of the lattice.
- With closed lattices (§16), a superimposed element may "wrap" around so that part of the superimposed element is at the end of the lattice and part of the element is at the beginning of the lattice. See the example in Section §16.
- No super_lord element is made when a drift element is split. Thus in the above example, there is no **D** super_lord and the two elements **D#1** and **D#1** are *not* **super_slaves**. Drifts are the only type of element where, if split, a super_lord element is not created.

14.2 Example 2

The second superposition example involves superposition of an element with finite length:

```
! Lattice File: superimpose2.bmad
beginning[beta_a] = 10.    ! m  a-mode beta function
beginning[beta_b] = 10.    ! m  b-mode beta function
beginning[e_tot] = 10e6    ! eV   Or can set p0c
parameter[geometry] = open      ! or closed

Q: quad, l = 4
D: drift, l = 12
S: solenoid, l = 8, superimpose, ref = Q, ele_origin = beginning
M: marker, superimpose, ref = S, offset = 1

lat: line = (Q, D)
use, lat
```

The superimposes a solenoid on top of a quadrupole and a drift. Start *Tao* as explained in section §6.2 with the lattice file **superimpose2.bmad**.

```
Tao> show lat
      Values at End of Element:
 Index  name      key                    s       l    beta     phi     eta ...
                                                          a       a       a ...
     0  BEGINNING Beginning_Ele      0.000     ---   10.00   0.000   0.00 ...
     1  Q#1       Quadrupole         2.000   2.000   10.40   0.197   0.00 ...
     2  Q\S       Sol_Quad           4.000   2.000   11.60   0.381   0.00 ...
     3  S#1       Solenoid           7.000   3.000   14.90   0.611   0.00 ...
     4  M         Marker             7.000   0.000   14.90   0.611   0.00 ...
     5  S#2       Solenoid          10.000   3.000   20.00   0.785   0.00 ...
     6  D#1       Drift             16.000   6.000   35.60   1.012   0.00 ...
     7  END       Marker            16.000   0.000   35.60   1.012   0.00 ...
Lord Elements:
     8  Q         Quadrupole         4.000   4.000   11.60   0.381   0.00 ...
     9  S         Solenoid          10.000   8.000   20.00   0.785   0.00 ...
 Index  name      key                    s       l    beta     phi     eta ...
                                                          a       a       a ...
      Values at End of Element:
```

The **Q\S** super_slave element has both quadrupole **Q** and solenoid **S** elements as super_lords. This makes **Q\S** a **sol_quad** or combination solenoid and quadruple element.

```
Tao> show ele q\s
 Element #                2
 Element Name: Q\S
 Key: Sol_Quad
... etc...

Slave_status: Super_Slave
Associated Super_Lord(s):
   Index   Name                                Type
       8   Q                                   Quadrupole
       9   S                                   Solenoid
Lord_status:  Not_a_Lord
```

Notes:

- This superposition works since Bmad has a **sol_quad** element which is a combination solenoid/quadrupole. On the other hand, it is not possible to superimpose a quadrupole with a sextupole since Bmad does not have a combination quadrupole/sextupole element.
- **Jumbo** superposition can be used to superimpose elements that whose combination cannot be represented by a corresponding *Bmad* element. The drawback in this case is that the particle tracking through this element must be done via a Runge-Kutta or similar tracking method. (§18). See the *Bmad* manual for more details.

14.3 Exercises

14.1 Create a lattice file with

14.2 Create a lattice file with an element that uses **jumbo** superposition.

15 Multipass

Some lattices have the beam recirculating through the same element multiple times. For example, an Energy Recovery Linac (ERL) will circulate the beam back through the LINAC part to retrieve the energy in the beam. In *Bmad*, this situation can be simulated using the concept of **multipass**. Another situation where multipass is useful is for modeling the interaction region in a colliding beam machine. In the Bmad manual multipass is discussed in the "Superposition and Multipass" chapter.

15.1 What is Multipass and What is it Good For?

Consider the following lattice:

```
A: quadrupole
ll: line = (A, A)
use, ll
```

The lattice has two quadrupoles both called **A**. These two elements, even though they have the same name, are independent:

```
Tao> change ele 1    k1 0.01   ! Can modify first A element.
Tao> change ele A##2 k1 0.02   ! And can modify second A ele independently.
```

Now consider an ERL. With an ERL, the beam will go through the linac section multiple times. An ERL lattice might look like:

```
linac: line = (...)
arc: line = (...)
dump: line = (..)
erl_line: line = (injector, linac, arc, linac, dump)
```

Here you don't want the elements of the fist **linac** in **erl_line** to be treated as separate from the second **linac** in **erl_line** since they represent the same set of physical elements. This is where **multipass** comes in. Multipass is used to describe the situation where multiple elements to be tracked through are actually the same physical element and you want that fact to be enforced when element parameters are varied.

In this case, the solution is to mark the **linac** line as multipass to tell Bmad that the first instance of **linac** in **erl_line** contains the same physical elements as the second instance of **linac**:

```
linac: line[multipass] = (...)
```

With a **multipass** line, *Bmad* will setup appropriate multipass lords and multipass slaves to connect together all elements which represent the same physical element.

15.2 Example

```
! Lattice File: multipass.bmad

beginning[beta_a] = 100.   ! m  a-mode beta function
beginning[beta_b] = 100.   ! m  b-mode beta function
beginning[p0c] = 10e6   ! eV
parameter[geometry] = open      ! or closed

cavity: lcavity, l = 1, voltage = 10e6

linac: line[multipass] = (cavity)
erl: line = (linac, linac)
use, erl

expand_lattice
cavity\2[phi0_multipass] = 0.5
```

Start *Tao* as explained in section §6.2 with the lattice file **multipass.bmad**. The lattice looks like:

```
Tao> show lat
      Values at End of Element:
 Index  name       key                   s       l    beta     phi    eta ...
                                                          a       a      a ...
     0  BEGINNING  Beginning_Ele     0.000     ---  100.00   0.000   0.00 ...
     1  CAVITY\1   Lcavity           1.000   1.000   78.87   0.011  -0.00 ...
     2  CAVITY\2   Lcavity           2.000   1.000   42.09   0.028  -0.00 ...
     3  END        Marker            2.000   0.000   42.09   0.028  -0.00 ...
Lord Elements:
     4  CAVITY     Lcavity           0.000   1.000    0.00   0.000   0.00 ...
 Index  name       key                   s       l    beta     phi    eta ...
                                                          a       a      a ...
  Values at End of Element:
```

Bmad creates a **multipass_lord** called **cavity** to control the **multipass_slaves** called **cavity\1** and **cavity\2**:

```
Tao> show ele 4
 Element #               4
 Element Name: CAVITY
... etc...
Slave_status: Free
Lord_status:  Multipass_Lord
Slaves:
   Index   Name        Type
       1   CAVITY\1    Lcavity
       2   CAVITY\2    Lcavity
```

Since the **cavity** element represents the physical element, any change in the parameters of **cavity** will be be reflected in the slaves (just like superposition lords and slaves). As an example, changing the attribute of the lord:

```
Tao> set ele cavity x_offset = 0.001
```

changes the corresponding attributes of the slaves:

```
Tao> show ele 2
 Element #                2
 Element Name: CAVITY\2
 Key: Lcavity
 S_start, S:    1.000000,    2.000000
 Ref_time:  6.675633E-09

 Attribute values [Only non-zero/non-default values shown]:
    1   L          =  1.000E+00 m
... etc...
   36   X_OFFSET   =  1.000E-03 m        57   X_OFFSET_TOT  =  1.000E-03 m
... etc...
```

The exception to the rule that the multipass_lord completely controls the multipass_slave attributes is the **phi0_multipass** attribute of **lcavity** and **rfcavity** elements. **phi0_multipass** allows for different settings of the RF phase for different passes through the cavity element. From the above lattice:

```
expand_lattice                 ! cavity\2 is created during lattice expansion
cavity\2[phi0_multipass] = 0.5 ! Shifts the RF phase for cavity\2 by 180^deg
```

The **expand_lattice** command "expands" the lattice to create **cavity\2** (see the *Bmad* manual for more details) and the next line shifts the phase of **cavity\2** by 180 degrees.

This 180 degrees phase shift makes **cavity\2** decelerating instead of accelerating. Thus the reference energy after cavity **cavity\2** will be the same as the reference energy at the start of the lattice:

```
Tao> show lat -attrib e_tot
      Values at End of Element:
 Index  name      key                      s       l           e
                                                              tot
     0  BEGINNING Beginning_Ele        0.000     ---  1.0013E+07
     1  CAVITY\1  Lcavity              1.000   1.000  2.0013E+07
     2  CAVITY\2  Lcavity              2.000   1.000  1.0013E+07
     3  END       Marker               2.000   0.000  1.0013E+07
Lord Elements:
     4  CAVITY    Lcavity              0.000   1.000  2.0013E+07
 Index  name      key                      s       l           e
                                                              tot
```

Notes:

- Bmad does not demand that the global position of the multipass_slaves of a multipass_lord be in the same position in the global coordinate system.
- Since the reference energy is changing, the transfer matrix through a **lcavity** will not be symplectic.

15.3 Exercises

15.1 Modify **multipass.bmad** so that there are two element in the **linac** line called **cavity** and verify that *Bmad* does the proper bookkeeping (that is, there are two **cavity** multipass lords.

15.2 **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 (which is the normalized momentum deviation from the reference §13.1) does change.

16 Lattice Geometry

The **parameter[geometry]** parameter in a lattice file sets the lattice topology to be **open** or **closed**:

open
: For **open** lattices, *Bmad* computes the reference orbit and Twiss parameters by taking the **beginning[...]** Twiss and **particle_start[...]** orbit settings as the initial values and propagates them to the end of the lattice (like you would do for a linac).

closed
: For **closed** lattices, *Bmad* calculates the Twiss and orbit periodic solution (like you would in a storage ring). In this case, *Bmad* will ignore Twiss and orbit settings in the lattice file.

Example:

```
! Lattice File: geometry.bmad
parameter[p0c] = 1e9
parameter[geometry] = closed

d: drift, l = 2
q1: quad, l = 0.5, k1 = 3, hkick = 0.001, superimpose
q2: quad, l = 0.5, k1 = -3, vkick = 0.002, superimpose, offset = 1

lat: line = (d)
use, lat
```

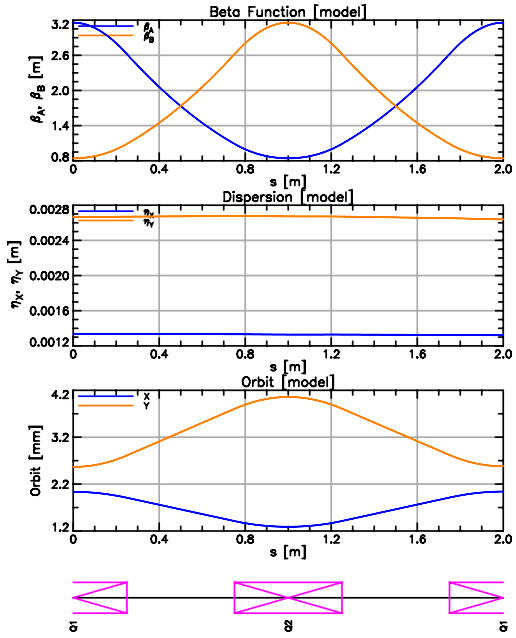


Figure 16: *Bmad* computes the periodic Twiss and orbits for **closed** lattices.

The lattice looks like:

```
Tao> show lat
      Values at End of Element:
 Index   name      key                    s       l    beta     phi     eta ...
                                                          a       a       a ...
     0   BEGINNING Beginning_Ele       0.000     ---    5.93   0.000   0.00 ...
     1   Q1#2      Quadrupole          0.250   0.250    5.57   0.043   0.00 ...
     2   D#1       Drift               0.750   0.500    4.32   0.145   0.00 ...
     3   Q2        Quadrupole          1.250   0.500    4.32   0.266   0.00 ...
     4   D#2       Drift               1.750   0.500    5.57   0.368   0.00 ...
     5   Q1#1      Quadrupole          2.000   0.250    5.93   0.411   0.00 ...
     6   END       Marker              2.000   0.000    5.93   0.411   0.00 ...
Lord Elements:
     7   Q1        Quadrupole          0.250   0.500    5.57   0.043   0.00 ...
 Index   name      key                    s       l    beta     phi     eta ...
                                                          a       a       a ...
```

The result is shown in Figure 16. The **q1** quadrupole has been superimposed placing its center at the origin **s** = 0. This results in **q1** being "wrapped around" so that first half of **q1**, **q1#1**, comes at the *end* of the tracking part of the lattice and the second half, **q1#2**, comes at the beginning of the lattice:

```
Tao> show ele q1
 Element #                7
 Element Name: Q1
 Key: Quadrupole
 S_start, S:    1.750000,    0.250000
... etc...

Slave_status: Free
Lord_status:  Super_Lord
Slaves:
   Index    Name         Type
       5    Q1#1         Quadrupole
       1    Q1#2         Quadrupole
```

Notes:

- Bmad does not demand that a closed lattice be closed in the sense that the global position at the end of the lattice be the same as the beginning. This makes sense since sometimes you want to take a lattice section and get the periodic solutions even though the section is not physically closed.

16.1 Exercises

16.1 It is sometimes convenient to switch the lattice geometry from closed to open while maintaining the same beginning Twiss and orbit values. Create an open geometry lattice from the **geometry.bmad** lattice. While it is possible to code the beginning Twiss and orbit values by hand, an easier way is to have *Tao* write a bmad lattice file using the **write bmad** command. This new file will contain the proper settings for the beginning Twiss and orbit.

17 Forks and Branches

A **fork** or **photon_fork** element marks the point where multiple lines can merge or branch off from. Forking elements can be used to describe such things as X-ray lines branching from storage rings (see Figure 17a), injection or extraction lines, etc.

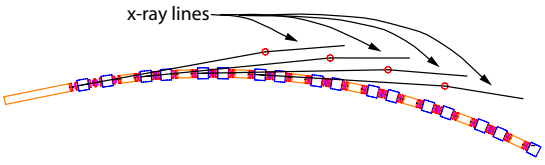
17.1 Example

```
! Lattice File: fork.bmad

beginning[beta_a] = 10.0        ! m  a-mode beta function
beginning[beta_b] = 10.0        ! m  b-mode beta function
beginning[e_tot] = 10e6         ! eV
parameter[geometry] = open      ! or closed

b: sbend, l = 2, angle = pi/3
f: fork, to_line = extract_line, superimpose, offset = 0.4
q quadrupole, l = 2

extract_line: line = (q)       ! The line forked to.
extract_line[geometry] = open

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

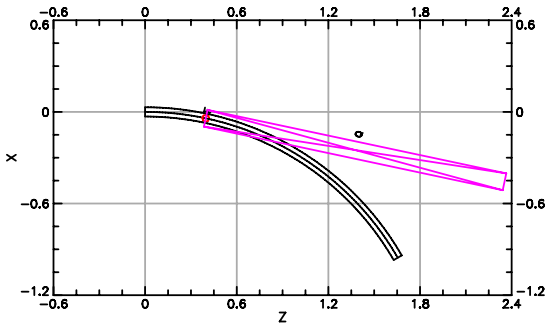
In this example The **lat** line is used as the basis for the lattice due to the "**use, lat**" statement. This line contains the bend **b** and, via superposition, the fork element **f**. The fork element **f** connects to the **to_line** called **extract_line** which contains a single quadrupole element called **q**.

To see the geometry of the lattice, start *Tao* as explained in section §6.2 with the lattice file **fork.bmad** and create a **floor_plan** plot:

```
Tao> place r11 floor
```



(a) Fork elements can be used to construct interconnected lines like x-ray lines branching from a storage ring.



(b) Simple fork example.

Figure 17

The result is shown in Figure 17b.The **fork** element is the red circle. The lattice is looks like:

```
Tao> show lat
      Values at End of Element:
 Index   name       key                    s        l     beta      phi     eta ...
                                                            a        a       a ...
     0   BEGINNING  Beginning_Ele        0.000      ---    10.00    0.000   0.00 ...
     1   B#1        Sbend                0.400    0.400     9.77    0.040   0.03 ...
     2   F          Fork                 0.400    0.000     9.77    0.040   0.03 ...
     3   B#2        Sbend                2.000    1.600     5.32    0.249   0.75 ...
     4   END        Marker               2.000    0.000     5.32    0.249   0.75 ...
Lord Elements:
     5   B          Sbend                2.000    2.000     5.32    0.249   0.75 ...
 Index   name       key                      s        l     beta      phi     eta ...
                                                            a        a       a ...
      Values at End of Element:
```

The **show lat** output does not show **q**. Where is the line that was forked to? The answer is that Bmad creates a set of **branches** to hold the different lines. Branches are assigned an index starting from 0 and information on them can be seen with the **show branch** command:

```
Tao> show branch
                    N_ele  N_ele   ...              Live
  Branch            Track    Max   ...    Geometry  Branch  From_Fork
  0: LAT                4      5   ...    Open       T
  1: EXTRACT_LINE       2      2   ...    Open       T      0>>2


                                                                  Defines
  Fork_Element      Forking_To                          Direction   To_Branch?
  0>>2: LAT>>F      1>>0: EXTRACT_LINE>>BEGINNING         1           T
```

This shows that the lattice has two branches. When there are multiple branches, elements are indexed using the notation:

```
branch_index>>element_index
```

so that, for example, "**0**>>**2**" represents element number 2 in branch 0. That is, the fork element **f**.

Each branch has its own set of parameters like the geometry, reference energy, etc. These may be set using the syntax

```
  branch_name[parameter] = ...
```

For example the "**extract_line[geometry]**" was used to set the geometry of **extract_line** in **fork.bmad**.

The **show lat** branch, by default, shows branch 0. To see other branches use the -**branch** option:

```
Tao> show lat -branch 1
      Values at End of Element:
 Index   name       key                    s        l     beta      phi     eta ...
                                                            a        a       a ...
```

```
    0  BEGINNING Beginning_Ele       0.000      ---     9.77    0.040    0.03 ...
    1  D         Drift               2.000    2.000     8.04    0.268    0.34 ...
    2  END       Marker              2.000    0.000     8.04    0.268    0.34 ...
Index  name      key                     s        l     beta     phi     eta ...
                                                          a       a       a ...
     Values at End of Element:
```

Forked lines can, in turn, have forks to other lines. And lines can connect back to existing lines. In this way an entire accelerator complex can be simulated.

Notes:

- The difference between **fork** and **photon_fork** is that the default species for **fork** is the same as the line forked from while for a **photon_fork** the default species are photons.
- A fork element is not restricted to forking to the beginning of a line. The place where a fork element connects can be set by the **to_element** attribute.

17.2 Exercises

17.1 Using the **fork.bmad** lattice, vary the beginning Twiss and orbit (using **set** and/or **change** commands) and verify that the Twiss and orbit in branch 1 varies appropriately.

17.2 For a more complicated example, play around with the lattice:

```
examples/tutorial_bmad_tao/lattice_files/
                                wave_analysis/chess-u_6000mev_20181120.lat
```

This is a lattice that is used for simulation of the Cornell CESR storage ring. The lattice includes X-ray lines so that the effect on the X-ray beams due to things such as magnet misalignments can be simulated.

18 Tracking Methods

For each lattice element one can vary the method used to track particles through the element. This is useful, among other things for optimizing speed and/or accuracy. There are several element parameters that control tracking. These are:

```
tracking_method      ! How a particle is tracked through the element.
mat6_calc_method     ! How the element's transfer matrix is calculated.
spin_tracking_method ! How a particle's spin is tracked through an element.
field_calc           ! How the electric and/or magnetic field is calculated.
```

Example:

```
q1: quadrupole, l = 0.6, ..., tracking_method = runge_kutta
```

For the **tracking_method** parameter some possible values are:

```
bmad_standard  ! Fast, thick element formulas.
symp_lie_ptc   ! Symplectic Lee integration tracking.
taylor         ! Taylor map.
linear         ! Linear tracking.
custom         ! Tracking with custom code.
runge_kutta    ! Track through fields.
etc...
```

Much more information in the Bmad manual in the Chapter on “Tracking, Spin, and Transfer Matrix Calculation Methods”.

19 Optimization with Tao

"Optimization" is the process of varying (model) lattice parameters to create a lattice with a certain set of properties as close to "ideal" as possible. For example, orbit flattening involves optimizing steering setting in the lattice so that the orbit, as calculated from the **model** lattice, matches the measured orbit. The steering strengths in the optimized lattice can then be used to vary the actual steerings and therefore to correct the actual orbit in the machine.

Another example of optimization is lattice design where, say, sextupole magnet strengths in the lattice are adjusted to give maximum dynamic aperture.

Optimization involves "**data**" and "**variables**". **Data** is the parameters to be optimized. For example, orbit positions when flattening an orbit or the value of beta at the interaction point when designing a lattice. **Variables** are what is to be varied which can be steering strengths, quadrupole strengths, etc.

Optimization involves minimizing one or more "objectives" or "merit functions". In the case of orbit flattening, typically there is a single merit function that is a function of the differences between the measured and calculated orbits. In other situations, multiple objectives may be desired. *Tao* itself implements "single objective" optimization. For "multiple objective" optimization, there is a separate program called **moga** that you can use (§5).

The general form of the merit function **M** is

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

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 data and variable differences which will be discussed in detail below.

The sum over the variables in Eq. (1) is used to keep the values of the variables "reasonable" in case there are degeneracies or near degeneracies in the effects of the variables. For example, when flattening an orbit, if there are two steerings close to one another, then it may be the case that the calculated orbit is reasonable even when one variable is large and positive while the other variable is large and negative. The variable sum in Eq. (1) can, in this case, drive the steerings towards zero to avoid large "unphysical" steering strengths. Often the best way to determine what the relative values for the weights should be comes from varying the weights to see what works best.

There are several different optimizers that can be used with *Tao*. The one optimizer that is good for finding global merit function minima is the **de** (differential evolution) optimizer. All of the others are good for finding local minima.

Note: Optimization is covered in detail in the "Lattice Correction and Design" chapter in the *Tao* manual.

19.1 Example Optimization Files

The example files used to illustrate optimization are in the directory

```
examples/tutorial_bmad_tao/lattice_files/lattice_optimization
```

Copy these files to your working directory. Here the main initialization file is named **tao.init**. Since this is the default name for initialization files (§6.4), and since the **tao.init** file contains the name of the lattice file, *Tao* can be started without the **-lat** option (§6.2).

There are five files here:

```
lat.bmad          ! Lattice file.
setup.tao         ! Command file run at startup.
tao.init          ! Primary Tao initialization file.
tao_plot.init     ! Secondary initialization file.
optimized_var.out ! Output optimized values
```

Consider the file **tao.init** first. The file is divided into three parts. The first part sets some general parameters while the next two sections setup data §19.2 and variable §19.3 lists.

```
&tao_start
  startup_file = "setup.tao"
/

&tao_design_lattice
  n_universes = 1
  design_lattice(1)%file = "lat.bmad"
/

&tao_plot_page
  plot_page%size = 500, 400
  place(1) = "layout", "lat_layout"
  place(2) = "r12", "beta"
  place(3) = "r22", "key"
/
```

Namelist format is used as explained in Section §6.4. A single **universe** will be used (§10.3) and the lattice file name for the universe is "**lat.bmad**". The command file "**setup.tao**" will be run after all other initialization is complete.

The **tao_plot_page** namelist sets some plotting parameters. The setting of **plot_page%size** overrides the default size of the plot window and the **place(1)**, **place(2)**, and **place(3)** settings define initial placement of plots (§9.2).

The startup command file **setup.tao** defines an **alias** commands:

```
alias opt run lm
```

This defines the command "**opt**" to be equivalent to "**run lm**". The **run** command starts optimization and the "**lm**" option specifies the Levenburg-Marquardt optimizer which is a good optimizer for finding a local minimum. See the **Tao Commands** chapter in the *Tao* manual for more details.

19.2 Data in Tao

In order to optimize, you must tell Tao what **data** will contribute to the merit function. A detailed description on how to do this is given in the "**Data**" chapter of the *Tao* manual.

The data that is used in the present example is defined in the middle section of the **tao.init** file:

```
&tao_d2_data
  d2_data%name = "twiss"
  n_d1_data = 2
/

&tao_d1_data
  ix_d1_data = 1
  d1_data%name = "a"
  datum(1) =  "beta.a"     "" "" "END"   "target"  12.0   1e1
  datum(2) =  "alpha.a"    "" "" "END"   "target" -0.4    1e2
/

&tao_d1_data
  ix_d1_data = 2
  d1_data%name = "b"
  datum(1) =  "beta.b"     "" "" "END"   "target"  12.0   1e1
  datum(2) =  "alpha.b"    "" "" "END"   "target" -0.4    1e2
/
```

In general, the **data** is grouped into a three level tree as illustrated in Figure 18. Nodes at the the highest level are instances of what is called **d2_data** structures. For example, a **d2_data** structure may be setup to hold **orbit** data. In the present case, there is a single **d2_data** structure named "**twiss**".

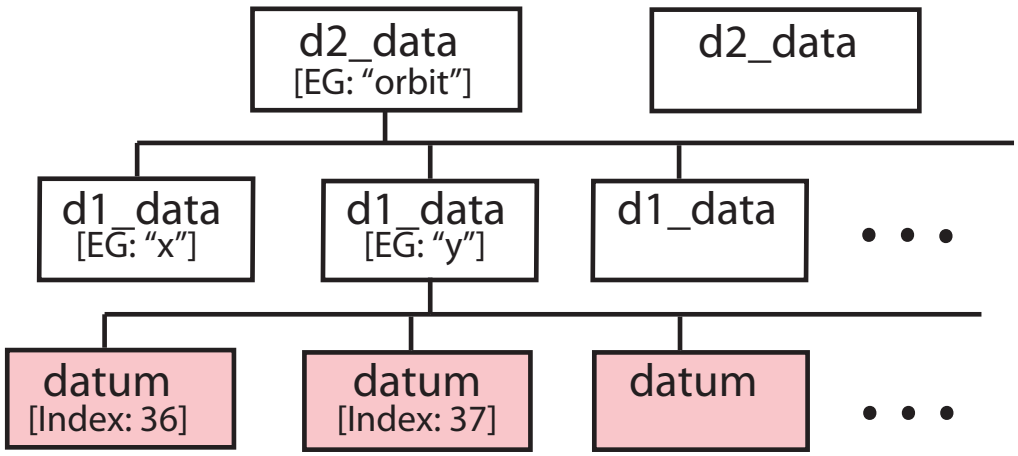


Figure 18: Data is grouped into a three level tree. A **d2_data** structure holds a set of **d1_data** structures. A **d1_data** structure holds an array of **datums**.

A **d2_data** structure will hold an array of one or more **d1_data** structures. For example, a **orbit d2_data** structure may hold "**x**" and "**y**" **d1_data** structures which represent horizontal and vertical orbit data. In the present case, the **twiss d2_data** structure has two **d1_data_structures** named "**a**" and "**b**" representing the two transverse normal modes of oscillation. The syntax to refer to a particular **d1_data** structure is:

```
d2-data-name.d1-data-name
```

So with the above example, the **a** structure would be referred to as **twiss.a** .

A **d1_data** structure will hold an array of one or more **datum** structures. For example, the **x d1_data** structure contained in an **orbit d2_data** structure may be setup with an array of datums, one for each beam position monitor in the machine, with each datum representing a horizontal orbit measurement at a specified BPM. In this case, both **a** and **b d1_data** structures hold two datums, one representing β and α Twiss values. To refer to an individual **datum** use the syntax:

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

where **datum-index** is the index for the datum. So with the above example, the first datum in **twiss.a**, which is the **a**-mode β, would be referred to as **twiss.a[1]** .

An individual **datum** is structure that has a number of components. With the present **tao.init** file, seven components of each datum are set. These components are, in order:

```
data_type       ! Type of data: "orbit.x", etc.
ele_ref_name    ! Name of reference lattice element
ele_start_name  ! Name of starting lattice element when there is a range
ele_name        ! Name of the lattice element where datum is evaluated.
merit_type      ! Type of constraint: "target", "max", "min", etc.
meas            ! Measured datum value.
weight          ! Weight for the merit function term
```

Thus for the **twiss.b[2]** datum which is set on the line:

```
datum(2) =  "alpha.b"    "" "" "END"   "target" -0.4    1e2
```

The **data_type** component is set to "**alpha.b**" [For a list of data types that *Tao* recognizes, see the **Tao Data Types** section (§5.8) of the **Data** chapter of the *Tao* manual.], the **ele_ref_name** and **ele_start_name** are set to the blank string. These parameters are not used in the present example.

The **ele_name** component for **twiss.b[2]** is set to "**END**" which is where the datum is to be evaluated. The **target** type merit means that δD in Eq. (1) is evaluated using the equation

$$\delta D = \text{model} - \text{meas}$$

where **model** is the value as calculated from the **model** lattice and **meas** is the "**measured**" value as set on the datum line. For **twiss.b[2]**, **meas** is set to -0.4. Also the weight w for **twiss.b[2]** is set to 100. Thus if the model b-mode alpha function is, say, 1.0 at element **END**, then this datum would contribute $100 * (1.0 - 0.4)^2$ to the merit function.

Start *Tao* (remember, no -**lat** argument needed). The **d2_data** structures can be shown with the command **show data**:

```
Tao> show data

  Name                                  Using for Optimization
  twiss.a[1:2]                                   Using: 1:2
  twiss.b[1:2]                                   Using: 1:2
```

To see a list of datums for an individual **d1_data** structure append the **d1_data** name after **show data**. For example:

```
Tao> show data twiss.b
Data name: twiss.b
                     ...                                                |   Useit
                     ...  Ele          Meas        Model       Design | Opt  Plot
1  beta.b <target>   ...  END     1.200E+01   9.292E+00   9.292E+00     T     F
2  alpha.b <target>  ...  END    -4.000E-01  -4.427E-01  -4.427E-01     T     F
                     ...  Ele          Meas        Model       Design | Opt  Plot
                     ...                                                |   Useit
```

To see the parameters of an individual datum append the datum name after **show data**. For example:

```
Tao> show data twiss.a[2]

%ele_name           = END
... etc...
%data_type          = alpha.a
... etc...
%model              =  -4.42701763E-01
%design             =  -4.42701763E-01
... etc...
%good_model         = T
%good_design        = T
%good_base          = T
%good_meas          = T
%good_ref           = F
%good_user          = T
%good_opt           = T
%good_plot          = F
%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. The user also as control as to whether a datum will be used in optimization and this is controlled by setting the **good_user** component. The commands that control this are **use**, **veto** and **restore**. Example:

```
Tao> veto data twiss.a
  twiss.a[1:2]                                   Using:
  twiss.b[1:2]                                   Using: 1:2
```

In this example the two **twiss.a** datums have been vetoed. It can be checked that the **twiss.a** datums now have their **good_user** components set to False.

Notice that it does not matter to the optimization process how data is divided into **d1_data** and **d2_data** groups. It is only a matter of convenience to the user. Also a given **d1_data** group of data does not have to contain data of a single type. Thus the **twiss.a** datums include both **beta** and **alpha** type data.

19.3 Variables in Tao

In order to optimize, you must tell Tao what **variables** you want to vary to minimize the merit function. A detailed description on how to construct **variables** is given in the "Variables" chapter of the *Tao* manual.

The variables that are used in the present example are defined in the bottom section of the **tao.init** file:

```
&tao_var
  v1_var%name = "quad"
  search_for_lat_eles = "Quad::*"
  default_step = 1e-4
  default_attribute = "k1"
  default_merit_type = "limit"
  default_low_lim = -50
  default_high_lim = 50
  default_weight = 1
  ix_min_var = 1
  default_key_delta = 1e-2
  default_key_bound = T
/
```

Just like **data**, **variables** are grouped into a tree but in this case there are only two levels. The top level nodes of the tree are called **v1_var** structures. In this example a **v1_var** structure is defined called **quad** which controls the **k1** attribute of all element whose name matches **quad::***. This will match to all quadrupoles. In this case, the lattice has 6 quadrupoles named **Q1** through **Q6**. Thus there will be an array of 6 **variables** associated with the **quad v1_var** structure.

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]**.

The parameters like **default_step** in the above namelist establish a default value for the **step** attribute of each variable that is created for the **quad** structure. The **step** attribute is used by *Tao* to calculate derivatives that are used by some of the optimizers. Essentially, to calculate derivatives, *Tao* varies the variable by $\pm$**step** and looks at the changes in the **data**. Like many attributes associated with **optimization** it is important that the **step** attribute be set properly. To

small a setting and round-off error can throw off the derivative calculation. On the other hand, if the value of **step** is too large, nonlinearities can throw off the calculation.

The **weight** of a variable sets the value of w_j in Eq. (1). Since the **merit_type** in this case is **limit**, the δV used in the merit function is:

$$\delta V = \begin{cases} \text{model} - \text{high\_lim} & \text{model} > \text{high\_lim} \\ \text{model} - \text{low\_lim} & \text{model} < \text{low\_lim} \\ 0 & \text{Otherwise} \end{cases} \tag{2}$$

That is, the contribution to the merit function will be zero if the value of the variable is between **low_lim** and **high_lim** which in this case is -50 and 50.

Running *Tao*, the **v1_var** structures can be shown with the command **show variable**:

```
Tao> show var
       Name                                        Using for Optimization
    quad[1:6]                                      1:6
```

To see a list of individual variables of a given **v1_var** structure, append the **v1_var** name to the **show variable** command:

```
Tao> sho var quad
Variable name:  quad

 Index  Controlled Attribs(s)    Meas         Model        Design  Useit_opt
     1  Q1[K1]                8.6924-311    0.0000E+00    0.0000E+00       F
     2  Q2[K1]                8.6924-311    0.0000E+00    0.0000E+00       F
     3  Q3[K1]                8.6924-311    0.0000E+00    0.0000E+00       F
     4  Q4[K1]                8.6924-311    0.0000E+00    0.0000E+00       F
     5  Q5[K1]                8.6924-311    0.0000E+00    0.0000E+00       F
     6  Q6[K1]                8.6924-311    0.0000E+00    0.0000E+00       F
 Index  Controlled Attribs(s)    Meas         Model        Design  Useit_opt
```

To see the parameters of an individual variable, append the variable name to the **show var** command. Example:

```
Tao> sho var quad[2]
%ele_name         = Q2
%attrib_name      = K1
... etc...
%exists           = T
%good_var         = T
%good_user        = T
%good_opt         = T
%useit_opt        = T
... etc...
```

The **useit_opt** logical indicates whether the variable will be used in the optimization process. The user has some control as to whether a variable will be used in optimization and this is controlled by setting the **good_user** component. The commands that control this are **use**, **veto** and **restore**. Example:

```
Tao> veto var quad[4]
  quad[1:6]                                          Using: 1:3 5:6
```

Variable properties can also be changed within Tao. For example,

```
Tao> set var quad[1:4]|low_lim = -1
```

will set the lower limit for the first four quads.

Notice that, like data, it does not matter to the optimization process how variables are divided groups. It is only a matter of convenience to the user. Also a given **v1_var** instance, the array of associated **variables** does not all have to be of a single type.

If you want to have one *Tao* **variable** control a set of parameters, construct an **overlay** or **group** element (§11) and then have the *Tao* variable control the overlay or group. For example, the following overlay gangs the **k1** parameters of elements **Q1** and **Q3** together:

```
ps1: overlay = {Q1, Q3}, var = {k1}, k1 = 0.8
```

19.4 Key Bindings

Tao has two modes for entering commands. In **single mode**, each keystroke represents a command. That is, with a few exceptions, the user does not have to press the carriage control key to signal the end of a command. This is to be contrasted with **line mode**, which you have been using up to now, where *Tao* waits until the return key is depressed to execute a command. **single mode** is useful for quickly varying parameters to see how they affect a lattice but the number of commands in single mode is limited. **Single mode** is covered in detail in the "Single Mode" chapter in the *Tao* manual.

The main purpose of **single mode** is to associate certain keyboard keys with certain variables so that the pressing of these keys will change their associated model value of the variable. This is called a **key binding** and is illustrated in Figure 19.

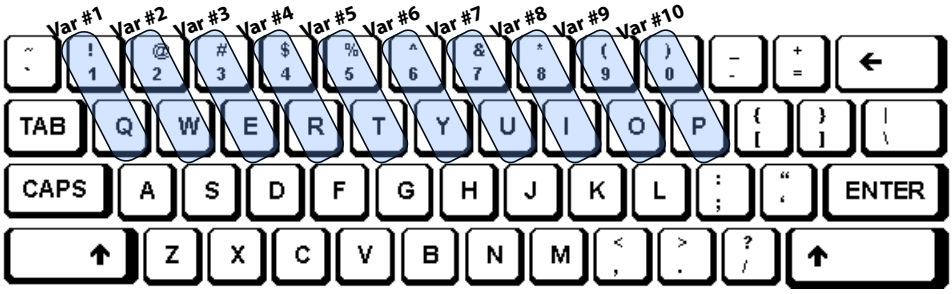
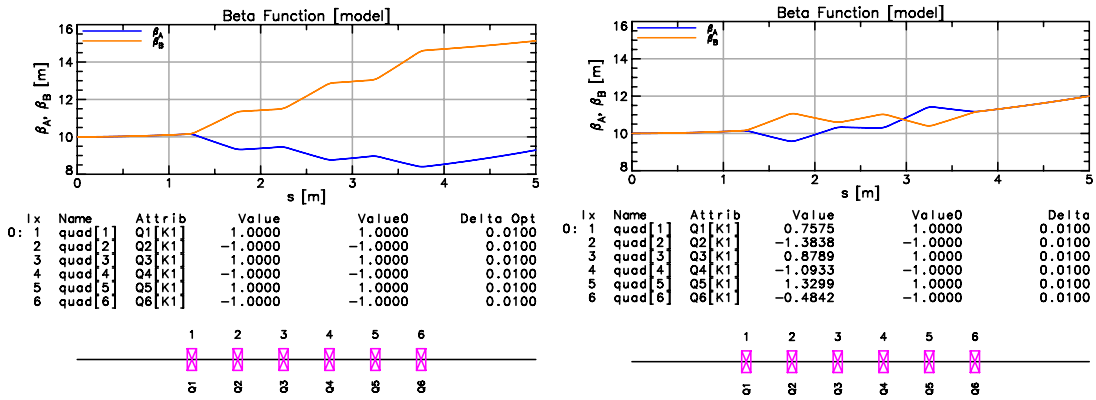


Figure 19: Ten pairs of keys on the keyboard are bound to ten variables so that pressing a key of a given pair will either increment or decrement the associated variable. The first key pair bound to variable number 1 are the **1** and **Q** keys, etc.



(a) Initial plot window showing the initial beta functions.

(b) Plot window after optimization showing the optimized beta functions.

Figure 20

Start *Tao* using the example optimization files (§19.1) or use the **reinit tao** command to reinitialize *Tao*. The plot window should look like Figure 20a. The **key_table** plot in the middle shows what variables (§19.3) have been bound to what keyboard keys. In this instance the **quad[1]** variable is bound so that pressing the "1" key will change **quad[1]** by +**delta**, and pressing the "q" key will change **quad[1]** by -**delta**. Pressing the shift key when pressing the "1" or "q" keys will change **quad[1]** by +**10**×**delta** and -**10**×**delta** respectively. Similarly, the **quad[2]** variable is bound to the **2** and **w** keys, etc.

Single mode and key bindings are useful for getting a feel for how variables affect the lattice. Get into single mode by issuing the **single_mode** command. Play around with varying variables. To get out of single mode press capital **Z**.

19.5 Running an optimization

Start *Tao* using the example optimization files (§19.1) or use the **reinit tao** command to reinitialize *Tao*. The plot window should look like Figure 20a.

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

```
Tao> show opti
Data Used:
  twiss.a[1:2]                                  Using: 1:2
  twiss.b[1:2]                                  Using: 1:2
Variables Used:
  quad[1:6]                                     Using: 1:6

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
  %lmdif_eps                     =   9.99999996E-13
  %merit_stop_value              =  -1.00000000E+00
  %svd_cutoff                    =   9.99999975E-06
... etc...
```

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 what the biggest contributions to the merit function are use the **show top10** command:

```
Tao> show top

Constraints                    ...   Ele/S     Target      Value      Merit
twiss.b[1]  beta.b <target>    ...   END       1.200E+01   1.513E+01  9.82E+01
twiss.a[1]  beta.a <target>    ...   END       1.200E+01   9.292E+00  7.33E+01
twiss.b[2]  alpha.b <target>   ...   END      -4.000E-01  -2.527E-01  2.17E+00
twiss.a[2]  alpha.a <target>   ...   END      -4.000E-01  -4.427E-01  1.82E-01
quad[6]     Q6[K1]             ...   3.80     -5.000E+01  -1.000E+00  0.00E+00
quad[5]     Q5[K1]             ...   3.30      5.000E+01   1.000E+00  0.00E+00
... etc...

 figure of merit: 1.738388E+02

List of non-zero contributors to the Merit Function:
Name                           Merit           Sigma [= sqrt(Chi^2/N)]
twiss.b                        1.0034E+02      2.2179E+00
twiss.a                        7.3499E+01      1.9149E+00
```

This shows, among other things, that the value of the merit function is 173.8 and that the largest contributor to the merit function is **twiss.b[1]** which has a contribution of 98.2 which is over 50%.

To run, say, the **lm** optimizer use the **run lm** command. In this case, this command has been

aliased to the **opt** command by the **setup.tao** command file that was run at initialization (§19.1):

```
Tao> opt
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    5.4532E-01  1.00E-04
    2    8.7067E-02  1.00E-05
... etc...
   19    3.5034E-04  1.00E-22
   20    2.9333E-04  1.00E-23
Written: var1.out
   21    2.9333E-04  0.00E+00
```

The optimization has managed to reduce the merit function from to 2.9E-4 or about six orders of magnitude. Further reductions in the merit function can be had by running the optimizer repeatedly. At the end of optimization, *Tao* creates a file **var1.out** which contains the optimized variable values:

```
 Q1[K1] =   7.76217267682967E-01
 Q2[K1] =  -1.40330956873957E+00
 Q3[K1] =   8.78773428675319E-01
 Q4[K1] =  -1.07943627873167E+00
 Q5[K1] =   1.29072113653010E+00
 Q6[K1] =  -4.57469896955973E-01
... etc...
```

This file should be virtually identical to the **optimized_var.out** file (§19.1). The format of this file conforms to *Bmad* lattice file syntax so this file can be used to create a lattice with the optimized values. One way to form a lattice with optimized values is to create a new lattice file that calls the original lattice and **var1.out**. That is, the new file would look like:

```
call, file = lat.bmad
call, file = var1.out
```

Note: At any time to print in Bmad format the variable values used in the optimization use the command:

```
show var -bmad -good
```

To save directly to a file, add the write option:

```
show -write solution.bmad var -bmad -good
```

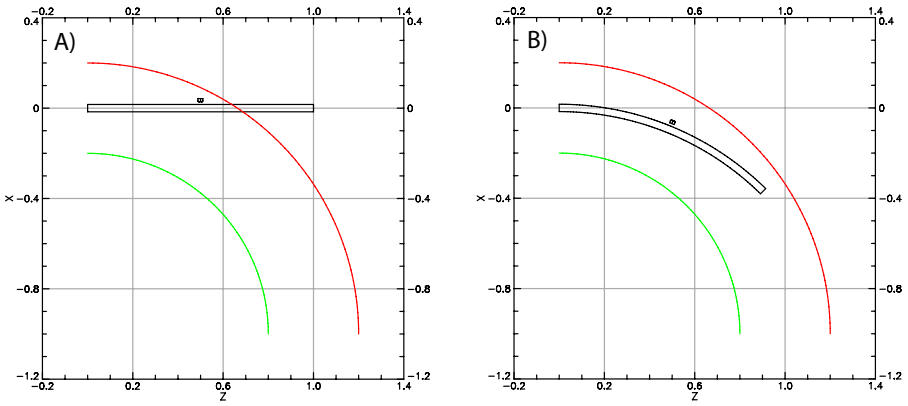


Figure 21: Optimization to keep a machine within existing building walls. A) Initial setup. The end of the machine (a 1 meter long dipole) is outside the walls (red and green semi circles). B) After optimization the machine is safely inside the walls.

19.6 Exercises

19.1 Start with the files in **examples/tutorial_bmad_tao/lattice_files/lattice_optimization**. Change all the quadrupole strengths to zero in **lat.bmad** and change the alpha Twiss **meas** targets to -1. Now run with the **opt** command and verify that the optimizer does not not find a good solution! Why is this? The problem is that the good solutions (and there is more than one) are outside of the local minimum that the optimizer is stuck in. Note: To easily reset the lattice use the command:

```
set lattice model = design
```

19.2 Start with the situation in Exercise 19.1 and find a good solution by first using **single mode** to vary the quadrupole strengths to find an approximate solution. Then run the **lm** optimizer to polish the results. This is a general strategy, often there is no single method that will work so a combination of methods is what is needed.

19.3 Start with the situation in Exercise 19.1 and find a good solution by first using the **de** optimizer which can find global minimums. Then run the **lm** optimizer to polish the results. Warning: Success here depends upon finding the right **de** parameter settings to use. This will take some thought and experimentation so successful completion of this exercise will not be quick.

19.4 This exercise shows how to do an optimization with a constraint that the machine stay within existing building walls. If you get stuck, a working example can be viewed at:

```
$ACC_ROOT_DIR/tao/examples/building_wall_optimization
```

(a) Construct a lattice with a single element which is a 1 meter long bend with zero bend angle.

(b) Setup a *Tao* input file that defines two building wall sections as shown in Figure 21A. The sections are two circular arcs of radius 0.8 meters and 1.2 meters.

(c) Setup two datums: **wall[1]** and **wall[2]**. The first datum constrains the end of the lattice to be to the inside of the outside (left) wall with a 0.1 meter clearance. The second datum constrains the end of the lattice to be to the outside of the inside (right) wall with a 0.1 meter clearance.

(d) Setup a variable to vary either the **g** or **angle** component of the bend.

(e) Run the **lm** optimizer to produce Figure 21B. Voila! The machine is within the walls with the desired clearance.

(f) When you startup *Tao* and the dipole is unbent, you should get the warning:

```
[WARNING] tao_init:
    DATUM EXISTS BUT CANNOT COMPUTE A MODEL VALUE: wall[2]
       INVALID SINCE: No wall section found in the transverse plane
                      of the evaluation point.
```

Explain why you are getting this warning. Also explain why this warning goes away when the dipole gets bent enough. [Hint: Read carefully the description of how the datum value is computed.] Note that not being initially able to compute a model value does not hinder the optimization.

20 Beam tracking in Tao

Tao has two basic particle tracking modes: **single** and **beam**. **Single** particle tracking is the default mode. In this mode a single particle is tracked and this tracking is used for orbit and Twiss calculations. This is the mode that has been used up to now in this tutorial.

With **beam** tracking, *Tao* does the same single particle tracking as in **single** particle tracking mode and, in addition, *Tao* tracks a **beam** of particles. A particle beam is made up of a number of **bunches** with each bunch being made up of some number of **particles**. Typically beams with only a single bunch are simulated. **Beam** tracking allows for interparticle effects to be simulated, for example, Coherent synchrotron Radiation (CSR).

The example files used to illustrate optimization are in the directory

```
examples/tutorial_bmad_tao/lattice_files/beam_tracking
```

There are five files here:

```
lat.bmad          ! Lattice file.
setup.tao         ! Command file run at startup.
tao.init          ! Primary Tao initialization file.
tao_plot.init     ! Secondary initialization file.
beam.tao          ! Command file to track a beam.
```

These are similar to the files used for lattice optimization (§19.1.

The initial beam distribution is determined by the settings of the **beam_init** structure in the **tao_beam_init** namelist. With the present example, **beam_init** is set in the **tao.init** file:

```
! Simple Gaussian beam
&tao_beam_init
  beam_init%n_particle = 1000
  beam_init%a_norm_emit = 1.0e-6. ! 1 mm-mrad
  beam_init%b_norm_emit = 1.0e-6. ! 1 mm-mrad
  beam_init%bunch_charge = 1e-9   ! 1 nC
  beam_init%sig_pz = 1e-3         ! 10^-3 relative
  beam_init%sig_z = 0.00059958    ! 2 ps * cLight
  beam_saved_at =  "*"     ! Save distribution at all elements.
/
```

Documentation on setting the **beam_init** structure is in the **Beam Initialization** chapter of the *Bmad* manual. The initial beam distribution can be set from a file of particle positions or by specifying general parameters like the emittance, etc. In this case, there is a single bunch with 1000 particles with normalized emittances of 1 mm-mrad in both planes, etc.

When the beam is tracked, beam distribution statistics like the centroid are calculated at every lattice element. Since saving the particle distribution is memory intensive when there is a large number of particles, the particle distribution is only saved at lattice elements specified by the **beam_saved_at** parameter in the **tao_beam_init** namelist. In this case the particle distribution is saved at the exit end of every lattice element. To, say, just save at every marker element, **beam_saved_at** could be set to "**marker::***".

The **tao.init** file specifies that the **setup.tao** file should be run at startup. This file defines some aliases. To switch to beam tracking mode,

```
Tao> set global track_type = beam
```

This will immediately track these particles to the end of the lattice. Statistics from beam tracking are stored at the end of every element, and seen from the **show beam** command:

```
Tao> show beam q5
Cached bunch parameters:
  Parameters for bunch:       1
  Particles surviving:        1000
  Particles lost:             0
  Particles lost (%):         .000
  Charge live (C):              1.00000000E-09
  Centroid:  6.45361481E-12 -5.64470815E-13 -1.24096868E-11 -8.79861917E-13 ...
  RMS:       6.75168709E-04  8.93956548E-05  8.21313385E-04  1.12761136E-04 ...
             norm_emitt           beta             alpha
  a:         9.98698154E-07  7.24443183E+00  5.10465184E-01
  b:         9.98705861E-07  1.07200175E+01 -1.22746781E+00
  x:         9.98699589E-07  8.92076260E+00  6.28587261E-01
  y:         9.98704411E-07  1.32005818E+01 -1.51150120E+00
  z:         1.17181728E-05  5.99704551E-01

Sigma Mat       x              px               y              py
X     4.55852786E-07 -3.21209371E-08 -1.42999858E-12  1.89319887E-13 ...
Px   -3.21209371E-08  7.99158310E-09  4.14153134E-13  1.10347243E-14 ...
Y    -1.42999858E-12  4.14153134E-13  6.74555677E-07  7.72383921E-08 ...
Py    1.89319887E-13  1.10347243E-14  7.72383921E-08  1.27150739E-08 ...
Z    -9.15470557E-12  5.66620835E-13  9.99050861E-12  5.43154863E-13 ...
Pz   -6.75696744E-12  5.60363622E-13  1.20598961E-11  8.99073770E-13 ...

Note: Individual particle positions are saved at this element.
```

The **beam_saved_at** element list will save the full particle distribution at the end of the matching elements, in this case all elements.

These particles and their statistics can be plotted. For example,

```
Tao> place r12 bunch_sigma_xy
```

will display line plots along the beamline. The $x - p_x$ phase space can be plotted as:

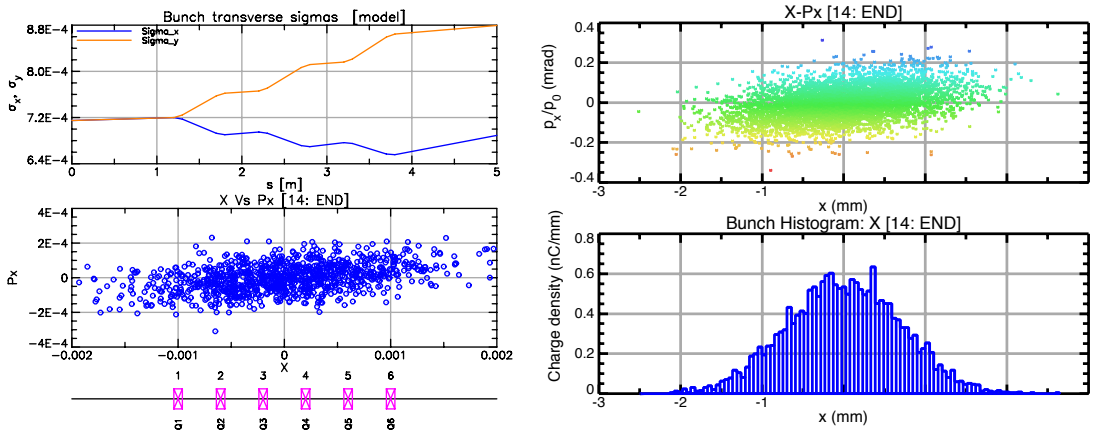
```
Tao> place r22 bunch_x_px
```

With this type of beam, the number of particles can be changed by:

```
Tao> set beam_init n_particle = 5000
```

The reference element for the plotting can be changed by:

```
Tao> set curve r22.g.c ele_ref_name = Q5
```



(a) Basic beam size and phase space

(b) Custom plot templates, with histogram

Figure 22: Beam plotting. Particles can be colored by their attributes, in this case simply by the p_x coordinate.

Any changes to the lattice (including optimization steps) will result in re-tracking the beam. To return to single particle tracking mode, set:

```
Tao> set global track_type = single
```

21 Wave Analysis

Wave analysis is a method for finding isolated "kick errors" in a machine by analyzing the appropriate data. For example, consider the orbit of a beam. In some region of the machine, assuming are no orbit kicks in the region (and assuming no x-y coupling), the horizontal orbit $x(s)$ of the beam will be "wave" given by the standard formula

$$x(s) = A \sqrt{\beta_x(s)} \cos(\phi_x s + \phi_{x0}) \tag{3}$$

where A and ϕ_0 depend upon the position of the beam at the beginning of the region and $\beta(s)$ and $\phi_x(s)$ are the standard beta and betatron phase parameters. Consider then choosing some region of the machine and fitting the data from an orbit measurement in this region to Eq. (3) using A and ϕ_{x0} as fitting parameters. If there indeed where no kicks in this region, and if the data is perfect, a plot of the measured orbit minus the fit orbit will be zero in the region. If this plot of orbit minus fit is extended to the entire machine, the plot will become non-zero after any point where there is a kick to the beam. In other words, a plot of **orbit - fit** can be used to locate where a kick is happening. This is the essence of wave analysis [1]. In practice, two fits are done to two different regions on either side of where a kicker is thought to be located. This allows for a more accurate calculation of where the kick is.

Wave analysis can be used other types of data besides orbits:

Measurement Type	*Error Type*
`Orbit`	`Steering errors`
`Betatron phase differences`	`Quadrupolar errors`
`Beta function differences`	`Quadrupolar errors`
`Horizontal/Vertical Coupling`	`Skew quadrupolar errors`
`Dispersion differences`	`Sextupole errors`

Table 1: Types of measurements that can be used in a wave analysis and the types of errors that can be diagnosed.

Wave analysis can not only find errors in a machine it can be used to measure calibration constants in magnets. For example, by measuring the coupling at two different setting of a given skew quadrupole magnet, the magnets calibration can be determined. For further information, see the **Wave Analysis** chapter (§8) in the *Tao* manual.

21.1 Example Analysis

The example wave analysis presented here uses actual data taken at the Cornell storage ring CESR. Two measurement of the betatron phase were made the second one taken about two days after the first. The betatron phase is measured by shaking the beam at the betatron resonance frequencies and measuring the turn-by-turn response at the BPM detectors[2]. The amplitude of the sinusoidal response gives the beta function and the phase of the response gives the betatron phase. The horizontal/vertical coupling can also be extracted from the data

using the ratio of the response in the two planes. In practice, the betatron phase data is less noisy than the beta measurement since the phase is fairly insensitive to errors in measuring the oscillation amplitude. This being the case, it is the betatron phase that will be analyzed here.

Input files for this example are in the directory:

```
examples/tutorial_bmad_tao/lattice_files/wave_analysis
```

The relevant files are

```
chess-u_6000mev_20181120.lat    ! Lattice file
setup.tao                       ! Startup command file
tao.init                        ! Tao init file
wave_anal.tao                   ! Wave analysis commands.
```

A wave analysis works on data so the **tao.init** file sets up data for the betatron phase:

```
&tao_d2_data
  d2_data%name = 'phase'
  n_d1_data = 2
/

&tao_d1_data
  ix_d1_data = 1
  d1_data%name = 'a'
  search_for_lat_eles = "type::BPM*"
/

&tao_d1_data
  ix_d1_data = 2
  d1_data%name = 'b'
  use_same_lat_eles_as = "phase.a"
/
```

Setting **search_for_lat_eles** to "**type::BPM***" works since, By convention, BPMs are represented in CESR lattices by **marker** elements whose **type** attribute begins with "**BPM**".

The phase data is set in the **setup.tao** file:

```
set data phase.a[1:20]|ref = [-54.2921, -52.0007, -51.8171, ...
set data phase.a[21:40]|ref = [-34.7387, -34.2289, -33.0380, ...
set data phase.a[41:60]|ref = [-16.7037, -16.6183, -16.0310, ...
...

set data phase.a[1:20]|meas = [-54.2560, -51.9583, -51.7726, ...
set data phase.a[21:40]|meas = [-34.7252, -34.2236, -33.0035, ...
set data phase.a[41:60]|meas = [-16.6726, -16.5874, -16.0207, ...
...
```

The **ref** data represents the first measurement and the **meas** data represents the second.

The **setup.tao** file also modifies the betatron phase plot:

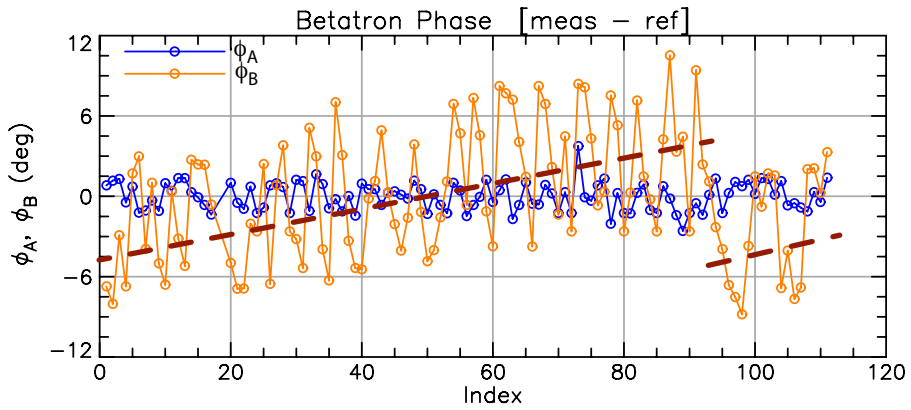


Figure 23: Betatron phase difference between two measurements taken two days apart in the Cornell CESR storage ring. The dashed lines indicate approximately the oscillation centroid of the **B**-mode phase difference.

```
set plot phase x_axis_type = index
set graph phase x%label = 'Index'
set graph phase component = meas - ref
set curve phase data_source = data
set curve phase draw_symbols = T
place r11 phase
```

The setting of **x_axis_type** to **index** switches the **x**-axis variable from **s**-position to data index. This is done for convenience later on.

Start *Tao* and the plot window should look like Figure 23 which shows the change in the "horizontal-like" **a**-mode phase ϕ_a (blue curve) along with the change in the "vertical-like" **b**-mode phase ϕ_b (orange curve). In the figure, a dashed line in red has been added to approximately show the average of the oscillations of ϕ_b. Even before doing a phase analysis, a lot can be learned by looking at the plot:

- In a region where there have been no changes in quadrupole strength, the oscillation centroid of the phase difference is constant. In Figure 23, the oscillation centroid for the ϕ_b difference is shown approximately by the red dashed line. As can be seen, there is a large jump in the centroid near detector 90 which indicates a quadrupolar field change in that region.
- The **a**-mode phase oscillations are small to compared to the **b**-mode phase indicating that the quadrupolar field change near detector 90 is at a point with large β_b compared to β_a.
- With the exception of the discontinuity near detector 90, the oscillation centroid of the **b**-mode phase has a slope but is not obviously discontinuous. This indicates that here has been small changes to the strength of many quadrupole magnets distributed throughout the ring.

Now run the wave analysis on the **phase.b** data:

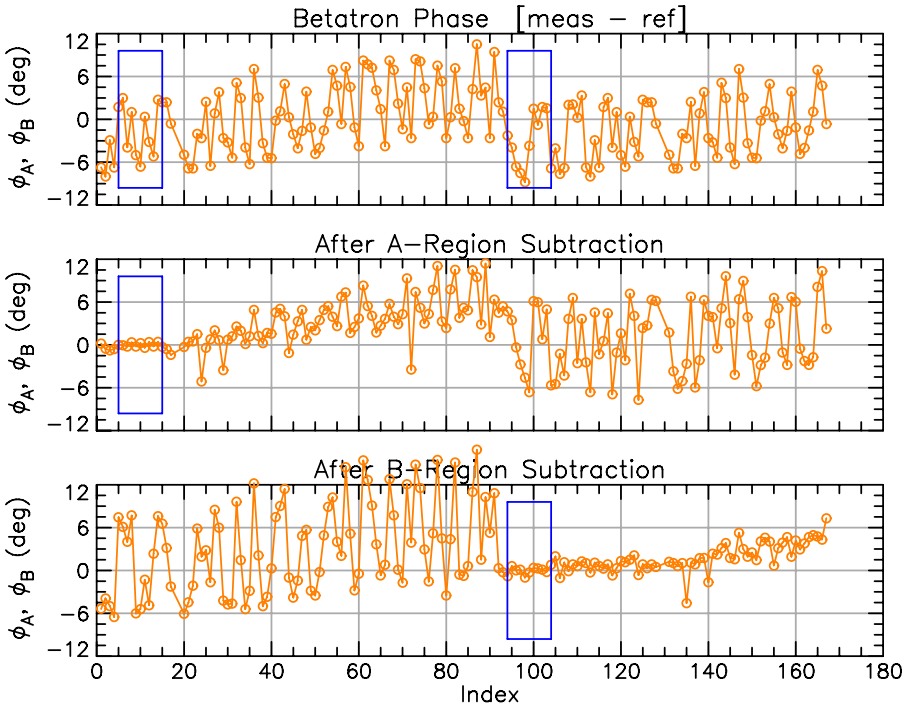


Figure 24: Wave analysis of the **phase.b** difference data. The blue boxes show the locations of the **A** and **B** fit regions. The data has been extended past the end of the lattice to allow a wave analysis for the region near the ends of the lattice. Top graph: The original data. Middle graph: The data with the **A**-region fit subtracted off. Bottom graph: The data with the **B**-region fit subtracted off. Notice that the **y**-axis label is simply taken from the original plot so that the fact that the label mentions ϕ_A should be ignored.

```
Tao> wave phase.b
```

The result is shown in Figure 24. There are two regions, called **A** and **B** where the data is fit to a betatron phase wave[1]. Initially, the placement of these two regions are somewhat arbitrarily chosen by *Tao*. In this case, the **A**-region is from datum 5 to datum 15 and the **B** region is from datum 94 to datum 104. Notice that, in order to be able to analyze the region near the ends of the lattice, the data has been extended by 1/2 of the length of the data array. In this case the **phase.b** data range was from 1 to 111. Thus in the extended curves in Figure 24, datum 112 is derived from datum 1, datum 113 is derived from datum 2, etc.

In Figure 24 the top plot is the original **phase.b** difference data, the middle plot is The difference data with the **A**-region fit subtracted off, and the bottom plot is the difference data with the **B**-region fit subtracted off. Since the difference between the data and the **A**-region fit is near zero in the **A**-region, and similarly for the **B**-region, this shows that both the **A** and **B** regions are well fitted. That is, there were no significant quadrupole changes in the fit regions in the time period between the two measurements. The goodness of the fits, "**Sigma_Fit/Amp_Fit**" is printed as

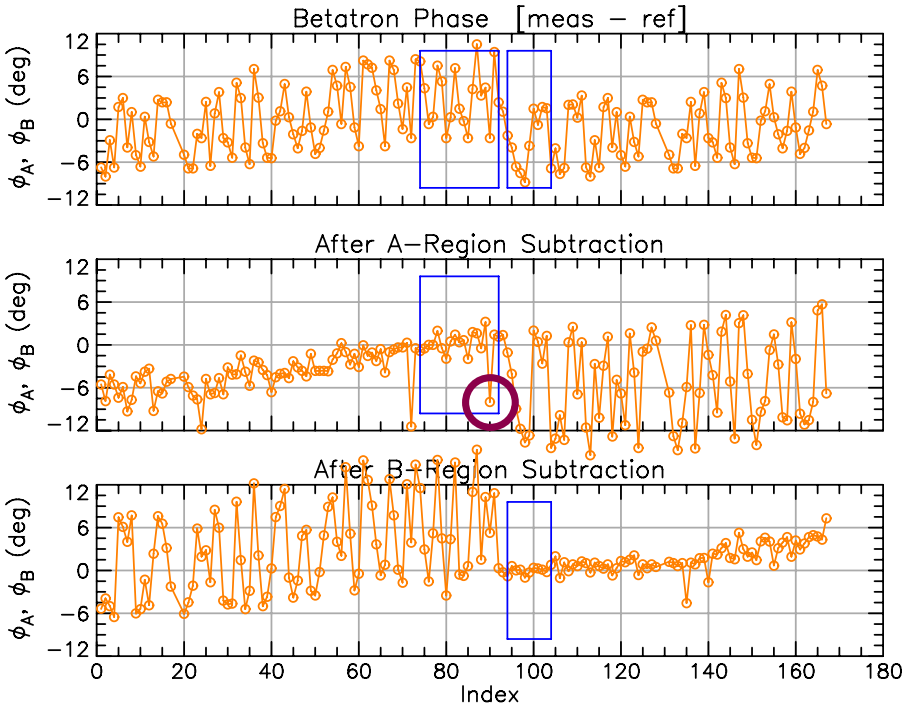


Figure 25: Wave analysis of the **phase.b** difference data after the fit regions have been adjusted to bracket the quadrupole error. The circle in the middle graph marks a bad data point.

part of the output of the **wave** command:

```
ix_a:    5  15
ix_b:   94 104
A Region Sigma_Fit/Amp_Fit:      0.035
B Region Sigma_Fit/Amp_Fit:      0.067
Sigma_Kick/Kick:     0.033
Sigma_phi:           0.020
Chi_C:               0.804 [Figure of Merit]

Normalized Kick = k * l * beta [dimensionless]
   where k = quadrupole gradient [rad/m^2].
After Dat#    Norm_Kick      s-pos    ele@kick                          phi
       20          0.13      77.48    248   D083                      16.340
       23         -0.13     103.39    290   B16W                      17.911
       25          0.13     126.07    307   D104                      19.482
       27         -0.13     137.80    320   B20W                      21.053
       30          0.13     161.15    344   B23W                      22.624
... etc...
```

A value of 1 indicates a poor fit and a value of zero indicates a good fit. In this case, with the goodness of the fits are both below 0.1 indicating a good fit. See the section on **Wave Analysis**

Commands and Output (§8.4) in the *Tao* manual for more details.

Since the fit regions are far apart, there are many possible error locations. To narrow down the possibilities, the fit regions need to be moved as close as possible while still maintaining good fits. Since the **B**-region fit shows a strong error just to the left of the **B**-regions left edge, it makes sense to move the **A**-region towards the **B** region.

The reader is invited to play around with adjusting the fit regions. Or just set:

```
set wave ix_a = 74 92    ! Set A-region boundaries
set wave ix_b = 94 104   ! Set B-region boundaries
```

The result is shown in Figure 25. A unique solution has been bracketed:

```
ix_a:  74  92
ix_b:  94 104
...
Normalized Kick = k * l * beta [dimensionless]
   where k = quadrupole gradient [rad/m^2].
After Dat#    Norm_Kick     s-pos    ele@kick                  phi
       93          0.27    683.95    956   D408               63.314
```

This shows a quadrupole error at about **s** = 684 meters. Nearby elements are:

```
Tao> show lat -s 681:687
# Values shown are for the Exit End of each Element:
# Index  name           key                       s       l
#
    946  D404#2         Drift               682.171   1.684  ...
    947  D14BE_RFA1_SEG Marker              682.171   0.000  ...
    948  D404#3         Drift               682.350   0.179  ...
    949  DOG_LEG_14E1   Kicker              682.594   0.244  ...
    950  D405           Drift               682.856   0.262  ...
    951  DET_13E        Marker              682.856   0.000  ...
    952  D406           Drift               682.874   0.018  ...
    953  SEX_13E        Sextupole           683.146   0.272  ...
    954  D407           Drift               683.208   0.062  ...
    955  Q13E           Quadrupole          683.808   0.600  ...
    956  D408           Drift               683.998   0.189  ...
```

So the likely candidate is quadrupole **Q13E**. The **show element** command shows that the **b**-mode beta function at this element is about a factor of 10 larger than the **b**-mode beta which explains the small **phase.a** signal. Subsequent investigation showed that there was a ground problem which was corrected and further observation showed that this fixed the problem.

Notes:

- In the middle plot in Figure 25, there is a data point, marked by a red circle, within the **A**-region whose value is far from the its neighboring points. This indicates a bad data point. To get a better fit, this data point can be removed from the plot and the fit using the command

  ```
  veto data phase.b[90]
  ```

- To make the **setup.tao** command file run faster, lattice and replotting calculations are suspended by the following at the top of the command file:

```
set global lattice_calc_on = F   ! Stop lattice calculations.
set global plot_on = F           ! Stop replotting.
```

  with corresponding commands at the end of the file to re-enable calculations. This is not a big factor here but in other cases can save a significant amount of time.

- For the wave analysis to work the right edge of the **A** region must be to the left of the left edge of the **B**-region.

- To increase the accuracy of the analysis, the quadrupoles and skew quadrupoles in the**model** lattice should be varied to fit the **model** Twiss parameters to one of the measurements. This is especially true when calibrating skew quadrupoles since the the effect of a skew quadrupole is proportional to the tune difference between the **a** and **b** normal modes.

- Data where there are multiple error locations that are well enough separated can be analyzed by varying the **A** and **B** regions to successively to bracket the individual error locations.

21.2 Exercises

21.1 Create your own data for doing a wave analysis. A easy way to do this is to introduce errors into the model lattice and then set **meas** (or **ref**) data:

```
set data phase.a|meas = phase.a|model + 0.1 * ran_gauss()
```

The **ran_gauss()** function adds noise to the data so you can experiment with how noise degrades the analysis.

21.2 By creating your own data, experiment with how well you can resolve two errors that are close together.

22 References

[1] D. Sagan, “Betatron phase and coupling correction at the Cornell Electron/Positron Storage Ring,” *Physical Review Special Topics - Accelerators and Beams*, vol. 3, p. 102801, Oct. 2000.

[2] R. L. D. Sagan, R. Meller and D. Rubin, “Betatron phase and coupling measurements at the Cornell electron/positron storage ring,” *Phys. Rev. ST Accel. Beams*, vol. 3, p. 092801, 2000. Note: While the paper does not mention it, Bmad is the software being used here for the analysis.