Revision: November 8, 2019

The Tao Graphic User Interface Manual

John Mastroberti & David Sagan

Contents

List of Figures

List of Tables

Chapter 1

Overview

Tao is an open source general purpose program for charged particle and X-ray simulations in accelerators and storage rings. It is built on top of the *Bmad* toolkit (software library) which provides the needed computational routines needed to do simulations. Essentially you can think of *Tao* as a car and *Bmad* as the engine that powers the car. In fact *Bmad* powers a number of other simulation programs but that is getting outside of the scope of this manual.

Documentation for *Bmad* and *Tao*, as well as information for downloading the code if needed is given on the *Bmad* web site

`https://www.classe.cornell.edu/bmad`

Tao by itself is a command line program. To make it more readily accessible, a graphic user interface (GUI) has been developed and this is the subject of this manual. The GUI is written in Python. The Python `tkinter` package is used for windowing. `Tkinter` being an interface layer for the `Tk` widget toolset. The Python based plotting package MatPlotLib can be used for graphics. Alternatively, the PGPLOT/PLPLOT plotting that comes standard with Tao can be used.

It is assumed in this manual that the reader already has some familiarity with *Bmad* and *Tao*. If not, there are manuals for *Bmad* and *Tao* posted on the *Bmad* web site along with a beginner's tutorial.

The coding the of GUI was a joint development project with John Mastroberti, Kevin Kowalski, and David Sagan. Thanks must go to Thomas Gläße for helping with the inital development.

Chapter 2

GUI Installation

2.1 Obtaining the Source Code

Source code and documentation for *Bmad*, *Tao* and the GUI for *Tao* can be obtained, if needed, at the *Bmad* web site at:

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

2.2 Building Tao

As a prerequisite, if not already available, *Tao* must be built before using the GUI. Build instructions are available on the *Bmad* web site. There are two ways for the GUI scripts (written in Python) to interact with *Tao*. One way is to use the `pexpect` module which is a communications layer that interfaces to *Tao*'s command line interface. The other way is to use `ctypes` (an interface between Python and C) to communicate directly with the *Tao* subroutine library (the *Tao* program is built by linking to the *Tao* library).

The advantage of using `ctypes` is that it is faster. The drawback is that `ctypes` requires a version of the *Tao* library that is `shared object`. If you are using a *Bmad* "`Distribution`" (a package that is downloaded from the Web containing *Bmad*, *Tao*, associated libraries, etc.), the default is *not* to build shared object libraries. This default can, of course, be changed but if you do not have control of how things are built, you may have to use `pexpect`. To check if there is a shared object library built, issue the following command:

```
ls $ACC_ROOT_DIR/production/lib/libtao.*
```

[This assumes that you are not setting `ACC_LOCAL_ROOT` as discussed in Sec. §2.4.] In all cases you will see a file `libtao.a`. This is a static library which is always built but not of use. A file with an extension `.so` (UNIX) or `.dylib` (Mac) is a shared object library.

2.3 Python Requirements

Minimum Python version for the GUI is Python 3.6.

The GUI depends upon a number of modules that may have to be downloaded:

```
  tkinter
  ttk (may be called pyttk)
  pexpect         # If using pexpect instead of ctypes.
  matplotlib
  cycler
  ateutil
  tkagg
```

Note: The GUI uses the TkAgg backend for matplotlib. There may be a problem with Python finding the TkAgg backend. On the mac, using macports, the solution is to install matplotlib with the `tkinter` variant. Something like:

```
  sudo port uninstall py36-matplotlib           # May not be needed.
  sudo port install  py36-matplotlib +tkinter   # This is when using Python version 3.6
```

For more information see:

```
  https://matplotlib.org/tutorials/introductory/usage.html#backends
```

If one of the modules is missing, python will generate an error message. For example:

```
> python ../../gui/main.py
Exception in Tkinter callback
Traceback (most recent call last):
  File "/opt/local/Library/Frameworks/Python.framework/Versions/3.7/lib/
                          python3.7/tkinter/__init__.py", line 1705, in __call__
    return self.func(*args)
  File "../../gui/main.py", line 372, in param_load
    from tao_interface import tao_interface
  File "/Users/dcs16/Bmad/bmad_dist/tao/gui/tao_interface.py", line 4, in <module>
    from tao_pipe import tao_io
  File "/Users/dcs16/Bmad/bmad_dist/tao/python/tao_pexpect/tao_pipe.py", line 14, in <module>
    import pexpect
ModuleNotFoundError: No module named 'pexpect'
```

Notice that the last line shows that the pexpect module is needed.

How to install missing modules on the mac: [Note: The exact installation commands will depend upon which version of python is being used. Use the "python –version" command to see what version you are using.

Using macports and python 3.6:

```
  sudo port install py36-tkinter
  sudo port install py36-pexpect
```

Using pip (or pip3):

```
  sudo pip install pytkk
  sudo pip install pexpect
```

WARNING: it can be dangerous to use pip to install/modify modules in your system Python. A much safer way to install the modules you need is to set up a python virtual environment. On Linux, you may also be able to find versions of the required modules in your system package manager, which are tailored to your Linux distribution and will not break your system python.

2.4 Environmental Variables

To run the GUI (or even to run *Tao* without the GUI), certain environmental varibles must be set. This is the same initialization that is done when compiling *Bmad* and *Tao*. See your local Guru or the

Bmad web site for more details. To see if the environmental variables have been set, run the `accinfo` command.

It may be desireable to specify a local build tree as the place for the python scripts to find the *Tao* executable or *Tao* shared object library. To accomplish this, set the environmental variable `ACC_LOCAL_ROOT` to the base directory of your local build tree.

```
export ACC_LOCAL_ROOT=/home/dcs16/bmad_dist
```

The standard place for the GUI script files is at:

```
"$ACC_ROOT_DIR/tao/python/pytao/gui
```

When doing GUI development work, the default location can be changed by setting `ACC_PYTHONPATH`. Example:

```
export ACC_PYTHONPATH="$ACC_LOCAL_ROOT/tao/python/pytao/gui"
```

[This assumes that `ACC_LOCAL_ROOT` has been set.] `ACC_PYTHONPATH` must be set before *Bmad* is initialized. That is, if *Bmad* is initialized in the `.bashrc` file, `ACC_PYTHONPATH` must be initialized in the `.bashrc` file before the *Bmad* initialization.

To check that `PYTHONPATH` has the correct value use the command:

```
printenv |grep PYTHONPATH
```

2.5 Installation Troubleshooting

Got error:

```
ImportError: cannot import name '_tkagg'
```

Solution: Uninstall and then reinstall matplotlib. For example, if using pip:

```
sudo pip uninstall matplotlib
sudo pip install matplotlib
```

Chapter 3

GUI Startup

3.1 Starting the GUI

The GUI can be started from the system shell by executing the command

```
python -m pytao.gui
```

You can also specify any of the command line options that *Tao* supports. For example,

```
python -m pytao.gui -init_file ~/bmad_dist/tao/examples/cesr/tao.init -rf_on
```

This will prefill the settings for `init_file` and `rf_on`. The GUI starts with the window shown in Figure 3.1. From here, all of the command-line settings that Tao supports can be set (settings that are left blank are omitted when Tao is started). The parameters above the horizontal separator bar are *Tao* parameters and the parameters below the bar are GUI specific parameters.

Towards the bottom of the window, below the horizontal separator, are some settings that are specific to the GUI. The "Interface to Tao" setting controls whether the `ctypes` or `pexpect` backend for communicating with Tao will be used. Below it, the "Shared Library" (if "Interface to Tao" is set to "ctypes") or "Tao Executable" (if "Interface to Tao" is set to "pexpect") setting points the GUI to the correct executable or shared object library to use. In most cases, the GUI will prefill this box by referencing the ACC_LOCAL_ROOT and ACC_ROOT_DIR environmental variables.

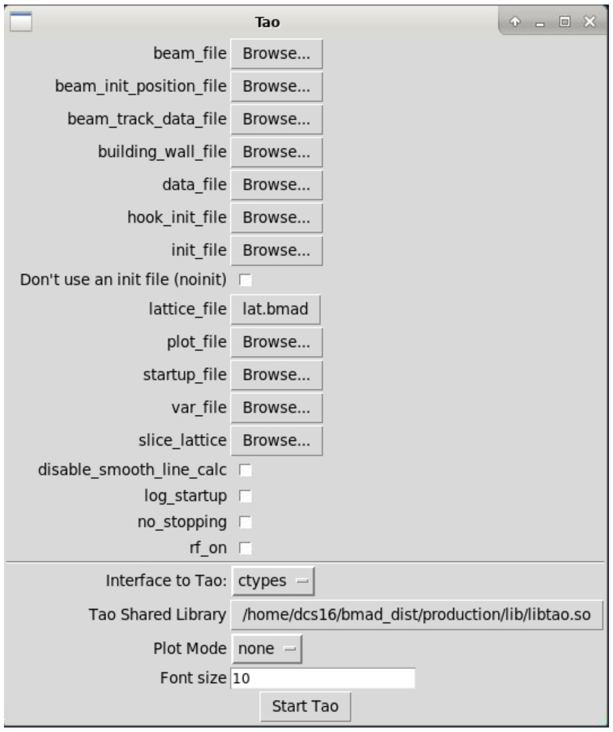
The setting of `Plot Mode` controls what plotting environment is used. Possible setting are:

```
matplotlib
pgplot/plplot
none
```

`MatPlotLib` is a python based package which is the most versatile. The `pgplot/pgplot` option is the standard tao plotting used with the command line version of *Tao*. Note: If the *Tao* `-noplot` option is present on the command line when the GUI is started, `Plot Mode` will be set to `none`. Note: Which plotting package is used with `pgplot/plplot`, either `pgplot` or `plplot`, is determined by how *Tao* was compiled.

Finally, the font size can be set as desired. Hitting Enter/Return while the font size box is in focus will adjust the font size of the startup window to give the user a sense of what the chosen font size will look like.

Once all of the startup settings have been set, clicking "Start Tao" will initialize Tao and bring the user to the main GUI window.



Tao	
beam_file	Browse...
beam_init_position_file	Browse...
beam_track_data_file	Browse...
building_wall_file	Browse...
data_file	Browse...
hook_init_file	Browse...
init_file	Browse...
Don't use an init file (noinit)	☐
lattice_file	lat.bmad
plot_file	Browse...
startup_file	Browse...
var_file	Browse...
slice_lattice	Browse...
disable_smooth_line_calc	☐
log_startup	☐
no_stopping	☐
rf_on	☐
Interface to Tao:	ctypes
Tao Shared Library	/home/dcs16/bmad_dist/production/lib/libtao.so
Plot Mode	none
Font size	10
	Start Tao

Figure 3.1: The GUI startup window. In this example, the init file that tao should use has been specified.

3.2 GUI Initialization File

To speed up the initialization process, you can make an init file for the GUI. This file should be called "gui.init", and it should be in the same directory from which you start the GUI.

gui.init should have each option on a separate line, and each option should be listed in the form "parameter:value". For example, the text below would constitute a good gui.init file:

```
#MY GUI INIT FILE
beam_file:/path/to/beam/file
data_file:/path/to/data/file
#THIS IS A COMMENT
disable_smooth_line_calc:T
rf_on:T
tao_executable:/path/to/executable
```

For *Tao* specific parameters, the names in the `gui.init` file correspond to the names in the startup window (Fig. 3.1) or in one case, the name in parentheses. For GUI specific parameters (below the horizontal separator), replace blanks between words and make all letters lower case. For example, "`Interface to Tao`" in the startup window becomes `interface_to_tao` in the `gui.init` file. Note: The startup window can be bypassed altogether by setting `start_tao` in `gui.init` to `True`.

The order in which you list options in gui.init is not important.

File paths should be specified in full to be safe, but you can specify paths relative to the directory from which you launch main.py. For example, "/home/username/file", "subfolder/my_file", and "../../path/to/another/file" would all be acceptable file paths. You can also use your environmental variables and "~", as in "~/Documents/my_file" and "$DIST_BASE_DIR/tao/file".

For logical parameters, for example, "rf_on", Use T/F or True/False as the parameter's value.

You can also include comments with #. Anything after a # character will be ignored.

To get a list of parameters that can be set in a `gui.init` file, start Tao with the command

```
  tao -help
```

The corresponding gui.init parameter is the Tao parameter with the leading dash "-" removed and a colon ":" between the parameter and the parameter's value.

In order of precedence, parameters on the command line when the GUI is started take precedence over paramters defined in the GUI initialization file and these take precedence over parameters defined in *Tao* startup files.

Chapter 4

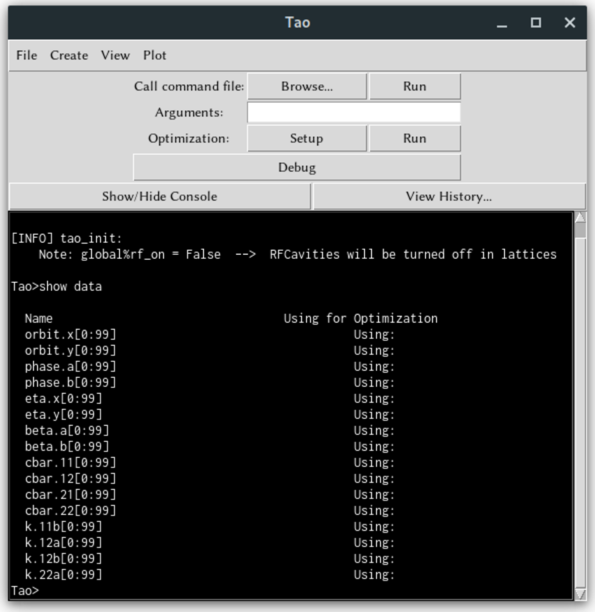
Main Window

4.1 Parameter Input

In general, real parameter values can be set using expressions just like parameter values can be set using expressions on the *Tao* command line.

4.2 The Main GUI Window

The main window for the GUI is shown in Figure 4.1. From here, the user has access to all of the GUI's features. Command files can be called by browsing for them and then clicking "Run", with



```
[INFO] tao_init:
    Note: global%rf_on = False  -->  RFCavities will be turned off in lattices

Tao>show data

  Name                                  Using for Optimization
  orbit.x[0:99]                                   Using:
  orbit.y[0:99]                                   Using:
  phase.a[0:99]                                   Using:
  phase.b[0:99]                                   Using:
  eta.x[0:99]                                     Using:
  eta.y[0:99]                                     Using:
  beta.a[0:99]                                    Using:
  beta.b[0:99]                                    Using:
  cbar.11[0:99]                                   Using:
  cbar.12[0:99]                                   Using:
  cbar.21[0:99]                                   Using:
  cbar.22[0:99]                                   Using:
  k.11b[0:99]                                     Using:
  k.12a[0:99]                                     Using:
  k.12b[0:99]                                     Using:
  k.22a[0:99]                                     Using:
Tao>
```

Figure 4.1: The main GUI window, showing the results of `show data` on the console.

arguments specified in the "Arguments" below. In the future, the user will also be able to set up and run optimization routines from this window, although this feature is not currently available.

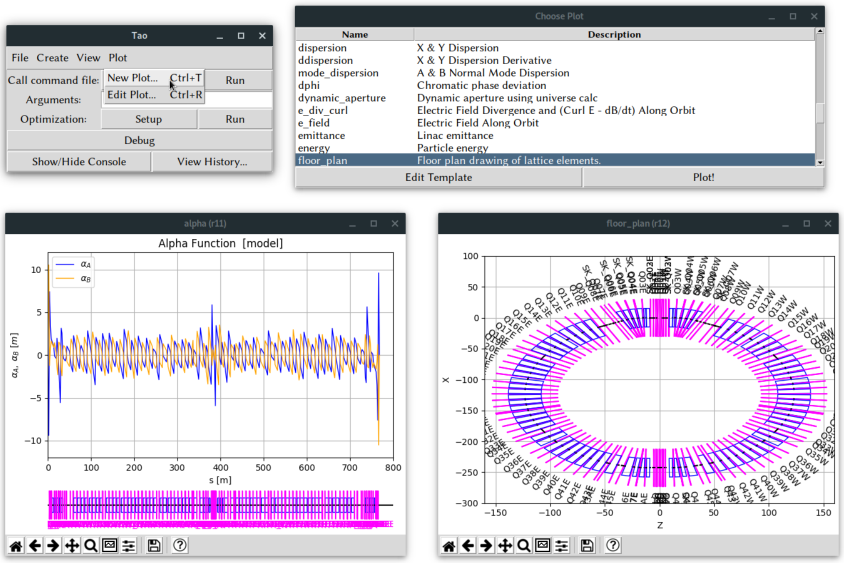
The main window also has a console, where commands can be run in Tao exactly like in regular Tao. The console will also display warning messages if a command produces an error.

Chapter 5

Plotting

One of the GUI's strong suits is the ability to view an manipulate plots using matplotlib. Existing plots can be viewed and edited, and new templates can be defined on the fly.

Note: the GUI does support classic PG/PLplots, but it is recommended to use matplotlib as the plotting engine due to the extra fatures it offers.



Name	Description
dispersion	X & Y Dispersion
ddispersion	X & Y Dispersion Derivative
mode_dispersion	A & B Normal Mode Dispersion
dphi	Chromatic phase deviation
dynamic_aperture	Dynamic aperture using universe calc
e_div_curl	Electric Field Divergence and (Curl E - dB/dt) Along Orbit
e_field	Electric Field Along Orbit
emittance	Linac emittance
energy	Particle energy
floor_plan	Floor plan drawing of lattice elements.

Figure 5.1: Viewing plots with the GUI. Top left: accessing the plot template list from the root window. Top right: the plot template window. Bottom left: a plot of the alpha function with the lat_layout shown below. Bottom right: the floor_plan plot.

5.1 Viewing Plots

Figure 5.1 shows how to plot existing templates in the GUI. The plot template window lists all of the currently defined plot templates as well as their descriptions. A template can be plotted either by double clicking on it, or by selecting it and then clicking the "Plot" button. This will open a window displaying the selected plot (see Section 5.2 for more information). The user can also edit an existing template by selecting it in the tempalte window and clicking "Edit Template".

5.2 Interacting with Plots

Interacting with plots in the GUI is primarily done using the toolbar along the bottom of the plotting window.

Starting from the left, the home button returns the graph to the starting view. The back button returns to the previous view of the graph, working as an undo button. The forward button undoes the effects of the back button. All of these buttons also have keyboard shortcuts, 'h' or 'r' for home, 'c' or 'left arrow' for back, and 'v' or 'right arrow' for forward.

The pan/zoom button allows panning of the graph by holding a left click on the graph and dragging the mouse around. his mode also allows zooming of the graph by holding a right click on the graph and dragging the mouse around. These actions can be restricted to the horizontal axis by holding 'x' while dragging, or the vertical axis by holding 'y'. Holding 'control' while dragging the mouse will preserve aspect ratio. Clicking on the toolbar button again will get out of this mode.

The zoom to rectangle button allows a rectangle to be selected by holding a left click on the graph and dragging the mouse, which will then fill the graph window. Holding 'x' or 'y' while selecting a rectangle will only effect the horizontal or vertical axis respectively.

The save button allows the graph window to be saved as an image file. Using 'ctrl+s' has the same effect.

If any floor plan or lat layout is present, double clicking on an element will open a window to view or edit element parameters.

Floor plans contain a slider to scale the size of elements away from the element centerline. Clicking on the slider will adjust the width of the displayed elements.

5.3 Plotting Initialization

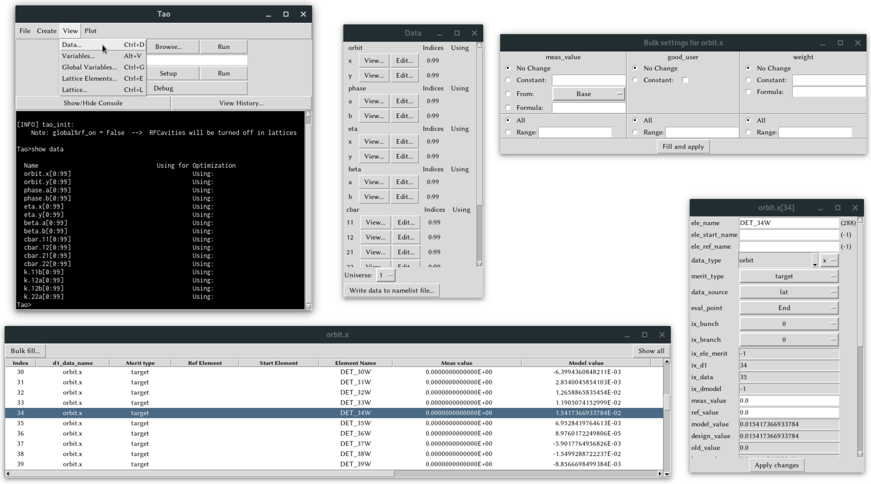
When operating the GUI in matplotlib mode, you can specify a list of template plots to plot in matplotlib as soon as tao starts. These templates should be listed in a file called plot.gui.init, which should be in the same directory from which you launch the gui.

plot.gui.init should have one template listed per line and absolutely nothing else on the line. The templates listed in plot.gui.init are checked against the list of templates that tao can plot, and if a template is not recognized it is simply ignored.

Chapter 6

Data

The GUI provides several windows for viewing and editing data arrays. This chapter assumes you are familiar with *Tao* data nomenclature as discussed in the `Data` chapter of the *Tao* manual.

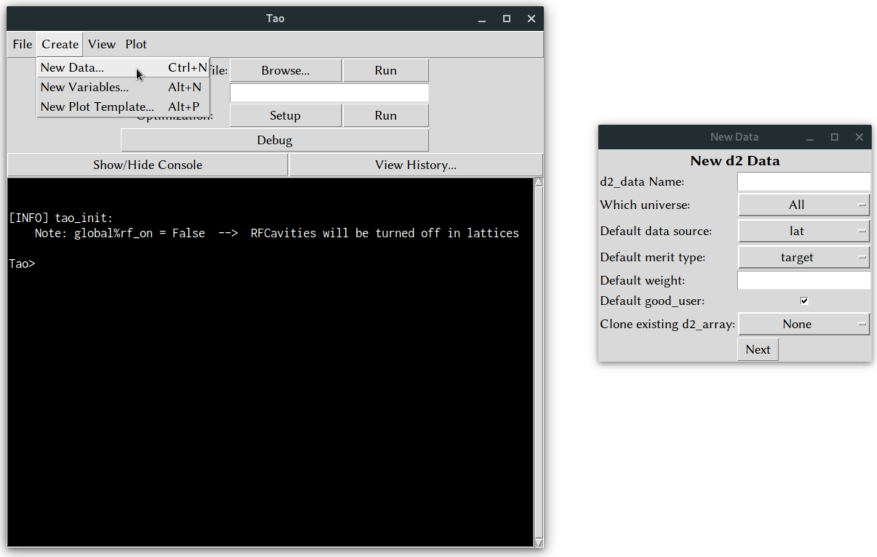


```
[INFO] tao_init:
   Note: global%rf_on = False  -->  RFCavities will be turned off in lattices

Tao>show data

  Name                                  Using for Optimization
  orbit.x[0:99]                                   Using:
  orbit.y[0:99]                                   Using:
  phase.a[0:99]                                   Using:
  phase.b[0:99]                                   Using:
  eta.x[0:99]                                     Using:
  eta.y[0:99]                                     Using:
  beta.a[0:99]                                    Using:
  beta.b[0:99]                                    Using:
  cbar.11[0:99]                                   Using:
  cbar.12[0:99]                                   Using:
  cbar.21[0:99]                                   Using:
  cbar.22[0:99]                                   Using:
  k.11b[0:99]                                     Using:
  k.12a[0:99]                                     Using:
  k.12b[0:99]                                     Using:
  k.22a[0:99]                                     Using:
Tao>
```

Index	d1_data_name	Merit type	Ref Element	Start Element	Element Name	Meas value	Model value
30	orbit.x	target			DET_30W	0.0000000000000E+00	-6.3994360848211E-03
31	orbit.x	target			DET_31W	0.0000000000000E+00	2.8540045854103E-03
32	orbit.x	target			DET_32W	0.0000000000000E+00	1.2658865835454E-02
33	orbit.x	target			DET_33W	0.0000000000000E+00	1.1905074152999E-02
34	orbit.x	target			DET_34W	0.0000000000000E+00	1.5417366933784E-02
35	orbit.x	target			DET_35W	0.0000000000000E+00	6.9528419764613E-03
36	orbit.x	target			DET_36W	0.0000000000000E+00	8.9760172249806E-05
37	orbit.x	target			DET_37W	0.0000000000000E+00	-5.9017764956826E-03
38	orbit.x	target			DET_38W	0.0000000000000E+00	-1.5499288722237E-02
39	orbit.x	target			DET_39W	0.0000000000000E+00	-8.8566698499384E-03

ele_name	DET_34W	(288)
ele_start_name		(-1)
ele_ref_name		(-1)
data_type	orbit	x
merit_type	target	
data_source	lat	
eval_point	End	
ix_bunch	0	
ix_branch	0	
ix_ele_merit	-1	
ix_d1	34	
ix_data	35	
ix_dmodel	-1	
meas_value	0.0	
ref_value	0.0	
model_value	0.015417366933784	
design_value	0.015417366933784	
old_value	0.0	

Figure 6.1: The GUI's data viewing windows. Top left: the Tao root window and the menu shortcut for viewing data. Top middle: The d2_data_array window, which list the currently defined d2_data_-arrays for each universe. This window also includes links to view and edit (see section 6.2) existing d1_data_arrays. Bottom left: The d1_data_array window (in this case for orbit.x), showing all of the datums in the array orbit.x. Bottom right: The individual datum window (in this case for orbit.x[34]) displaying detailed datum properties and allowing the user to edit some of these properties. Top right: The bulk edit window (in this case for orbit.x) providing controls to quickly edit a few key properties for multiple datums in a d1_data_array.



```
[INFO] tao_init:
   Note: global%rf_on = False  -->  RFCavities will be turned off in lattices

Tao>
```

New d2 Data

d2_data Name:	
Which universe:	All
Default data source:	lat
Default merit type:	target
Default weight:	
Default good_user:	✓
Clone existing d2_array:	None

Figure 6.2: Left: the data creation window can be accessed from the root window's menubar. Right: The first pane of the data creation window.

6.1 Viewing Data

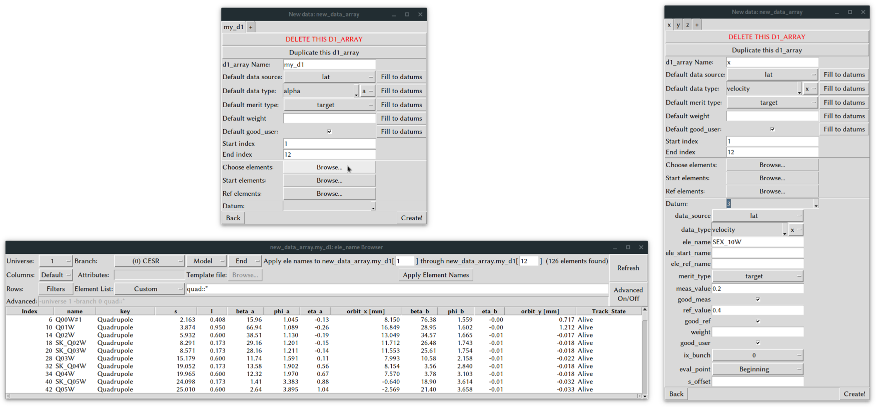
Figure 6.1 shows the various windows that the GUI provides for viewing data and making minor changes to data arrays. The d2_data_array window (top middle in Figure 6.1) displays all data arrays for a given universe. To view any existing d1_data_array, click on its "View" button. This window also provides the ability to edit any existing data array in detail (see Section 6.2), as well as functionality for writing existing data to a namelist (see Section 9.1).

The d1_data_array window (bottom left in Figure 6.1) allows the user to view an existing d1_data_-array. This window displays important properties of each datum in the array, such as element name, meas, model, and design values, and weight, in a scrollable table. To view a datum in detail, double click on its row in the d1_data_array window. This will open the individual datum window for that datum, displaying all of its properties and allowing some of them to be editted.

The d1_data_array window also allows the user to edit a few key properties of the datums in the array all at once using the bulk settings window (top right in Figure 6.1). This window is accessed by clicking on the "Bulk fill" button in the d1_data_array window. From here, the meas_value, good_user, and weight settings for the datums in the array can be edited in bulk. Changes may be applied to every datum in the array, or to only a specific range of datums using the range specifier. Once the desired settings have been specified, clicking the "Fill and apply" button will edit the d1_data_array as necessary, and changes will be reflected in the d1_data_array window.

6.2 Creating and Editing Data

Tao data structures can be defined via an initialization file or on-the-fly via the GUI. For setting up a data initialization file, see the `Tao Initialization` chapter in the *Tao* manual. To initialize data via the GUI, open the `New D2 Data` window



d1_array Name: my_d1
Default data source: lat
Default data type: alpha a
Default merit type: target
Default weight:
Default good_user: ✓
Start index 1
End index 12
Choose elements:
Start elements:
Ref elements:
Datum:

Universe: 1 Branch: (0) CESR Model End Apply ele names to new_data_array.my_d1[1] through new_data_array.my_d1[12] (126 elements found)
Columns: Default Attributes: Template file:
Rows: Element List: Custom quad::*
Advanced: -universe 1 -branch 0 quad::*

Index	name	key	s	l	beta_a	phi_a	eta_a	orbit_x [mm]	beta_b	phi_b	eta_b	orbit_y [mm]	Track_State
6	Q00W#1	Quadrupole	2.163	0.408	15.96	1.045	-0.13	8.150	76.38	1.559	-0.00	0.717	Alive
10	Q01W	Quadrupole	3.874	0.950	66.94	1.089	-0.26	16.849	28.95	1.602	-0.00	1.212	Alive
14	Q02W	Quadrupole	5.932	0.600	38.51	1.130	-0.19	13.049	34.57	1.665	-0.01	-0.017	Alive
18	SK_Q02W	Quadrupole	8.291	0.173	29.16	1.201	-0.15	11.712	26.48	1.743	-0.01	-0.018	Alive
20	SK_Q03W	Quadrupole	8.571	0.173	28.16	1.211	-0.14	11.553	25.61	1.754	-0.01	-0.018	Alive
28	Q03W	Quadrupole	15.179	0.600	11.74	1.591	0.11	7.993	10.58	2.158	-0.01	-0.022	Alive
32	SK_Q04W	Quadrupole	19.052	0.173	13.58	1.902	0.56	8.154	3.56	2.840	-0.01	-0.018	Alive
34	Q04W	Quadrupole	19.965	0.600	12.32	1.970	0.67	7.570	3.78	3.103	-0.01	-0.018	Alive
40	SK_Q05W	Quadrupole	24.098	0.173	1.41	3.383	0.88	-0.640	18.90	3.614	-0.01	-0.032	Alive
42	Q05W	Quadrupole	25.010	0.600	2.64	3.895	1.04	-2.569	21.40	3.658	-0.01	-0.033	Alive

d1_array Name: x
Default data source: lat
Default data type: velocity x
Default merit type: target
Default weight:
Default good_user: ✓
Start index 1
End index 12
Choose elements:
Start elements:
Ref elements:
Datum: 3
data_source lat
data_type velocity x
ele_name SEX_10W
ele_start_name
ele_ref_name
merit_type target
meas_value 0.2
good_meas ✓
ref_value 0.4
good_ref ✓
weight
good_user ✓
ix_bunch 0
eval_point Beginning
s_offset

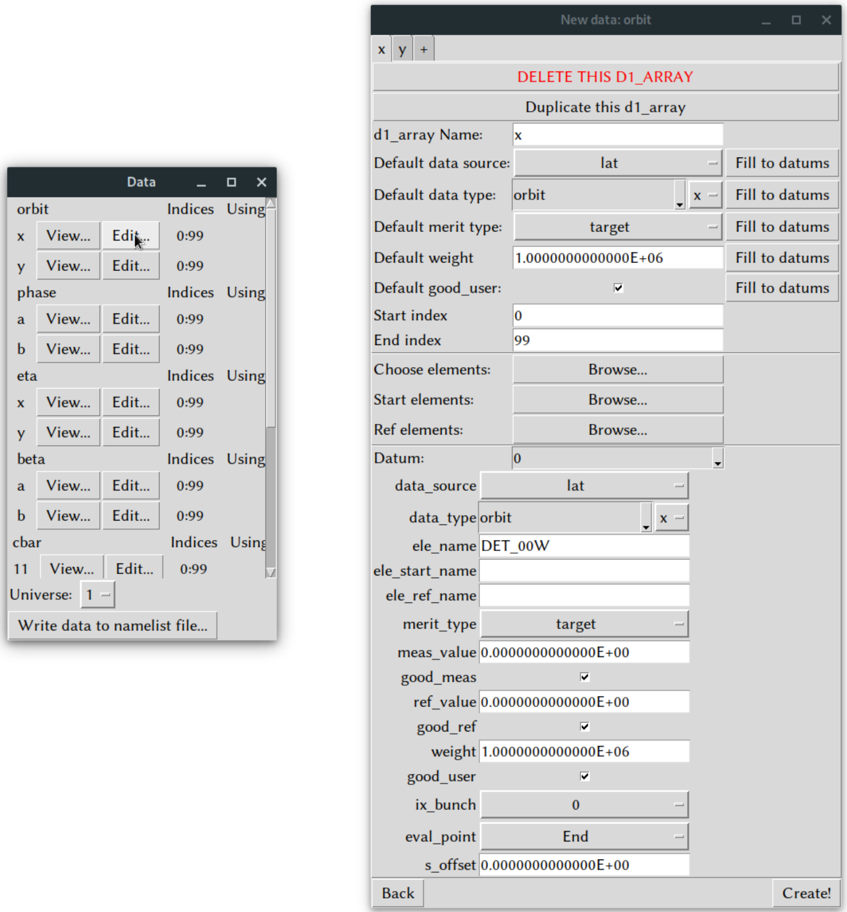
Figure 6.3: The d1_array pane of the data creation window. Top left: Here, only one d1_array has been created (called my_d1), its default data type has been set to alpha.a, and the start and end indices have been set to 1 and 12 respectively.
Bottom left: the lattice browser for the ele_names that will be used with my_d1. Right: Here, the user has defined three d1_arrays: x, y, and z. The data type for new_data_array.x has been set to velocity.x, and the start and end indices have been set to 1 and 12. Here, the user is currently editing new_data_array.x[3], where the meas value has been set to 0.2 and the ref value has been set to 0.4.

The GUI also supports the creation of data arrays on the fly through the create dat window. This window can be accessed as shown in figure 6.2. In the first pane of the data creation window, the user can input the desired settings for the new d2_data_array. The user can also select and existing d2_data_array to clone. This will copy the d2 properties of that array, as well as the d1 properties and all of the datums for each d1 array. Once this information has been input, the user can hit the "Next" button to go to the d1_data_array pane.

The d1_array pane of the data creation window is where most of the data array's properties are set. This pane is shown in Figure 6.3. The d1_array pane of the data creation window displays each d1_array in its own tab. To add a tab, click on the "+" tab at the top of the window. Tabs can also be removed by navigating to them and then clicking on their delete button. An existing tab can also be duplicated by clicking on the duplicate button right under the delete button. This may be useful if you want to define several d1_arrays with many of the same properties, but want them each to have a different data type, for example.

The next section of the window holds the d1-level settings for the array. Here, the d1 name, start index, and end index can be set, as well as the default data_source, data_type, merit type, weight, and good user value for the d1_array.

The next section allows the users to set the ele_name, ele_start_name, and ele_ref_name for the d1_array en-masse. Clicking on these buttons will bring up the lattice browser window (bottom left in Figure 6.3). This window is essentially identical to the main lattice window for the GUI (see Section ??), with a few additions. Towards the top right of the window, the user can specify which indices to read the element names into. Clicking "Apply Element Names" will then write the ele names that are currently in the table sequentially into the d1_array's datums. In the example shown in Figure 6.3, new_data_array.my_d1[1]|ele_name will be set to "Q00W#1", new_data_array.my_d1[2]|ele_name will be set to "Q01W", and so on. If there are more elements in the table than there are datums to write to, the table will be truncated and only the first elements in the table will be used. If there are less elements in the table than there are datums to write to, the elements in the table will be looped



orbit			Indices	Using
x	View...	Edit...	0:99	
y	View...	Edit...	0:99	
phase			Indices	Using
a	View...	Edit...	0:99	
b	View...	Edit...	0:99	
eta			Indices	Using
x	View...	Edit...	0:99	
y	View...	Edit...	0:99	
beta			Indices	Using
a	View...	Edit...	0:99	
b	View...	Edit...	0:99	
cbar			Indices	Using
11	View...	Edit...	0:99	

Universe: 1

d1_array Name:	x
Default data source:	lat
Default data type:	orbit x
Default merit type:	target
Default weight	1.0000000000000E+06
Default good_user:	✓
Start index	0
End index	99
Choose elements:	Browse...
Start elements:	Browse...
Ref elements:	Browse...
Datum:	0
data_source	lat
data_type	orbit x
ele_name	DET_00W
ele_start_name	
ele_ref_name	
merit_type	target
meas_value	0.0000000000000E+00
good_meas	✓
ref_value	0.0000000000000E+00
good_ref	✓
weight	1.0000000000000E+06
good_user	✓
ix_bunch	0
eval_point	End
s_offset	0.0000000000000E+00

Figure 6.4: Editting an existing d2_data_array.

through so that each datum gets an element name.

The bottom portion of the d1_array pane of the data creation window allows the user to set the properties of the individual datums in the array. Once a start and end index have been specified, the "Datum" drop down menu will be populated with all of the datum indices. Selecting an index will bring up the datum settings for that datum, as shown in the right of Figure 6.3. Note that any settings that have a d1-level default value are automatically filled in. Once the user edits a property of a datum, that property will no longer be auto-filled from the d1-level default settings, even if those default values are subsequently edited. If the user wants to explicitly fill a d1 setting to that d1_array's datums, they may do so with the corresponding "Fill to datums" button.

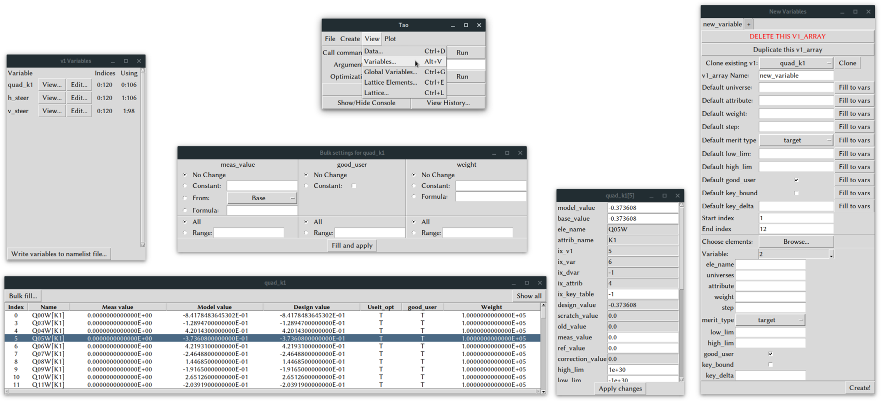
Once all of the data settings have been adjusted as necessary, the user must click the "Create" button to create the d2_array in Tao. Doing so will close the data creation window.

The data creation window can also be accessed from the d2_data window discussed in Section 6.1. Clicking on the "Edit" button for any d2 array will load that array into the data creation window, just as if the user had cloned that array from the d2 pane of the data creation window. This is shown in Figure 6.4 Note that any changes made in the data creation window will not take effect in Tao until the user clicks the "Create" button. For example, clicking the delete button for the orbit.x array would not actually delete the array in Tao until the user clicks "Create".

Chapter 7

Variables

Variables are viewed and edited in the GUI almost exactly the same as data, as shown in Figure 7.1. An important exception is that, since there are only v1 variables arrays and no v2 arrays of v1 arrays, the variable creation window only has one pane, and separate v1 arrays created in different tabs of the window are completely independent. Also, the user can clone an existing v1 array with the drop down menu below the "Duplicate" button.



Variable			Indices	Using
quad_k1	View...	Edit...	0:120	0:106
h_steer	View...	Edit...	0:120	1:106
v_steer	View...	Edit...	0:120	1:98

meas_value	good_user	weight
No Change	No Change	No Change
Constant:	Constant:	Constant:
From: Base		Formula:
Formula:		
All	All	All
Range:	Range:	Range:

Index	Name	Meas value	Model value	Design value	Useit_opt	good_user	Weight
0	Q00W[K1]	0.0000000000000E+00	-8.4178483645302E-01	-8.4178483645302E-01	T	T	1.0000000000000E+05
3	Q03W[K1]	0.0000000000000E+00	-1.2894700000000E-01	-1.2894700000000E-01	T	T	1.0000000000000E+05
4	Q04W[K1]	0.0000000000000E+00	4.2014300000000E-01	4.2014300000000E-01	T	T	1.0000000000000E+05
5	Q05W[K1]	0.0000000000000E+00	-3.7360800000000E-01	-3.7360800000000E-01	T	T	1.0000000000000E+05
6	Q06W[K1]	0.0000000000000E+00	4.2193100000000E-01	4.2193100000000E-01	T	T	1.0000000000000E+05
7	Q07W[K1]	0.0000000000000E+00	-2.4648800000000E-01	-2.4648800000000E-01	T	T	1.0000000000000E+05
8	Q08W[K1]	0.0000000000000E+00	1.4468500000000E-01	1.4468500000000E-01	T	T	1.0000000000000E+05
9	Q09W[K1]	0.0000000000000E+00	-1.9165000000000E-01	-1.9165000000000E-01	T	T	1.0000000000000E+05
10	Q10W[K1]	0.0000000000000E+00	2.6512600000000E-01	2.6512600000000E-01	T	T	1.0000000000000E+05
11	Q11W[K1]	0.0000000000000E+00	-2.0391900000000E-01	-2.0391900000000E-01	T	T	1.0000000000000E+05

model_value	-0.373608
base_value	-0.373608
ele_name	Q05W
attrib_name	K1
ix_v1	5
ix_var	6
ix_dvar	-1
ix_attrib	4
ix_key_table	-1
design_value	-0.373608
scratch_value	0.0
old_value	0.0
meas_value	0.0
ref_value	0.0
correction_value	0.0
high_lim	1e+30
low_lim	-1e+30

Clone existing v1:	quad_k1
v1_array Name:	new_variable
Default universe:	
Default attribute:	
Default weight:	
Default step:	
Default merit type	target
Default low_lim:	
Default high_lim	
Default good_user	✓
Default key_bound	☐
Default key_delta	
Start index	1
End index	12
Choose elements:	Browse...
Variable:	2
ele_name	
universes	
attribute	
weight	
step	
merit_type	target
low_lim	
high_lim	
good_user	✓
key_bound	☐
key_delta	

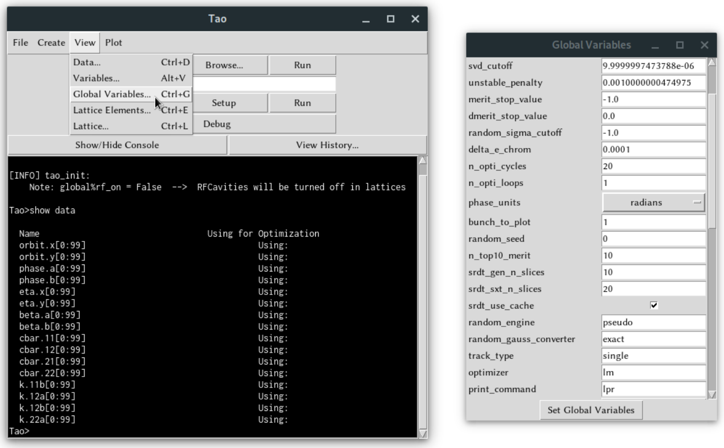
Figure 7.1: The variable viewing and editing windows in the GUI. Top middle: accessing the v1 variable window from the root window (console hidden). Top left: the v1 variable window, showing the currently defined v1 variable arrays. Bottom left: the variable window for quad_k1, displaying each of its variables and their key attributes. Middle: the bulk fill window for quad_k1. Bottom middle: the individual variable window for quad_k1[5], showing all of its settings and allowing the user to edit some of them. Right: The variable array creation window.

Chapter 8

Other Windows

8.1 Global Variable Window

Global variables in Tao can be viewed and modified from the global variables window as shown in Figure 8.1. Once you have editted the global variables, clicking the "Set Global Variables" button will set the variables in Tao as appropriate.



```
[INFO] tao_init:
    Note: global%rf_on = False  -->  RFCavities will be turned off in lattices

Tao>show data

  Name                                   Using for Optimization
  orbit.x[0:99]                                    Using:
  orbit.y[0:99]                                    Using:
  phase.a[0:99]                                    Using:
  phase.b[0:99]                                    Using:
  eta.x[0:99]                                      Using:
  eta.y[0:99]                                      Using:
  beta.a[0:99]                                     Using:
  beta.b[0:99]                                     Using:
  cbar.11[0:99]                                    Using:
  cbar.12[0:99]                                    Using:
  cbar.21[0:99]                                    Using:
  cbar.22[0:99]                                    Using:
  k.11b[0:99]                                      Using:
  k.12a[0:99]                                      Using:
  k.12b[0:99]                                      Using:
  k.22a[0:99]                                      Using:
Tao>
```

svd_cutoff	9.9999997473788e-06
unstable_penalty	0.0010000000474975
merit_stop_value	-1.0
dmerit_stop_value	0.0
random_sigma_cutoff	-1.0
delta_e_chrom	0.0001
n_opti_cycles	20
n_opti_loops	1
phase_units	radians
bunch_to_plot	1
random_seed	0
n_top10_merit	10
srdt_gen_n_slices	10
srdt_sxt_n_slices	20
srdt_use_cache	✓
random_engine	pseudo
random_gauss_converter	exact
track_type	single
optimizer	lm
print_command	lpr

Figure 8.1: View and edit global variables with the Global Parameters window.

Chapter 9

Miscelaneous

9.1 Writing Namelists

Chapter 10

GUI Development

10.1 GUI Development

10.1.1 Profiling the GUI

To profile the GUI, first create a `gui.init` file with:

```
skip_setup:T
do_mainloop:F
```

then use the python script:

```
from pytao import gui
import cProfile
cProfile.run('gui.tao_root_window()', 'profile.dat')
```