ISBN: 978-3-95450-231-8 ISSN: 2673-5490 doi: 10.18429/JACoW-IPAC2023-WEPA086

PYTAO: THE PYTHON INTERFACE TO TAO

C. E. Mayes, H. H. Slepicka, SLAC National Accelerator Laboratory, 94025 Menlo Park, CA, USA
D. C. Sagan, Cornell University, 14850 Ithaca, NY, USA

Abstract

PyTao is a Python interface to the Bmad based Tao program for accelerator design and simulation. This enables advanced design and optimization beyond the normal capabilities of Tao and simplifies the use of Tao as an online model for an operating accelerator. Here we will describe this interface and some of its applications, including online models for the LCLS and LCLS-II at SLAC National Accelerator Laboratory.

BMAD AND TAO

Tao [1] 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 [2] toolkit (software library) which provides the computational routines needed for accelerator physics calculations.

The basic Tao interface uses a command line for input. Output is text printed on the command line window and/or plots displayed on a separate plot window. Besides this plot window, Tao provides limited scripting capabilities, such as allowing text files with commands to be issued in sequence.

Tao implements a number of commands to do such things as setting lattice parameters, manipulating plots, etc. For displaying information, the main command is the "`show`" command. The `show` command displays information in a compact human-readable format. The `show` command is complicated by the fact that there is a large amount of information to be communicated — anything from lattice parameter values, radiation integrals, Twiss parameters, optimization settings, transfer maps, plot settings, etc. The `show` command has over 40 categories (sub-commands) of information to show and most sub-commands have a number of options that can be used to modify how and what information is displayed.

Using the output of the `show` command to interface with a scripting language such as Python is problematic because, in order to maximize human readability, there is no standard format for displaying information for the various `show` sub-commands. This means that separate code must be developed on the Python side[1] for each sub-command and for the different options that each sub-command implements. This is a huge endeavor, and the problem is made worse by the fact that as Tao evolves to meet changing simulation needs, the `show` command output will evolve, requiring the attendant maintenance workload for the interface code.

In order to simplify communication with Python, Tao provides a special "`python`" command to return structured text that is suitable for parsing by another program. For example, the command

```
Tao> python ele:head end
```

will give the output

```
universe;INT;F;1
1^ix_branch;INUM;F;0
ix_ele;INT;I;869
key;ENUM;F;Marker
...
```

The information here is returned in a semicolon-delimited list which is easily parsed. Each line gives information for a single parameter: name, type (e.g., integer, real number), settability (whether this parameter may be varied by the user), and value. While this was developed for a Python interface, any language could be used to parse this output. Tao's `python` command has over 100 sub-commands to display different types of information, and most sub-commands return information using the four-field, semicolon-delimited list as discussed above. Differences occur with vector information and for sub-commands that are used to set Tao parameters. However, the number of different formats is kept to a minimum to limit interface maintenance on the Python side. Furthermore, the syntax of Tao's `python` command is structured to minimize breakage of existing scripts when things change. For example, if a change in Tao necessitates adding parameters to the output of the `ele:head` sub-command as shown above, a script that accesses, say, the value of the existing `ix_ele` parameter will not be affected by the change.

Python can communicate with Tao by running the Tao executable within Python and then using Python's `pexpect` [3] module to communicate through the virtual command line. Alternatively, Tao can be compiled as a shared library object that Python can link to at run time. Communication is then through Python's `ctypes` module [4]. To further speed up communication when there is a large amount of data to be transferred, for selected Tao `python` sub-commands, the string output can be bypassed and array data stored in a temporary array that can be accessed by Python using C functions provided by Tao. For example,

```
Tao> python lat_list -array_out * ele.s
```

writes s-position data for all elements into this array.

PYTAO

PyTao is an open-source Python package that allow users to control Tao from Python [5]. It does this using the functionality previously described by loading the compiled Tao library via Python's `ctypes` module.

[1] Henceforth, it is assumed that the scripting language is Python, but the discussion is applicable to any scripting language.

ISBN: 978-3-95450-231-8 ISSN: 2673-5490 doi: 10.18429/JACoW-IPAC2023-WEPA086

```
tao -init tao.init

Tao> set element  Q00W k1 = -0.85

Tao> show element end
Element # 869
Element Name: END
Key: Marker
...
```

(a) Tao

```
python
>>> from pytao import Tao
>>> tao = Tao("-init tao.init")

>>> tao.cmd("set element  Q00W k1 = -0.85")

>>> tao.ele_head("end")
{"universe": 1, "1^ix_branch": 0,
 "ix_ele": 869,
 "key": "Marker",
 ...
```

(b) PyTao

Figure 1: a) Example of accessing Tao through the basic command line. b) Example of accessing Tao via PyTao and the Python command line. Any initialization options for the `tao` program can be used to initialize the `Tao` Python object. This object also has special methods such as `ele_head` to extract information as Python data types. There are approximately 100 such methods.

PyTao's primary capability is provided by the "`Tao`" Python object which is instantiated using the same arguments as used with the basic command line interface. Figure 1 shows basic examples of Tao and the equivalent PyTao syntax for instantiating a session, issuing an arbitrary command, and returning data.

The Python "`Tao`" object hosts approximately 100 methods for exchanging data with Tao. Anything that a user would normally type at a `Tao>` prompt can be sent as a string through the `Tao.cmd` method, with a primary return as a list of strings. Other methods return custom types. For example,

```
>>>tao.matrix("beginning", "end")
{"mat6": array([[-9.85453321e-01,
                 ..., ]])},
"vec0": array([ 3.92874185e-04,
                ... ]) }
```

returns the analyzed linear transfer matrix and vector components from two elements in the lattice as a `dict` of NumPy arrays [6].

JUPYTER INTEGRATION

An alternative way to interact with Tao is via a Jupyter notebook [7] using the `%tao` magic command. This is automatically registered to the existing Jupyter notebook session when a `Tao` object is created. Multiple lines can be executed in the same block. Figure 2 shows how this is used.

OPENPMD-BEAMPHYSICS INTEGRATION

`OpenPMD` [8] is a standard that has been developed in cooperation with a number of laboratories for facilitating exchange of particle- and mesh-based data. Bmad implements openPMD to create files for the exchange of particle beam data. On the Python side, the openPMD-beamphysics Python package [9] can be used for bunch analysis, plotting, and data conversion. Although PyTao does not depend on this package, it does provide a method for easily extracting complete particle coordinates in a standard format. This is illustrated in Fig. 3.

EXAMPLE USAGE

PyTao is used in a variety of situations, including

- advanced scripting,
- dataset generation for machine learning,
- graphical user interfaces,
- external optimization,
- online modeling.

Figure 4 shows an advanced scripting example from the PyTao documentation for exploring asymmetrical quadrupole strengths in a FODO lattice. Custom plots, such as the one in the figure, are easily generated using standard plotting tools such as Matplotlib [10]. Further tasks, such

```
[1]: from pytao import Tao

[2]: tao = Tao('-init tao.init')

[3]: %%tao
     show element end

     ------------------------
     Tao> show element end
     Element # 869
     Element Name: END
     Key: Marker
     S_start, S:    768.426421,    768.426421
```

Figure 2: PyTao usage in a Jupyter notebook with the `%tao` magic command. This allows the user to use the more efficient native Tao command syntax and human-readable printing within a notebook environment.

ISBN: 978-3-95450-231-8 ISSN: 2673-5490 doi: 10.18429/JACoW-IPAC2023-WEPA086


```
from pmd_beamphysics import ParticleGroup

P = ParticleGroup(data=tao.bunch_data('end'))

P.plot('delta_t', 'delta_energy')
```

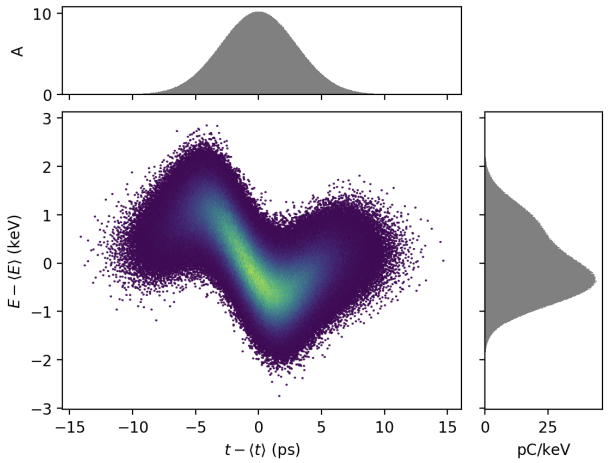

Figure 3: openPMD-beamphysics integration [9].

as tracking particles, can be easily done in the same Python process.

One motivation for PyTao development was to support a GUI for Tao using standard Python tools. Although this is not currently developed, many of the components needed are available in the code and could be utilized in the future.

For machine learning, the PyTao machinery enabled Edelen et al. [11] to make large datasets of longitudinal phase space projections by tracking particles through a Bmad model of the LCLS with various compression settings. These datasets were used to train artificial neural network surrogate models for use in online tuning.

Tao includes several native numerical optimizers. However, for slow-running calculations or more advanced optimization, it is useful to use external optimization packages such as Xopt [12]. Gulliford et al. [13] show an example of PyTao being used to optimize the Electron Ion Collider (EIC) electron cooler using one of Xopt's CNSGA algorithms.

Another PyTao usage is with the LCLS-Live [14] environment to model the LCLS and LCLS-II machines online. PyTao is used extensively in this case. In the same Python process, this is done by 1) instantiating design Bmad models; 2) acquiring live values for relevant process variables using PyEPICS [15]; 3) translating them to simulation settings and applying to the model; and 4) serving the data over EPICS [16]. Figure 5 shows bunch tracking statistics of a live model of LCLS-II during commissioning at the end of 2022.

CONCLUSION

PyTao is a Python interface to the Tao accelerator simulation program. By integrating the capabilities of Python with Tao, PyTao enables advanced design and optimization beyond the normal capabilities of Tao and simplifies the use of Tao as an online model for an operating accelerator.

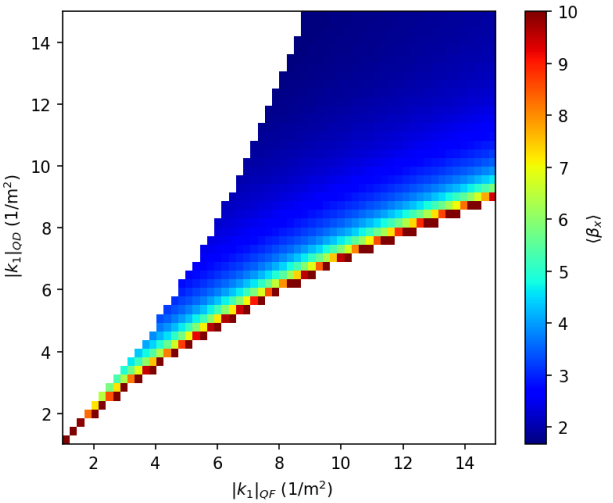

Figure 4: Example from the PyTao documentation to compute the average horizontal beta function as a function of quadrupole strengths k_1 for an asymmetric FODO lattice with elements named QF and QD.

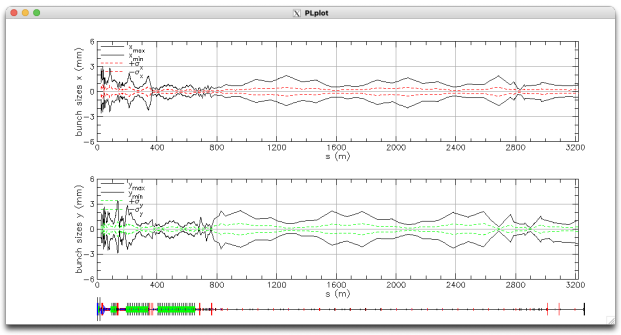

Figure 5: Screenshot of particle statistics from live particle tracking of the LCLS-II `sc_bsyd` beampath using PyTao. In a continuous loop, values from several thousand process variables are acquired from the EPICS control system and translated into simulation settings using LCLS-Live [14]. 10,000 particles are tracked, and summary statistics are plotted using Tao's native plotting window. During commissioning this is updated at about 1 Hz.

ACKNOWLEDGEMENTS

This work was supported by the U.S. Department of Energy, under DOE Contract No. DE-AC02-76SF00515 and the Office of Science, Office of Basic Energy Sciences. In addition, this work was supported by Department of Energy grant DE-SC0018370 and National Science Foundation award DMR-1829070.

REFERENCES

[1] D. Sagan and J. Smith, "The Tao Accelerator Simulation Program," *PAC05 Particle Acc. Phys. Conf., Knoxville TN, 2005*, vol. 0505161, p. 4159, 2005.

[2] D. Sagan, "Bmad: A relativistic charged particle simulation library," *Nucl. Instrum. Methods Phys. Res., Sect. A*, vol. 558,

ISBN: 978-3-95450-231-8 ISSN: 2673-5490 doi: 10.18429/JACoW-IPAC2023-WEPA086

no. 1, pp. 356–359, 2006. DOI: 10.1016/j.nima.2005.11.001.

[3] N. Spurrier and the Pexpect development team, *Pexpect: A pure python expect-like module*, 2023. https://github.com/pexpect/pexpect

[4] Python Software Foundation, *Ctypes — a foreign function library for python*, 2021. https://docs.python.org/3/library/ctypes.html

[5] C. Mayes, D. Sagan, and H. Slepicka, *PyTao*, 2023. https://bmad-sim.github.io/pytao/

[6] C. R. Harris *et al.*, "Array programming with NumPy," *Nature*, vol. 585, no. 7825, pp. 357–362, Sep. 2020. DOI: 10.1038/s41586-020-2649-2.

[7] *Project Jupyter*, 2023. https://jupyter.org/

[8] A. Huebl *et al.*, *openPMD: A meta data standard for particle and mesh based data*, https://github.com/openPMD, 2015. DOI: 10.5281/zenodo.591699. https://www.openPMD.org

[9] C. Mayes, C. Gulliford, W. Lou, F. Ji, and J. Duris, *ChristopherMayes/openPMD-beamphysics: openPMD-beamphysics v0.7.5*, version v0.7.5, Apr. 2023. DOI: 10.5281/zenodo.7823796. https://github.com/christophermayes/openPMD-beamphysics/

[10] J. D. Hunter, "Matplotlib: A 2d graphics environment," *Computing in Science & Engineering*, vol. 9, no. 3, pp. 90–95, 2007. DOI: 10.1109/MCSE.2007.55.

[11] A. Edelen, N. Neveu, D. Ratner, C. Emma, and C. Mayes, "Machine learning models for optimization and control of x-ray free electron lasers," in *The Machine Learning and the Physical Sciences 2019 Workshop at the 33rd Annual Conference on Neural Information Processing Systems (NeurIPS 2019)*, Vancouver, Canada, 2019. https://ml4physicalsciences.github.io/2019/files/NeurIPS_ML4PS_2019_90.pdf

[12] C. Mayes *et al.*, *Christophermayes/xopt: Xopt v1.4.0*, version v1.4.0, May 2023. DOI: 10.5281/zenodo.7894002. https://github.com/christophermayes/xopt

[13] C. Gulliford *et al.*, "Design and optimization of an erl for cooling eic hadron beams," in *Proceedings of the 23rd International Particle Accelerator Conference (IPAC23)*, Venice, Italy, 2023, to appear in MOPA016.

[14] C. Mayes, M. Gibbs, J. Garrahan, and N. Neveu, *LCLS-Live*, 2022. https://slaclab.github.io/lcls-live/

[15] M. Newville *et al.*, *Pyepics/pyepics:* version 3.5.1, Feb. 2022. DOI: 10.5281/zenodo.5949248. https://doi.org/10.5281/zenodo.5949248

[16] L. R. Dalesio *et al.*, "The experimental physics and industrial control system architecture: Past, present, and future," *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, vol. 352, no. 1, pp. 179–184, 1994, ISSN: 0168-9002. DOI: https://doi.org/10.1016/0168-9002(94)91493-1. https://www.sciencedirect.com/science/article/pii/0168900294914931