ISBN: 978-3-95450-231-8 ISSN: 2673-5490 doi: 10.18429/JACoW-IPAC2023-WEPL127

PROGRESS ON THOR SCSI DEVELOPMENT

P. Schnizer[*], W. Sulaiman Khail, J. Bengtsson, M. Ries
Helmholtz-Zentrum Berlin, BESSY, Berlin, Germany
L. Deniau, CERN, Genève, Switzerland

Abstract

Tracy, the code base used for designing synchrotron light sources with predictable performance, has been significantly refactored. Furthermore it now uses MAD-NG's GTPSA library.

We describe the achieved progress, discuss its python interface. We show how to use it for achieving a robust design for a modern synchrotron light source.

INTRODUCTION

The Helmholtz Zentrum Berlin is looking into modernising and upgrading its own synchrotron light source BESSY II next to the one it operates on behalf of PTB, the light source MLS. These will now transit to a 4th and 3rd generation light source. In parallel the existing light sources BESSY II and MLS are constantly upgraded. These upgrades require proper tools for modelling these accelerators for predictable results.

Influenced by the design choices of MAD-NG [1, 2] Tracy 2 [3, 4] is being revamped in to a modern code base called Thor Scsi. This code base is building on tools made available by others. We report on the progress made and on design choice along the line. Furthermore it profits from the scientific working environment provided the python world.

GTPSA

Many accelerator design tasks require the study of the influence of different parameters on the performance of a particular beam. The Truncated Power Series Algebra simplifies this task [5]. For the Next Generation version of the MAD series, the truncated power series have been tailored and implemented in a new way in order to speed up the code and make appropriate choices by splitting variables and parameters [6].

This power series objects implement methods for basic arithmetic operations. Given that C++ provides operator overloading many physics equations can be implemented in a way that it is readable to physicists. Using C++ templates the same code can be used for efficient single particle tracking using C++ primitives or parameter dependence using truncated power series objects as base ones.

Wrappers for C++ and python

The GTPSA library is implemented in C [6]. For Thor Scsi a shallow C++ wrapper was created, which uses C++'s smart pointers (i.e. std::shared_ptr and std::unique_ptr). The wrapper then uses the allocation and deletion functions provided by GTPSA.

[*] pierre.schnizer@helmholtz-berlin.de

A shallow python wrapper was made using [7], which allows using tpsa (real) and ctpsa (complex) objects as any other python objects. A considerable subset of GTPSA functionalities is provided in its current implementation.

SUPPORTING PARAMETER STUDIES

Thor, a version of Tracy 2, already allowed studying the influence of different parameters using TPSA objects. The parameter to be studied had to be set at compile time.

The development of last year made Thor Scsi a python library. All beam elements are implemented in C++. GTPSA allows defining an ample amount of parameters to study (on one of the authors laptop up to 300 parameters). A straightforward solution would be to make all parameters TPSA objects. The solution chosen here was to wrap gtpsa::tpsa and double objects in a std::variant. Then simple wrappers were provided that so that the different accelerator Lego building blocks can use efficient basis types of tpsa objects at the user selection.

Using pybind11 a python interface was developed which allows now using a parameter as a double or gtpsa object. In it current form the user needs to instantiated this wrapper if required. In a next step it will be modified that the wrapper will make conversions on the users behalf.

The different types need to be wrapped in a common type as C++ is statically typed. The work required to implement and handle such a type show once more that a language using dynamic typing make implementing such use cases much more straight forward. The user will either use a basic value or a GTPSA object to his preference and the rest will be sorted out by dynamic typing.

A Just in Time (JIT) engine will provide the flexibility of a dynamic language parred with a fast evaluation speed. Given LuaJIT [8], a fast JIT for Lua embedded and extended in MAD-NG [9], will make this flexibility overhead nearly negligible.

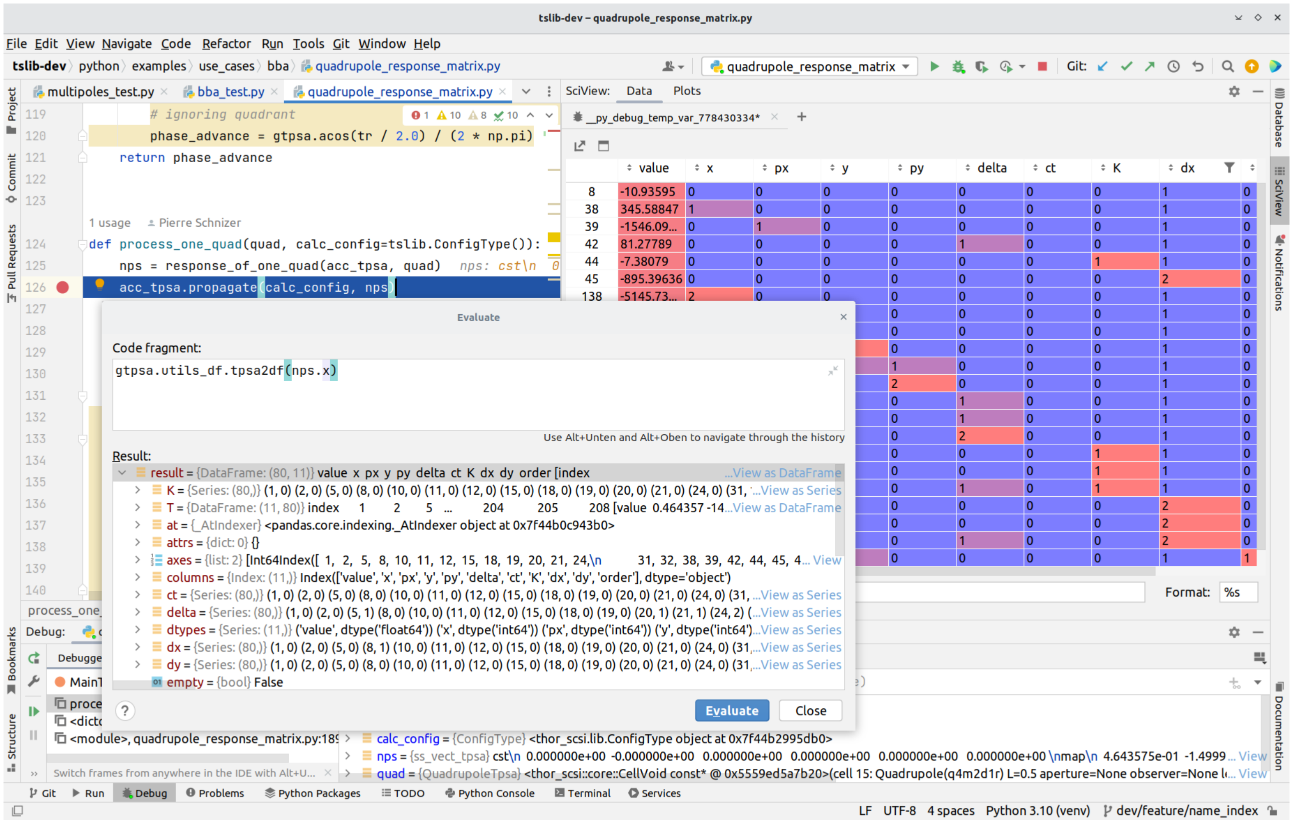
GTPSA: PYTHON USER INTERFACE

Python packages as pandas dataframes [10, 11] or xarray [12] simplify the access and make tables or array data elements easier to access for humans. xarray, targeting making multidimensional arrays easier to implement and access went further and allowed naming the dimensions.

The python wrapper of the GTPSA object next to the state space object used for tracking, now uses an "IndexMapping" object.

```
d = dict(x=0, px=1, y=2, py=3, delta=4, ct=5,
         K=6, dx=7, dy=8)
named_index = gtpsa.IndexMapping(d)
```

ISBN: 978-3-95450-231-8 ISSN: 2673-5490 doi: 10.18429/JACoW-IPAC2023-WEPL127


tslib-dev – quadrupole_response_matrix.py
File Edit View Navigate Code Refactor Run Tools Git Window Help
tslib-dev 〉python 〉examples 〉use_cases 〉bba 〉quadrupole_response_matrix.py
multipoles_test.py
bba_test.py
quadrupole_response_matrix.py
quadrupole_response_matrix
Git:
SciView: Data Plots
__py_debug_temp_var_778430334*
Project
Commit
Pull Requests
Bookmarks
Structure
Database
SciView
Notifications
Documentation
1 usage
Pierre Schnizer
Evaluate
Code fragment:
Use Alt+Unten and Alt+Oben to navigate through the history
Result:
Evaluate
Close
Format: %s
process_one_
Debug:
Debugger
MainT
proce
<dictc
Switch frames from anywhere in the IDE with Alt+U...
Git Run Debug Problems Python Packages TODO Python Console Terminal Services
LF UTF-8 4 spaces Python 3.10 (venv) dev/feature/name_index

```
        # ignoring quadrant
        phase_advance = gtpsa.acos(tr / 2.0) / (2 * np.pi)
    return phase_advance


def process_one_quad(quad, calc_config=tslib.ConfigType()):
    nps = response_of_one_quad(acc_tpsa, quad)   nps: cst\n 0
    acc_tpsa.propagate(calc_config, nps)
```

	value	x	px	y	py	delta	ct	K	dx	
8	-10.93595	0	0	0	0	0	0	0	1	0
38	345.58847	1	0	0	0	0	0	0	1	0
39	-1546.09...	0	1	0	0	0	0	0	1	0
42	81.27789	0	0	0	0	1	0	0	1	0
44	-7.38079	0	0	0	0	0	0	1	1	0
45	-895.39636	0	0	0	0	0	0	0	2	0
138	-5145.73...	2	0	0	0	0	0	0	1	0
					0	0	0	0	1	0
					0	0	0	0	1	0
					0	0	0	0	1	0
					1	0	0	0	1	0
					2	0	0	0	1	0
					0	1	0	0	1	0
					0	1	0	0	1	0
					0	2	0	0	1	0
					0	0	0	1	1	0
					0	0	0	1	1	0
					0	1	0	1	1	0
					0	0	0	0	2	0
					0	0	0	0	2	0
					0	1	0	0	2	0
					0	0	0	0	1	1

```
gtpsa.utils_df.tpsa2df(nps.x)
```

```
result = {DataFrame: (80, 11)} value x px y py delta ct K dx dy order [index   ...View as DataFrame
  K = {Series: (80,)} (1, 0) (2, 0) (5, 0) (8, 0) (10, 0) (11, 0) (12, 0) (15, 0) (18, 0) (19, 0) (20, 0) (21, 0) (24, 0) (31, '...View as Series
  T = {DataFrame: (11, 80)} index    1     2     5  ...     204     205     208 [value  0.464357 -14...View as DataFrame
  at = {_AtIndexer} <pandas.core.indexing._AtIndexer object at 0x7f44b0c943b0>
  attrs = {dict: 0} {}
  axes = {list: 2} [Int64Index([ 1,  2,  5,  8, 10, 11, 12, 15, 18, 19, 20, 21, 24,\n        31, 32, 38, 39, 42, 44, 45, 4... View
  columns = {Index: (11,)} Index(['value', 'x', 'px', 'y', 'py', 'delta', 'ct', 'K', 'dx', 'dy', 'order'], dtype='object')
  ct = {Series: (80,)} (1, 0) (2, 0) (5, 0) (8, 0) (10, 0) (11, 0) (12, 0) (15, 0) (18, 0) (19, 0) (20, 0) (21, 0) (24, 0) (31, ...View as Series
  delta = {Series: (80,)} (1, 0) (2, 0) (5, 1) (8, 0) (10, 0) (11, 0) (12, 0) (15, 0) (18, 0) (19, 0) (20, 1) (21, 1) (24, 2) (...View as Series
  dtypes = {Series: (11,)} ('value', dtype('float64')) ('x', dtype('int64')) ('px', dtype('int64')) ('y', dtype('int64')...View as Series
  dx = {Series: (80,)} (1, 0) (2, 0) (5, 0) (8, 1) (10, 0) (11, 0) (12, 0) (15, 0) (18, 0) (19, 0) (20, 0) (21, 0) (24, 0) (31,...View as Series
  dy = {Series: (80,)} (1, 0) (2, 0) (5, 0) (8, 0) (10, 0) (11, 0) (12, 0) (15, 0) (18, 0) (19, 0) (20, 0) (21, 0) (24, 0) (31,...View as Series
  empty = {bool} False
```

```
<module>, quadrupole_response_matrix.py:18
  calc_config = {ConfigType} <thor_scsi.lib.ConfigType object at 0x7f44b2995db0>
  nps = {ss_vect_tpsa} cst\n 0.000000e+00 -0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 \nmap\n 4.643575e-01 -1.4999... View
  quad = {QuadrupoleTpsa} <thor_scsi::core::CellVoid const* @ 0x5559ed5a7b20>(cell 15: Quadrupole(q4m2d1r) L=0.5 aperture=None observer=None l... View
```

Figure 1: Phase space inspection within a modern programming environment. The code is stopped by the debugger. The phase space x component (note access by name) is exported to a pandas dataframe. The IDE supports the user inspecting this dataframe.

Here you can see that the first 6 dimensions follow the standard naming used for single particle tracking, which allows supports 2.5 dimensional studies. The “IndexMapping” object checks internally if this standard selection is followed, unless the user sets an optional flag to override this behaviour. One can see that here further indices are available for “K”, “d_x” and “d_y”. This could be names chosen for estimating expected orbit distortions as found during a beam based alignment session.

GTPSA provides methods that instrument its objects as required for parametric studies. The following code would set the gtpsa variable δ to 10^{-3}

```
delta.set_variable(1e-3, "delta") .
```

Furthermore it will set the coefficient of the first order derivative in δ to 1. Similar functionality is provided for knobs e.g.

```
dx.set_knob(1e-3, "dx") .
```

GTPSA provides a clean straightforward interface to access all coefficients. The python gtpsa packages provides now utilities to convert these to numpy record arrays and then in turn into Pandas dataframe[1].

[1] The separation was a deliberate design choice so that gtpsa does only depend on numpy. Support for the conversion to a pandas dataframe is provided as an option.

Modern integrated development environments (IDE), e.g. PyCharm’s professional offering then allow inspection of the coefficients of one of the phase space components selecting filters on the different coefficients (see Fig. 1). Indexing with numbers will provide the same results. Indexing by names can be seen as more intuitive and thus less error prone.

USE CASE EXAMPLES: BBA

Beam based alignment is a standard procedure at synchrotron light sources to measure the position of the quadrupoles versus the beam orbits. This is typically done by measuring the closed beam orbit at slightly different quadrupole strength. The gradient change can be typically neglected. The correlation of the orbit change to the gradient allows estimating the “feed down” dipole of the quadrupole due the misalignment of the quadrupole with respect to the reference orbit.

Different to the original method [13], at BESSY II the machine model is used to estimate the orbit deviation due to the gradient change. This orbit deviation requires checking that the quadrupole gradient change applied on the machine is the same as in the model. The tune of the BESSY II machine can be precisely measured using dedicated hardware . The change in phase advance can be simply calculated by instrumenting each quadrupole in turn by using:

ISBN: 978-3-95450-231-8 ISSN: 2673-5490 doi: 10.18429/JACoW-IPAC2023-WEPL127

```
k_org = magnet.get_main_multipole_strength()
k = gtpsa.ctpsa(desc, po,
                mapping=named_index)
k.set_knob(k_org, "K")
muls = magnet.get_multipoles()
muls.set_multipole(2,
                   gtpsa.CTpsaOrComplex(k))
```

Then the phase space object is traced trough the machine. At the end the phase advance dependence on K is a simple lookup of using

```
dq_dK = ps.ct.get(K=1)
```

The variable dq_dK now contains the value of $\partial Q/\partial K$ with Q the Floquet space coordinate and $K = G/(B\rho)$, with G the quadrupole gradient, B the dipole field and ρ the beam curvature within the dipoles. This data is evaluated for every different quadrupole and can be compared to. This allows checking if the model used is consistent with the machine. In case of BESSY II a dedicated power converter is connected to a dedicated winding on the quadrupoles. In this manner it can be checked if no polarity error was introduced in one of the windings. The required computation for all 144 quadrupoles requires less than a minute using a single core on a standard laptop CPU of today (Intel Core I7).

As the chosen beam based alignment is model based one more check was introduced. One quadrupole was moved in the model by 0.1 mm. The quadrupole's feed-down effect was compensated adding a dipole component of opposite sign. Orbit tracking showed that the orbit was zero all along the ring. Now the strength of this very quadrupole was changed by 1 % and following calculations were made:

- the expected orbit distortion was estimated using Courant-Snyder's equation describing the closed orbit for a single distorted quadrupole
- the fixed point was searched by tracking a phase space of doubles and recorded for the positions of the beam position monitors.
- the effect of the displaced quadrupole was estimated instrumenting its position in x and y as knobs.

The results are depicted in Fig. 2. One can see that the lines agree pretty much to each other.

Beam based alignment procedures are standard at many facilities; this approach here shows a method, how a model based one can be validated in a pretty straightforward manner. Given its simplicity it can be a good choice for a unit test.

EXTENSION OF USE CASES

Since long different parameters of the machine model were used as knobs. Thanks to GTPSA's implementation many parameters can be used: the coefficient is selected or set by the powers associated to it [6]. Different gtpsa objects are combined into a state space for tracking, typically representing the different canonical variables.

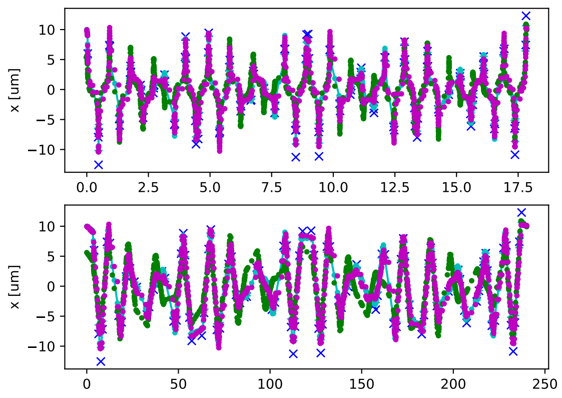


Figure 2: Orbit distortion due to quadrupole displacement calculated using parameter dependence or finding the fixed point and tracking it through the ring.

Beam based alignment data can be retrieved at BESSY II as for many standard light sources. The description above allows estimating the contribution of the linear dependency on the position. Subtracting the estimation of this effect allows studying the influence of higher oder terms.

Similar reasoning applies to other challenges of light source optimisation: e.g. minimising the impact of sextupole response, LOCO, orbit perturbations or conducting a robustness analysis.

Results of these studies on the BESSY II machine will be published elsewhere.

CONCLUSION

Thor Scsi developments targeted to use GTPSA library instead of Tracy 2's TPSA libray. As Thor Scsiis implemented in C++ a shallow wrapper implemented for gtpsa. Based on this wrapper a python interface was developed. This gives access to a considerable subset of GTPSA's library.

Parameter dependence study support was added to Thor scsi and made accessible via the python interface. It now allows turning many parameters of the beam dynamics elements into user knobs and study their impact on the beam. It was first used on testing and evaluating the beam based alignment procedure. Further use cases are studying optics perturbation and for engineering tolerances and robustness of the design.

REFERENCES

[1] L. Deniau, *MAD-NG's Reference Manual*. https://cern.ch/mad/releases/madng/html/

[2] L. Deniau, *MAD-NG Source Repository*. https://github.com/MethodicalAcceleratorDesign/MAD

[3] J. Bengtsson, "The sextupole scheme for the swiss light source," Paul Scherrer Institute, Tech. Rep., 1997. https://ados.web.psi.ch/slsnotes/sls0997.pdf

[4] J. Bengtsson, W. Rogers, and T. Nicholls, *A CAD tool for linear optics design: A controls engineer's geometric approach to hill's equation*, 2021. doi:10.48550/ARXIV.2109.15066

[5] M. Berz, "The method of power series tracking for the mathematical description of beam dynamics," *Nucl. Instrum. Methods Phys. Res., Sect. A*, vol. 258, no. 3, pp. 431–436, 1987. doi:10.1016/0168-9002(87)90927-2

[6] L. Deniau and C. I. Tomoiaga, "Generalised Truncated Power Series Algebra for Fast Particle Accelerator Transport Maps," in *Proc. IPAC'15*, Richmond, VA, USA, May 2015, pp. 374–377. doi:10.18429/JACoW-IPAC2015-MOPJE039

[7] J. Wenzel, J. Rhinelander, and D. Moldovan, *Pybind11 – seamless operability between C++11 and python*, https://github.com/pybind/pybind11, 2017.

[8] M. Pall, *The LuaJIT Project*. https://luajit.org

[9] D. D'Andrea, "Behavioural Analysis of Tracing JIT Compiler Embedded in the Methodical Accelerator Design Software," Presented 2019, 2019. https://cds.cern.ch/record/2692915

[10] The pandas development team, *Pandas-dev/pandas: Pandas*, version latest, 2020. doi:10.5281/zenodo.3509134

[11] W. McKinney, "Data Structures for Statistical Computing in Python," in *Proceedings of the 9th Python in Science Conference*, 2010, pp. 56–61. doi:10.25080/Majora-92bf1922-00a

[12] S. Hoyer and J. Hamman, "Xarray: N-D labeled arrays and datasets in Python," *J. Open Res. Software*, vol. 5, no. 1, 2017. doi:10.5334/jors.148

[13] G. Portmann, D. Robin, and L. Schachinger, "Automated Beam Based Alignment of the ALS Quadrupoles," in *Proc. PAC'95*, Dallas, TX, USA, May 1995, pp. 2693–2695. doi:10.1109/PAC.1995.505662