




1 cppTPSA/pyTPSA: a C++/Python package for 2 truncated power series algebra

3 He Zhang  ^{1¶}

4 ¹ Thomas Jefferson National Accelerator Facility, Newport News, VA 23606, USA ¶ Corresponding
5 author

DOI: [10.xxxxxx/draft](https://doi.org/10.xxxxxx/draft)

Software

- [Review](#) 
- [Repository](#) 
- [Archive](#) 

Editor: [Daniel S. Katz](#) 

Reviewers:

- [@CFGrote](#)
- [@mbkumar](#)

Submitted: 27 August 2021

Published: unpublished

License

Authors of papers retain copyright
and release the work under a
Creative Commons Attribution 4.0
International License ([CC BY 4.0](#)).

6 Summary

7 The truncated power series algebra (TPSA), also referred to as differential algebra (DA), is a
8 well-established and widely used method in particle accelerator physics and astronomy. The
9 most straightforward usage of TPSA/DA is to calculate the Taylor expansion of a given function
10 at a specific point up to order n , based on which more sophisticated methods have been
11 developed, e.g. symplectic tracking ([Berz, 1991a](#)), normal form analysis ([Berz, 1991b](#)), verified
12 integration ([Berz & Makino, 1998](#)), global optimization ([Makino & Berz, 2003](#)), fast multipole
13 method for pairwise interactions between particles ([Zhang & Berz, 2011](#)) etc. The cppTPSA
14 package implements the TPSA/DA in C++11 and provides the developers a convenient library
15 to build the advanced TPSA/DA-based method ([Zhang, 2021a](#)). A Python 3 library, pyTPSA,
16 has also been developed based on the C++ lib and is available in a separate GitHub repository
17 ([Zhang, 2021b](#)).

18 Background

19 In the following, we give a very brief introduction on TPSA/DA from a practical perspective
20 of computation. Please refer to ([Berz, 1999](#)) and ([Chao, 2002](#)) for the complete theory with
21 more details.

22 The fundamental concept in DA is the DA vector. To make the concept easier to understand,
23 we can take a DA vector as the Taylor expansion of a function at a specific point.

24 Considering a function $f(\mathbf{x})$ and its Taylor expansion $f_T(\mathbf{x}_0)$ at the point \mathbf{x}_0 up to the order
25 n , we can define an equivalence relation between the Taylor expansion and the DA vector as
26 follows

$$[f]_n = f_T(\mathbf{x}_0) = \sum C_{n_1, n_2, \dots, n_v} \cdot d_1^{n_1} \cdot \dots \cdot d_v^{n_v},$$

27 where $\mathbf{x} = (x_1, x_2, \dots, x_v)$, and $n \geq n_1 + n_2 + \dots + n_v$. Here d_i is a special number and it
28 represents a small variance in x_i . Generally one can define a DA vector by directly setting
29 values to respective terms, without defining the function f . The addition and multiplication of
30 two DA vectors can be defined straightforwardly. To add two DA vectors, we simply add the
31 coefficients of the like terms. To multiply two DA vectors, we multiply each term in the first
32 one with all the terms in the second one and combine like terms while ignoring all terms above
33 order n . So given two DA vectors $[a]_n$ and $[b]_n$ and a scalar c , we have the following formulas:

$$\begin{aligned} [a]_n + [b]_n &:= [a + b]_n, \\ c \cdot [a]_n &:= [c \cdot a]_n, \\ [a]_n \cdot [b]_n &:= [a \cdot b]_n, \end{aligned} \tag{1}$$

34 According to the fixed point theorem (Berz, 1999), the inverse of a DA vector that is not
35 infinitely small can be calculated iteratively in a limit number of iterations.

36 The derivation operator ∂_v with respect to the v^{th} variable can be defined as

$$\partial_v[a]_n = \left[\frac{\partial}{\partial x_v} a \right]_{n-1},$$

37 which can be carried out term by term on $[a]_n$. The operator ∂_v satisfies the chain rule:

$$\partial_v([a] \cdot [b]) = [a] \cdot (\partial_v[b]) + (\partial_v[a]) \cdot [b].$$

38 The inverse operator ∂_v^{-1} can also be defined and carried out easily in a term-by-term manner.
39 Once the fundamental operators are defined, the DA vector can be used in calculations just as
40 a number.

41 Statement of need

42 The TPSA/DA methods for particle beam dynamic analysis was developed in 1980s. The
43 tools are available in several popular programs for particle accelerator design and simulations,
44 e.g. COSY Infinity 9 (Makino & Berz, 2006), MAD-X (Deniau et al., 2017), PTC (Forest
45 et al., 2002), etc. In recent years, the use of TPSA/DA has been extended in other fields,
46 which motivates building TPSA/DA libraries in popular programming languages. The existing
47 programs are not convenient for developers in other fields. MAD-X is specifically developed
48 for the accelerator design and cannot be used as a general programming language. Although
49 COSY Infinity can be used as a general programming languages, it lacks some convenient
50 programming features in a modern language, such as C++ or Python. It also does not have
51 abundant libraries and a large supporting community. PTC includes a TPSA/DA library in
52 Fortran 90 but it does not have a user-friendly interface. The TPSA/DA library in C++ is
53 rare. DACE Massari & Wittig (2021) is one alternative. The DACE repository on GitHub had
54 been created but no codes had been released when the author started to develop cppTPSA.
55 Now DACE is available to the public. DACE provides the fundamental DA operations as well
56 as some advanced algorithms based on DA but it has not supported the complex DA vectors,
57 which is useful in the normal form analysis. To the best knowledge of the author, there is no
58 other TPSA/DA library in Python 3.

59 Features

60 This library is composed of a C++ library that performs the TPSA/DA calculations and a
61 Python wrapper (in a separate repository). Users can compile the source code into a static or
62 shared library or generate a Python library for Python 3 environment. The readme file in each
63 repository describes how to compile the C++ library and the Python library respectively.

64 The C++ library is based on Lingyun Yang's TPSA code, which was included in the previous
65 versions of MAD-X. In the development, we try to make minimal changes on the original code,
66 but had to revise or rewrite some functions for better efficiency and/or consistency. One big
67 change is the memory management. In Yang's code, the pointers to all the DA vectors are
68 stored in a vector. Each time a new DA vector is needed, the program will search in the vector
69 to find the first empty pointer and allocate the memory. Once the DA vector is out of scope,
70 the memory is freed. In this library, we allocate the memory pool for all DA vectors (number
71 defined by the user) in the very beginning when we initialize the DA environment. The address
72 for the slots, each for one DA vector, in the pool are saved in a linked-list. Whenever we need
73 to create a new DA vector, we take out a slot from the beginning of the list. Whenever a DA

74 vector goes out of the scope, its destructor will set all value in the slot to zero and put it back
75 to the end of the list. The memory pool is managed simply by manipulating the two pointers
76 that points to the beginning and the end of the list. In this way, the repetitive searching and
77 allocation/deallocation operations are avoided and better efficiency can be achieved.

78 Some new features have been added, which are listed in the following.

- 79 1. Add a DA vector data type and define the commonly used math operators for it, so that
80 users can use a DA vector as simple as a normal number in calculations.
- 81 2. Support the complex DA vector defined by the C++ complex template.
- 82 3. More math functions are supported. (A list of the overloaded math functions can be
83 found in the readme file of the repository.)
- 84 4. Add new functions that perform the composition of (complex) DA vectors, which can
85 carry out multiple compositions in a call.
- 86 5. A Python wrapper is provided.

87 The following C++ code shows an example of a simple TPSA/DA calculation. After initializing
88 an environment that can contain at most 400 three dimensional DA vectors up to the 4-th
89 order, two DA vectors x1 and x2 and a complex DA vector y1 are defined, some trigonometric
90 functions are performed on them, and the results are output to the screen.

```
#include "da.h"
da_init(4, 3, 400);
DAVector x1, x2;
x1 = da[0] + 2*da[1] + 3*da[2];
x2 = sin(x1);
x1 = cos(x1);
auto y1 = x1 + x2*1i;
std::cout<<x1<<x2<<std::endl;
std::cout<<sin(y1)<<std::endl;
```

91 A Python example doing the same calculation is presented as follows.

```
import tpsa
tpsa.da_init(4, 3, 400)
da = tpsa.base()
x1 = da[0] + 2*da[1] + 3*da[2]
x2 = tpsa.sin(x1)
x1 = tpsa.cos(x1)
y1 = tpsa.complex(x1, x2)
print(x1)
print(x2)
print(tpsa.sin(y1))
```

92 More examples can be found in the respective repository.

93 Verification

94 This library has been verified with COSY Infinity 9.0. As an example, the outputs of calculating
95 $\sin(0.3+da[0]+2\times da[1])$ up to the fourth order by both programs are presented in [Figure 1](#)
96 and [Figure 2](#) respectively. [Figure 1](#) shows the result by COSY Infinity, while [Figure 2](#) shows the
97 result by cppTPSA. The two programs give out exactly the same result. Here we want to note
98 (1) for some functions, e.g. arcsin, one may observe difference in the results at orders of 10^{-15}
99 or 10^{-16} , which is due to the different algorithms used in the calculation and is considered
100 acceptable in practice and (2) the sequence of the terms may be different when outputting a
101 DA vector from cppTPSA and from COSY Infinity.

Table with 3 columns: I, COEFFICIENT, ORDER EXPONENTS. It contains 15 rows of numerical data and their corresponding orders and exponents.

Figure 1: COSY Infinity 9.0 output.

Table with 4 columns: I, V [36], Base, [15 / 15]. It contains 15 rows of numerical data in scientific notation and their corresponding base values.

Figure 2: cppTPSA output.

Acknowledgements

The author would like to thank Dr. Lingyun Yang for providing his source code. This material is based upon work supported by the U.S. Department of Energy, Office of Science, Office of Nuclear Physics under contract DE-AC05-06OR23177.

References

Berz, M. (1991a). Symplectic tracking in circular accelerators with high order maps. Nonlinear Problems in Future Particle Accelerators, 288.
Berz, M. (1991b). High-order computation and normal form analysis of repetitive systems, in: M. Month (Ed), physics of particle accelerators (Vol. 249, p. 456). American Institute of Physics.
Berz, M. (1999). Modern map methods in particle beam physics. Academic Press.

- 114 Berz, M., & Makino, K. (1998). Verified integration of ODEs and flows using differential
115 algebraic methods on high-order Taylor models. *Reliable Computing*, 4(4), 361–369.
116 <https://doi.org/10.1023/A:1024467732637>
- 117 Chao, A. W. (2002). *Lecture notes on topics in accelerator physics*. Stanford Linear Accelerator
118 Center, Menlo Park, CA (US). <https://doi.org/10.2172/812598>
- 119 Deniau, L., Grote, H., Roy, G., & Schmidt, F. (2017). *The MAD-X program, version 5.07.00,*
120 *user's reference manual*. CERN.
- 121 Forest, E., Schmidt, F., & McIntosh, E. (2002). Introduction to the polymorphic tracking
122 code. *KEK Report*, 3, 2002.
- 123 Makino, K., & Berz, M. (2003). Verified global optimization with Taylor model methods.
124 *International Journal of Computer Research*, 12,2, 245–252.
- 125 Makino, K., & Berz, M. (2006). COSY INFINITY version 9. *Nuclear Instruments and Methods*,
126 558, 346–350. <https://doi.org/10.1016/j.nima.2005.11.109>
- 127 Massari, M., Di Lizia, P., Cavenago, F., & Wittig, A. (2018). Differential algebra software library
128 with automatic code generation for space embedded applications. In *2018 AIAA information*
129 *systems-AIAA infotech@ aerospace* (p. 0398). <https://doi.org/10.2514/6.2018-0398>
- 130 Massari, M., & Wittig, A. (2021). DACE: The differential algebra computational toolbox. In
131 *GitHub repository*. GitHub. <https://github.com/dacelib/dace>
- 132 Zhang, H. (2021a). cppTPSA: A C++ TPSA lib. In *GitHub repository*. GitHub. <https://github.com/zhanghe9704/tpsa>
133 <https://github.com/zhanghe9704/tpsa>
- 134 Zhang, H. (2021b). pyTPSA: A Python TPSA lib. In *GitHub repository*. GitHub. <https://github.com/zhanghe9704/tpsa-python>
135 <https://github.com/zhanghe9704/tpsa-python>
- 136 Zhang, H., & Berz, M. (2011). The fast multipole method in the differential algebra framework.
137 *Nuclear Instruments and Methods A* 645, 338–344. <https://doi.org/10.1016/j.nima.2011.01.053>
138 <https://doi.org/10.1016/j.nima.2011.01.053>