The Journal of Open Source Software

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

**He Zhang** [1¶]

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

**DOI:** 10.21105/joss.04818

**Software**

- Review
- Repository
- Archive

**Editor:** Daniel S. Katz

**Reviewers:**

- @CFGrote
- @mbkumar

**Submitted:** 27 August 2021
**Published:** 29 February 2024

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

## Summary

The truncated power series algebra (TPSA), also referred to as differential algebra (DA), is a well-established and widely used method in particle accelerator physics and astronomy. The most straightforward usage of TPSA/DA is to calculate the Taylor expansion of a given function at a specific point up to order $n$. In recent years, as the application of TPSA/TA has been extended to other fields, a reusable implementation of TPSA/DA as a modern C++ library or other high level programming language like Python has become desirable. The cppTPSA package implements TPSA/DA in C++11 and provides developers a convenient library with which to build advanced TPSA/DA-based methods. A Python 3 library, pyTPSA, has also been developed based on the C++ lib.

## Background

In the following, we give a very brief introduction on TPSA/DA from a practical computational perspective. Please refer to Berz (1999) and Chao (2002) for the complete theory with more details.

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

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

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

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: it represents a small variance in $x_i$. Generally one can define a DA vector by directly setting values to respective terms, without defining the function $f$. The addition and multiplication of two DA vectors can be defined straightforwardly. To add two DA vectors, we simply add the coefficients of the like terms. To multiply two DA vectors, we multiply each term in the first one with all the terms in the second one and combine like terms while ignoring all terms above order $n$. So given two DA vectors $[a]_n$ and $[b]_n$ and a scalar c, we have the following formulae:

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

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

The derivation operator $\partial_v$ with respect to the $v^{\text{th}}$ variable can be defined as

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

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

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

The inverse operator $\partial_v^{-1}$ can also be defined and carried out easily in a term-by-term manner. Once the fundamental operators are defined, the DA vector can be used in calculations just as a number. More sophisticated methods using DA have been developed, *e.g.* symplectic tracking (Berz, 1991a), normal form analysis (Berz, 1991b), verified integration (Berz & Makino, 1998), global optimization (Makino & Berz, 2005,), fast multipole method for pairwise interactions between particles (Zhang & Berz, 2011).

## Statement of need

TPSA/DA methods for particle beam dynamic analysis were developed in the 1980s. These tools are available in several popular programs for particle accelerator design and simulations, such as COSY Infinity 9 (Makino & Berz, 2006), MAD-X (Deniau et al., 2017; Grote & Schmidt, 2003), and PTC (Forest et al., 2002). In recent years, the application of TPSA/DA has been extended to other fields, motivating the development of TPSA/DA libraries in popular programming languages. However, the existing programs are not convenient for developers from other fields. For instance, MAD-X is specifically developed for accelerator design and cannot be used as a general programming language. Although COSY Infinity can be used as a general programming languages, it lacks some of the convenient programming features found in modern languages, such as C++ or Python, along with abundant libraries and a large supporting community. PTC does include a TPSA/DA library in Fortran 90 but it lacks a user-friendly interface. TPSA/DA libraries in C++ are rare. DACE (Massari et al., 2018; Massari & Wittig, 2021) is one alternative. The DACE repository on GitHub had been created but no codes had been released when the author began developing cppTPSA(Zhang, 2021). Now DACE is available to the public, providing fundamental DA operations and some advanced algorithms based on DA. However, it does not support complex DA vectors, which are useful in normal form analysis. To the author's best knowledge, there is no other TPSA/DA library in Python 3.

## Features

This library consists of a C++ library that performs TPSA/DA calculations and a Python wrapper. Users can compile the source code into a static or shared library or generate a Python library for a Python 3 environment. The readme file in the repository provides detailed instructions on how to compile both the C++ library and the Python library, respectively.

The C++ library is based on Lingyun Yang's TPSA code (Yang, 2009), which is also incorporated into MAD-X (Deniau et al., 2017). During development, we tried to make minimal changes from the original code, but had to revise or rewrite some functions for better efficiency and/or consistency. One big change is the memory management. In Yang's code, the pointers to all the DA vectors are stored in a vector. Whenever a new DA vector is required, the program searches this vector for the first empty pointer and allocates the memory. Once a DA vector is out of scope, the memory is freed. In contrast, our library initiates a memory pool for all DA vectors (with the number defined by the user) at the very start, during the initialization of the DA environment. The addresses for the slots, each for one DA vector, in the pool are maintained in a linked list. Whenever we need to create a new DA vector, we take out a slot from the beginning of the list. Whenever a DA vector goes out of the scope, its destructor will set all value in the slot to zero and put it back at the end of the list. The memory pool is managed simply by manipulating the two pointers: one pointing to the start and the other to the end of the list. This method eliminates the repetitive searching and allocation/deallocation operations, thereby achieving better efficiency.

Some new features have been added, as follows:

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

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

```cpp
#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;
```

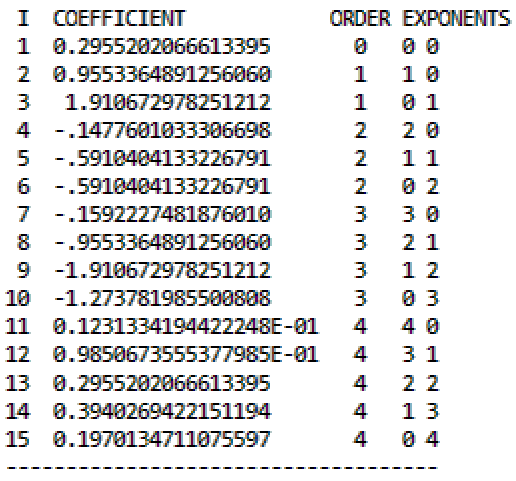
A Python example doing the same calculation is presented as follows.

```python
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))
```

More examples can be found in the repository.

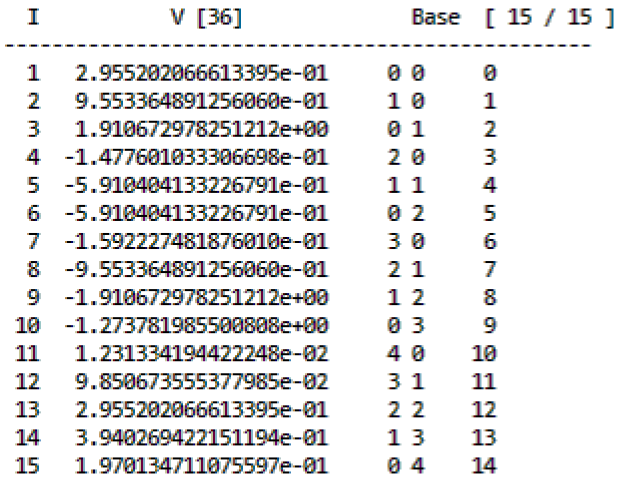
## Verification

This library has been verified with COSY Infinity 9.0. As an example, the outputs of calculating sin (0.3+da[0]+2×da[1]) up to the fourth order by both programs are presented in Figure 1 and Figure 2 respectively. Figure 1 shows the result generated by COSY Infinity, while Figure 2 shows the result generated by cppTPSA. The two programs give exactly the same result. In most cases, the two programs agree to the machine's precision. However, one may observe difference in the coefficients at orders of $10^{-15}$ or $10^{-16}$ for some special functions such as arcsin. This is because different numerical recipes are used in calculation. For example, a special function may be approximated by different series. This small deviation is usually considered acceptable in practice. If higher precision is desired, one could/should consider the Taylor Model (TM) datatype in COSY Infinity. The TM vector calculates a DA vector together with its error band. However, it is outside the scope of this code. Please note the sequence of the terms may be different when outputting a DA vector from cppTPSA and from COSY Infinity.



```
  I  COEFFICIENT             ORDER EXPONENTS
  1  0.2955202066613395        0   0 0
  2  0.9553364891256060        1   1 0
  3   1.910672978251212        1   0 1
  4  -.1477601033306698        2   2 0
  5  -.5910404133226791        2   1 1
  6  -.5910404133226791        2   0 2
  7  -.1592227481876010        3   3 0
  8  -.9553364891256060        3   2 1
  9  -1.910672978251212        3   1 2
 10  -1.273781985500808        3   0 3
 11  0.1231334194422248E-01    4   4 0
 12  0.9850673555377985E-01    4   3 1
 13  0.2955202066613395        4   2 2
 14  0.3940269422151194        4   1 3
 15  0.1970134711075597        4   0 4
 ------------------------------------
```

**Figure 1:** COSY Infinity 9.0 output.



```
  I          V [36]            Base  [ 15 / 15 ]
--------------------------------------------------

  1   2.955202066613395e-01    0 0     0
  2   9.553364891256060e-01    1 0     1
  3   1.910672978251212e+00    0 1     2
  4  -1.477601033306698e-01    2 0     3
  5  -5.910404133226791e-01    1 1     4
  6  -5.910404133226791e-01    0 2     5
  7  -1.592227481876010e-01    3 0     6
  8  -9.553364891256060e-01    2 1     7
  9  -1.910672978251212e+00    1 2     8
 10  -1.273781985500808e+00    0 3     9
 11   1.231334194422248e-02    4 0    10
 12   9.850673555377985e-02    3 1    11
 13   2.955202066613395e-01    2 2    12
 14   3.940269422151194e-01    1 3    13
 15   1.970134711075597e-01    0 4    14
```

**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. https://doi.org/10.1063/1.41975

Berz, M. (1999). *Modern map methods in particle beam physics*. Academic Press. https://doi.org/10.1016/s1076-5670(08)x7018-1

Berz, M., & Makino, K. (1998). Verified integration of ODEs and flows using differential algebraic methods on high-order Taylor models. *Reliable Computing*, *4*, 361–369. https://doi.org/10.1023/A:1024467732637

Chao, A. W. (2002). *Lecture notes on topics in accelerator physics*. Stanford Linear Accelerator Center, Menlo Park, CA (US). https://doi.org/10.2172/812598

Deniau, L., Skowronski, P., Roy, G., & others. (2017). MAD-X: Methodical accelerator design. In *GitHub repository*. GitHub. https://doi.org/10.5281/zenodo.7900975

Forest, E., Schmidt, F., & McIntosh, E. (2002). Introduction to the polymorphic tracking code. *KEK Report*, *3*, 2002. https://inspirehep.net/literature/591979

Grote, H., & Schmidt, F. (2003). MAD-X-an upgrade from MAD8. *Proceedings of the 2003 Particle Accelerator Conference*, *5*, 3497–3499. https://doi.org/10.1109/PAC.2003.1289960

Makino, K., & Berz, M. (2005). Verified global optimization with Taylor model based range bounders. *Transactions on Computers*, *11*(4), 1611–1618. https://www.bmtdynamics.org/pub/papers/GOM05/GOM05.pdf

Makino, K., & Berz, M. (2006). COSY INFINITY version 9. *Nuclear Instruments and Methods*, *558*, 346–350. https://doi.org/10.1016/j.nima.2005.11.109

Massari, M., Di Lizia, P., Cavenago, F., & Wittig, A. (2018). Differential algebra software library with automatic code generation for space embedded applications. In *2018 AIAA information systems-AIAA infotech@ aerospace* (p. 0398). https://doi.org/10.2514/6.2018-0398

Massari, M., & Wittig, A. (2021). DACE: The differential algebra computational toolbox. In *GitHub repository*. GitHub. https://github.com/dacelib/dace

Yang, L. (2009). Array based truncated power series package. *Proceedings of the 10th Internaltional Computational Accelerator Physics Conference*, 371–373. https://https://accelconf.web.cern.ch/ICAP2009/papers/thpsc059.pdf

Zhang, H. (2021). cppTPSA: A C++ TPSA lib. In *GitHub repository*. GitHub. https://github.com/zhanghe9704/tpsa

Zhang, H., & Berz, M. (2011). The fast multipole method in the differential algebra framework. *Nuclear Instruments and Methods A 645*, 338–344. https://doi.org/10.1016/j.nima.2011.01.053