The Journal of Open Source Software

SDA: a symbolic differential algebra package in C++

He Zhang [1¶]

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

DOI: 10.xxxxxx/draft

Software

- Review
- Repository
- Archive

Editor: Open Journals

Reviewers:

- @openjournals

Submitted: 01 January 1970
Published: unpublished

Summary

The truncated power series algebra (TPSA), or differential algebra (DA), is a well-established and widely used method in particle accelerator physics. It is typically used to generate high-order map of a nonlinear dynamical system but is also used in symplectic tracking, normal form analysis, verified integration, global optimization, fast multipole method, etc. This package is the first to carry out symbolic DA-based calculations. Symbolic DA makes it possible to trace the contribution of initial conditions to the final result in a DA calculation process and improve the efficiency of repeated DA calculations, which potentially broadens the usage of DA.

Background

The fundamental concept in DA is the DA vector, which can be considered as the Taylor expansion of a function at a specific point from a practical perspective.

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

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

where $\mathbf{x} = (x_1, x_2, \ldots, x_v)$, and $n \geq n_1 + n_2 + \cdots + n_v$. Here d_i is a special number: it represents a small variance in x_i. 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 then combine like terms while ignoring all terms above order n. Given two DA vectors $[a]_n$ and $[b]_n$ and a scalar c, we have

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

According to the fixed point theorem (Berz, 1999), the inverse of a DA vector that is not infinitely small can be calculated in a finite number of iterations. The derivation operator ∂_v with respect to the v^{th} variable and its inverse operator $\partial^{-1}v$ can be defined and carried out on a term by term basis on $[a]_n$. A DA vector can be used in calculations just as a number.

The symbolic DA combines DA with symbolic calculation. Any coefficient of a Symbolic DA (SDA) vector is an explicit expression of the symbols in lieu of a number.

Statement of need

DA has been used in particle beam dynamic analysis since1980s and gradually extended to other fields. DA provides powerful analyzing tools (Berz, 1991a, 1991b), for a dynamic system. It is also used in various numerical algorithms (Berz & Makino, 1998; Makino & Berz, 2005; Zhang & Berz, 2011). DA tools are available in particle accelerator simulators (Deniau et al., 2017; Forest et al., 2002; Grote & Schmidt, 2003; Makino & Berz, 2006), or as stand-alone libraries (Massari et al., 2018; Massari & Wittig, 2021; Zhang, 2024). All of them only perform numerical DA calculation. This is the first and only library for symbolic DA calculations.

We developed SDA to improve the efficiency of repetitive DA processes. By performing the computation once with SDA, we obtain an explicit expression for how the final DA vector depends on the initial inputs. Evaluating this expression—rather than rerunning the full DA process for each new input—significantly reduces computation time (Zhang, 2025). SDA also provides an efficient method for computing higher-order derivatives of any given function.

Features

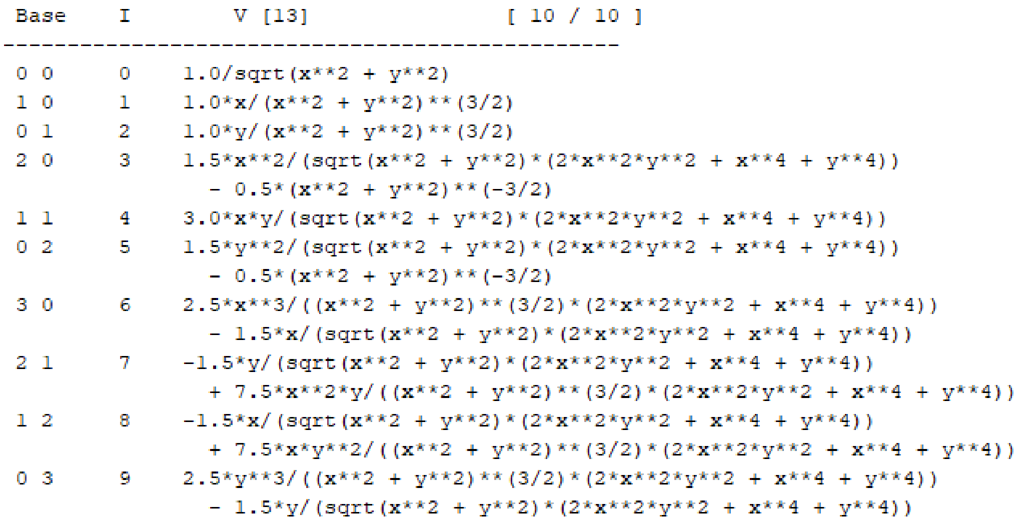
This library is based on the numerical DA library, cppTPSA (Zhang, 2024). All the DA calculations are carried out on symbols using exactly the same algorithms in cppTPSA by employing the SymEngine library (Fernando et al., 2024). Users can compile the source code into a static or shared library and install it on their system. The main features of this library include the following:

1. Define the SDA vector data type and overload common math operators/functions for it.
2. Support the composition, derivation, and inverse derivation on SDA vectors.
3. Obtain the explicit expression of a partial derivative for a function.
4. Obtain the callable function on the symbols from an SDA.
5. Obtain the numerical DA from an SDA by assigning values to all the symbols.

The following code shows how to calculate the SDA vector of $1/\sqrt{x^2+y^2}$ up to the third order. After initializing a memory pool for 400 SDA vectors and defining the symbols, x and y, the SDA vector f is calculated and printed. The value of the SDA vector is shown in Figure 1. The first column shows the orders of the bases, the second column displays the index of each term in the SDA vector, and the last column lists the coefficient of each term as a function of x and y.

```
#include <iostream>
#include <sda.h>
typedef SymbDA::DAVector SDA;
using SymEngine::Expression;

int main() {
    int order{3}, dim{2}, pool{400};
    SymbDA::da_init(order, dim, pool);
    auto& sda = SymbDA::da;
    Expression sx("x"), sy("y");
    SDA f = 1/sqrt((sx+sda[0])*(sx+sda[0]) + (sy+sda[1])*(sy+sda[1]));
    std::cout<<f;
}
```

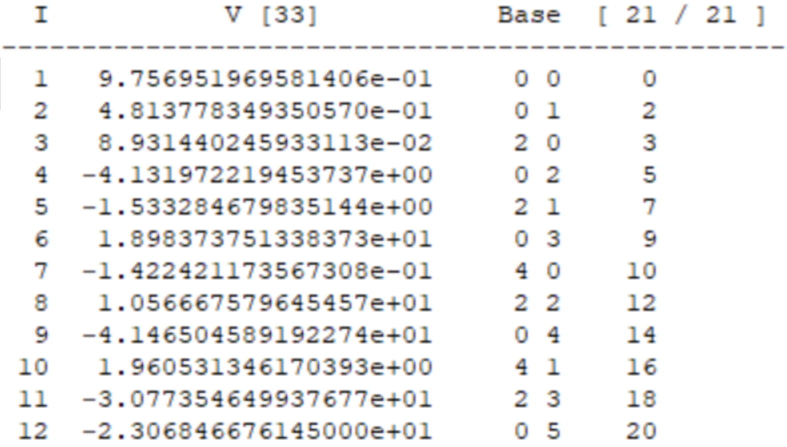


```
Base     I        V [13]                    [ 10 / 10 ]
-------------------------------------------------
0 0      0     1.0/sqrt(x**2 + y**2)
1 0      1     1.0*x/(x**2 + y**2)**(3/2)
0 1      2     1.0*y/(x**2 + y**2)**(3/2)
2 0      3     1.5*x**2/(sqrt(x**2 + y**2)*(2*x**2*y**2 + x**4 + y**4))
                 - 0.5*(x**2 + y**2)**(-3/2)
1 1      4     3.0*x*y/(sqrt(x**2 + y**2)*(2*x**2*y**2 + x**4 + y**4))
0 2      5     1.5*y**2/(sqrt(x**2 + y**2)*(2*x**2*y**2 + x**4 + y**4))
                 - 0.5*(x**2 + y**2)**(-3/2)
3 0      6     2.5*x**3/((x**2 + y**2)**(3/2)*(2*x**2*y**2 + x**4 + y**4))
                 - 1.5*x/(sqrt(x**2 + y**2)*(2*x**2*y**2 + x**4 + y**4))
2 1      7     -1.5*y/(sqrt(x**2 + y**2)*(2*x**2*y**2 + x**4 + y**4))
                 + 7.5*x**2*y/((x**2 + y**2)**(3/2)*(2*x**2*y**2 + x**4 + y**4))
1 2      8     -1.5*x/(sqrt(x**2 + y**2)*(2*x**2*y**2 + x**4 + y**4))
                 + 7.5*x*y**2/((x**2 + y**2)**(3/2)*(2*x**2*y**2 + x**4 + y**4))
0 3      9     2.5*y**3/((x**2 + y**2)**(3/2)*(2*x**2*y**2 + x**4 + y**4))
                 - 1.5*y/(sqrt(x**2 + y**2)*(2*x**2*y**2 + x**4 + y**4))
```

Figure 1: Example code output.

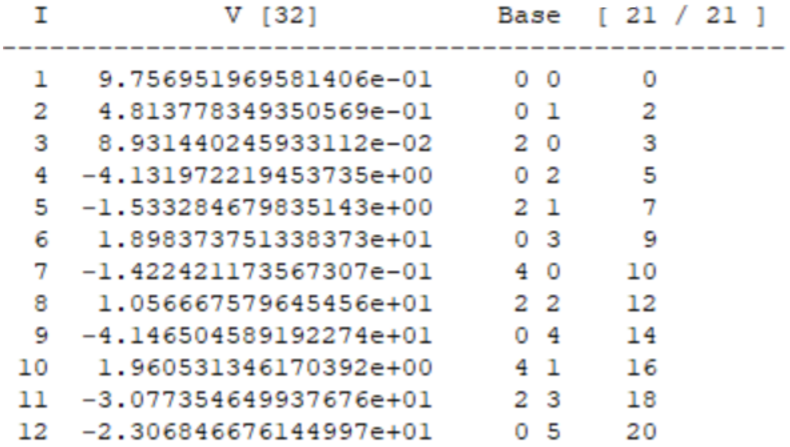
Verification

This library has been verified with the numerical DA library, cppTPSA, by assigning values to all the symbols in an SDA vector and checking it against direct calculation using cppTPSA. As an example, For example, Figure 2 and Figure 3 show the DA vector $v = \exp(1.5926 + d_1^2 + 5.3897 \cdot d_2)$ to the fifth order, calculated by SDA and DA respectively. We calculate the relative error for each non-zero coefficient. This procedure is repeated 1,000 times for all the math functions in the SDA lib and the absolute values of the relative errors are all below 1×10^{-15}.



```
  I            V [33]             Base   [ 21 / 21 ]
----------------------------------------------------
  1    9.756951969581406e-01      0 0       0
  2    4.813778349350570e-01      0 1       2
  3    8.931440245933113e-02      2 0       3
  4   -4.131972219453737e+00      0 2       5
  5   -1.533284679835144e+00      2 1       7
  6    1.898373751338373e+01      0 3       9
  7   -1.422421173567308e-01      4 0      10
  8    1.056667579645457e+01      2 2      12
  9   -4.146504589192274e+01      0 4      14
 10    1.960531346170393e+00      4 1      16
 11   -3.077354649937677e+01      2 3      18
 12   -2.306846676145000e+01      0 5      20
```

Figure 2: SDA output.



```
  I          V [32]          Base  [ 21 / 21 ]
--------------------------------------------------
  1   9.756951969581406e-01     0 0     0
  2   4.813778349350569e-01     0 1     2
  3   8.931440245933112e-02     2 0     3
  4  -4.131972219453735e+00     0 2     5
  5  -1.533284679835143e+00     2 1     7
  6   1.898373751338373e+01     0 3     9
  7  -1.422421173567307e-01     4 0    10
  8   1.056667579645456e+01     2 2    12
  9  -4.146504589192274e+01     0 4    14
 10   1.960531346170392e+00     4 1    16
 11  -3.077354649937676e+01     2 3    18
 12  -2.306846676144997e+01     0 5    20
```

Figure 3: cppTPSA output.

Acknowledgements

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

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

Fernando, I., Čertík, O., & others. (2024). *SymEngine*. https://github.com/symengine/symengine

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

Zhang, H. (2024). cppTPSA/pyTPSA: A C++/Python package for truncated power series algebra. *Journal of Open Source Software*, *9*(94), 4818. https://doi.org/10.21105/joss.04818

Zhang, H. (2025). *Boosting the efficiency of the differential AlgebraBased fast multipole method operators using symbolic calculation*. Presented at the meeting of SIAM CSE25, Fort Worth, Texas, U.S. https://www.siam.org/media/fyvh3qlf/cse25_abstracts.pdf

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