A FLEXIBLE-VARIABLE TRUNCATED POWER SERIES ALGEBRA IN Zlib *

Y.T. Yan, Y. Cai, and J. Irwin
Stanford Linear Accelerator Center, Stanford University, Stanford, CA 94309 USA

Abstract

Zlib is a numerical library for Truncated Power Series Algebra (TPSA) and Lie Algebra for application to nonlinear analysis of single particle dynamics. The first version was developed in 1990 with the use of the One-Step Index Pointers (OSIP's). The OSIP's form the Zlib nerve that offers optimal computation and allow order grading as well as flexible initialization of the global number of variables for the TPSA. While the OSIP's are still kept for minimum index passing to achieve efficient computation, Zlib has been being upgraded to allow flexible and gradable local number of variables in each C++ object of the Truncated Power Series (Tps) class. Possible applications using Zlib are discussed.

1 INTRODUCTION

The first Zlib version was developed in 1990 [1] in Fortran language. Its development was aimed at fast computation of the one-turn Taylor map extraction of the Superconducting Super Collider (SSC) lattices and high-order nonlinear mapping analyses. The core is the Zlib nerve system which consists of the One-Step Index Pointers for minimum index pass to achieve efficient computation of the TPSA that involves some fundamental operations such as addition, subtraction, multiplication, division, inverse, sine, cosine, square root, power, exp, log, derivative, integral, etc. of the Tps's, and some derived operations such as concatenation, Poisson bracket, various formats of Lie generators in Cartesian or Action-Angle coordinates, map tracking, etc. The Tps orders were gradable. The Memories for the One-Step Index Pointers and necessary internal auxiliary arrays were dynamically allocated at the minimum required level per user's input for the maximum order and number of variables.

In 1993, the basic part (about 20%) of the Zlib Fortran subroutines were faithfully translated into C++ codes that form two fundamental classes of the TPSA [2]. These two classes were named ZSeries and ZMap which handles the algebra of truncated power series and vector truncated power series respectively. Recently at SLAC, aiming at further development in C++ for mapping analysis, while keeping the original Zlib One-Step Index Pointers for efficient index passing, we have been re-designing the Zlib based on a newly invented formulation [3] that allows for both order and variable grading. Major classes implemented and to be implemented are Tps (Truncated Power Series), Vps (Vector Tps), Aps (Tps in Action-Angle Variables), Lie and a variety of derived Lie classes. The truncated power series coefficients are commonly understood as a double type. Indeed, they can be a Tps type, too, making classes Tps<Tps>, Vps<Tps>, Lie<Tps>, etc. such that the canonical coordinate variables form a class of Tps of which each coefficient is another class of Tps of certain parameter variables whose optimal values are to be determined after parameterized mapping analysis.

2 ONE-STEP INDEX POINTERS

There are, fundamentally, three kinds of data structures used in programming the TPSA in beam physics community, the hybrid procedure of Cosy Infinity developed in late 1990's, the One-Step-Index-Pointer procedure of Zlib developed in 1990, and the Link-List procedure of MXYZPPLK developed in 1990, The recent elegant TPSA "look-back table" description of Dragt is fundamentally similar to Zlib "One-Step Index Pointer".

Through appropriate labeling and indexing, a Tps can be represented by a one-dimensional array of real (double) numbers one-to-one corresponding to the coefficients of the Tps. For allowing order grading, low order coefficients would be indexed first, followed by the next high-order and then the next high-order coefficients until the preset (or derived) maximum order is reached. In a mathematical formula, an n-variable, P-order Tps can be written as

$$T(n,P) = C_0 + H(n,1) + H(n,2) + ... + H(n,P),$$

where $H(n,i), i = 1,2,...,P$, is the homogeneous polynomial of order i. and C_0 is the constant term. This formula allows for order grading (adding or truncating high-order terms does not affect the low order structure) but not for variable grading. Of course, in the above formula, one can let n be the fixed order and P be the maximum number of variables, then it allows for variable grading, but the order has to be fixed. Such a flexible-variable but fixed-order formulation is most suitable for use in Taylor map extraction. Indeed, the Zlib Fortran version developed in 1990 has two sub libraries, one for gradable number of variables (the TPALIB) and the other for gradable orders (the ZPLIB) [1].

The key to fast speed TPSA is to have One-Step Index Pointers prepared only once for repeated use such that for any coefficient involved in a given calculation, it can be identified with a minimum index path. Taking Tps multiplication as an example, let Tps C = A * B, where A and B are two given Tps's that may be with different orders, then the task is to obtain all of the coefficients of C to a

* Work supported by the Department of Energy under Contract No. DE-AC03-76SF00515

specified order derived from the orders of A and B and the preset cap order. To obtain $C[j]$ (jth-term of C), the "backward" scheme would go over a loop i to sum over exactly the number of contributing multiplication terms given by $A[aOSIP[i]] * B[bOSIP[i]]$, where $aOSIP$ and $bOSIP$ are One-Step Index Pointers (OSIP's) stored for repeated use to achieve fast multiplication.

3 FLEXIBLE NUMBER OF LOCAL VARIABLES

As has been described in the last section, the one-dimensional-array data structure for the Tps can either have variable grading or order grading but not both. If one chooses to have gradable orders, then the number of variables is fixed in the structure. Nonetheless, one can establish a set of order-gradable One-Step Index Pointers for each number of variables up to the maximum number of variables, and build up variable-crossing OSIP's that involves operations between two different number-of-variable structures. In addition to a fixed-variable version, Zlib has a C++ version with such a flexible-variable data structure. Therefore, for example, one can perform Tps multiplication directly between two Tps's A and B with nA and nB variables respectively, where nA may or may not be equal to nB. In comparison with the fixed-variable TPSA, such data structure allowing for operation among different number of variables would ease the flexible use of the TPSA dramatically. However, the price is paid with a larger memories for storing a larger set of the OSIP's. Also note that such a data structure does not allow variable grading though it allows for flexible number of variables.

To offset the large memories required for the OSIP's, We have been upgrading Zlib again based on a new formula give by [3]

$$\begin{aligned} T(n,P) &= C_0 + x_1 T(1,P-1) + x_2 T(2,P-1) + .. \\ &\quad + x_n T(n,P-1), \end{aligned}$$

where C_0 is the constant term and $x_1, x_2, ...x_n$ label the n variables respectively. The nice thing about this new formulation is that (a) it builds on the top of the above one-dimensional flexible-variable scheme such that the One-Step Index Pointers can still be used for achieving fast speed, (b) both the number of variables and the order are gradable simultaneously, and most importantly, (c) the OSIP's only need to be prepared and stored up to an order smaller than the maximum order desired by 1 or 2, saving memories dramatically. The most saving of memories comes from the Tps multiplication OSIP's. They only need to be prepared and stored up to an order that is smaller than the maximum order by 2. For detailed description of this new scheme, allowing both order and variable grading while still using One-Step Index Pointers, one can refer to the original article [3].

4 MAJOR CLASSES AND THEIR APPLICATIONS

Major classes in Zlib are Tps, Vps, Aps, Lie and its derivatives, and related parameterized classes Tps<Tps>, Vps<Tps>, Lie<Tps>. They are briefly described as follows.

The Tps Class

Tps is an abbreviated name for the truncated Power Series. It is the most fundamental class in Zlib. A Tps truncated at an order of P can be mathematically written as [1] [6]

$$U(\vec{z}) = \sum_{o=0}^{P} u(\vec{k}) \vec{z}^{\vec{k}},$$

where, assuming n variables, $\vec{z}$ represents the variables labeled as $z_1, z_2, ...z_n$, $\vec{k}$ represents the power indices $(k_1, k_2, ...k_n)$ and so $\vec{z}^{\vec{k}}$ represents $z_1^{k_1} z_2^{k_2} ... z_n^{k_n}$, and $\sum_{o=0}^{P}$ means summation over all possible monomials labeled by $\vec{k}$ with order given by $o = k_1 + k_2 + ... + k_n$ that is less than or equal to P.

The Tps class in Zlib is designed to manipulate Tps represented by the above-described coefficients. Both the order and the number of variables are local (object) member data, that is, each object of Tps can have its own order and number of variables that may be different from another Tps object.

Besides providing the basic functions for the other classes in Zlib, this class is frequently used for extracting Taylor maps for beamlines. With a suitable Tps initialization, one simply replace the the double type declaration of related variables with the Tps declaration in the tracking routine to obtain Taylor maps.

The Vps Class

Vps is an abbreviated name for the Vector truncated Power Series. A Vps truncated at an order of P can be mathematically written as [1] [6]

$$\vec{U}(\vec{z}) = \sum_{o=0}^{P} \vec{u}(\vec{k}) \vec{z}^{\vec{k}}, \tag{1}$$

that is, each component of the Vps is a Tps represented by coefficients described in the last section. For example, the i^{th} component would be represented by

$$U_i(\vec{z}) = \sum_{o=0}^{P} u_i(\vec{k}) \vec{z}^{\vec{k}},$$

The Vps class in Zlib C++ version is designed to manipulate Vps described above. It can represent a Taylor map and therefore have member functions for concatenation and Taylor-map tracking.

The Aps Class

Aps is an abbreviated name for the truncated power series in action-angle variable space. A class named Aps in Zlib C++ version is actively under implementation. Some of the important member functions in this class are the nPB tracking and the extraction of the normalized resonance basis

coefficients which was coded before in Fortran and have been used intensively for PEP-II lattice studies [7].

The Lie Classes

Application of TPSA to nonlinear single-particle dynamics usually goes with the Lie algebraic analysis. Therefore, majority of the Zlib classes are to be for Lie algebras such as single Lie generators, Dragt-Finn factorizations [8], nonlinear normal forms [9], kick factorizations [10], integrable polynomial factorization [11], etc. Most of these functions are achieved with derived Lie classes.

The parameterized Classes

In mapping analysis of a beam line lattice, in addition to the canonical phase-space variables, we often would like to have parameter variables which are constant but not specified with a value. Their values are either to be determined after the analysis or are dynamical (time dependent) to allow additional studies such as for synchrotron oscillation, power supply ripple, and ground motion at lower computational price. Treatment of such parameterized map in Fortran is tedious and usually uses semi-parameterization methods. Although some fully parameterized (coefficients of the power series in canonical space are treated as power series in parameter space) algorithms were written for treating both linear (but nonlinear in parameter space) and nonlinear cases [12], there were no implementation of such fully parameterized methods in the Zlib Fortran version. However, with the object-oriented capability, it is easier to code such fully parameterized algorithms since one can consider each of the coefficients in the canonical space as an object of Tps instead of a double. These fully parameterized mapping methods are currently under active development in Zlib C++ version. The major part of this effort involves classes Tps<Tps>, Vps<Tps>, Lie<Tps> and more. We plan to make the Tps, Vps, and Lie classes as template classes so that no extra classes are needed for achieving these parameterized functions.

5 SUMMARY

Through several rounds of upgrading, we may have lead Zlib to the ultimate design of the TPSA in terms of speed and flexibility. The Zlib nerve system consisting of the One-Step Index Pointers provides very efficient computational speed. The implementation based on the newly discovered formula /citegrade given in Section III, allows flexible and gradable number of variables and orders.

Possible applications of Zlib are broad. Besides performing various nonlinear mapping analyses, it can be easily linked to a C++ tracking code to extract one-turn or one-section Taylor maps.

Zlib has been linked to LEGO (a modular accelerator design code) [13] for extracting nonlinear parameterized maps and for non-linear mapping analysis.

6 ACKNOWLEDGEMENT

We would like to thank Scott Berg, Alex Dragt, Jim Holt, Chris Islin, Leo Michelotti, Nick Walker, and Johannes van Zeijts for many useful discussions. Special thanks are given to N. Malitsky for his interest in translating the Zlib Fortran version into the first C++ version.

7 REFERENCES

[1] Y. Yan and C. Yan, "*Zlib — A Numerical Library for Differential Algebra*," SSC Laboratory Report SSCL-300 (1990); Y.T. Yan, "*Zlib and Related Programs for Beam Dynamics Studies*," in *Computational Accelerator Physics*, AIP Conf. Proc. No. 297, p.279 (1993), R. Ryne, eds.

[2] N. Malitsky, A. Reshetov, and Y. Yan, "*ZLIB++: an Object-Oriented Numerical Library for Differential Algebra*," SSCL-659 (1994).

[3] Y.T. Yan, "*A New Formula for Ultimate Design of the Truncated Power Series Algebra with Both Order and Number-of-Variable Gradability*," SLAC-PUB-7435, 1997; Y.T. Yan, "*Letter to the Editors*," ICFA Beam Dynamics Newsletter, No. 13, P. 8, April, 1997, K. Hirata, J. Jowett, and S.Y. Lee eds.

[4] M. Berz, "*Differential Algebraic Description of Beam Dynamics to Very High Orders*," *Particle Accel.* **24**, 109 (1989).

[5] L. Michelotti, "*MXYZPPLK: a C++ version of differential algebra*," Fermi National Accelerator Laboratory Report FN-535 (1990).

[6] Y.T. Yan, "*Applications of Differential Algebras to Single-Particle Dynamics in Storage Rings*," SSCL-500, in *The Physics of Particle Accelerator*, AIP Conf. Proc. No. 249, p. 378 (1992), M. Month and M. Dienes, eds.

[7] Y.T. Yan, J. Irwin, and T. Chen, "*Resonance Basis Maps and nPB Tracking for Dynamics Aperture Studies*," Particle Accelerators, Vol. 55, p. 263 (1996).

[8] A. Dragt and J. Finn, "*Lie Series and Invariant Functions for Analytic Symplectic Maps*," *J. Math. Phys.*, **17**, 2215 (1976).

[9] E. Forest, M. Berz, and J. Irwin, "*Normal Form Methods for Complicated Periodic Systems*," *Particle Accelerators*, **24**, 91 (1989).

[10] J. Irwin, "*A Multi-Kick Factorization Algorithm for Nonlinear Maps*," in *Accelerator Physics at the SSC*, AIP Conf. Proc. No. 326, edited by Y.T. Yan et al. (AIP, New York, 1995), p. 662; D. Abell and A.J. Dragt, to be published.

[11] J. Shi and Y.T. Yan, "*Explicitly Integrable Polynomial Hamiltonians and Evaluation of Lie Transformations*," *Phys. Rev. E*, **48**, 3943 (1993).

[12] In Chapters 5 and 6 of Ref. 5.

[13] Y. Cai, M. Donald, J. Irwin, and Y.T. Yan, "*LEGO: A Modular Accelerator Design Code*," in these proceedings.