SLAC-PUB-7363
November 1996
CONF-9609256--13

# Zlib: a Numerical Library for Optimal Design of Truncated Power Series Algebra and Map Parameterization Routines

Yiton T. Yan
Stanford Linear Accelerator Center, Stanford University, Stanford, CA 94309

**Abstract**

A brief review of the Zlib development is given. Emphasized is the Zlib nerve system which uses the One-Step Index Pointers (OSIP's) for efficient computation and flexible use of the Truncated Power Series Algebra (TPSA). Also emphasized is the treatment of parameterized maps with an object-oriented language (e.g. C++). A parameterized map can be a Vector Power Series (Vps) or a Lie generator represented by an exponent of a Truncated Power Series (Tps) of which each coefficient is an object of truncated power series.

## Introduction

Zlib Fortran version was developed in 1990 [1]. Its development was originated at fast computational speed for nonlinear analyses of high-order power-series maps of the Superconducting Super Collider (SSC) lattices. Since Supercomputers, such as Cray computers, were used, the algorithms used for manipulating truncated power series and Lie algebras were optimized for scalar, vector, and parallel computing. Of the most important part in achieving such optimization was the Zlib nerve system consisted of One-Step Index Pointers for optimized computation of TPSA routines such as multiplication, concatenation, partial derivative, Taylor map tracking, etc. 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. By the time of the CAP93 Conference, there were more than 200 dynamically usable subroutines in Zlib Fortran version for TPSA and Lie algebraic mapping analysis [2].

In late 1993, upon termination of the SSC, about 20% of the Zlib Fortran subroutines were faithfully translated into C++ codes that form two fundamental classes of the TPSA [3]. These two classes were named ZSeries and ZMap which handles the algebra of truncated power series and Vector truncated power series respectively and have been included in Malitsky's Unified Library [4]. Recently at SLAC, aiming at further development for mapping analysis, the two classes ZSeries and ZMap have been rewritten and named as Tps and Vps and added or to be added upon them are many other classes for treating Lie algebras. Many of these classes are translated or to be translated from Zlib Fortran version developed in early 90's.

## Truncated Power Series - Tps

Tps is an abbreviated name for the Truncated Power Series. A Tps truncated at an order of $\Omega$ can be mathematically written as [1] [5]

*Work supported by Department of Energy, contract DE-AC03-76SF00515.
+presented at The 1996 Computational Accelerator Physics Conference, 24-27 September, 1996, Williamsburg, VA; to appear in CAP96 Proceedings.

## DISCLAIMER

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

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 $\sum_{o=0}^{\Omega}$ 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 $\Omega$. For an n-variable, $\Omega$-order Tps, there are a total of $m(n, \Omega) = (n + \Omega)!/(n!\Omega!)$ monomials. In an optimized computation for the TPSA, the first step is to allocate minimum possible memory for storing the Tps coefficients. To achieve this goal, Zlib uses an integer sequence j's that starts from 0 to $m(n, \Omega) - 1$ (or from 1 to $m(n, \Omega)$ for the Fortran version) to label the the Tps coefficients, that is, there is a one-to-one correspondence between j's and $\vec{k}$'s. Such labeling is the same for all Tps's except that there may be order differences and so the label sequence starts with the lowest order (the 0th order) and then go on to the next order and so on. For example, for a 3-variable case, the corresponding labels between j's and $\vec{k}$'s up to third order are: $0 \equiv (0,0,0)$, $1 \equiv (1,0,0)$, $2 \equiv (0,1,0)$, $3 \equiv (0,0,1)$, $4 \equiv (2,0,0)$, $5 \equiv (1,1,0)$, $6 \equiv (1,0,1)$, $7 \equiv (0,2,0)$, $8 \equiv (0,1,1)$, $9 \equiv (0,0,2)$, $10 \equiv (3,0,0)$, $11 \equiv (2,1,0)$, $12 \equiv (2,0,1)$, $13 \equiv (1,2,0)$, $14 \equiv (1,1,1)$, $15 \equiv (1,0,2)$, $16 \equiv (0,3,0)$, $17 \equiv (0,2,1)$, $18 \equiv (0,1,2)$, $19 \equiv (0,0,3)$. In Zlib, the relation between j's and $\vec{k}$'s is governed by a simple formula which is only used to generate One-Step Index Pointers.

The Tps class in Zlib C++ version is designed to manipulate Tps represented by the above-described coefficients.

## Vector truncated Power Series - Vps

Vps is an abbreviated name for the Vector truncated Power Series. A Vps truncated at an order of $\Omega$ can be mathematically written as [1] [5]

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

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

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

The Vps class in Zlib C++ version is designed to manipulate Vps described above.

## One-Step Index Pointers

For optimized computation of the TPSA, besides efficient memory mangement, one would also like to achieve efficient calculation for each of the related algebras such

as Tps multiplication, Vps concatenation, and Taylor map tracking, etc. The key 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. For example, let A and B be two Tps's (may be with different orders), such that Tps C = A * B. The task is to obtain all of the coefficients of C to a specified order derived from the orders of A and B and the preset cap order. Assuming the minimum and the maxmum orders for C derived are minimumOrderOfC and maximumOrderOfC, to obtain C, the code in Zlib would look like as follows.

```
for (order = minimumOrderOfC; order <= maximumOrderOfC; ++order) {
   lowOrder = MaximumOrderOf (order-maximumOrderOfB, minimumOrderOfA);
   highOrder = MinimumOrderOf (maximumOrderOfA, order-miniMumOrderOfB);
   for (j = monomialBegin[order]; j < monomial[order]; ++j) {
      for (i = ipBegin[j][lowOrder]; i < ipEnd[j][highOrder]; ++i) {
        C[j] += A[aOSIP[i]] * B[bOSIP[i]];
      }
   }
}
```

where except A, B, C (assumed cleared), all other variables are integers and are assumed to have been declared. Note that except lowOrder and highOder (obtained with negligible computer time), all other indices are obtained by assignment only (minimum index path) through One-Step Index Pointers (OSIP's). Note that, if a supercomputer is used, then the innest (i-) loop is vectorized through automatic gather while the j-loop is parallelized. This original Zlib One-Step Index Pointers scheme for Tps multiplication may be categorized as a "backward" scheme. Recently, Dragt seems to be interested in exploring a similar scheme which may be categorized as a "forward" scheme [6].

As another example of the OSIP's, let V be a Vps with a dimension of n (an n-dimensional Tps) reresenting a one-turn Taylor map, and z be a vector respenting the phase-space coordinates of a particle. Assuming no parameters, that is, in phase space, z has the same dimension as V, then a one-turn Taylor map tracking is to update the phase-space coordinates represented by z through evaluation of the Vps given by Eq. 1. The code in Zlib would look like as follows.

```
   for (j = 1; j < monomial[orderOfV]; ++j) xx[j] = z[iOSIP[j]]*xx[jOSIP[j]];
   for (i = 0; i < n; ++i) z[i] = V[i][0];
   for (i = 0; i < n; ++i)
      for (j = 1; j < monomial[orderOfV]; ++j) z[i] += V[i][j]*xx[j];
   }
```

where again iOSIP and jOSIP are One-Step Index Pointers and in the double loop, the inner one is vectorized while the outer one is parallelized. It was this fast

computational process that allowed the fast Taylor-map tracking for the SSC to high orders (11- or 12-th order) with a computational speed two orders of magnitude faster than the conventional element-by-element tracking [5].

## Action-angle variable truncated Power Series - Aps

Aps is an abbreviated name for the truncated power series in action-angle variable space. A class named Aps in Zlib C++ version is nearly completed. 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].

## 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.

## Parameterized maps - the Tps of Tps

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 capability of the object orientation, 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 in stead of a double. These fully parameterized mapping methods are currently under active development in Zlib C++ version.

## Zlib Future Direction

While Zlib Fortran version will still be kept for optional use, there will be no more development. The future direction is to develop a more complete C++ version for Zlib. Currently, there are more than 30 classes in Zlib that are under active improvement and/or development.

## Acknowledgement

Systematical programming use of differential algebras (truncated power series algebras) was introduced to the particle accelerator community by Berz [13]. On the physics side, the Lie algebraic application to accelerator beam dynamics was introduced by Dragt [14]. E. Forest and J. Irwin and others have also made contributions for more Lie algebraic applications. Michelotti made the most effort in advocating C++ programming for beam line studies and has used a link list method to develop a differential algebra package [15].

Highly acknowledgement is given to N. Malitsky for his participation in Zlib C++ version development.

## References

[1] Y. Yan and C. Yan, "*Zlib — A Numerical Library for Differential Algebra,*" SSC Laboratory Report SSCL-300 (1990).

[2] 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.

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

[4] N. Malitsky, in these proceedings.

[5] 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.

[6] A. Dragt, private communication.

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

[14] A.J. Dragt, Annual Rev. Nucl. Part. Sci. **38**, 455 (1988).

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