Available online at www.sciencedirect.com

ScienceDirect

Physics Procedia 1 (2008) 333–338

Physics Procedia

www.elsevier.com/locate/procedia

Proceedings of the Seventh International Conference on Charged Particle Optics

Mathematica program for extracting one-turn Lie generator map. application of TPSA.

Dobrin Kaltchev*

TRIUMF 4004 Wesbrook Mall, Vancouver, B.C., Canada V6T2A3

Received 9 July 2008; received in revised form 9 July 2008; accepted 9 July 2008

Abstract

The Lie Algebra package LieMath, written in the *Mathematica* language, constructs the one-turn nonlinear map for a given lattice of optical elements. The method used is a BCH-based map concatenation. Truncated power series algebra (TPSA) techniques have been implemented to compute the Poisson bracket and extract the map faster than when one relies on the symbolic capabilities of *Mathematica* to operate on truncated multivariate Taylor series. In addition, this makes possible obtaining parameter-dependent numerical maps and optimization of nonlinear parameters.

PACS: 41.85.-p; 02.70.Wz

Keywords: Beam optics, Lie algebra, Nonlinear maps

1. Introduction

The Lie algebraic method allows us to create the nonlinear map of a beamline, either in Lie operator form, or as Taylor expansions of final coordinates in terms of initial coordinates. In a map concatenation based on the Baker-Cambell-Hausdorff (BCH) expansion, the most computationally intensive part is calculation of the Poisson brackets (PB). Therefore, differential algebra libraries are needed to carry out fast computations with polynomials and vector functions of polynomials. To build the map, a symbolic computational system may be used such as Mathematica [1], [2], [3]. The most straightforward approach involves encoding the PB as an operator on multivariate polynomials produced by the truncated series expansion of the (piecewise constant) Hamiltonian. One then relies on the symbolic engine to perform the product and derivative needed for the PB. Using a symbolic system gives additional flexibility: easy switching between numerical and analytical calculations and also getting analytical dependence on parameters [1],[5]. The main disadvantage is speed, which is slow for large lattices.

In automatic differentiation, or Truncated power series algebra (TPSA) [6–8], the array of Taylor coefficients in the expansion of a multivariate function is computed not by symbolic differentiation, or by numerical

* Corresponding author: +1-604-222-7319
E-mail address: kaltchev@triumf.ca

doi:10.1016/j.phpro.2008.07.113

approximations, but by manipulating the coefficient arrays of other (simpler) functions. In our case this means replacing the symbolic PB with an operator on coefficient arrays, called in this paper derivative structures (DS). These differ from the differential algebra vectors in [6] only in the order in which the entries are stored in the computer memory. Such an inexpensive modification of the code LieMath [3] increases its speed by a factor of around ten and also provides parameter dependent maps, while preserving its general Lie-algebraic algorithm.

Section 2 shows how to implement the pyramid structure of coefficients of multivariate polynomials and efficiently handle the index set manipulation. One needs to assign addresses for all possible monomials in a polynomial. We use a direct addressing with an index array and create and store reference arrays that list the indices needed for each operation. Such a "preprocessing" is performed only once, at the beginning of execution. This method saves computing time at some expense of space and only requires implementation of a few manipulations on vectors of integers.

Section 3 describes the algorithm of LieMath. Comparisons with other codes and sample applications can be found in [3] and [4].

An important notice is that both the choice of addressing and the maximum achievable improvement in computing time (Section 5) have been substantially influenced by the Mathematica's built-in means of handling sparse structures.

2. Implementation of TPSA

This section describes the algorithm allowing one to replace the operations on polynomials in the Poisson bracket with operations on coefficient arrays (derivative structures). For a beamline of optical elements with lengths L_p and Hamiltonians $\boldsymbol{H}_p$, $p = 1, 2, ...$, the polynomials in question are the truncated at order m expansions of the individual-element Lie generators – $L_p\boldsymbol{H}_p(x)$ in the components of x. The coordinate vector x is of dimension n and, besides the six phase-space coordinates, may include some parameters, i.e. $n > 6$ in the case of maps with knobs described in Section 4. If a parameter dependence is not sought for, then $n = 6$ and the order of the Taylor map to be generated is $M = m - 1$.

2.1. Pyramid of Coefficients – Derivative Structure

The expansion of a function of n variables $x = \{x_1, x_2, \ldots, x_n\}$, truncated at order m, is

$$f(x) = \sum_{|\boldsymbol{k}|\le m} F_{\boldsymbol{k}} x_1^{k_1} \cdots x_n^{k_n} = \sum_{|\boldsymbol{k}|\le m} F_{\boldsymbol{k}} \boldsymbol{x}^{\boldsymbol{k}} \qquad (1)$$

$$|\boldsymbol{k}| = k_1 + k_2 + \ldots + k_n.$$

Here $\boldsymbol{k} = \{k_1, k_2, \ldots k_n\}$ is the index vector: a vector of nonnegative integers indicating the term in the series and F_k is the coefficient array or (with accuracy to factorial factors) the array of all partial derivatives up to order m. This array can be visualized as an n-dimensional pyramid where n specifies the dimension and m the size [7]. The total number of entries is given by the binomial coefficient: $L = \binom{m+n}{m}$ In what follows, we call F_k the derivative structure, or simply the pyramid of f.

The index set as an array joining all index vectors: $\Gamma_n^m \equiv \{\boldsymbol{k}_1, \boldsymbol{k}_2, \cdots \boldsymbol{k}_L\}$. Let p be an integer or zero. A composition of p of order n is a particular arrangement of n nonnegative integers whose sum is p. We build the index set in the following way:

$$\Gamma_n^m = \texttt{Join}\quad \texttt{Compositions[p,n]}$$
$$(p = 0, 1, \cdots, m) \qquad (2)$$

Where `Compositions[p,n]` [10] are all compositions for fixed p. For example, the expansion ($n = 3$, $m = 2$):

$$F_{\{0,0,0\}}1 + F_{\{1,0,0\}}x_1 + F_{\{0,1,0\}}x_2 + F_{\{0,0,1\}}x_3 \\ +F_{\{2,0,0\}}x_1^2 + F_{\{1,1,0\}}x_1x_2 + F_{\{1,0,1\}}x_1x_3 \\ +F_{\{0,2,0\}}x_2^2 + F_{\{0,1,1\}}x_2x_3 + F_{\{0,0,2\}}x_3^2$$

the index set (with compositions shown on different rows) is:

$$\Gamma_3^2 = \{\ \{0,0,0\}, \\ \{0,0,1\},\{0,1,0\},\{1,0,0\}, \\ \{0,0,2\},\{0,1,1\},\{0,2,0\},\{1,0,1\},\{1,1,0\},\{2,0,0\}\ \} \qquad (3)$$

The dimensions *n* and *m* are fixed at the start of the program, hence all pyramids have the same index set which can be represented as a matrix $\Gamma_{l,i} = (\boldsymbol{k}_l)_i$.

Although in Mathematica it is not a syntactic problem to index directly one array with another, even when their dimensions are arbitrary, we choose to work with a linear index *l* and a linear array *F(l)*, where *l* has a corresponding entry $F(l) = F_{\boldsymbol{k}l}$. For example in (3) $\Gamma_{3,*} = \{0, 1, 0\}$ and $F(3) = F_{\{0,1,0\}}$.

2.2. TPSA Poisson bracket

All polynomials (*f, h, u, . . .*) are associated with corresponding pyramids (*F,H, U, . . .*), all indexed with the same set Γ. If *h* is given as an arithmetic operation combining *u* and *v* and *U* and *V* are known, then one needs to define a corresponding operation on *U* and *V* that yields *H*. For the Poisson bracket, such operations are only the product and derivative since addition and multiplication by a number are performed coordinatewise, as in matrix addition and multiplication by a constant. Preprocessing (see the Introduction) of an operation means creating subsets of the index array and storing them together with information on what to do with the subset. Such a subset corresponds to a group of pyramid entries called a box. When the product or derivative is actually called, only the box elements are operated upon. For the product of polynomials, the elements of `H=PROD[U,V]` are found by the Leibniz' rule. From Eqn (1) substituted into $h(\boldsymbol{x}) = u(\boldsymbol{x})v(\boldsymbol{x})$:

$$h(\boldsymbol{x}) = \sum_{\boldsymbol{j}} U_{\boldsymbol{j}}\boldsymbol{x}^{\boldsymbol{j}} \sum_{\boldsymbol{i}} V_{\boldsymbol{i}}\boldsymbol{x}^{\boldsymbol{i}} = \sum_{\boldsymbol{k}} \left(\sum_{\boldsymbol{j}\le\boldsymbol{k}} U_{\boldsymbol{j}}V_{\boldsymbol{k}-\boldsymbol{j}} \right) \boldsymbol{x}^{\boldsymbol{k}}$$

we get:

$$H_{\boldsymbol{k}} = \textstyle\sum_{\boldsymbol{j}\le\boldsymbol{k}} U_{\boldsymbol{j}}V_{\boldsymbol{k}-\boldsymbol{j}} \qquad (4)$$

A similar rule can be deduced for the derivative with respect to the i^{th} th variable: $h(\boldsymbol{x}) = \frac{\partial f(x)}{\partial \boldsymbol{x_i}}$, which defines the derivative operator: `H=DER[F,i]`. Implementation details are given in the Appendix. The Poisson bracket can be defined as

```
PBDS[F, G] := PROD[DER[F,1], DER[G,2]] - PROD[DER[F,2], DER[G,1]] + ...   (5)
```

or, in a similar fashion, it can be encoded via its own reference arrays.

2.3. Other method of indexing

A question is in order whether the above method is optimum for sparse arrays which occur since in some polynomials only a few of the variables are present. Therefore, we have tried the same addressing as for the differential algebra vector in [6], i.e. according to decimals in base *m*+1. This brings the advantage that, for the polynomial product for example, the resultant coefficient is automatically stored into the correct location thus avoiding multiple operations on zeros. No improvement in speed is observed, attributed to the already built in abilities of *Mathematica* to operate on sparse arrays.

3. LieMath Code

The input to LieMath conforms closely to the standard format for accelerator lattices [16], made possible by a lattice parser written in the same language. The map transforms a vector of canonical coordinates $\{x, p_x, y, p_y, c\tau, p_\tau\}$ – deviations from the reference trajectory with design energy E_0, momentum p_0 and local curvature $h(s)$ (s is the path length along the reference trajectory). The motion of a particle with charge e and energy E, assumed to be a constant, is governed by the Hamiltonian:

$$H = -(1+hx)\left[1 - \frac{2p_\tau}{\beta_0} + p_\tau^2 - p_x^2 - p_y^2\right]^{1/2} - \frac{p_\tau}{\beta_0} - \frac{eA_S}{p_0} \tag{6}$$

where the Hamiltonian and momenta have been scaled by the design momentum, $\tau = t - t_0$ is the time of flight relative to the reference particle and $p_\tau = -(E - E0)/p_0c$. Above $A_s = \vec{A}.\vec{s}(1+hx)$ (the canonical vector potential) with $\vec{A}$ being the vector potential and $\vec{s}$ the unit vector in direction tangent to the reference trajectory. For a drift, bend (curvature h), quadrupole (strength k_1) and sextupole (strength k_2), $\vec{A}.\vec{s}$ is given by (see. eg. [13]):

$$-hx + \frac{(h^2+k_1)x^2}{2} - \frac{k_1 y^2}{2} - \frac{xy^2k_2}{2} + \frac{x^3(-3h^3 + k_2 - hk_2)}{6}$$

The algorithm is as outlined in [3], [11], [14] with the BCH formula replaced by its DS equivalent BCHDS. At first, Γ is created and the generators $-L_p\boldsymbol{H}_p(x)$ are series-expanded to order $m = M + 1$ and converted into pyramids. The kick factorization of an element is made by replacing it with a Lie operator and a following linear matrix. All matrices are moved to the end of the lattice, which changes each Lie operator with a similarity transform. Next, all nonlinear generators are combined into one with BCHDS applied in a loop over the elements.

$$BCHDS[F,S] = F + G + \frac{1}{2}:F:G + \frac{1}{12}:F:^2 G + \frac{1}{12}:G:^2 F + \cdots + \frac{1}{120}:G:^2:F:^2 G + O((F,G)^6);$$
$$:A:B \equiv PBDS[A,B], \tag{7}$$

where F and G are the DS of two polynomials whose linear terms are removed and $O((f, g)^6)$ means terms of order ≥ 6. Applying on this generator the formal expansion of the Lie exponent (in DS form) yields the Taylor map. For periodic cells this one-turn map is converted to a normal form by using symbolic transformations, see [5].

4. Numerical nonlinear optimization

One natural extension of the above algorithm allows us to get numerical maps depending on nonlinear parameters (maps with knobs [9]). Consider a beamline for which Q parameters nl_1, nl_2, . . . , nl_Q have been declared to be variables. These may be the strengths of some nonlinear elements. To create dependence (as a power expansion) on these parameters, the above names are appended to the coordinate vector:

$$x = \{x, p_x, y, p_y, c\tau, p_\tau, nl_1, nl_2, \ldots nl_Q\} \tag{8}$$

and the value of m is increased. All steps in the algorithm in Section 3 are preserved: for the new m and $n = 6+Q$, Γ is constructed and the DS are filled in by differentiating analytically the Hamiltonians with respect the new x. The final map elements depend on parameters and can be optimized numerically. For instance, increasing the map order from two to three will produce linear dependence on sextupole strengths, making it possible to match second order map elements and linear chromaticities.

5. Computing time

The computing time (`TIMING` command) per one `BCHDS` call is independent on the kind of optical elements concatenated. On a 1-GHz processor, for six-dimensional numerical maps of third order, this time is 0.22 seconds.

For the same maps (n=6, M=3) the gain in total execution time due to TPSA is around one order of magnitude even if (as it is logical to assume) the result from the symbolic BCH is truncated at order m after each element. Table 1 shows timings of the truncated BCH and BCHDS loops for a beamline of 70 elements including strong interleaved sextupoles [1].

Table 1
CPU time (seconds) of the concatenation loop for a lattice of 70 elements ([1])

M	BCHDS	truncated symbolic BCH
3	15	125
4	52	846

For large n and m the effect of implementing TPSA increases (Table 1), which is explained with the increasing fraction of nonzero coefficients. By taking now the untruncated BCH, Figure 1 shows the ratio of times versus the pyramid fraction populated with non-zero real numbers (the nonzero entires are chosen randomly). The plot illustrates the ability of *Mathematica* to operate efficiently on sparse numerical arrays. The ratio reaches several orders of magnitude for large and dense pyramids (m=5, filling fraction >0.5). The maximum effect is observed when all coefficients are symbols (filling fraction =1).

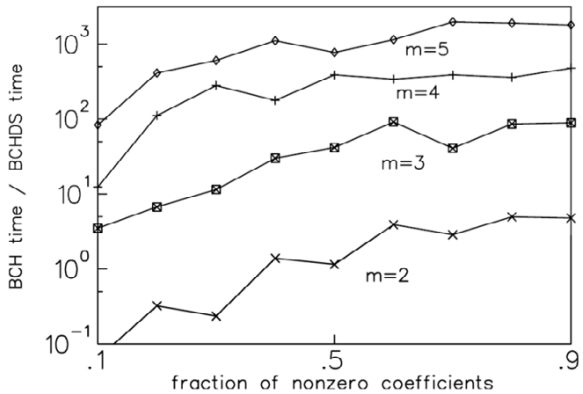

Fig. 1. Time needed for one call of the untruncated BCH divided by the same time for BCHDS. Here n=4.

References

[1] N. J. Walker, J. Irwin, M. Woodley, Analysis of Higher Order Optical Aberrations in the SLC Final Focus, using Lie Algebra Techniques, Proc. Of PAC 1993.
[2] J. Irwin, Analytic Nonlinear Methods for Beam Optics, in Proc. PAC (1997).
[3] D. Kaltchev, Building Truncated Taylor Maps with Mathematica and Applications to FFAG, in Proc. EPAC (2004).
[4] D. Kaltchev,Implementation of TPSA in the Mathematica Code LieMath, in Proc. EPAC (2006).
[5] Chunxi Wang and Alex Chao, Analytic Second- and Third-Order Achromat Designs, Proc. of PAC (1995).

[6] Berz, M. Differential algebraic description of beam dynamics to very high orders, Particle Accelerators 24, 109 (1989)
[7] Richard Neidinger, An efficient method for the numerical evaluation of partial derivatives of arbitrary order, ACM Transactions on Mathematical Software (TOMS) (1992).
[8] D. Kalman, R. Lindell, Recursive Multivariate Automatic Differentiation, Optimization Methods and Software, 6 (1995) 161.
[9] M. Berz and K. Makino, COSY Infinity Version 8.1, http://cosy.pa.msu.edu
[10] The Mathematica Book, Wolfram Media, Fifth edition, 2003
[11] Tanaji Sen, Y.T. Yan, J. Irwin, Liemap : A Program for Extracting a Oneturn Single Exponent Lie Generator map, IEEE 1991 Particle Accelerator Conference, San Francisco, May 1991.
[12] J. Irwin, Computation of Lattice maps Using Modular BCH and Similarity Composition Rules, PAC 5 (1995) 2871.
[13] Johan Bengtsson, Doctorate Thesis CERN 88-05.
[14] A. Dragt and E. Forest, Computation of nonlinear behavior of Hamiltonian systems using Lie algebraic methods, J. Math. Phys. 24 (12), (1983).
[15] J. Murray, K. Brown, T. Fieguth, The Completed Design of the SLC Final Focus System , SLAC-PUB-4219 (1987).
[16] D. Carey, F. Iselin,Standard Input Language for Particle Beam and Accelerator Computer Programs, Proc of Snow-mass, Colorado, (1984).

6. Appendix

Preprocessor of product: The product is defined by Eqn. 4. For each $k \leftrightarrow l$ we find all vectors $\boldsymbol{j}$ that are smaller or equal to k coordinate-wise and store their linear indices in the reference array $r_{l,j}$; $l = 1, \ldots, L$, $j = 1, \ldots, J^{(l)}$. Here "coordinate-wise" means $\boldsymbol{j} \le \boldsymbol{k}$ if and only if $j_i \le k_i$ for all $i = 1, 2, \ldots n$. For a fixed l, the vector $r_{l,*}$ defines the lth box.

Product: $H(l)$ is the scalar product between the lth box for U and the same box for V, but with its order inverted:

Hk _ H(l) =
J(l) X
j=1
U(rl,j)V (rl,J(l)−j+1).

For example, in (3) $\Gamma_{5,*} = \{0, 0, 2\}$ and one finds $r_{5,*} = \{1, 2, 5\}$. Hence $H(5) = U(1)V(5) + U(2)V(2) + U(5)V(1)$, or by going back to multi-indices: $H(5) = U_{\{0,0,0\}}\, V\{0,0,2\} + U_{\{0,0,1\}}\, V_{\{0,0,1\}} + U_{\{0,0,2\}}\, V_{\{0,0,0\}}$.

Preprocessor of derivative: The subset of indices which defines the ith box consists of all vectors $\boldsymbol{k}$ that are present in both $\Gamma_{l,i}$ and $\Gamma_{l,i+1}$. We again denote this subset by $r_{i,j}$, where $j = 1, \ldots, J$.

Derivative: The first J entries of H are:

$$H(j) = (\Gamma_{j,i} + 1)F(r_{i,j}),\ 1 \le j \le J \qquad \text{(A.1)}$$

and the rest (for $j > J$) are zero.

For example, to compute `DER[F,1]` we add unity to all first elements of n-vectors in (3). One finds $r_{1,*} = \{4, 8, 9, 10\}$ and `DER[F, 1]` $= \{F_{\{1,0,0\}}, F_{\{1,0,1\}}, F_{\{1,1,0\}}, 2\,F_{\{2,0,0\}}, 0, \ldots, 0\}$.