Truncated Power Series Algebra

April 29, 2010, Alex Chao

Introduction

In the study of nonlinear dynamics, high-order nonlinear maps are extremely useful:

- To study long-term stability of particle motion (dynamic aperture) by tracking with nonlinear maps.

- To extract various nonlinear dynamics quantities analytically (Courant-Snyder, Lie algebra, etc.) from the nonlinear one-turn map.

But, regardless of the purposes, question is: How to generate these high order maps efficiently in the first place?

One answer: TPSA technique, first introduced for accelerators by M. Berz, 1989. Since then, a large number of computer codes/libraries were written using TPSA.

TPSA is a powerful computation tool, not only for accelerators but also for any computational algorithm which relates some outputs to some inputs:

Once the algorithm is given, TPSA allows one to generate power series expressions of the outputs in terms of the inputs. The order of the power series Ω is limited only by memory space of a computer and is specified by the user.

Note, however, that TPSA does not contain physics; it is a computational technique.

What algorithms? The answer is any algorithm that calculates definitively the numerical values of Y once the numerical values of X are given.

Examples

- Explicit functions
  $y_i = y_i(x_1, x_2, \ldots, x_n)$
- A computer code with $X=(x_1, x_2, \ldots, x_n)$ as inputs, and $Y=(y_1, y_2, \ldots, y_m)$ as outputs. This computer code can be 10,000 lines, and parallel-processing.
- Random number generation is included if it is definitive.

This algorithm is a map $X \rightarrow Y$. We want to approximate the map by a Taylor expansion up to order Ω,

$$Y \approx \sum_{j}^{\Omega} C_j X^j$$

i.e. a truncated Taylor map from X to Y. TPSA technique is a way to calculate the coefficients C_j once the original algorithm is specified.

This nonlinear map then replaces the original algorithm (at least approximately).

In accelerators, one might (but does not have to) have $X=(x,x',y,y',z,\delta)$ of a particle at some starting position in a storage ring. By following X to obtain $Y=(x,x',y,y',z,\delta)$ through one beamline element, we obtain a single-element nonlinear map. By following X for one revolution, one obtains the nonlinear one-turn map.

If you have a tracking program, TPSA allows you to extract one-turn Taylor map $X \rightarrow Y$ up to some order Ω (e.g. $\Omega=10$). Subsequent tracking using this Taylor map is likely to be much faster than the original tracking code.

Note that

- if the original program contains a bug, the TPSA would still work, but the resulting Taylor map contains the same bug.
- the original tracking code relates Y to X numerically, while TPSA map relates Y to X algebraically.
- It is obvious that the coefficients C_j are related to the derivative of Y with respect to X. So TPSA really allows one to calculate the high order derivatives of the outputs with respect to the inputs,

For example, you can calculate

$$\frac{\partial^{10} y_5}{\partial^3 x_1 \partial^3 x_2 \partial^4 x_6}$$

Using TPSA -- not easy to do for a 10,000-line tracking code!

Consider a single input *x* and single output. Let *y* be expressed as a truncated Taylor series around a reference point $x=a$,

$$y = f(x) = f(a) + (x-a)f'(a) + \frac{1}{2}(x-a)^2 f''(a) + ... \quad + \frac{1}{\Omega!}(x-a)^{\Omega} f^{(\Omega)}(a)$$

TPSA will calculate all the derivatives of *f(x)* at $x=a$. Both Ω and *a* are specified by the user.

Note the one-to-one equivalence between the function *f(x)* and the vector

$$(f(a), f'(a), \ldots f^{(\Omega)}(a))$$

The vector, with $\Omega+1$ coefficients, is the TPSA representation of the function.

One could imagine calculating the derivatives numerically:

$$f'(a) \approx \frac{f(a+\epsilon) - f(a)}{\epsilon}$$

but loses accuracy rapidly for high order derivatives. TPSA is so remarkable because it does not subtract two nearly equal numbers, and calculates high order derivatives to 12 digits!

Before TPSA, high order maps of magnet elements were obtained by solving the particle's equation of motion. This yielded long analytic expressions, and were limited to low orders (TRANSPORT 2nd order, MARYLIE 3rd order, COSY 5th order). TPSA makes this approach obsolete.

After truncation, the Taylor map is generally nonsymplectic. This Taylor map needs to be symplectified. One way is to apply Lie algebra. The combined application of TPSA and Lie algebra revolutionized nonlinear dynamics research in accelerators in 1980's.

TPSA

Consider

$$f(x) = \frac{1}{x + \frac{1}{x}}$$

Suppose we want to find the derivative *f'(2)*.

Method 1: Do it analytically

$$f'(x) = -\frac{1 - \frac{1}{x^2}}{(x + \frac{1}{x})^2} \quad \rightarrow \quad f'(2) = -\frac{3}{25}$$

But this is not always doable.

Method 2: Do it numerically.

$$f'(2) \approx \frac{f(2.1) - f(2)}{2.1 - 2} = \frac{0.38817 - 0.4}{2.1 - 2} = -0.1183$$

But one loses accuracy quickly.

To compute *f'(2)* using TPSA, let us first form a vector *v=(2,1)* and try to find *f(v)*. The first component is *2* is because we want to compute *f'(2)*. The second component is always *1*.

$$f(v) = \frac{1}{v + \frac{1}{v}}$$

As we will establish later, the vectors are manipulated according to the rules:

$$\frac{1}{(a_1, a_2)} = (\frac{1}{a_1}, -\frac{a_2}{a_1^2}) \quad (11)$$

$$(a_1, a_2) + (b_1, b_2) = (a_1 + b_1, a_2 + b_2) \quad (12)$$

Thus,

$$f(v) = \frac{1}{(2,1) + \frac{1}{(2,1)}} = \frac{1}{(2,1) + (\frac{1}{2}, -\frac{1}{4})}$$

$$= \frac{1}{(\frac{5}{2}, \frac{3}{4})} = (\frac{2}{5}, -\frac{3}{25})$$

The two components in the final vector are miraculously equal to *f(2)* and *f'(2)*!

Nowhere explicit expressions of *f'(x)* was used. Nowhere subtraction of nearly-equal numbers was executed. The fact that is possible is counter-intuitive, but has a deep mathematical origin.

The simplest version of the TPSA technique is thus

$$f(v) = (f(a), f'(a)), \qquad \text{for any function } f(x), \quad \text{and } v = (a, 1) \qquad (14)$$

The secret of TPSA is contained in the two vector manipulation rules. How are those rules established?

Step 1

Consider the simplest case: the identity function f*(x)* =*x*. We of course want (14) to hold for this function. Let *v=(α,β)*, then LHS = *f(v)* = *v* = *(α,β)*.

We want this to be equal to RHS = *(f(a), f'(a))*, but we know *f(a)=a* and *f'(a) =1*. This means we must choose the input *(α,β) =(a,1)*. This is what we did.

It is easy to show that the constant function *f(x)=c* satisfies (14).

Step 2

Suppose we now have two functions $f(x)$ and $g(x)$, each satisfying (14). We now want to establish a rule which allows the new function $h(x)=f(x)+g(x)$ to satisfy (14) also.

LHS = $h(v) = f(v) + g(v) = (f(a), f'(a)) + (g(a), g'(a))$

RHS = $(f(a) + g(a), f'(a)+g'(a))$

Obviously $h(x)$ satisfies (14) if and only if we establish the vector addition rule (12).

We now know that all functions of the type $f(x)=c+nx$ satisfy (14).

Step 3

With $f(x)$ and $g(x)$ satisfying (14), we now want $h(x) = f(x)\, g(x)$ to satisfy (14).

LHS $= h(v) = (f(a), f'(a))\ (g(a), g'(a))$
RHS $= (h(a), h'(a)) = (f(a)g(a), f(a)g'(a)+f'(a)g(a))$

Thus we require the vector multiplication rule

$$(a_1,a_2)\ (b_1,b_2) = (a_1b_1,\ a_2b_1+a_1b_2)$$

Take $f(x)=g(x)=x$ ➔ $h(x)=x^2$ satisfies (14) ➔ $h(x) = x^n$ satisfies (14)

The two rules together then ➔ All power series of x satisfy (14).

We have thus established the TPSA for any arbitrary function which is expandable into a Taylor series. Just substitute *v=(a,1)* into *f(v)* and follow two simple rules to obtain *f(a)* and *f'(a)*!

Note *v=(a,1)* is only the initial input of vector. Once inserted into *f(v)*, it of course no longer has the form *(a,1)*.

How about a division rule? Answer: use multiplication rule to obtain (11).

Let $\frac{1}{(a_1, a_2)} = (x, y)$, then

$$
\begin{aligned}
& (a_1, a_2)\ (x, y) = (1, 0) \\
\Longrightarrow \quad & (a_1 x, a_1 y + a_2 x) = (1, 0) \\
\Longrightarrow \quad & a_1 x = 1, \quad a_1 y + a_2 x = 0 \\
\Longrightarrow \quad & x = \frac{1}{a_1}, \quad y = -\frac{a_2}{a_1^2}, \qquad \text{Q.E.D.}
\end{aligned}
$$

Higher Orders

Higher derivatives are obtained by higher order TPSA.

To Ω-th order, we first form the vector $v = (a,1, 0,0,\ldots\ 0)$ with $\Omega+1$ elements. We then demand:

$$f(v) = (f(a), f'(a), f''(a), \ldots\ , f^{(\Omega)}(a))$$

Following similar 3 steps as before, it is easy to establish the higher order sum and multiplication rules:

$$(a_0, a_1, a_2, \ldots\ , a_\Omega) + (b_0, b_1, b_2, \ldots\ , b_\Omega) = (a_0 + b_0, a_1 + b_1, a_2 + b_2, \ldots\ , a_\Omega + b_\Omega)$$

$$(a_0, a_1, a_3, \ldots\ , a_\Omega)\ (b_0, b_1, b_3, \ldots\ , b_\Omega) = (c_0, c_1, c_3, \ldots\ , c_\Omega)$$

$$c_m = \sum_{k=0}^{m} \frac{m!}{k!(m-k)!} a_k b_{m-k}$$

$$c_0 = a_0 b_0, c_1 = a_1 b_0 + a_0 b_1, c_2 = a_2 b_0 + 2a_1 b_1 + a_0 b_2,\ \ldots$$

A constant c is identified as $(c,0,0, \ldots\ 0)$.

The remarkable thing is that once these two rules are established, high order derivatives of an arbitrary function *f(x)* can be computed by substituting *x* by *v=(a,1,0,0,...)* --- no subtracting of nearly-equal numbers and no analytic calculation of derivatives.

Once the high order derivatives are obtained, Taylor series expansion follows.

Special Functions

Is it really true that we only need two rules, addition & multiplication rules? What about special functions, e.g. e^x?

One can expand e^x into a power series, then substitute $x \rightarrow v$. But this power series is infinite series.

We will use a trick to calculate e^v exactly with a finite number of steps!

The trick is to note that any vector whose first component is zero is a "small" vector,

$$(0, \times, \times, \times, \ldots)^k = (0, 0, 0, \ldots \times, \times)$$

There are k zeros on the RHS. This means $(0,\times\times\times\ldots)$ raised to $(\Omega+1)$-th power is exactly zero in TPSA.

Using this trick, we obtain

$$
\begin{aligned}
e^{(a_0,a_1,a_2,\ldots\ a_\Omega)} &= e^{a_0}\sum_{k=0}^{\Omega}\frac{1}{k!}(0,a_1,a_2,\ldots\ a_\Omega)^k \\
\ln(a_0,a_1,a_2,\ldots\ a_\Omega) &= (\ln a_0,0,0,0\ldots,0) \\
&\quad +\sum_{k=1}^{\Omega}(-1)^{k+1}\frac{1}{k}(0,\frac{a_1}{a_0},\frac{a_2}{a_0},\ldots,\frac{a_\Omega}{a_0})^k \\
\sqrt{(a_0,a_1,a_2,\ldots\ a_\Omega)} &= \sqrt{a_0}\left[(1,0,0,0\ldots0)+\frac{1}{2}(0,\frac{a_1}{a_0},\frac{a_2}{a_0},\ldots,\frac{a_\Omega}{a_0})\right. \\
&\quad \left.+\sum_{k=2}^{\Omega}(-1)^k\frac{(2k-3)!!}{(2k)!!}(0,\frac{a_1}{a_0},\frac{a_2}{a_0},\ldots,\frac{a_\Omega}{a_0})^k\right] \\
\sin(a_0,a_1,a_2,\ldots\ a_\Omega) &= \sin a_0\sum_{k=0}\frac{(-1)^k}{(2k)!}(0,a_1,a_2,\ldots,a_\Omega)^{2k} \\
&\quad +\cos a_0\sum_{k=0}\frac{(-1)^k}{(2k+1)!}(0,a_1,a_2,\ldots,a_\Omega)^{2k+1} \\
\cos(a_0,a_1,a_2,\ldots\ a_\Omega) &= \cos a_0\sum_{k=0}\frac{(-1)^k}{(2k)!}(0,a_1,a_2,\ldots,a_\Omega)^{2k} \\
&\quad -\sin a_0\sum_{k=0}\frac{(-1)^k}{(2k+1)!}(0,a_1,a_2,\ldots,a_\Omega)^{2k+1}
\end{aligned}
$$

All series terminate, and all can be evaluated readily by applying the two rules.

Multiple input and output variables

So far we discussed a single input x and a single output y. We now generalize to multiple variables.

Multiple outputs is trivial. Each output can be treated independently of the other output variables. We concentrate on just one of them and consider $(x_1,x_2, ..., x_n) \rightarrow y$.

We need vectors of much larger dimension. To keep track of the multiple indices, while using minimum computer storage, is extremely complex for TPSA codes --- the code must work for arbitrary number of input and output variables, to arbitrary order Ω.

We will illustrate only with the case $(x_1,x_2) \rightarrow y$

The TPSA requires that when input x_1 is substituted by vector v_1 and x_2 by v_2, we want to obtain an output vector

$$\begin{aligned} y(v_1, v_2) &= \Big(y(a_0, b_0), \frac{\partial y}{\partial x_1}(a_0, b_0), \frac{\partial y}{\partial x_2}(a_0, b_0), \\ & \frac{\partial^2 y}{\partial x_1^2}(a_0, b_0), \frac{\partial^2 y}{\partial x_1 \partial x_2}(a_0, b_0), \frac{\partial^2 y}{\partial x_2^2}(a_0, b_0), \\ & .... , \frac{\partial^\Omega y}{\partial x_2^\Omega}(a_0, b_0) \Big) \end{aligned} \qquad (51)$$

where a_0,b_0 are prespecified reference positions for x_1,x_2. All derivatives are to be evaluated at (a_0,b_0).

For (51) to hold for the two identity functions $y(x_1,x_2)= x_1$ and $y(x_1,x_2)= x_2$, we must choose the initial vectors

$$\begin{aligned} v_1 &= (a_0, 1, 0, 0, 0, ...0) \\ v_2 &= (b_0, 0, 1, 0, 0, ...0) \end{aligned}$$

The vector addition and multiplication rules are

$$
\begin{aligned}
& (a_{00}, a_{10}, a_{01}, a_{20}, a_{11}, a_{02}, \cdots, a_{0\Omega}) + (b_{00}, b_{10}, b_{01}, b_{20}, b_{11}, b_{02}, \cdots, b_{0\Omega}) \\
= \quad & (a_{00} + b_{00}, a_{10} + b_{10}, a_{01} + b_{01}, \cdots, a_{0\Omega} + b_{0\Omega}) \qquad (53) \\
& (a_{00}, a_{10}, a_{01}, a_{20}, a_{11}, a_{02}, \cdots, a_{0\Omega})\ (b_{00}, b_{10}, b_{01}, b_{20}, b_{11}, b_{02}, \cdots, b_{0\Omega}) \\
= \quad & (c_{00}, c_{10}, c_{01}, c_{20}, c_{11}, c_{02}, \cdots, c_{0\Omega})
\end{aligned}
$$

$$
c_{mn} = \sum_{s=0}^{m} \sum_{t=0}^{n} a_{st} b_{m-s,n-t} \frac{m!n!}{s!(m-s)!t!(n-t)!} \qquad (54)
$$

To find a high order derivative, we substitute v_1 and v_2 into $y(x_1,x_2)$, applying (53,54), and the output vector contains the result.

Applications

1. Consider (x,x',y,y',z,δ) at some starting location as inputs. Consider (x,x',y,y',z,δ) of one turn later as outputs. A nonlinear Taylor map is then obtained. This map can be used to track particles, or to calculate some nonlinear dynamics quantities (such as tune shifts with betatron amplitudes, nonlinear chromaticities, nonlinear resonance strengths) analytically. Examples: SSC 6-D 10-th order one-turn map, high order map over PEP-2 interaction region.

2. Find the dependence of the one-turn nonlinear map on the strength S of some special magnet. Include S as one of the input variables. The map obtained is a Taylor expansion in S in addition to the other input variables. Can be useful for example to study sensitivity to magnet errors.

3. Consider strength S of a quadrupole in a lattice design as one input variable. This allows varying S to match the β-functions or the betatron tunes if they are chosen as output variables.

4. Other applications.