RUNGE-KUTTA DA INTEGRATOR IN MATHEMATICA LANGUAGE

D. Kaltchev and R. Baartman, TRIUMF, Vancouver, Canada*

Abstract

The method of Truncated Power Series Algebra is applied in a `Mathematica` code to compute the transfer map for arbitrary equations of motion describing a charged particle optical system. The code is a non-symplectic integrator – a combination between differential algebra module and a numerical solver of the equations of motion. Using the symbolic system offers some advantages, especially in the case of non-autonomous equations of motion (element with fringe-fields). An example is given – a soft-fringe map of a magnetic quadrupole.

INTRODUCTION

We have developed a `Mathematica` code to compute the transfer map for arbitrary equations of motion (EOM) describing a charged particle optical system. The code is a non-symplectic integrator – a combination between differential algebra (DA) module and numerical solver of EOM, which is is a well-established technique in map-building codes – most notably `COSY-`∞[1]. For magnetic quadrupoles with fringe fields, the code has been benchmarked against numerical integration of individual trajectories and against high-order maps generated with `COSY-`∞.

Although some form of forward *automatic differentiation* (AD) has already been implemented in `Mathematica`'s intrinsic operations on multivariate series, we preferred to create our own stand-alone package `ad_tools.m`. It uses the Truncated power series algebra (TPSA) technique – [2], [3], [4], and can operate on polynomials of arbitrary number of variables to an arbitrary order.

This paper discusses how to create a DA-integrator package by introducing only minor modifications in some conventional numerical solver. We use the 4-5 order Runge-Kutta-Fehlberg algorithm – `rkf.m`. Replacing it with a higher order Runge-Kutta algorithm is expected to improve its performance [5].

For systems with fringe-fields (non-autonomous EOM), the dependence on the path-length along the reference trajectory s is usually described via some auxiliary functions approximating data from calculations or magnetic measurements. The final map depends on the high-order derivatives of these functions. Using an analytical computational system offers an advantage in this case since the right-hand sides of EOM, in their differential algebra form, contain only *analytic* expressions for the above derivatives. In such a way, no accuracy is lost and the approximating functions may be arbitrary.

* TRIUMF receives funding via a contribution agreement through the National Research Council of Canada.

The alternative Lie algebraic approach [6] is based on results obtained by Dragt and Forest and consists in solving numerically the ordinary differential equations for the Lie polynomials.

AUTOMATIC DIFFERENTIATION (`AD_TOOLS.M`)

Automatic differentiation (AD) finds the derivative of an expression without finding an expression for the derivative. There are no small-difference approximations and the result is accurate to machine precision. For this, expressions are replaced with their unique truncated Taylor series, i.e. multivariate polynomials. Consider for example this expansion containing all powers of $m = 2$ dynamical variables up to order $n = 2$:

$$U_{\{0,0\}} + p_x U_{\{0,1\}} + p_x^2\, U_{\{0,2\}} + x\, U_{\{1,0\}} + p_x\, x\, U_{\{1,1\}} + \\ + x^2\, U_{\{2,0\}} \tag{1}$$

The set of coefficients $U_{\vec{k}}$ represents a differential algebra (DA) variable [2] – we denote it by U. Assuming $U_{\vec{k}}$ are real, the multi-index $\vec{k}$ is used to define an address where a coefficient is stored in some array (of computer memory). This storage array may have a linear shape:

$$U \equiv \{U_{\{0,0\}}, U_{\{0,1\}}, U_{\{0,2\}}, U_{\{1,0\}}, U_{\{1,1\}}, U_{\{2,0\}}\} \tag{2}$$

and then a numerical constant c is represented with $\{c, 0, 0, 0, 0, 0\}$ and the dynamic variable x, with $\{0, 0, 0, 1, 0, 0\}$. In a similar way, all dynamic variables $x, p_x, y, p_y \ldots$ can be replaced by corresponding DA variables. Moreover, arbitrary parameters describing the optical system may be added. Next, the AD (or TPSA) rules tell us how to replace any function of these variables by its DA counterpart. Such rules can be established by using the differential equation that the function obeys [3], which is the method chosen for `ad_tools.m`. For example, if U and V are DA variables (2), then the DA product and square root are computed from:

$$\texttt{PROD}[U,V] = \tag{3}$$
$$\{U_{\{0,0\}}\, V_{\{0,0\}}, U_{\{0,1\}}\, V_{\{0,0\}} + U_{\{0,0\}}\, V_{\{0,1\}}, U_{\{0,2\}}\, V_{\{0,0\}} + \\ + U_{\{0,1\}}\, V_{\{0,1\}} + U_{\{0,0\}}\, V_{\{0,2\}}, U_{\{1,0\}}\, V_{\{0,0\}} + U_{\{0,0\}}\, V_{\{1,0\}}, \\ U_{\{1,1\}}\, V_{\{0,0\}} + U_{\{1,0\}}\, V_{\{0,1\}} + U_{\{0,1\}}\, V_{\{1,0\}} + U_{\{0,0\}}\, V_{\{1,1\}}, \\ U_{\{2,0\}}\, V_{\{0,0\}} + U_{\{1,0\}}\, V_{\{1,0\}} + U_{\{0,0\}}\, V_{\{2,0\}}\}$$

$$\texttt{SQRT}[U] = \tag{4}$$
$$\left\{ \sqrt{U_{\{0,0\}}}, \frac{U_{\{0,1\}}}{2\sqrt{U_{\{0,0\}}}}, \frac{\frac{-U_{\{0,1\}}^2}{8\,U_{\{0,0\}}} + \frac{U_{\{0,2\}}}{2}}{\sqrt{U_{\{0,0\}}}}, \frac{U_{\{1,0\}}}{2\sqrt{U_{\{0,0\}}}}, \right. \\ \left. \frac{\frac{-U_{\{0,1\}}\, U_{\{1,0\}}}{4\,U_{\{0,0\}}} + \frac{U_{\{1,1\}}}{2}}{\sqrt{U_{\{0,0\}}}}, \frac{\frac{-{U_{\{1,0\}}}^2}{8\,U_{\{0,0\}}} + \frac{U_{\{2,0\}}}{2}}{\sqrt{U_{\{0,0\}}}} \right\}$$

Expressions as (3) and (4) have the same vector structure as (2), each element being the series coefficient or, with accuracy to some numerical factors, the partial derivative of a composite function. For example, by taking a square root of (1) and expanding – the result is (4). Obviously, (3) and (4) can easily be generated on a symbolic computational system (`Series` command in `Mathematica`). However the native `Mathematica` implementation has the disadvantage that the maximum number of variables n and the order of the polynomials m need to be fixed and one needs to prepare a dedicated notebook for every particular problem.

We have created a package `ad_tools`, where the result of practically arbitrary DA operation is built recursively. To see that this might be possible, choose some component of (3) or (4) and notice its position. The element found there does not depend on the elements in (2) that are on the right of this position. The package is less than 100 lines long, it has the same speed as the `Series` command and (with however a considerable programming effort) may be transfered to other languages. Other features of `ad_tools`, also absolute prerequisites for an automatic differentiation package (see [8]), are:

- m and n are arbitrary input parameters – all arrays resize automatically depending on the problem at hand;
- a method of indexing that provides speed;
- DA operations: `PRODuct`, `POWER`, `DERivative` (see [7]) and `DIVision`;
- functions of DA variables: `EXP`, `LOG`, `SQRT`, `SINCOS`;
- concatenation of polynomial maps.

RUNGE-KUTTA INTEGRATOR (`RKF.M`)

In practice any calculational algorithm which relates output quantities to initial quantities may be converted via TPSA to generate power series expansions of the output quantities in terms of initial ones.

Consider the initial value problem for an optical element with fringe fields and denote by $\mathbf{der}$ the vector of the right-hand sides (derivatives) of the non-autonomous EOM:

$$y'_i = \mathtt{der}_i(\boldsymbol{y}, s), \quad \boldsymbol{y}(s_{\rm ini}) = \boldsymbol{y}_0, \tag{5}$$

where $\boldsymbol{y} \equiv \{x, p_x, \dots\}$ and $'$ denotes derivative with respect to s. We choose a numerical solver of (5) which has automatic step size control and the ability to go backwards over the independent variable s, so that one call: $\mathtt{rkf}[s_{\rm ini}, s, \mathbf{der}]$, produces a numerical approximation to the exact solution $y(s)$.

To obtain a DA integrator, the first step is to replace all occurrences of y_i in $\mathbf{der}$ and in the initial condition in (5) with DA variables (capital cases): $Y_i = \mathtt{MK}(y_i)$, where `MK` is the "make-DA-variable" operator in `ad_tools`. For the x component, the result can be found in the previous Section. Next, one has to replace all nontrivial[1] arithmetic operations and elementary functions in $\mathbf{der}$ with their DA counterparts. The `Mathematica`'s substitution rules have been found very useful at this point and, omitting the details, we denote the result by $\mathbf{DER}(\boldsymbol{Y}, s)$. Now a single call $\mathtt{rkf}[s_{\rm ini}, s, \mathbf{DER}]$ generates the Taylor map of the system:

$$\mathcal{M}_{s_{\rm ini}\to s} \ .$$

If the right-hand-sides are power series in the components of $\boldsymbol{y}$, then all non-trivial DA operations that one needs are power and product. This is obvious for autonomous EOM. For non-autonomous EOM, such as (5), this is again possible if the s-dependence is described with an auxiliary function $\mathcal{K}(s)$, all derivatives of which are known to our computational system in analytic form. Now the right-hand sides of EOM are given by $\mathbf{der}(\boldsymbol{y}, \mathcal{K}(s), \mathcal{K}'(s), \mathcal{K}''(s), \dots)$ and contain polynomials with coefficients depending on $\mathcal{K}^{(p)}$, where p denotes the order of the derivative over s. Since the transformation $\mathbf{der} \to \mathbf{DER}$ involves no differentiation, but only consists in copying a coefficient into its correct position in the DA array, all functions $\mathcal{K}^{(p)}(s)$ enter $\mathbf{DER}$ in *symbolic* (and hence analytic) form making it possible to compute $\mathbf{DER}$ exactly at each point s.

If the right-hand-sides of EOM contain elementary functions of the components of $\boldsymbol{y}$, these need be encoded as DA operators in the same way as this was done with `SQRT` in the previous section.

EXAMPLE: HIGH ORDER SOFT-FRINGE MAP OF A MAGNETIC QUADRUPOLE

In this section we compute the third-order map for an on-momentum particle passing through a quadrupole with position-dependent gradient $k(s)$. The 4th order map terms are zero due to the symmetry. The vectors $\mathbf{der}$ and $\mathbf{der0} \equiv \mathbf{der}|_{k(s)\to 0}$ are derived via straightforward differentiation of this expanded Hamiltonian [9]:

$$\mathcal{H} = \frac{1}{2}\,(p_x^2 + p_y^2) + \frac{1}{2}\,k(s)\,(x^2 - y^2) + \frac{1}{8}\,(p_x^2 + p_y^2)^2 - \\ - \frac{1}{12}\,k''(s)\,(x^4 - y^4) - \frac{1}{4}\,k'(s)\,(xp_x - yp_y)(x^2 - y^2). \tag{6}$$

Consider a bell-shaped function $k(s)$ approaching a constant $k_0 = k(0)$ in the centre of the quadrupole. The effective field boundaries are at $s = -L$ and $s = L$ (effective magnetic length $2L$) and the full aperture is D. Since our intention is to compare with `COSY`-∞, describe the field shape with an Enge function E:

$$k[s] = k_0\,(E[s - L, D] + E[-s - L, D] - 1) \ ; \\ E[z, D] \equiv \frac{1}{1 + \exp\left[\sum_{k=0}^{5} a_k (\frac{z}{D})^k\right]}. \tag{7}$$

The transfer map from $-L$ to L is a composition (denoted by $\circ$) of three maps:

$$\mathcal{M}_Q = \mathcal{M}^{(0)}_{-L\to -s_\infty} \circ \mathcal{M}_{-s_\infty \to s_\infty} \circ \mathcal{M}^{(0)}_{s_\infty \to L} \tag{8}$$

[1] Addition and subtraction are trivial operations – they are performed coordinate-wise.

with $s = \pm s_\infty$ being some locations far from the origin ($s = 0$).

For a test, we take $k_0 = 1\,\mathrm{m}^{-1}$, $2L = 0.5\,\mathrm{m}$, $D = 0.05\,\mathrm{m}$, $s_\infty = 1.5L$ and choose the set of Enge coefficients a_k to be the COSY-∞ default fringe-field option FR=3 (Fig. 1 shows $k(s)$):

$$a_k = \{0.296471, 4.533219, -2.270982, 1.068627, -0.036391, 0.022261\}.$$

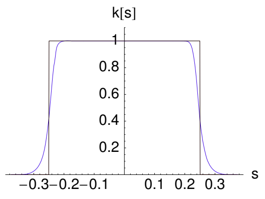

Figure 1: Function $k(s)$ and a stepwise function: $-L < s < L$.

The Taylor map (8) to fourth order is generated with the script in Figure 2 (right) and the result is shown in Figure 3 (zero elements are not shown). The corresponding COSY-∞ commands, Figure 2 (left), produce the map in Figure 4. They are in good agreement

```
ap:=0.05/2; k0:=1;
OV 4 2 0 ;
RPE 1000 ;
Bt:=k0^2*CHIM*ap;
UM ;
fr  3;
mq 0.5 Bt ap;
Pm 6 ;
```

```
n = 4; m = 4;
X={x,px,y,py};
<< ad_tools.m
<< rkf.m
Y = {MK[x],MK[px],
     MK[y],MK[py]};
rkf[-L,   -Sinf, DER0];
rkf[-Sinf, Sinf, DER ];
rkf[ Sinf, L   , DER0];
PrintMap[Y];
```

Figure 2: Scripts to compute the Taylor map of a magnetic quadrupole: COSY-∞ (left) and the rkf package (right). Here Sinf$\equiv s_\infty$ and DER0$\equiv$ DER $|_{k(s)\to 0}$.

```
x            px           y            py
0.877581     −0.479236    0            0            {1, 0, 0, 0}
0.479623     0.877581     0            0            {0, 1, 0, 0}
0            0            1.12762      0.521299     {0, 0, 1, 0}
0            0            0.52088      1.12762      {0, 0, 0, 1}
0.01138      −0.125797    0            0            {3, 0, 0, 0}
−0.157662    −0.154298    0            0            {2, 1, 0, 0}
−0.208893    0.187407     0            0            {1, 2, 0, 0}
0.202422     −0.00676766  0            0            {0, 3, 0, 0}
0            0            −0.0516499   −0.445424    {2, 0, 1, 0}
0            0            −0.0618326   −0.36368     {1, 1, 1, 0}
0            0            0.122494     0.220189     {0, 2, 1, 0}
0            0            0.251901     −0.0376499   {2, 0, 0, 1}
0            0            −0.0150236   −0.113106    {1, 1, 0, 1}
0            0            0.279616     0.108483     {0, 2, 0, 1}
−0.0621854   −0.507696    0            0            {1, 0, 2, 0}
−0.252471    −0.0774142   0            0            {0, 1, 2, 0}
−0.0616344   −0.362365    0            0            {1, 0, 1, 1}
−0.016282    −0.127517    0            0            {0, 1, 1, 1}
−0.127225    −0.276544    0            0            {1, 0, 0, 2}
0.217048     −0.142442    0            0            {0, 1, 0, 2}
0            0            0.0359988    −0.199175    {0, 0, 3, 0}
0            0            0.367646     −0.214046    {0, 0, 2, 1}
0            0            0.297545     −0.328965    {0, 0, 1, 2}
0            0            0.30663      −0.00807945  {0, 0, 0, 3}
```

Figure 3: Taylor map output – package rkf.m.

```
x             px             y             py
0.8775805     −0.4792355     0.000000      0.000000       1000
0.4796232     0.8775805      0.000000      0.000000       0100
0.000000      0.000000       1.127623      0.5212993      0010
0.000000      0.000000       0.5208796     1.127623       0001
0.01139819    −0.1257492     0.000000      0.000000       3000
−0.1576746    −0.1542672     0.000000      0.000000       2100
−0.2088959    0.1874384      0.000000      0.000000       1200
0.2024207     −0.006760539   0.000000      0.000000       0300
0.000000      0.000000       −0.05164986   −0.4454244     2010
0.000000      0.000000       −0.06183257   −0.3636798     1110
0.000000      0.000000       0.1224945     0.2201890      0210
0.000000      0.000000       0.2519014     −0.03764989    2001
0.000000      0.000000       −0.01502357   −0.1131063     1101
0.000000      0.000000       0.2796165     0.1084832      0201
−0.06218542   −0.5076964     0.000000      0.000000       1020
−0.2524710    −0.07741421    0.000000      0.000000       0120
−0.06163436   −0.3623650     0.000000      0.000000       1011
−0.01628207   −0.1275172     0.000000      0.000000       0111
−0.1272251    −0.2765446     0.000000      0.000000       1002
0.2170479     −0.1424416     0.000000      0.000000       0102
0.000000      0.000000       0.03598315    −0.1992755     0030
0.000000      0.000000       0.3676672     −0.2141294     0021
0.000000      0.000000       0.2975509     −0.3290218     0012
0.000000      0.000000       0.3066314     −0.008089533   0003
```

Figure 4: Taylor map output – COSY-∞.

REFERENCES

[1] M. Berz and K. Makino, COSY Infinity Version 8.1, http://cosy.pa.msu.edu

[2] Berz, M. " Differential algebraic description of beam dynamics to very high orders", Particle Accelerators 24, 109 (1989).

[3] R. Neidinger, "An efficient method for the numerical evaluation of partial derivatives of arbitrary order", ACM Transactions on Mathematical Software (TOMS), 1992.

[4] D. Kalman, R. Lindell, "A Recursive Approach to Multivariate Automatic Differentiation", Optimization Methods and Software, 6 (1995) 161-192.

[5] Berz, M. "Computational aspects of optics design and simulation: COSY INFINITY", NIM A298 (1990) 473-479.

[6] R. Ryne, A.J. Dragt, "Numerical Computation of Transfer maps using Lie Algebraic Methods", Proc. of PAC 1987.

[7] D. Kaltchev, "Implementation of TPSA in the Mathematica Code LieMath", EPAC'06, Edinburgh, July 2006

[8] http://www.autodiff.org/

[9] Irwin, J., Chun-Xi Wang, "Explicit soft fringe maps of a quadrupole" Proc. of PAC'95, May 1995.