Alternating Gradient Synchrotron Department
Relativistic Heavy Ion Collider Project
BROOKHAVEN NATIONAL LABORATORY
Upton, New York 11973

Spin Note

AGS/RHIC/SN No. 006
(January 19, 1996)

Application of a Differential Algebra Approach to a RHIC Helical Dipole

N. Malitsky

December 8, 1994

For Internal Distribution Only

RHIC/AP/51

Application of a adifferential algebra approach to a RHIC helical dipole.

Nikolay Malitsky

December 8, 1994

1 Introduction

This paper describes an object-oriented method, that enables one to obtain Taylor maps for arbitrary optical elements and include them in different accelerator algorithms. The approach is based on the differential algebraic (DA) technique, which in the accelerator physics was suggested by Martin Berz and implemented by him in the program COSY INFINITY[1]. In order to make an efficient use of DA operations, COSY INFINITY includes a special DA precompiler, which transforms arithmetic operations containing DA variables into a sequence of calls to FORTRAN subroutines. In ZLIB++[2], the object-oriented version of the numerical library for differential algebra, truncated power series and Taylor maps are considered as corresponding C++ classes **ZSeries** and **ZMap** with overloaded assignment, additive and multiplicative operators. These objects were implemented directly in the numerical integrator instead of DOUBLE variables and used to derive a Taylor map for the RHIC helical dipole.

2 DA Integrator

The particle motion in the magnetic field is described by the following set of equations, written in terms of the MAD coordinates:

$$
\begin{aligned}
\frac{d}{ds}x &= \frac{(1+hx)}{\frac{p_s}{p_0}}\left(\frac{p_x}{p_0}\right) \\
\frac{d}{ds}\left(\frac{p_x}{p_0}\right) &= \left[\frac{\vec{p}}{p_0}\times\left(\frac{e\vec{B}}{p_0 c}\right)\right]_x \frac{1+hx}{\frac{p_s}{p_0}} + h\frac{p_s}{p_0} \\
\frac{d}{ds}y &= \frac{(1+hx)}{\frac{p_s}{p_0}}\left(\frac{p_y}{p_0}\right) \\
\frac{d}{ds}\left(\frac{p_y}{p_0}\right) &= \left[\frac{\vec{p}}{p_0}\times\left(\frac{e\vec{B}}{p_0 c}\right)\right]_y \frac{1+hx}{\frac{p_s}{p_0}} \\
\frac{d}{ds}\sigma &= \frac{1}{(v_0/c)} - \frac{l'}{(v/c)} \\
\frac{d}{ds}\left(\frac{\Delta E}{p_0 c}\right) &= 0,
\end{aligned}
\tag{1}
$$

where

$$
\frac{p_s}{p_0} = \sqrt{1 + \frac{2}{\beta}\left(\frac{\Delta E}{p_0 c}\right) + \left(\frac{\Delta E}{p_0 c}\right)^2 - \left(\frac{p_x}{p_0}\right)^2 - \left(\frac{p_y}{p_0}\right)^2}
$$

A class template **RKIntegrator** was developed to perform the numerical integration through arbitrary magnetic field simultaneously for real and DA variables. Instances of template class **RKIntegrator** are initialized by some external function *elementField*, which will be used by the Runge-Kutta integrator to calculate the specific magnetic field:

RKIntegrator$< ZSeries, ZMap >$

$elementDAIntegrator(elementField, elementParameters, step);$

These functions can be collected in one library and shared by different users. In accordance with the basic principles of the object-oriented platform for accelerator codes PAC++[3] the integration is considered as an action of one object on another and defined as a multiplicative operator:

Map $m;\ \ m = 1;$

$m = elementDAIntegrator * m;$

3 Input language

The implementation of the DA integrator completes the object-oriented approach for the description of accelerator structure[3]. All lattice elements are

considered as instances of a C++ class **Element** and divided in three categories: **MAD**, **COSY** and **WILD**.

- **MAD** elements form the majority of all elements and can be defined as the superposition of standard MAD parameters:

  $$\textbf{Element}\ hb = length^{*}\mathbf{L} + 2 * PI/N^{*}\mathbf{ANGLE};$$

  where element *hb* is an object with length equal to *length m* and bend angle $2 * PI/N$ *rad.*

- **COSY** elements include "nonstandard" parameters, but can be defined by a Taylor map (e.g. helical dipole):

  $$\textbf{RKIntegrator}< ZSeries, ZMap >$$
  $$elementDAIntegrator(elementField, elementParameters, step);$$
  $$\textbf{Element}\ helix = helixDAIntegrator * map;$$

  The inclusion of the DA integrator in the object-oriented input language enables one to inherit the flexibility of COSY INFINITY. On the other hand, we keep the clarity of lattice description, because the accelerator or transfer line usually contains only a few "nonstandard" elements.

- **WILD** elements. This category contains elements, which cannot be completely described by the Taylor map (such as internal target, splitter magnet etc.) At the optical design level they must be replaced by elements of 1st or 2nd categories (e.g. target as drift):

  $$\textbf{Element}\quad target \quad = lengthOfTarget * \mathbf{L};$$
  $$\textbf{Teapot}\quad ring \quad = ... * target * ...;$$

  where the class **Teapot** is the object-oriented version of the program TEAPOT[4]. The particle-target interactions may be described by some external class **Target** and included in the general numerical simulation with the overloaded multiplicative operator $Target :: operator*(Particle\&\ particle)$:

  $$\textbf{Target}\quad tooth(toothParameters);$$
  $$ring.track(1, numberOfTarget - 1, particle);$$
  $$particle = tooth * particle;$$
  $$ring.track(numberOfTarget + 1, numberOfLastElement, particle);$$

  This approach enables one also to use the different object-oriented HEP libraries (as GIZMO) or test the new complicated algorithms without changing of the object-oriented accelerator programs.

4 RHIC helical dipole

The field $\vec{B}$ in the current-free region of a helical magnet can be expressed in Cartesian coordinate system as:

$$\begin{aligned} B_x &= B_r cos(\phi) - B_\phi sin(\phi) \\ B_y &= B_r sin(\phi) + B_\phi cos(\phi) \\ B_z &= B_z \end{aligned} \tag{2}$$

where[5]

$$\begin{aligned} B_r &= -k \sum_{m=1}^{\infty} m I_m'(mkr)(a_m cos(m\theta) + b_m sin(m\theta)) \\ B_z &= k \sum_{m=1}^{\infty} m I_m(mkr)(b_m cos(m\theta) - a_m sin(m\theta)) \\ B_\phi &= -\frac{1}{kr} B_z \end{aligned}$$

and $\theta = \phi - (kz + \phi_0)$, $x = rcos(\phi)$, $y = rsin(\phi)$. Unfortunately, these equations cannot be expressed directly in DA variables, because the inversion and square root functions for a DA variable $r = (r_0, r_1, r_2, ..., r_N)$ is defined as truncated power series[6]:

$$\begin{aligned} &\sqrt{(r_0, r_1, r_2, ..., r_N)} \\ &= \sqrt{r_0}\sqrt{1 + (0, \frac{r_1}{r_0}, \frac{r_2}{r_0}, ..., \frac{r_N}{r_0})} \\ &= \sqrt{r_0}\sum_{i=0}^{n}(-1)^i \frac{1 \cdot 3 \cdot ...(2i-3)}{2 \cdot 4 \cdot ...(2i)}(0, \frac{r_1}{r_0}, \frac{r_2}{r_0}, ..., \frac{r_N}{r_0})^i \end{aligned}$$

and depends on r_0. The expressions r^{2n}, $r^n cos(n\phi)$, and $r^n sin(n\phi)$ can be presented as simple functions of x and y variables. To extract them the equations (2) were transformed and written in the following form:

$$\begin{aligned} B_x &= \tilde{B}_r x - \tilde{B}_\phi y \\ &\quad + \sum_{m=1} \frac{(mk/2)^m}{(m-1)!} r^{m-1}\{b_m sin(\phi - m\theta) - a_m cos(\phi - m\theta)\} \\ B_y &= \tilde{B}_r y + \tilde{B}_\phi x \\ &\quad - \sum_{m=1} \frac{(mk/2)^m}{(m-1)!} r^{m-1}\{b_m cos(\phi - m\theta) + a_m sin(\phi - m\theta)\} \\ B_z &= -k\tilde{B}_\phi r^2 \\ &\quad + k \sum_{m=1} \frac{(mk/2)^m}{(m-1)!} r^m \{b_m cos(m\theta) - a_m sin(m\theta)\} \end{aligned} \tag{3}$$

where

$$\begin{aligned}
\tilde{B}_r &= -8\sum_{m=1}(km/2)^{m+2}(\tilde{I}_{m-1}(mkr) - \frac{m}{2}\tilde{I}_m(mkr)) \\
&\quad \cdot r^m\{a_m cos(m\theta) + b_m sin(m\theta)\} \\
\tilde{B}_\phi &= -\frac{8}{k}\sum_{m=1}(km/2)^{m+3}\tilde{I}_m(mkr) \\
&\quad \cdot r^m\{b_m cos(m\theta) - a_m sin(m\theta)\}
\end{aligned}$$

and

$$\tilde{I}_m(z) = \sum_{i=1}^{\infty}\frac{(z/2)^{2(i-1)}}{4\,i!\,(m+i)!}$$

The two sets of equations (2) and (3) were implemented in C++ and tested together in DOUBLE variables. When a perfect agreement between different functions was achieved, the last one was accepted as the template function *helixField* and located in the file *Field.hh*. The nominal design for the RHIC helical snake[7] consists of 4 helical dipoles of 2.4 m length, the B_0 field for the outer modules is 1.458 T and for the inner ones is 4 T. The Taylor map of one helical dipole was obtained by the short program presented in Figure 1. As a first step, only the influence of the main harmonic b_1 was considered. Results obtained with this approach agree with the first and second order transfer matrices derived by the SNIG program[8] via numerical integration of particle trajectories (see Figure 2). The input language described in Section 3 provides one with several methods to include the helical snake in the Teapot tracking procedure. The easiest one is presented in Figure 3.

5 Acknowledgment

I would like to thank R.Talman, S.Peggs, F.Pilat, and V.Ptitsin for many useful discussions.

References

[1] M. Berz, "User's Guide and Reference Manual to COSY INFINITY v.6".

[2] N. Malitsky, A. Reshetov, and Y. Yan, SSCL-659, 1994.

[3] N. Malitsky, A. Reshetov, G. Bourianoff, SSCL-675, 1994.

[4] L. Schachinger and R. Talman, Particle Accelerator,22,35(1987).

[5] V. Ptitsin, Note RHIC/AP/41(oct. 10,1994)

[6] M. Berz, Particle Accelerators,1989,Vol.24,pp.109-124

[7] A. Luccio, Presented at the Spin Accelerator Meeting, BNL, October 6,1994.

[8] A. Luccio, Private communication.

```
#include "RKIntegrator/RKIntegrator.hh"
#include "Field/Field.hh"

main()
{

 BEAM_DIM    = 4;
 ZLIB_ORDER  = 2;

// Helix Parameters ( main field b1)

 double gamma  = 27;
 double BR     = gamma*PROTON/0.3;          // 1/m

 double length = 2.4;                       // m
 double k      = 2*PI/length;               // 1/m
 double B0     = 1.458;                     // T

 double helixParameters[7];

 helixParameters[0] = 6.;                   // number of parameters = 2*order +4
 helixParameters[1] = length;               // Length
 helixParameters[2] = k;                    // k
 helixParameters[3] = 0.0;                  // phase
 helixParameters[4] = 0.0;                  // gap
 helixParameters[5] = -2.*B0/BR/k;          // b1 = -2*B0/BR/k
 helixParameters[6] = 0.0;                  // a1

// Initialization

 ZMap z;  z = 1.;
 RKIntegrator<ZSeries, ZMap> helix(helixField, helixParameters, 0.01);

// DA Runge-Kutta integrator

 cout << helix*z;

 return(1);
}
```

Figure 1. Example of main program for DA integration through RHIC helical dipole.

a. PAC++. RKIntegrator.

ZMap : order = 2 dimension = 4

1	4.867687e-06	2.028584e-06	-1.582709e-02	4.249226e-06	0	0	0	0
2	9.995705e-01	-1.791320e-04	2.390723e-04	-4.903340e-08	1	0	0	0
3	2.399748e+00	9.999996e-01	4.919799e-04	3.077267e-04	0	1	0	0
4	-3.075360e-04	-8.548339e-08	9.995705e-01	-5.369104e-04	0	0	1	0
5	-8.203138e-04	-2.394684e-04	2.399774e+00	9.991406e-01	0	0	0	1
6	3.290828e-05	1.680853e-05	-4.068043e-02	1.093308e-05	2	0	0	0
7	2.075623e-02	2.040447e-05	-1.031094e-06	8.135623e-02	1	1	0	0
8	2.713307e-02	2.183969e-05	1.205582e-05	5.031948e-06	1	0	1	0
9	5.137259e-05	-2.707133e-02	-2.071993e-02	5.593873e-06	1	0	0	1
10	1.246527e-02	-1.035920e-02	-3.952618e-03	9.765436e-02	0	2	0	0
11	1.080032e-06	-2.709228e-02	2.071587e-02	-3.707718e-05	0	1	1	0
12	-7.871256e-03	-6.503091e-02	-2.488749e-02	-2.078928e-02	0	1	0	1
13	1.899624e-05	9.459601e-06	-1.354658e-02	2.791449e-05	0	0	2	0
14	2.076924e-02	2.964764e-05	5.850425e-05	2.721552e-02	0	0	1	1
15	3.736429e-02	1.040335e-02	-1.181383e-02	3.265235e-02	0	0	0	2

b. SNIG program.

MATRIX

[x]	[4.8695E-03]	[0.9996	2.400	-3.0760E-04	-8.2046E-04]	[xo]
[u]	[2.0305E-03]	[-1.7920E-04	1.000	-1.7083E-07	-2.3977E-04]	[uo]
[y]	[-15.83]	[2.3913E-04	4.9206E-0	0.9996	2.400]	[yo]
[v]	[4.2507E-03]	[-2.4291E-08	3.0787E-04	-5.3702E-04	0.9991]	[vo]

3.209E-08	2.076E-05	2.714E-05	5.565E-08	1.242E-05	4.941E-09	-7.758E-06	1.434E-08	2.076E-05	3.722E-05
1.838E-08	2.961E-08	2.671E-08	-2.707E-05	-1.035E-05	-2.708E-05	-6.498E-05	1.486E-08	6.500E-08	1.046E-05
-4.069E-05	-2.901E-09	1.413E-08	-2.059E-05	-4.153E-06	2.074E-05	-2.465E-05	-1.355E-05	-9.675E-08	-1.291E-05
8.681E-09	8.135E-05	1.931E-09	-7.650E-09	9.765E-05	-4.656E-08	-2.075E-05	2.549E-08	2.721E-05	3.261E-05
xo**2	xo*uo	xo*yo	xo*vo	uo**2	uo*yo	uo*vo	yo**2	yo*vo	vo**2

Figure 2. Second order Taylor map for the helical dipole at $\gamma = 27$. B0 = 1.458 T, L = 2.4 m.

```
// Helical snake + Teapot tracking.

#include "Teapot/Teapot.hh"
#include "WildElements/HelicalSnake.hh"

main(){

  int i, j, turn;
  int numberParticles = 100;
  int numberTurns     = 100;

// Global

  ENERGY = 100;       // GeV

// Particle

  Particle** p;
  p = new Particle*[numberTurns+1];
  for(i=0; i <= numberTurns; i++)
    p[i] = new Particle[numberParticles+1];

  for(i=1; i <= numberParticles; i++)
   for(j=1; j <= TEAPOT_DIM; j++)
     p[0][i][j] = ...;

// Teapot

  Teapot rhic("fort.7");

  int nhel1 = rhic.number("helMarker1");
  int nhel2 = rhic.number("helMarker2");
  int nelem = rhic.numberElements();

// Tracking

  for(turn=1; turn <= numberTurns; turn++)
    for(i=1; i <= numberParticles; i++){
      p[turn][i] = rhic.track(1, nhel1-1, p[turn-1][i]);
      p[turn][i] = helicalSnake*p[turn][i];
      p[turn][i] = rhic.track(nhel2+1, nelem, p[turn][i]);
    }

...
}
```

Figure 3. Example of the inclusion of the helical snake in Teapot tracking .