



NEW FEATURES IN THE DESIGN CODE TLIE *

Johannes van Zeijts
Continuous Electron Beam Accelerator Facility
12000 Jefferson Avenue, Newport News, VA 23606

ABSTRACT

We present features recently installed in the arbitrary-order accelerator design code TLIE. The code uses the MAD input language, and implements programmable extensions modelled after the C language that make it a powerful tool in a wide range of applications: from basic beamline design to high precision - high order design and even control room applications.

The basic quantities important in accelerator design are easily accessible from inside the control language. Entities like parameters in elements (strength, current), transfer maps (either in Taylor series or in Lie algebraic form), lines, and beams (either as sets of particles or as distributions) are among the type of variables available. These variables can be set, used as arguments in subroutines, or just typed out. The code is easily extensible with new datatypes.

INTRODUCTION

Here we give a short introduction to the physics and algorithms used in the program. We use the Hamiltonian formalism and canonical variables introduced in the map code field in the program MARYLIE 3.0¹. The Hamiltonian describes in a compact way the dynamics of the particles in the full 6 dimensional phase space.

$$H = -\frac{P_r}{\beta} - \frac{A_z}{G} - \sqrt{1 - \frac{2P_r}{\beta} + P_r^2 - (P_x - \frac{A_x}{G})^2 - (P_y - \frac{A_y}{G})^2}, \quad (1)$$

G is the magnetic rigidity ' $B\rho$ ' of particles on the design orbit. It is given by the relation

$$G = \frac{p_0}{q}, \quad (2)$$

and has units of tesla meters. Also, β and γ are the standard relativistic factors for the design orbit. They are related to p_0 (the design momentum) and $p_t^0 = -H|_{\text{design orbit}}$ by the equations

$$\begin{aligned} p_0 &= \beta\gamma mc, \\ p_t^0 &= -\gamma mc^2. \end{aligned} \quad (3)$$

A_x , A_y , and A_z are the vector potentials; for example, the vector potential for a normal quadrupole with a gradient of b_2 tesla/meter is

$$A_x = -\frac{b_2}{2}(x^2 - y^2) \quad (4)$$

*Work supported by Department of Energy contract #DE-AC05-84ER40150

New physics is added by substituting more complex vector potentials. We provide vector potentials for all the simple elements and for realistic multipole magnets. In some of our multi-particle-effect applications the vector potential actually depends on the transported beam. However, the algorithms described below do not change no matter how complicated the vector potential becomes. The transfer maps for the 'time λ ' flow are generated in a variety of ways from the Hamiltonian:

- Transfer map generation for the case where the Hamiltonian H does not depend on the independent variable is most readily done by direct exponentiation of the formal solution

$$T(z) = e^{-\lambda H_1(z)} = z - \lambda[H, z] + \frac{1}{2!}\lambda^2[H, [H, z]] + \dots, \quad (5)$$

where $[,]$ is the Poisson bracket operator. This algorithm will eventually converge due to the $n!$ term in the denominator of the expansion of the exponential operator.

- Transfer map generation for the case where H does depend on the independent variable is implemented as:

$$T_{\lambda_1 \rightarrow \lambda_2}(z) = \int_{\lambda_2}^{\lambda_1} [H(z, \lambda), T(z)] d\lambda. \quad (6)$$

This algorithm is implemented in a particularly efficient way, which is the main reason we are able to generate high-order transfer maps for realistic systems in a time period practical for use in an optimization process (minutes for order 10). In practice we use a forward integration and calculate the inverse map $I = T^{-1}$

$$I_{\lambda_1 \rightarrow \lambda_2}(z) = \int_{\lambda_1}^{\lambda_2} [H(z, \lambda), I(z)] d\lambda. \quad (7)$$

The resulting Taylor series are converted into Lie algebraic form. This is necessary for the "concatenation" process and gives us a compact representation of the aberration coefficients. By default we split the Lie algebraic map up in homogeneous parts with the following standard 'Dragt-Finn' ordering

$$F(z) = e^{f_1}(z) = e^{f_1} M e^{f_2} \dots e^{f_n}(z), \quad (8)$$

where f_1 describes the misalignment part, f_2 the second order part, etc. The linear part M of the transfer map is carried as a symplectic matrix.

A number of other orderings of the Lie algebraic exponents are available and used in the program. The algorithms to translate between these different representations are most readily implemented to arbitrary order using Taylor series as intermediate steps. For instance to concatenate two maps we use the algorithm:

$$e^{f_1} e^{g_1}(z) = e^{f_1}(G(z)) = H(z) = e^{h_1}(z), \quad (9)$$

where all steps in the process are uniquely determined as long as the matrix part of the maps are symplectic.

EXTENSIONS TO THE PHYSICS

We have added the capability to calculate transfer maps for cylindrical current sheet magnets, the current sheets may be stacked so as to produce overlapping multipole fields. Together with the fast integration algorithm this allows us to model realistic systems including fringe fields to high precision in a practical time period.

The vector potential off axis, for a given multipole symmetry, is determined from the appropriate magnetic field gradients and their longitudinal derivatives on axis. In the following we will write expressions only for the normal multipoles (for $m \neq 0$). Skew multipoles correspond to $\cos(m\theta)$ terms in Eq.10. Given the Fourier expansion of the scalar potential

$$V(r, \theta, z) = \sum_{m=1}^{\infty} U_m(r, z) \sin(m\theta), \quad (10)$$

a vector potential giving the same field is

$$\begin{aligned} A_z &= - \sum_{m=1}^{\infty} \frac{\cos(m\theta)}{m} r \frac{\partial}{\partial r} U_m(r, z) \\ A_r &= \sum_{m=1}^{\infty} \frac{\cos(m\theta)}{m} r \frac{\partial}{\partial z} U_m(r, z). \end{aligned} \quad (11)$$

Here we have chosen a gauge where $A_\theta = 0$. The scalar potential off-axis may be written

$$U_m(r, z) = r^m \sum_{l=1}^{\infty} \frac{(-1)^l (m-1)!}{l!(l+m)!} \left(\frac{r}{2}\right)^{2l} \left(\frac{\partial}{\partial z}\right)^{2l} g_m(z), \quad (12)$$

where

$$g_m(z) = \lim_{r \rightarrow 0} \frac{m U_m(r, z)}{r^m} \quad (13)$$

represents the profile of the m^{th} multipole. This is a general solution to Maxwell equations order by order, for arbitrary $g_m(z)$. The problem is thus reduced to computing the generalized field gradients on axis for realistic magnet models. Subroutines to compute the required gradients are available for Halbach REC quadrupoles and for general multipoles, with the current distribution on a cylindrical surface specified by a shape function³.

One particular powerful feature of the program is the ability to specify user profiles for the field gradient as functions of the independent variable s in the control language. These profiles can be arbitrarily constructed functions of s , and the expressions are automatically differentiated to the order needed in the integration process. We give several examples below and show how a user profile is given interactively to specify the octupole component of one multipole. Remember that any number of different multipoles can be layered on top of each other to represent arbitrarily complicated profiles.

```

f1(s) = 1/(1+E^(b1 + b2*(s/d)+ b3*(s/d)^2))
f2(s) = 1/(1+(s/d)^2)^(3/2)
f3(s) = B0*(tanh(s/d)-tanh((s-10)/d))

multi: multipole, f1(), L = 1.0, m = 4, radius = 0.22

```

This feature for instance, allows users to install arbitrary fringe field profiles if he so chooses.

EXTENSIONS TO THE MAD LANGUAGE

We extend the MAD language in a variety of ways. First we introduce logic for loops, conditionals, subroutines, and functions returning real numbers. The control language is interpretive and fully programmable. It is based on a C like interpreter and written in the compiler generating language YACC⁴.

Secondly we allow for element parameters to be easily accessible in the language; i.e., name[L] gives the length of a particular element, name[K] gives the quadrupole strength, etc. These parameters can be read and set interactively.

Next we introduce a transfer map datatype; i.e., map1 = namedline.F creates a new map variable from a given line element, in this case a Lie map variable. We also have map2 = namedline.T, which returns the Taylor map. Entries in map variables can be accessed as: map1[x,x] for entries in the matrix part, map1[x³], map1[x Pt²] for higher order entries in the Liemap variables, and map2[x,x²] etc. for Taylor map variables. Since maps are checked for dependencies on element parameters and brought up to date when they are used, it is easy to write a code segment that studies the effect of changing a parameter on transfer map entries,

```

for(quad1[K] = 0; quad1[K] <= 3.0; quad1[K] += 0.1) {
    type quad1[K], namedline.F[x,x]-map1[x,x], namedline.F[x Pt^2]
}

```

where namedline is a line which is dependent on the strength of quad1.

Transfer maps are constructed by patching maps for single element bodies together with coordinate rotations, fringe-fields, etc. We provide all the basic elements in the program, but it is also possible to specify new constructions interactively by using operations on maps. These Lie map expressions (Fexpr) can be:

- name.F
- Lie map variable
- Liemap(identity), creates an identity liemap
- Fexpr + Fexpr
- Fexpr - Fexpr
- Fexpr / Fexpr, ignoring zero terms in denominator

- (Fexpr)
- Fexpr & Fexpr, result is the concatenated map.
- invert(Fexpr), returns the inverted Lie map.
- filter(Fexpr,expr), filters out entries less than expr
- prot(expr), the map for longitudinal reference plane rotation
- arot(expr), the map for transverse reference plane rotation
- monomial(Fexpr), the map in monomial factorized form
- reverse(Fexpr), returns a Lie map in reverse order
- flend(Fexpr), put the e^{f1} term at the tail end of the map
- standard(Fexpr), to convert back to Dragt-Finn factorization;

i.e., to get the relative difference between two Lie maps we simple ask for

```
type (F1-F2)/F2
```

We allow for the evaluation of subroutines along any given line. These subroutines expect several parameters as their arguments: the longitudinal distance, coordinates and angles with respect to the floor, an element type, the transfer map up to the particular point in the line, and the name of the element as a string variable. An example of this can be as simple as typing the names and coordinates of a line

```
proc typeName(real s1, real x1, real y1, real z1, real angxz, real angyz, real
angxy, real eltype, lienap lie1, string name)
(
    type s1,name,z1,x1,y1
)
```

Since the transfer map is available we can evaluate any changing variable along the beam line. A useful procedure is to calculate lattice functions and, for instance, find the maximum value of a lattice function along a line:

```
proc findmaxBeta(lienap $0) {
    bX = betaxF($0,bx1,ax1)
    bY = betayF($0,by1,ay1)
    if (bX > maxX) { maxX = bX; maxsX = $1}
    if (bY > maxY) { maxY = bY; maxsY = $1}
}
layout(namedline) with findmaxBeta()
```

Here we see how Liemap variables can be used as arguments in functions. Arguments can be accessed by name or by the order they appear in. The layout with procedure command can be used inside any procedure. Hence we can use the optimizer to, for instance, minimize the maximum lattice functions

```
func maxBeta(line $l) {
  maxX = maxY = -1.0
  maxsX = maxsY = -1.0
  layout($l) with findmaxBeta()
  if(maxY > maxX) return(maxY) else return(maxX)
}
```

CONTROL ROOM APPLICATIONS

By the addition of a few new parameters and keywords we turn the code into a package useful for control room applications. We add the ability to set and read currents of magnets, and read the horizontal and vertical position from beam position monitors. A suitable control logic library,⁵ and access to control room computers has to be available to be able to make use of these features.

The full power of the language is available to program experiments and data-analysis. For instance, a particularly simple application, to measure an entry in the transfer matrix between two points, is implemented as follows:

```
for(current1 = 0; current1 <= 3.0; current1 += 0.1) {
  kick1[I] = current1; sleep(1)
  type kick1[I], monitor1[x], monitor1[y]
}
```

where the `kick1` and `monitor1` keywords are declared as type `kick` resp. `monitor`. With suitable caution even the fitter and optimizer can be used for real-time control. This connection between a design code and a control code allows for measured data to be propagated using simulated transfer maps, which is useful in tuning algorithms. The interpretive nature of the code allows for many more applications to be implemented rapidly.

REFERENCES

1. A. J. Dragt, *MaryLie 3.0, A Program for Charged Particle Beam Transport Based on Lie Algebraic Methods*.
2. J. B. J. van Zeijts and F. Neri, *The Arbitrary Order Design Code Tlie*, presented at the Gosen High Order Optics Codes workshop, April 1992.
3. P. Walstrom, Filippo Neri and Tom Mottershead, *High Order Optics of Multipole Magnets*, LINAC Meeting Albuquerque 1990.
4. B.W. Kernighan and R. Pike, *The Unix Programming Environment*, (Prentice-Hall 1984).
5. M. Bickley, *Star Documentation*, CEBAF, May 1992.