Fermi National Accelerator Laboratory

FERMILAB-Conf-92/219

Accelerator Physics Analysis with an Integrated Toolkit

J.A. Holt, Leo Michelotti, T. Satogata

Fermi National Accelerator Laboratory
P.O. Box 500, Batavia, Illinois 60510

August 1992

Presented at the *XVth International Conference on High Energy Accelerators*, Hamburg, Germany, July 20-24, 1992

Operated by Universities Research Association Inc. under Contract No. DE-AC02-76CHO3000 with the United States Department of Energy

Disclaimer

Accelerator Physics Analysis with an Integrated Toolkit

J. A. Holt, Leo Michelotti, T. Satogata
Fermi National Accelerator Laboratory†
P. O. Box 500, Batavia, IL 90510, USA

Abstract

Work is in progress on an integrated software toolkit for linear and nonlinear accelerator design, analysis, and simulation. As a first application, "beamline" and "MXYZPTLK" (differential algebra) class libraries, were used with an X Windows graphics library to build an user-friendly, interactive phase space tracker which, additionally, finds periodic orbits. This program was used to analyse a theoretical lattice which contains octupoles and decapoles to find the 20^{th} order, stable and unstable periodic orbits and to explore the local phase space structure.

1 Introduction

There are a number of good general-purpose accelerator tracking codes in wide use such as SYNCH [1], MAD [2], or TEAPOT [3]. Each of these programs has an input format or language in which the user formulates the problem to be solved. This language is different from the language the code is written in. Frequently the accelerator physicist wants to solve a problem which does not fit within the constraints of the input language. The physicist must then duplicate the basic lattice handling functions which all codes must have as well as the special functions which solve the problem at hand. Also some problems can be solved more quickly when the input can be changed interactively and both the intermediate and final results be seen graphically. Most codes at present are a batch type operation; an input file is edited, the program is run and the final results are output. Exploring phase space to search for a separatrix for example, in this manner is a very time-consuming process.

We are developing a set of tools in which the input language is the same as the programming language. We have implemented our code in C++ because of the ease in creating new types that behave in every respect like fully functional variables of the language. We have integrated several types of class libraries to form a custom application to "solve" a particular problem. Different authors worked on different parts of the libraries. Because everything was encapsulated in objects, one author did not have to know any details of the objects created by another; only the interface to that object had to be known.

The following sections briefly describe the class libraries used to build the application. Some have been explained in greater detail elsewhere. We have used MXYZPTLK which is a set of **DA** classes which implement differential algebra, **beamline** which is a class used to simulate beamlines and rings, and a collection of X Window/Motif classes which are used for the user graphical interface.

2 Differential Algebra Class Library

The class library MXYZPTLK [4] is an implementation of differential algebra in C++ . There are two classes **DA** and **DAVector** which are derived from doubly linked lists. Each link in a list contains the "index array" for a particular non-zero derivative and its weighted value. When a **DA** variable is first declared, it is a list with no links. The links are created dynamically as calculations proceed. Full use is made of the function overload capability of C++ . All arithmetic operations including mixed-mode operations are handled automatically. No special function calls or precompiling is necessary. In addition, each **DA** variable keeps track of its own attributes, such as accuracy and reference point. For example, consider the simple function $x(m)$ defined by the equation

$$x(m) = \cos(m \cdot x(m)) \qquad (1)$$

† Operated by the Universities Research Association,Inc, under contract with the U.S. Department of Energy

Simple recursion can be used to construct $x(m)$ for m in the approximate range $m \in (-1.2, 1.2)$, determined by the condition $\mid m\sin(m \cdot x(m)) \mid < 1$. The same recursion, applied to **DA** variables, provides the derivatives as well. The C++ code fragment [5]

```
DA m, x;
m.setVariable( 0.5, 0);
x = cos( m * 0.9 );
for( i = 0; i < 15; i++ )  x = cos( m * x );
x.peekAt();
```

will evaluate derivatives of $x(m)$ at $m = 0.5$ ($x(0.5) = 0.900376\ldots$). Note that C++ automatically handles the mixed mode arithmetic in the third line. The **.peekAt** member function prints the desired derivatives.

3 Beamline Class Library

An accelerator is a collection of objects connected together to form beamlines, a structure modelled very naturally by C++ . The class **beamline**[6] is derived from two base classes; a doubly linked list and **bmlnElmnt**, which holds information common to all beamline elements, such as geometry. Because **beamline** is derived from **bmlnElmnt** it is easy to insert one **beamline** into another, thereby building complicated models hierarchically. As a trivial example, the following C++ code fragment is one way of constructing a five-cell FODO lattice.

```
double length      = 1.0, focalLength = 1.0;
drift           O( length );
thinQuadrupole  F( focalLength );
thinQuadrupole  D( -focalLength );

beamline A ( F );
A.append   ( O );
A.append   ( D );
A.append   ( O );

beamline B;
for( int i = 0; i < 5; i++) B.append( A );
```

The first lines declare variables **O**, **F**, **D** as the basic beamline elements. This is followed by a series of **.append** statements in which the elements are inserted into a cell, called **A**. The loop at the end contructs a beamline **B** consisting of five cells of **A**. In a beamline constructed in this manner, adjustment of **F** will adjust **F** everywhere.

Once declared, **beamline** objects can be used to do tracking, evaluate lattice functions, construct polynomial maps, and so forth. For example, the following code fragment belongs to a simple program which compares element by element tracking of a nonlinear beamline, called `E778`, with polynomial map evaluation.

```
beamline     E778;
proton       p;    DAproton     pd;
double       w[6], y[6], z[6], zero[6];
DA           zd[6];
 ...
// Construct polynomial map
pd.setState( zero );
E778.propagate( pd );
pd.getState( zd );
// Do tracking
for ( int i = 0; i < 50000; i++ ) {
//  -- with the map
  for ( int j = 0; j < 6; j++ )
          y[j] = zd[j].multiEval( z );
  for ( j = 0; j < 6; j++ ) z[j] = y[j];
// -- element by element
  E778.propagate( p );
  p.getState( w );
  ...
}
```

First, a polynomial map is constructed by propagating a **DAproton** object around the beamline and storing its resultant state in an array of `DA` variables. Within the loop, the **.multiEval** method is used to evaluate this map for comparison with element by element tracking, done by propagating a `proton` object around `E778`.

4 Graphic Class Library

Using Young's widgets [7] as foundation, we have implemented an X Window/Motif graphic interface in C++ . Classes such as fileselection, menubars, command buttons, popup windows and text input have been implemented making it easy for the user to customize the interface. The most important class is the **phaseSpace** class in which 4D phase space variables amplitude and phase (this can be changed to x-x', y-y' easily) are plotted. The user can click anywhere in the phase space region to set the initial conditions for tracking. This can be done while propagating through a beamline or map. Searches for a separatrix for example can be done very quickly.

The graphics interface communicates with the **beamline** class through a class called **genericMap**. This class has methods such as start and stop propagating, setting or retrieving the tunes, and finding periodic orbits. The user can change maps or beamlines or vari-

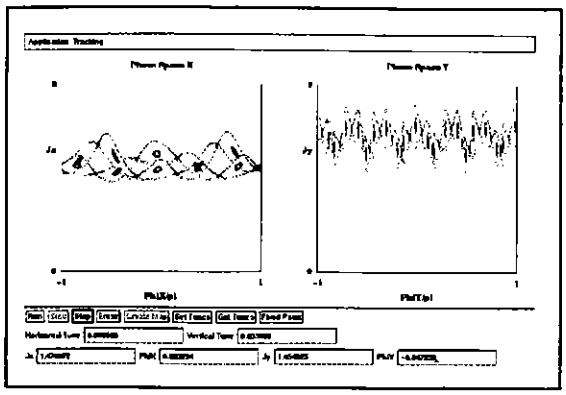

Figure 1: 2D Projection of 4D Phase Space

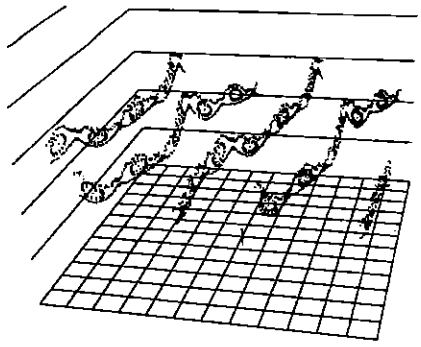

Figure 2: 3D Projection of 4D Phase Space

ables to be plotted without touching the graphics interface. It is also easy to add a new method to this class and connect this method to a command button or menu item in the interface.

5 A Simple Application

Figure 1 shows a simple application which was used to look for the periodic orbits of a theoretical lattice [8], the main elements of which are three octupoles separated by sixty degree phase advances and a decapole separated by a forty-five degree phase advance. A stable and an unstable period twenty orbit are displayed, along with the corresponding separatrix. One of the tori surrounding the stable orbit is also drawn.

Previous work by one of the authors [9] used coupled three-dimensional projections to explore four-dimensional phase space. Figure 2 shows the same lattice and periodic orbits using AESOP, a graphics shell originally written for the Evans and Sutherland PS390 terminal. AESOP, and the "GrafXPad" classes on which it is based, are being rewritten using X11R5 Phigs so as to make them available on any X workstation.

References

[1] A. A. Garren, A. S. Kenney, E. D. Courant, A. D. Russell, and M. J. Syphers. *SYNCH: A Computer System for Synchrotron Design and Orbit Analysis. Users Guide.* Los Alamos Accelerator Code Group. Los Alamos National Laboratory.

[2] Hans Grote and F. Christoph Iselin. *The MAD Program (Methodical Accelerator Design). User's Reference Manual.* European Organization for Nuclear Research, Geneva, Switzerland. CERN/SL/90-13.

[3] L. Schachinger and R. Talman, *TEAPOT: A Thin Element Accelerator Program for Optics and Tracking,* Superconducting Super Collider Laboratory, SSC-52 December 1985.

[4] L. Michelotti. *MXYZPTLK: A practical users-friendly C++ implementation of differential algebra: User's Guide.* Fermi Note FN-535, Fermilab, January 31, 1990; *MXYZPTLK: A C++ Hacker's Implementation of Automatic Differentiation.* In *Automatic Differentiation of Algorithms: Theory, Implementation, and Application.* SIAM, Philadelphia, PA. 1991.

[5] L. Michelotti, *A Note on the Automated Differentiation of Implicit Functions.* Fermilab TM-1742 (June 1991)

[6] L. Michelotti, *MXYZPTLK and Beamline: C++ Objects for Beam Physics.* In *Advanced Beam Dynamics Workshop on Effects of Errors in Accelerators, their Diagnosis and Correction.* (held in Corpus Christi, Texas, October 3-8, 1991) Published by American Institute of Physics, as Conference Proceedings No. 255. 1992.

[7] Douglas A. Young, *Object-Orientated Programing with C++ and OSF/Motif,* Prentice Hall, 1992.

[8] T. Satogata and S. Peggs, *Is Beta Modulation More or Less than Tune Modulation?* IEEE Particle Accelerator Conference 1989, page 476.

[9] L. Michelotti, *Exploratory Orbit Analysis.* In *Proceedings of the 1989 IEEE Particle Accelerator Conference.*
March 20-23, 1989, Chicago, IL. Vol.2, pp.1274-1276.