RECENT DEVELOPMENTS IN MAD VERSION 8

H. Grote and F. C. Iselin
CERN
CH-1211 Geneva 23

Abstract The MAD program [1] has been rewritten recently in most parts. The dynamic memory manager has been replaced by the CERN-written ZEBRA package which has been used successfully in more than 100 other programs for high-energy physics data analysis.

A new dynamic table handler keeps large tables in memory as long as they fit; it dumps them on disk or to the Cray SSD (Solid-State Storage Device) if necessary. The user accesses table lines in a transparent way by simple subroutine calls.

The input language has been extended to use an object-oriented approach. Accelerator elements are described in terms of classes of objects, providing easy selection of single elements in a large machine. A powerful mechanism has been implemented for writing tables of selected optical functions and/or element parameters in selected positions of the machine. These tables are generated and written under control of a dynamic table handler, and can be read directly by the LEP control system.

MAD contains a plot module with various options to plot data from internal tables. The tables can also be fed into a stand-alone plot program.

The rewrite has been completed in April 1990. New features with respect to previous versions include: Matching constraints for elements of the transfer matrix and for algebraic combinations of quantities; Calculation of Electron Beam Parameters; Deferred evaluation of lumps; Speed-up of time-critical subroutines.

Introduction

At CERN many programs exist which require dynamic management of computer memory in the framework of a FORTRAN program. For this purpose the ZEBRA package [2] has been developed. It has the advantage that the user need not be concerned about updating pointers when garbage collection becomes necessary. It has been used with success in over 100 data-analysis programs for high-energy physics and should therefore be reasonably bug-free. Since it is available on a large variety of computers it has been adopted as a tool for implementing the memory management in MAD.

The ZEBRA package allows to book "data banks" and to link them together in a tree-like structure. It supports linear lists, such that trees can have a variable number of branches from any given node. Furthermore ZEBRA supports the concept of reference links, i. e. pointers which may point to any data object, be it in the same or in a different tree.

ZEBRA also contains a package to read or write single data banks or complete trees on sequential or direct access files. These features make it easy to build up the data structures needed in MAD.

Dynamic Table Management

Most accelerator physics programs tend to build large tables for storing the results of their computations. These tables are then processed later in various ways.

Table 1: An OPTICS Output Table Example

```
@ GAMTR          %f     64.3336
@ ALFA           %f    0.241615E-03
@ XIY            %f    -.455678
@ XIX            %f     2.05279
@ QY             %f    0.250049
@ QX             %f    0.249961
@ CIRCUM         %f     79.0000
@ DELTA          %f    0.000000E+00
@ COMMENT        %20s "DATA FOR TEST CELL"
@ ORIGIN         %24s "MAD 8.01    IBM - VM/CMS"
@ DATE           %08s "19/06/89"
@ TIME           %08s "09.47.40"
* NAME        S               BETX              BETY
$ %16s        %f              %f                %f
  B1         36.6600         24.8427          126.380
  SF1        37.6200         23.8830          130.925
  QF1        39.5000         23.6209          132.268
  B2         75.8000         124.709          25.2153
  SD1        77.1200         130.933          23.8718
  QD1        79.0000         132.277          23.6098
```

A table normally has a three-dimensional structure: On the top level the table consists of one or more *segments*. Each segment contains a known number of *lines*, and each *table line* consists of a fixed number of *column entries* which may be numeric or alphanumeric. Both the number of lines per segment and the number of segments can be changed dynamically if required. Examples:

- Linear lattice functions after each element in the accelerator, one line for each position. There may be several segments, one for each value of the momentum error considered.
- Particle positions after each turn during tracking, one line for the phase space coordinates of each particle. The lines for each turn form a separate segment.
- The values of tunes, chromaticities, and other global quantities as a function of momentum error. This table contains one line for each momentum error, and there is only one segment.

All large tables are handled in MAD by the same table manager. There are subroutine calls to create, open, close, and delete a table. Tables can also be assigned various types of descriptors which can be accessed by their name. Such a descriptor might be the date or time of creation, a descriptive character string, a numeric value, etc. See Table 1 for an example of the TFS (Table File System for LEP control system) format [3].

Table creation begins with a subroutine call to assign the number of segments and the number of lines per segment and to define the desired columns. It ends with a close call (existing buffers remain in storage), or a dump call (all buffers are dumped to secondary storage).

Table access begins with a subroutine call to open the table, and ends with a close or dump call.

In either case the program selects segments and/or lines in any order. For each segment-line pair it tells the table manager whether reading, writing or updating is desired. The table manager automatically loads or dumps the required buffers behind the scene. The table buffers all reside in ZEBRA data banks.

The described interface makes it easy to implement new tables.

Plotting

MAD contains a general-purpose plot module which is able to plot any numeric table column against another. If there are several segments, the segment number may act as a parameter to plot several curves on the same or different frames. It is also possible to plot the frame parameter against the value of a variable in a fixed line, where the line number acts as a parameter.

This mechanism can generate all different plots which were possible with earlier versions of MAD, and many more. Examples are given in Figures 1 and 2.

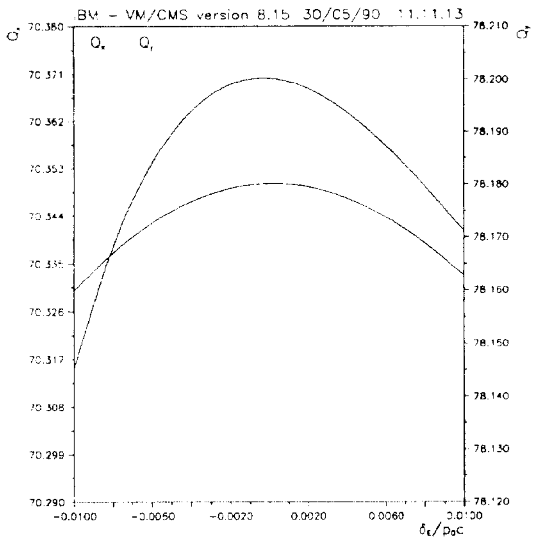

Figure 1: LEP tunes versus $\delta p/p$

Input Language Enhancements

For designing a new accelerator the input language used so far is perfectly adequate. However, when MAD runs in a control system computer, there should be a way to assign each single magnet a unique name. In principle one could enter an element definition for each magnet and then build up a beam line listing the names of all magnets without repetition. This method is however tedious and error-prone, and it quickly causes overflow of MAD's data pool.

The first part of the solution uses the class concept. It should be noted that an element keyword can be considered as the name of an element class, and an element definition as the creation of an object within this class. It is only a slight generalization to allow use of a previously defined object as a subclass, i. e. to define other objects using its name as a subclass name. The objects in

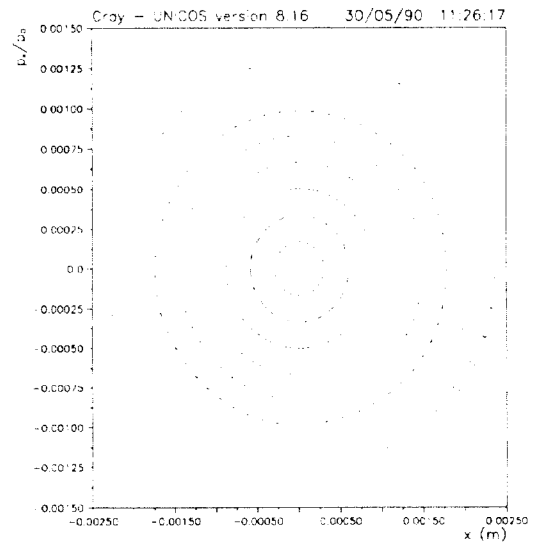

Figure 2: LEP tracking plot x versus p_x

a class inherit all attributes of the class, unless they are redefined differently. Examples:

```
MQ: QUADRUPOLE, L=1.6
QF: MQ, K1=0.0123
QD: MQ, K1=-0.0123
QT: MQ, L=1, TILT
```

Here MQ is the class of all machine quadrupoles which all have a length of 1.6 m. The classes QF and QD are the classes of the focussing and defocussing machine quadrupoles. Their length is inherited from the class MQ. The class QT is the class of all tilted (skewed) quadrupoles. Its length (1 m) is not inherited from MQ, since its definition explicitly mentions a length.

The second part of the solution is the introduction of the SEQUENCE construct. This looks roughly like the combination of element definitions and a line definition:

```
LEP: SEQUENCE
  ...
  QF1: QF, AT=25.3
  ...
  QD1: QD, AT=64.2
  ...
ENDSEQUENCE
```

From this construct MAD builds up a LINE definition. For each element listed it creates a definition. It also creates the necessary drift spaces to place the centres of the elements at the positions mentioned with "AT=...". This approach has three advantages:

- There is no need to allocate space for each new element. The program can rather use the same storage as for the class object, and just store the names in its name directory.
- There is no need to list an element name twice. This would be necessary if the element and line definitions were separated.
- The SEQUENCE can be easily generated from a data base.

Nevertheless it is possible to assign special attribute values to some element instances:

```
LEP: SEQUENCE
   ...
   CH27: HKICKER, AT=..., KICK=KCH27
   ...
ENDSEQUENCE
```

In this case MAD allocates extra space for the element, but this happens only for a few special elements.

Selective Output

The control system requires that selected element parameters and/or lattice functions are available in selected positions. Essentially this is an orthogonal selection from a table. It proceeds in two steps.

First the user has to specify the desired positions:

```
SELECT, FLAG=OPTICS, range1, ..., range5
```

"FLAG=OPTICS" tells MAD that the selection is valid for subsequent OPTICS commands. Up to five range selections select names of element classes or subclasses, with optional ranges of occurrence counts for output. All forms of range selections available in previous versions of MAD are also accepted.

Second the user specifies the quantities to be written:

```
OPTICS, CENTRE, COLUMN=name, ..., name
```

This causes running of the TWISS module, and stores the selected values for the selected positions in an internal table. The table is then written to disk in TFS format for further processing by other programs. The flag CENTRE causes output in the *centres* of the selected elements. If it is omitted, output occurs at the element *exits*. Further flexibility is added by replacing the SELECT commands by SPLIT commands:

```
SPLIT, range1, ..., range5, FRACTION=1/4
SPLIT, range1, ..., range5, FRACTION=1/2
SPLIT, range1, ..., range5, FRACTION=1/4
```

Like "SELECT, OPTICS" this selects elements for output, but in this example output occurs at 1/4, 1/2, and 3/4 of the length of selected elements. The same element can occur in several SPLIT commands.

Matching Enhancements

The most important enhancement is a new fitting method, LMDIF, which was borrowed from the DIMAD program [4]. This same method is also used for fitting in the HARMON part [5] of MAD version 8.

Several new matching constraints are now allowed:

- Constraints on position and slope of the (closed) orbit. These conditions may also be used to design orbit bumps.
- Constraints on elements of the transfer matrix between two positions, or on selected second-order terms thereof.
- The values of matching conditions may now themselves be variables. This permits interesting conditions to be applied.

As an example consider a spin rotator. Its design requires that the vertical phase advance from two vertical bends to the interaction point are 1.5 and 0.5 respectively. The elements (2, 1) of the transfer matrices for the same parts of the machine must have a specified ratio. This problem can be solved using the following commands.

```
RATIO := ...
RMATRIX, VB1/INTERACT, RM(2,1)=X, RM(3,4)=0
RMATRIX, VB2/INTERACT, RM(2,1)=RATIO*X, RM(3,4)=0
VARY, X
```

When MAD reads these conditions, it varies the parameter X together with the other variable parameters, and matches the two values RM(2,1) to the values of the specified expressions. This adjusts the ratio of the RM(2,1) values as desired. The conditions RM(3,4)=0 are equivalent to making the corresponding phase advances half-integers. Obviously the problem parameters should be chosen such that initially the phase advances are close to 0.5 and 1.5 respectively.

Tracking Enhancements

The enhancements in tracking mainly improve the tracking speed. It is now possible to specify lumping of a sequence before the constituents of the sequence are defined. The transfer map for the sequence will be evaluated when it is required for the first time, and it will automatically be recalculated when a system parameter changes.

Electron Beam Parameters

The program BEAMPARAM [6] has been adapted to become a module of MAD. It performs all the calculations it did as a stand-alone program, but it has no restrictions in the number of elements it can treat.

The parameters calculated include the tune shift, beam sizes, emittances, currents, luminosity, RF power, phase advance, bucket size, and the Touschek lifetime. As an option, the energy will be adapted to a given RF power.

Parameters such as the current, the beam sizes, and the emittances are available to MAD modules called subsequently.

References

[1] H. Grote and Ch. Iselin. *MAD Version 8, User's Reference Manual*. CERN/SL/90-13 (AP), CERN, 1990.

[2] R. Brun and J. Zoll. ZEBRA User Guide. CERN Computer Centre Program Library Long Write-up Q100. CERN, 1987.

[3] Ph. Defert, Ph. Hofmann, and R. Keyser. *The Table File System, the C Interfaces*. LAW Note 9. CERN, 1989.

[4] R. Servranckx et al. *User's Guide to the Program DIMAD*. SLAC 285 UC-28, SLAC, 1985.

[5] M. Donald and D. Schofield. *A User's Guide to the HARMON Program*. LEP Note 420. CERN, 1982.

[6] M. Hanney, J. M. Jowett, and E. Keil. *BEAMPARAM — A program for computing beam dynamics and performance of e^+e^- storage rings*. CERN/LEP-TH/88-2, CERN, 1988.