AN ORGANIZATION FOR NUCLEAR RESEARCH

CERN - SL DIVISION

CERN SL/92-46 (AP)

Status of MAD (Version 8.5) and Future Plans

F. C. Iselin

Abstract

The MAD project was started at CERN in 1981 as a flexible framework for optical computations.

The program is highly modular, it allows easy addition of new features. A format-free input language facilitates data preparation. This language is also understood by a variety of other programs; when moving data to other programs this avoids extensive translations of the data with the corresponding danger of making errors. The internal data structure describes the machine in a flexible way and access to this structure is simple. Canonical variables are used throughout the computations.

All this does not hamper the computational efficiency.

Geneva, Switzerland
23 September, 1992

SL/Div Reps

1 Introduction

The MAD project was started in 1981 at CERN to provide a flexible framework for optical computations. From the beginning the following features were considered important:

- Open-ended and modular design of the program to allow easy addition of new features.
- Format-free input language to facilitate data preparation. Later the requirement was added that the language should be understood by a variety of other programs. When moving data to other programs this would avoid extensive translations of the data with the corresponding danger of making errors.
- The internal data structure should describe the machine in a flexible manner and access to this structure should be simple.
- The program should be easy to maintain.
- Canonical variables should be used throughout the computations.

All these features should not hamper the computational efficiency. The Section 2 describes the available features of MAD. It is followed by two sections giving a general description of the "standard input language" and extensions to this language. Section 5 introduces the data structures while section 6 summarizes future plans.

2 Features of MAD

At present the MAD program [1] recognizes most physical elements which may constitute an accelerator:

- Marker, a do-nothing element, and drift space,
- Sector or parallel-face bending magnet,
- Long quadrupole, sextupole, or octupole,
- Thin multipole,
- Solenoid without fringing field,
- Orbit corrector, horizontal, vertical, or both,
- Orbit monitor, horizontal, vertical, or both,
- Accelerating cavity,
- Electrostatic separator,
- Rotation about longitudinal or vertical axis.

New element types can be added with moderate effort.
The executable commands can be loosely grouped as follows:

- Structure input (definition of physical elements and their order in the machine),
- Structure changes (installation, removal or motion of existing elements),
- Machine geometry calculations (survey),
- Machine imperfections (defined by systematic and/or random errors),
- Closed orbit finding,

- Linear lattice parameters including imperfections,
- Linear lattice matching including imperfections,
- Optical analysis in terms of uncoupled and fully coupled lattice functions,
- Tracking of beam envelopes,
- Complex constraints for matching (examples shown below),
- Electron beam parameters (equilibrium emittances and beam sizes),
- Tracking with TRANSPORT or Lie-algebraic methods,
- Orbit correction by the MICADO algorithm,
- Systematic energy loss due to radiation,
- Chromaticity correction by the methods of HARMON,
- Lie algebraic analysis (one-turn maps and normal form analysis),
- Spin kinematics by the SITF program,
- Plotting with various options.

3 Input Language

The "Standard Input Language" which is used in MAD has been defined at a workshop at SLAC in 1984, and described in [2]. This standard attempt to define an accelerator description which may be ported from one beam optics program to another without too much user intervention. It describes in some detail how elements and beam lines are defined. For executable commands however it only describes the format. This standard language has been adopted among others by the programs MAD, TRANSPORT, and DIMAD.

A command (definition or action) has the generic format

```
label:  keyword {attribute=value}
```

where the various parts have the meaning

label	Name given to the definition or command for later reference,
keyword	Selects the item to be defined or the action to be performed,
attribute	Keyword for a command attribute (length or excitation of a magnet, initial conditions for tracking, etc.),
value	Value given to the attribute.

The `attribute=value` group can be repeated as many times as required. The following is a complete example which computes the linear lattice functions for an accelerator cell:

```
TITLE,"Description of a test cell"

!  bending magnets:
B:  RBEND, L=35.09,ANGLE=0.011306116,&
    E1=0.005,E2=-.005,H1=.001,H2=.002
!  drift spaces:
L1: DRIFT,L=1.21
```

```
L2: DRIFT,L=0.56
L3: DRIFT,L=0.28
L4: DRIFT,L=1.57
!  quadrupoles:
QF: QUAD,L=1.6,K1=-0.02268553,TYPE=MQ
QD: QUAD,L=1.6,K1= 0.022683642,TYPE=MQ
!  accelerator cavity:
RF: RFCAV,L=0.1,VOLT=100,FREQ=114,LAG=0.4,&
    HARMON=30,TFILL=1,SHUNT=1
!  sextupoles:
SF1:SEXT,L=0.4,K2=-0.13129
SD1:SEXT,L=0.76,K2= 0.26328
!  order of occurrence of elements in the cell:
LIN(SF,SD):LINE=(L4,B,L2,SF,L3,QF,L1,B,L2,SD,L3,QD)
!  make this sequence active and compute lattice functions:
USE,LIN(SF1,SD1)
TWISS
```

4 Extensions to the Input Language

Over the years a number of extensions to the standard language have been adopted in MAD. They tend to increase the capacity of the language to express complex machines and complicated computations.

4.1 Element Classes

An element class can be thought of as a new keyword defining an element with given default attributes. Any element which has been previously defined can thus be used to define new elements. Assume we define three classes of quadrupoles:

```
MQF: QUADRUPOLE,L=LQM,K1=KQD   ! Defocusing quadrupoles
MQD: QUADRUPOLE,L=LQM,K1=KQF   ! Focusing quadrupoles
MQT: QUADRUPOLE,L=LQT,TILT     ! Tilted quadrupoles
```

These definitions define three classes of quadrupoles with given default lengths; the focusing and defocusing quadrupoles also have default strengths. These classes can be used to define the actual magnets:

```
QD1:  MQD          !  Defocusing quadrupoles
QD2:  MQD
QD3:  MQD
...
QF1:  MQF          !  Focusing quadrupoles
QF2:  MQF
QF3:  MQF
...
QT1:  MQT,K1=KQT1  !  Skewed quadrupoles
QT2:  MQT,K1=KQT2
```

...

Note that the derived magnets *inherit* the properties of their class. During computations, positions in the machine can be referred to by class name or by element names in the same right.

4.2 Output Selection

Additionally, every element carries also a second identifier called TYPE which may also be referred to. Together with the class mechanisms, the TYPE attribute provides a very flexible way to build output tables. Example:

```
   SELECT, OPTICS, {class} {element} {range} {type}
OPTICS, COLUMNS={name}
```

The above SELECT command sets output flags for the OPTICS command on all elements for which at least one of the following conditions is true:

- The element is a member or belongs to a subclass of one of the classes listed,
- The element's name is listed in **element**,
- The element occurs in one of the ranges in **range**,
- The element carries one of the TYPE identifiers in **type**.

The OPTICS command computes the lattice functions and writes all functions whose name is mentioned as one of the names after the keyword COLUMN. The result is stored as a "dynamic table" as described in the next subsection. This table can later be plotted, tabulated, or otherwise postprocessed. Example:

```
SELECT,OPTICS,MONITOR,KICKER
OPTICS,COLUMN=BETX,MUX,BETY,MUY
```

These commands create a table of the horizontal and vertical β and μ values for all beam position monitors and orbit correction kickers. It could be used in a controls program to compute orbit correctors; or it could be tabulated or plotted in various ways.

4.3 Dynamic Tables

Output tables can be created in the form of self-describing table files. These files have column headers, giving the names and formats of the columns, as well as table descriptors which contain global information. Example:

```
* S          NAME        BETX          MUX          BETY         MUY
$ %f         %16s        %f            %f           %f           %f
@ GAMTR      %f          64.3336
@ ALFA       %f          .241615E-03
@ XIY        %f          -.455669
@ XIX        %f          2.05279
@ QY         %f          .250049
@ QX         %f          .249961
```

```
@ CIRCUM      %lf          79.0000000000
@ DELTA       %f           .000000E+00
@ COMMENT     %20s         "DATA FOR TEST CELL"
@ ORIGIN      %28s         "MAD 8.5/4 Apollo - UNIX"
@ DATE        %08s         "20/03/92"
@ TIME        %08s         "10.29.18"
  .000000E+00 SEQ          132.276     .000000E+00  23.6099     .000000E+00
  1.57000     [000000]     124.847     .194446E-02  25.1829     .102546E-01
  36.6600     B1           24.8429     .110430      126.379     .117095
  37.2200     [000001]     24.2715     .114060      129.019     .117793
  37.6200     SF1          23.8831     .116704      130.925     .118283
  37.9000     [000002]     23.6210     .118580      132.268     .118621
  39.5000     QF1          23.6209     .129474      132.269     .120528
  40.7100     [000003]     24.8112     .137432      126.521     .122017
  75.8000     B2           124.710     .246071      25.2151     .228705
  76.3600     [000004]     127.328     .246779      24.6233     .232283
  77.1200     SD1          130.934     .247716      23.8717     .237273
  77.4000     [000002]     132.277     .248054      23.6098     .239150
  79.0000     QD1          132.276     .249961      23.6099     .250049
  79.0000     ENDM         132.276     .249961      23.6099     .250049
  79.0000     SEQ          132.276     .249961      23.6099     .250049
```

4.4 Arithmetic Expressions

All parameters with a real value can be specified to have the value of an arbitrary arithmetic expression in other parameters. The allowed operations include the basic operations add, subtract, multiply, divide and power, the most common elementary functions like sine, cosine, logarithm, exponential. MAD will evaluate all dependencies created whenever an independent variable used in an expression changes.

Several random functions are also allowed:

RANF()	Random number, uniformly distributed in [0,1],
GAUSS()	Random number, Gaussian distribution with unit standard deviation,
TGAUSS(X)	Random number, Gaussian distribution with unit standard deviation, truncated at X standard deviations,
USER0()	Random number, user-defined distribution without arguments,
USER1(X)	Random number, user-defined distribution with one argument,
USER2(X,Y)	Random number, user-defined distribution with two arguments.

Normally random generators immediately generate a new random value which is then stored as a constant. For machine imperfection definitions, however, new random values are generated as often as needed to assign errors to all specified elements.

The expression mechanism is very powerful. It allows to set up complex dependencies among variables, and to impose almost arbitrary matching conditions.

As a simple example consider the matching of a cell so as to have $\beta_x = \beta_y$:

```
! define a simple cell:
CEL: LINE=(...)
```

```
 !  make the cell active and initiate matching USE,CEL
CELL
   !  impose some constraints
   ...
   !  vary two quadrupoles
   VARY,QF[K1]
   VARY,QD[K1]
   !  impose equality of the two beta values
   VARY,AUX
   CONSTRAINT,#E,BETX=AUX,BETY=AUX
   !  invoke matching
   ...
ENDMATCH
```

The trick is to vary an auxiliary variable AUX, and to request that both β-function must be equal to this value.

A second example involves a spin matching condition. The positions VB1 and VB2 refer to two vertical bends. The condition imposes that the vertical phase advance between these two positions and the interaction point shall be an odd multiple of 1/2, and that the transfer matrix element (2,1) from VB2 to the interaction point IP differs by a given factor from the same element taken from VB1 to IP:

```
!  define the insertion to be matched
INS: LINE=(...,VB1,...,VB2,...,IP)
!  define the factor
FACTOR: CONSTANT=SIN(PI/3)
 !  make insertion the active line and initiate matching
USE,INS
MATCH,LINE=CEL
   !  vary an the auxiliary variable AUX and some K1's
   ...
   !  impose conditions on the transfer matrix
   VARY,AUX
   RMATRIX,VB1/IP,RM(3,4)=0,&!  phase advance condition
      RM(2,1)=AUX !  AUX becomes = to RM(2,1)
   RMATRIX,VB2/IP,RM(3,4)=0,&!  phase advance condition
      RM(2,1)=FACTOR*AUX !  RM(2,1) becomes = FACTOR * AUX
   !  invoke matching
   ...
ENDMATCH
```

The two conditions RM(3,4)=0 impose the proper phase advance, while the two conditions involving RM(2,1) and the auxiliary variable AUX impose the required scaling of this matrix element.

4.5 Beam Line Sequences

In a control system environment the sequence of elements in the accelerator is usually kept in a data base. It is the convenient to use a more rigid format for the sequence of elements in a more rigid format. For this purpose the notion of a beam sequence has been introduced:

```
POSB1=19.115

SEQ: SEQUENCE
   B1:  B,AT=POSB1
   SF1:  SF,AT=37.42
   QF1:  QF,AT=38.70
   B2:  B,AT=58.255,ANGLE=B1[ANGLE]
   SD1:  SD,AT=76.74
   QD1:  QD,AT=78.20
   ENDM: MARKER,AT=79.0
ENDSEQUENCE
```

The sequence must be preceded by *class* definitions. The sequence definition itself combines the process of element definition with their installation at given positions in the ring. The first name on each line is an unique name given to the element, the second name is an element class serving as a templated to define its attributes. The two names are followed by the longitudinal position in the machine where the element shall be placed, and optionally by additional element attributes which will override the class attributes. MAD generates the drift spaces automatically.

4.6 Sequence Editor

For machine studies beam sequences must often be modified by adding, removing, or displacing elements. This is made possible by using the sequence editor, which recognizes the following commands:

```
SEQEDIT,SEQUENCE=sequence


! remove single named element
REMOVE,CLASS=element
! remove whole element class
REMOVE,CLASS=class
! remove all elements matching pattern (see below)
REMOVE,PATTERN=pattern


! install at absolute position
INSTALL,ELEMENT=element,AT=expression
! install at position relative to an element
INSTALL,ELEMENT=element,AT=expr,FROM=element


! move to absolute position
```

```
MOVE,ELEMENT=element,TO=expr
!  move to position relative to an element
MOVE,ELEMENT=element,TO=expr,FROM=element
!  move relative to old position
MOVE,ELEMENT=element,BY=expr


ENDEDIT
```

4.7 "Wildcard" Strings

Wildcard strings are patterns similar to the patterns used in the UNIX "grep" utility. They can be used in the sequence editor for selectively removing all elements which match a pattern; or when saving matching results to select the definitions to be saved.

5 Data Structures

The internal data structures for MAD are based on the CERN-written ZEBRA package. This package provides:

- Dynamic memory allocation under FORTRAN,
- Data block (bank) format known to the system,
- Automatic garbage collection for deleted blocks,
- Automatic updating of pointers,
- Debugging tools (printing of data structures),
- Reference pointers across trees,
- Saving and retrieving of complete trees on disk,
- Structural pointers in the fashion of linear lists:

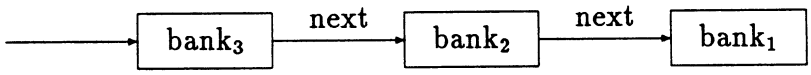

Figure 1: Example of a Linear List

- Structural pointers in the fashion of a tree:

Using these features very complex data structures can be built; This makes the implementation of the more exotic features easy.

6 Future Plans

At present we are working on the implementation of Differential Algebra facilities. It is expected that this will allow a simplified approach to the correction of non-linear aberrations. It may also serve to a certain extent help in determining the dynamic aperture, though this seems doubtful at the present time.

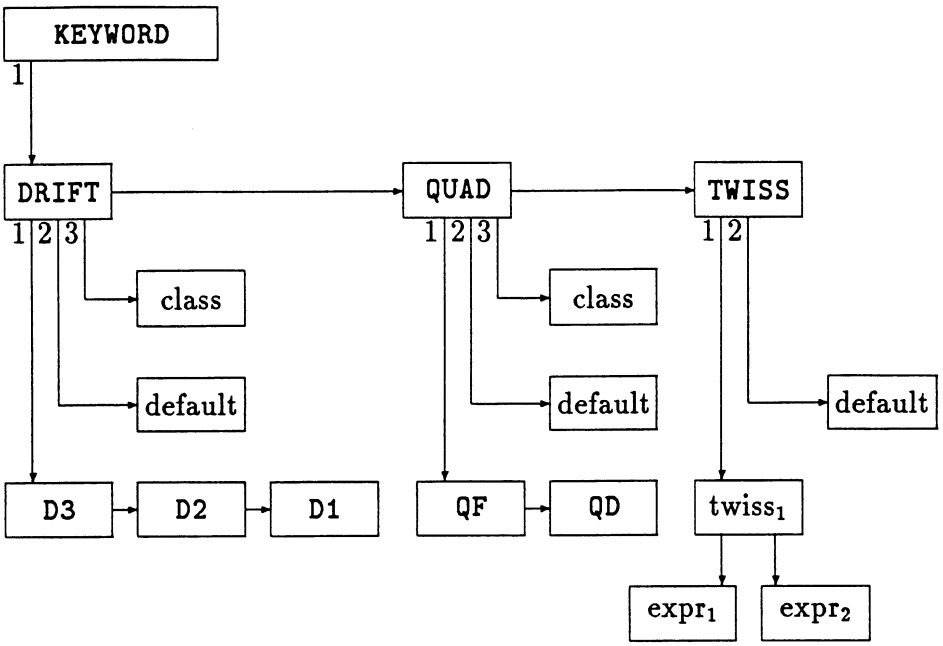

Figure 2: Example of a Tree Structure

Regardless of all its features, the ZEBRA system, being based on FORTRAN, is sometimes painful to use. As it does not support double-precision data with double-word alignment, it is necessary to copy all double-precision data to local storage before use. This can cause loss of computational speed, if care is not taken to organize these data moves properly. Also, bank pointers must always be kept in specific locations, which may also cause considerable difficulty for the programmer. We are therefore also investigating the possibility to rewrite MAD in a modern structured language. The obvious candidate would be C++, but for availability reasons we might have to consider another language, like plain C or FORTRAN-90.

References

[1] Grote H and Iselin F C 1991 *CERN Report* CERN/SL/90/13 (AP), Rev. 2.

[2] Carey D C and Iselin F C 1984 *CERN Report* LEP-TH/84-10; 1984 Summer Study on the Design and Utilization of the Superconducting Super Collider, Snowmass, Colorado, June 23–July 13.