EUROPEAN ORGANIZATION FOR NUCLEAR RESEARCH

USER'S GUIDE

- Table of contents

hansg, June 17, 2002

Standard CERN Copyright Notice:

CERN

EUROPEAN ORGANISATION FOR NUCLEAR RESEARCH

Program name:	MAD --- Methodical Accelerator Design
CERN program library entry:	T5001
Authors or contacts:	Frank.Schmidt@cern.ch SL Division CERN CH-1211 GENEVA 23 SWITZERLAND

---

hansg, January 24, 1997

EUROPEAN ORGANIZATION FOR NUCLEAR RESEARCH

Conventions

The accelerator and/or beam line to be studied is described as a sequence of beam elements placed sequentially along a reference orbit. The reference orbit is the path of a charged particle having the central design momentum of the accelerator through idealised magnets with no fringe fields (see Figure 1).

The reference orbit consists of a series of straight line segments and circular arcs. It is defined under the assumption that all elements are perfectly aligned. The accompanying tripod of the reference orbit spans a local curvilinear right handed coordinate system *(x,y,s)* The local *s*-axis is the tangent to the reference orbit. The two other axes are perpendicular to the reference orbit and are labelled *x* (in the bend plane) and *y* (perpendicular to the bend plane).

- Closed Orbit
- Global Reference System
- Local Reference System
- Sign Conventions for Magnetic Fields
- Variable
  - Canonical Variables Describing Orbits
  - Normalised Variables and other Derived Quantities
- Physical Units

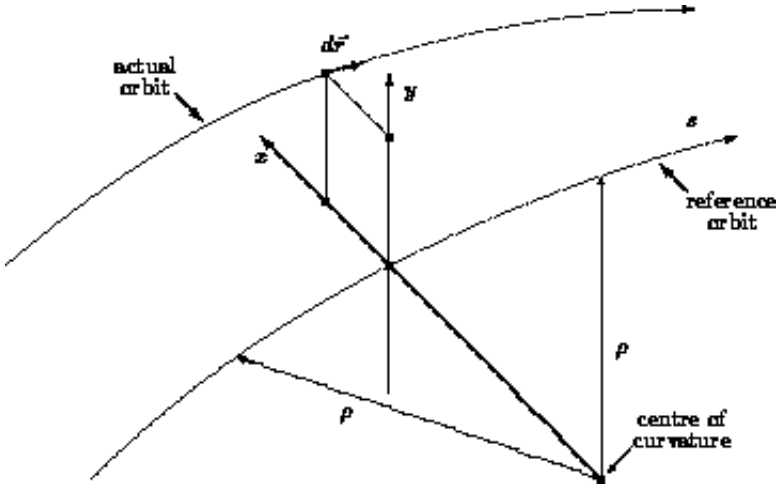


Figure 1: Local Reference System

hansg, May 8, 2001

EUROPEAN ORGANIZATION FOR NUCLEAR RESEARCH 

Closed Orbit

Due to various errors like misalignment errors, field errors, fringe fields etc., the closed orbit does not coincide with the reference orbit. It also changes with the momentum error. The closed orbit is described with respect to the reference orbit, using the local reference system (*x, y, s*). It is evaluated including any nonlinear effects.

MAD also computes the betatron and synchrotron oscillations with respect to the closed orbit. Results are given in the local (*x, y, s*)-system defined by the reference orbit.

hansg, January 24, 1997

EUROPEAN ORGANIZATION FOR NUCLEAR RESEARCH 

Global Reference System

The global reference orbit of the accelerator is uniquely defined by the sequence of physical elements. The local reference system (x, y, s) may thus be referred to a global Cartesian coordinate system (X, Y, Z) (see Figure 1). The positions between beam elements are numbered 0,...,i,...n. The local reference system (x_i, y_i, s_i) at position i, i.e. the displacement and direction of the reference orbit with respect to the system (X, Y, Z) are defined by three displacements (X_i, Y_i, Z_i) and three angles ($Theta_i$, Phi_i, Psi_i) The above quantities are defined more precisely as follows:

- X: Displacement of the local origin in X-direction.
- Y: Displacement of the local origin in Y-direction.
- Z: Displacement of the local origin in Z-direction.
- THETA: Angle of rotation (azimuth) about the global Y-axis, between the global Z-axis and the projection of the reference orbit onto the (Z, X)-plane. A positive angle THETA forms a right-hand screw with the Y-axis.
- PHI: Elevation angle, i.e. the angle between the reference orbit and its projection onto the (Z, X)-plane. A positive angle PHI correspond to increasing Y. If only horizontal bends are present, the reference orbit remains in the (Z, X)-plane. In this case PHI is always zero.
- PSI: Roll angle about the local s-axis, i.e. the angle between the intersection (x, y)- and (Z, X)-planes and the local x-axis. A positive angle PSI forms a right-hand screw with the s-axis.

The angles (THETA, PHI, PSI) are **not** the Euler angles. The reference orbit starts at the origin and points by default in the direction of the positive Z-axis. The initial local axes (x, y, s) coincide with the global axes (X, Y, Z) in this order. The six quantities (X_0, Y_0, Z_0, THETA_0, PHI_0, PSI_0) thus all have zero initial values by default. The program user may however specify different initial conditions.

Internally the displacement is described by a vector V and the orientation by a unitary matrix W. The column vectors of W are the unit vectors spanning the local coordinate axes in the order (x, y, s). V and W have the values:

$$V = \begin{pmatrix} X \\ Y \\ Z \end{pmatrix}, \qquad W = \Theta\Phi\Psi$$

where

$$\Theta = \begin{pmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{pmatrix}, \quad \Phi = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & \sin\phi \\ 0 & -\sin\phi & \cos\phi \end{pmatrix}, \quad \Psi = \begin{pmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{pmatrix}.$$

The reference orbit should be closed and it should not be twisted. This means that the displacement of the local reference system must be periodic with the revolution frequency of the accelerator, while the position angles must be periodic modulo(2 pi) with the revolution frequency. If PSI is not periodic module(2 pi), coupling effects are introduced. When advancing through a beam element, MAD computes V_i and W_i by the recurrence relations

$$V_i = W_{i-1} R_i + V_{i-1}, \; W_i = w_{i-1} S_i .$$

The vector R_i is the displacement and the matrix S_i is the rotation of the local reference system at the exit of the element i with respect to the entrance of the same element. The values of R_i and S_i are listed in the: straight reference system for each physical element type.

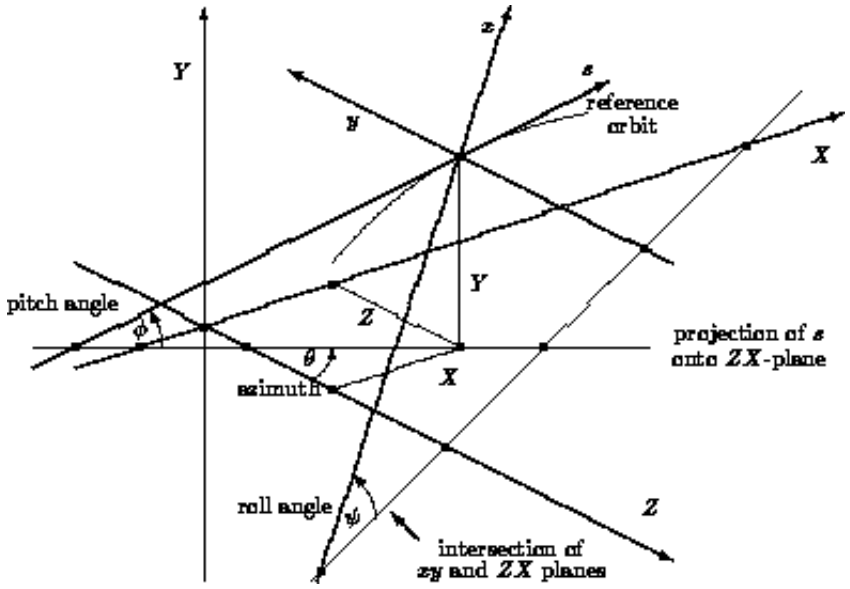


Figure 1: Global Reference System

hansg, January 24, 1997

Local Reference Systems

Reference System for Straight Beam Elements

In straight elements the local reference system is simply translated by the length of the element along the local *s*-axis. This is true for

- Drift space,
- Quadrupole,
- Sextupole,
- Octupole,
- Solenoid,
- RF cavity,
- Electrostatic separator,
- Closed orbit corrector,
- Beam position monitor.

The corresponding *R*, *S* are

$$\boldsymbol{R} = \begin{pmatrix} 0 \\ 0 \\ \boldsymbol{L} \end{pmatrix}, \qquad \boldsymbol{S} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}.$$

A rotation of the element about the *S*-axis has no effect on *R* and *S*, since the rotations of the reference system before and after the element cancel.

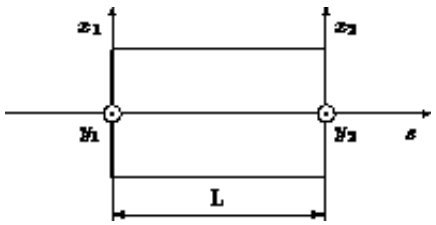


Figure 1: Reference System for Straight Beam Elements

Reference System for Bending Magnets

Bending magnets have a curved reference orbit. For both rectangular and sector bending magnets

$$R = \begin{pmatrix} \rho(\cos\alpha - 1) \\ 0 \\ \rho \sin\alpha \end{pmatrix}, \qquad S = \begin{pmatrix} \cos\alpha & 0 & -\sin\alpha \\ 0 & 1 & 0 \\ \sin\alpha & 0 & \cos\alpha \end{pmatrix},$$

where alphais the bend angle. A positive bend angle represents a bend to the right, i.e. towards negative *x* values. For sector bending magnets, the bend radius is given by rho, and for rectangular bending magnets it has the value

rho = *L* / 2 sin(alpha/2).

If the magnet is rotated about the *s*-axis by an angle psi, *R* and *S* are transformed by

$R^* = T\,R$, $S^* = T\,S\,T^{-1}$.

where *T* is the orthogonal rotation matrix

$$T = \begin{pmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{pmatrix}.$$

The special value psi = pi/2 represents a bend down.

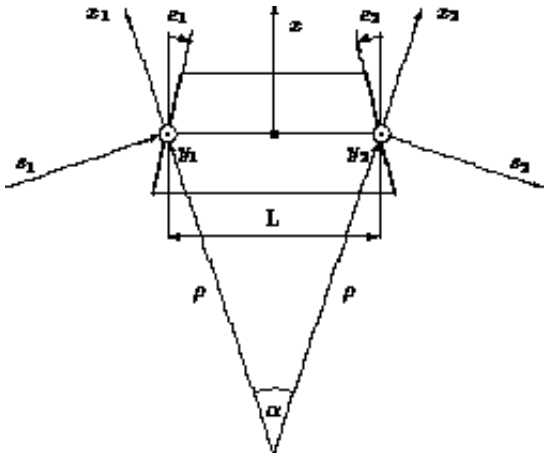


Figure 2: Reference System for Rectangular Bends; The signs of the pole-face rotations are positive as shown.

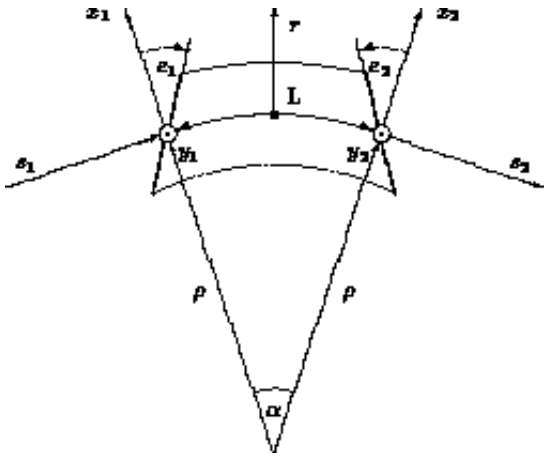


Figure 3: Reference System for Sector Bends; The signs of the pole-face rotations are positive as shown.

Elements which do not Change the Local Reference

MARKER elements do not affect the reference orbit. They are ignored for geometry calculations.

hansg, January 24, 1997

Drift Space

```
label: DRIFT,L=real;
```

A DRIFT space has one real attribute:

- L: The drift length (default: 0 m)

Examples:

```
DR1:   DRIFT,L=1.5;
DR2:   DRIFT,L=DR1[L];
```

The length of DR2 will always be equal to the length of DR1. The straight reference system for a drift space is a cartesian coordinate system.

hansg, January 24, 1997

Quadrupole

```
label: QUADRUPOLE,L=real,K1=real,K1S=real,TILT=real;
```

A QUADRUPOLE has four real attributes:

- L: The quadrupole length (default: 0 m).
- K1: The normal quadrupole coefficient

  K_1 = 1/(*B* rho) (ðB_y/ð x).

  The default is 0 m**(-2). A positive normal quadrupole strength implies horizontal focussing of positively charged particles.
- K1S: The skew quadrupole coefficient

  K_{1s} = 1/(2 *B* rho) (ðB_x/ð x - ðB_y/ð y)

  where (x,y) is now a coordinate system rotated by -45^o around s with respect to the normal one. The default is 0 m**(-2). A positive skew quadrupole strength implies defocussing (!) of positively charged particles in the (x,s) plane rotated by 45^o around s (particles in this plane have x = y > 0).

- TILT: The roll angle about the longitudinal axis (default: 0 rad, i.e. a normal quadrupole). A positive angle represents a clockwise rotation. A TILT=pi/4 turns a positive normal quadrupole into a negative skew quadrupole.

  Please note that contrary to MAD8 one has to specify the desired TILT angle, otherwise it is taken as 0 rad. This was needed to avoid the confusion in MAD8 about the actual meaning of the TILT attribute for various elements.

Note also that K_1/K_{1s} can be considered as the normal or skew quadrupole components of the magnet on the bench, while the TILT attribute can be considered as an tilt alignment error in the machine. In fact, a positive K_1 with a tilt=0 is equivalent to a positive K_{1s} with positive tilt=+pi/4.

Example:

```
QF: QUADRUPOLE,L=1.5,K1=0.001;
```

The straight reference system for a quadrupole is a cartesian coordinate system.

hansg, frs, August 28, 2003

EUROPEAN ORGANIZATION FOR NUCLEAR RESEARCH 

Sextupole

```
label: SEXTUPOLE,L=real,K2=real,K2S=real,TILT=real;
```

A SEXTUPOLE has four real attributes:

- L: The sextupole length (default: 0 m).
- K2: The normal sextupole coefficient

  K_2 = 1/(*B* rho) ($\partial^2 B_y / \partial x^2$).

  (default: 0 m**(-3)).
- K2S: The skew sextupole coefficient

  K_{2S} = 1/(2 *B* rho) ($\partial^2 B_x / \partial x^2$ - $\partial^2 B_y / \partial y^2$).

  where (x,y) is now a coordinate system rotated by -30^o around s with respect to the normal one. (default: 0 m**(-3)). A positive skew sextupole strength implies defocussing (!) of positively charged particles in the (x,s) plane rotated by 30^o around s (particles in this plane have x > 0, y > 0).

- TILT: The roll angle about the longitudinal axis (default: 0 rad, i.e. a normal sextupole). A positive angle represents a clockwise rotation. A TILT=pi/6 turns a positive normal sextupole into a negative skew sextupole.

  Please note that contrary to MAD8 one has to specify the desired TILT angle, otherwise it is taken as 0 rad. This was needed to avoid the confusion in MAD8 about the actual meaning of the TILT attribute for various elements.

Note also that K_2/K_{2s} can be considered as the normal or skew sextupole components of the magnet on the bench, while the TILT attribute can be considered as an tilt alignment error in the machine. In fact, a positive K_2 with a tilt=0 is equivalent to a positive K_{2s} with positive tilt=+pi/6.

Example:

```
S: SEXTUPOLE,L=0.4,K2=0.00134;
```

The straight reference system for a sextupole is a cartesian coordinate system.

hansg, frs, August 28, 2003

EUROPEAN ORGANIZATION FOR NUCLEAR RESEARCH 

Octupole

```
label: OCTUPOLE,L=real,K3=real,K3S=real,TILT=real;
```

An OCTUPOLE has four real attributes:

- L: The octupole length (default: 0 m).
- K3: The normal octupole coefficient

  K_3 = 1/(*B* rho) ($\partial^3 B_y/\partial x^3$).

  (default: 0 m**(-4)).
- K3S: The skew octupole coefficient

  K_3 = 1/(2 *B* rho) ($\partial^3 B_x/\partial x^3$ - $\partial^3 B_y/\partial y^3$).

  where (x,y) is now a coordinate system rotated by -22.5^o around s with respect to the normal one. (default: 0 m**(-4)). A positive skew octupole strength implies defocussing (!) of positively charged particles in the (x,s) plane rotated by 22.5^o around s (particles in this plane have x > 0, y > 0).

- TILT: The roll angle about the longitudinal axis (default: 0 rad, i.e. a normal octupole). A positive angle represents a clockwise rotation. A TILT=pi/8 turns a positive normal octupole into a negative skew octupole.

  Please note that contrary to MAD8 one has to specify the desired TILT angle, otherwise it is taken as 0 rad. This was needed to avoid the confusion in MAD8 about the actual meaning of the TILT attribute for various elements.

Note also that K_3/K_{3s} can be considered as the normal or skew quadrupole components of the magnet on the bench, while the TILT attribute can be considered as an tilt alignment error in the machine. In fact, a positive K_3 with a tilt=0 is equivalent to a positive K_{3s} with positive tilt=+pi/8.

Example:

```
O3: OCTUPOLE,L=0.3,K3=0.543;
```

The straight reference system for a octupole is a cartesian coordinate system. Octupoles are normally treated as thin lenses, except when tracking by Lie-algebraic methods.

hansg, frs, August 28, 2003

EUROPEAN ORGANIZATION FOR NUCLEAR RESEARCH 

Solenoid

`label: SOLENOID, L=real, KS=real;` (**thick** version)
`label: SOLENOID, L=0, KS=real, KSI=real;` (**thin** version)

A SOLENOID has two (three) real attributes:

- L: The length of the solenoid (default: 0 m)
- KS: The solenoid strength K_s (default: 0 rad/m). For positive KS and positive particle charge, the solenoid field points in the direction of increasing *s*.
- KSI: The solenoid integrated strength K_s*L (default: 0 rad). This additional attribute is needed only when using the thin solenoid, where *L=0*!
- *KNL & KSL: Take note that one can specify multipole coefficients but they have no effect in MAD-X proper but are used for solenoids with multipoles in PTC.*

Example:

```
SOLO: SOLENOID, L=2., KS=0.001;
THINSOLO: SOLENOID, L=0, KS=0.001, KSI=0.002;
```

The straight reference system for a solenoid is a cartesian coordinate system.

hansg, January 27, 1977

EUROPEAN ORGANIZATION FOR NUCLEAR RESEARCH 

RF Cavity

```
label: RFCAVITY, L=real,VOLT=real,LAG=real,HARMON=integer,FREQ=real;
```

An RFCAVITY has eight real attributes and one integer attribute:

- L: The length of the cavity (default: 0 m)
- VOLT: The peak RF voltage (default: 0 MV). The effect of the cavity is

  delta(E) = VOLT * sin(2 pi * (LAG - HARMON * f_0 t)).

- LAG: The phase lag [2pi] (default: 0).
- FREQ: The frequency [MHz] *fenergy*(no default). Note that if the RF frequency is not given, it is computed from the harmonic number and the revolution frequency f_0 as before. However, for accelerating structures this makes no sense, and the frequency is mandatory.
- HARMON: The harmonic number h (no default). Only if the frequency is not given.
- *Please take note, that the following MAD8 attributes: BETRF, PG, SHUNT and TFILL are currently not implemented in MAD-X!<*
- *Note as well that twiss is 4D only. As a consequence the TWISS parameters in the plane of non-zero dispersion may not close as expected. Therefore, it is best to perform TWISS in 4D only, i.e. with cavities switched off. If 6D is needed one has to use the ptc_twiss command.*

A cavity requires the particle energy (ENERGY) and the particle charge (CHARGE) to be set by a BEAM command before any calculations are performed.

Example:

```
BEAM,PARTICLE=ELECTRON,ENERGY=50.0;
CAVITY:   RFCAVITY,L=10.0,VOLT=150.0,LAG=0.0,HARMON=31320;
```

The straight reference system for a cavity is a cartesian coordinate system.

hansg, January 24, 1997

EUROPEAN ORGANIZATION FOR NUCLEAR RESEARCH 

BEAM: Set Beam Parameters

Many commands in MAD-X require the setting of various quantities related to the beam in the machine. Therefore, MAD-X will stop with a fatal error if an attempt is made to expand (USE) a sequence for which no BEAM command has been issued before.

The quantities are entered by a BEAM command:

```
BEAM, PARTICLE=name,MASS=real,CHARGE=real,
      ENERGY=real,PC=real,GAMMA=real,
      EX=real,EXN=real,EY=real,EYN=real,
      ET=real,SIGT=real,SIGE=real,
      KBUNCH=integer,NPART=real,BCURRENT=real,
      BUNCHED=logical,RADIATE=logical,BV=integer,SEQUENCE=name;
```

Warning: BEAM updates, i. e. it replaces attributes explicitely mentioned, but does not return to default values for others! To reset to beam value defaults, use RESBEAM. The particle restmass and charge are defined by:

- PARTICLE: The name of particles in the machine. MAD knows the restmass and the charge for the following particles:
  - POSITRON: The particles are positrons (MASS=m_e, CHARGE=1),
  - ELECTRON: The particles are electrons (MASS=m_e, CHARGE=-1),
  - PROTON: The particles are protons (MASS=m_p, CHARGE=1),
  - ANTIPROTON: The particles are anti-protons (MASS=m_p, CHARGE=-1).
  - POSMUON: The particles are positive muons (MASS=m_{mu}, CHARGE=1),
  - NEGMUON: The particles are negative muons (MASS=m_{mu}, CHARGE=-1).

Therefore neither restmass nor charge can be modified for these predefined particles. On the other hand, for ions and all other user defined particles the name, restmass, and charge can be entered independently.

By default the total particle energy is 1 GeV. A different value can be defined by one of the following:

- ENERGY: The total energy per particle in GeV. If given, it must be greater then the particle restmass.
- PC: The momentum per particle in GeV/c. If given, it must be greater than zero.
- GAMMA: The ratio between total energy and rest energy of the particles: GAMMA = E / m_0. If given, it must be greater than one. If the restmass is changed a new value for the energy should be entered. Otherwise the energy remains unchanged, and the momentum PC and the quantity GAMMA are recalculated. The emittances are defined by:
- EX: The horizontal emittance E_x (default: 1 m).
- EY: The vertical emittance E_y (default: 1 m).
- ET: The longitudinal emittance E_t (default: 1 m). The emittances can be replaced by the

normalised emittances and the energy spread:

- EXN: The normalised horizontal emittance [m]: $E_{xn} = 4\ (\mathrm{GAMMA}^2 - 1)^{1/2}\ E_x$ (ignored if E_x is given).
- EYN: The normalised vertical emittance [m]: $E_{yn} = 4\ (\mathrm{GAMMA}^2 - 1)^{1/2}\ E_y$ (ignored if E_x is given).
- SIGT: The bunch length c sigma(t) in [m].
- SIGE: The *relative* energy spread sigma(E)/E in [1].

Certain commands compute the synchrotron tune Q_s from the RF cavities. If Q_s is non-zero, the relative energy spread and the bunch length are

sigma(E) / $p_0\ c = (2\ pi\ Q_s\ E_t\ /\ ETA\ C)^{1/2}$,

c sigma(t) =$(\mathrm{ETA\ C\ E_t\ /\ 2\ pi\ Q_s})^{1/2}$.

C is the machine circumference, and

$ETA = \mathrm{GAMMA}^{-2} - \mathrm{GAMMA(transition)}^{-2}$.

The order of precedence in the parameter evaluation is given below:

```
particle->(mass+charge)
energy->pc->gamma->beta
ex->exn
ey->eyn
current->npart
et->sigt->sige
```

where any item to the left takes precendence over the others.

Finally, the BEAM command accepts

- KBUNCH: The number of particle bunches in the machine (default: 1).
- NPART: The number of particles per bunch (default: 0).
- BCURRENT: The bunch current (default: 0 A).
- BUNCHED: A logical flag. If set, the beam is treated as bunched whenever this makes sense.
- RADIATE: A logical flag. If set, synchrotron radiation is considered in all bipolar magnets.
- BV: an integer specifying the direction of the particle movement in a beam line; either +1 (default), or -1. For a detailed explanation see under bv flag.
- SEQUENCE: this attaches the beam command to a specific sequence; if the name is omitted, the BEAM command refers to the default beam always present. Sequences without attached beam use this default beam. When updating a beam, the corresponding sequence name, if any, must always be mentioned.

The BEAM command changes only the parameters entered. The command RESBEAM resets all beam data to their beam value defaults.

Examples:

```
BEAM,   PARTICLE=ELECTRON,ENERGY=50,EX=1.E-6,EY=1.E-8,SIGE=1.E-3;
 ...
BEAM,   RADIATE;
 ...
RESBEAM;
BEAM,   EX=2.E-5,EY=3.E-7,SIGE=4.E-3;
```

The first command selects electrons, and sets energy and emittances. The second one turns on synchrotron radiation. The last two select positrons (by default), set the energy to 1 GeV (default), clear the synchrotron radiation flag, and set the emittances to the values entered.

Some program modules of MAD-X may also store data into a beam data block. Expressions may refer to data in this beam data block using the notation

```
BEAM->attribute-name
```

or

```
BEAM%sequence->attribute-name.
```

This notation refers to the value of attribute-name found in the default BEAM resp. the beam belonging to the sequence given. This can be used for receiving or using values, e.g.

```
value,beam%lhcb2->bv;
```

or for storing values in the beam (this does NOT trigger an update of dependent variables !), e.g.

```
beam->charge=-1;
```

The current values in the BEAM bank can be obtained by the command

```
show,beam;
```

resp.

```
show,beam%sequence;
```

hansg 11.9.2000

EUROPEAN ORGANIZATION FOR NUCLEAR RESEARCH 

ELSEPARATOR: Electrostatic Separator

```
label: ELSEPARATOR,L=real,EX=real,EY=real,TILT=real;
```

An ELSEPARATOR (electrostatic separator) has four real attributes:

- L: The length of the separator (default: 0 m).
- EX: The horizontal electric field strength (default: 0 MV/m). A positive field increases p_x for positive particles.
- EY: The vertical electric field strength (default: 0 MV/m). A positive field increases p_y for positive particles.
- TILT: The roll angle about the longitudinal axis (default: 0 rad). A positive angle represents a clockwise of the electrostatic separator.

A separator requires the particle energy (ENERGY) and the particle charge (CHARGE) to be set by a BEAM command before any calculations are performed.

Example:

```
BEAM,PARTICLE=POSITRON,ENERGY=50.0;
SEP:    ELSEPARATOR,L=5.0,EY=0.5;
```

The straight reference system for a separator is a cartesian coordinate system.

hansg, frs, August 28, 2003

Closed Orbit Correctors

Three types of closed orbit correctors are available:

- HKICKER, a corrector for the horizontal plane,
- VKICKER, a corrector for the vertical plane,
- KICKER, a corrector for both planes.

```
label:   HKICKER, L=real,KICK=real,TILT=real;
label:   VKICKER, L=real,KICK=real,TILT=real;
label:   KICKER,  L=real,HKICK=real,VKICK=real,TILT=real;
```

The type KICKER should not be used when an orbit corrector kicks only in one plane.
The attributes have the following meaning:

- L: The length of the closed orbit corrector (default: 0 m).
- KICK: The kick angle for either horizontal or vertical correctors. (default: 0 rad).
- HKICK: The horizontal kick angle for a corrector in both planes (default: 0 rad).
- VKICK: The vertical kick angle for a corrector in both planes (default: 0 rad).
- TILT: The roll angle about the longitudinal axis (default: 0 rad). A positive angle represents a clockwise rotation of the kicker.

A positive kick increases p_x or p_y respectively. This means that a positive horizontal kick bends to the left, i.e. to positive x which is opposite of what is true for bends.
It should be noted that the kick values assigned to an orbit corrector like above are not overwritten by an orbit correction using the CORRECT command. Instead the kicks computed by an orbit correction and the assigned values are added when the correctors are used.

Examples:

```
HK1:   HKICKER, KICK=0.001;
VK3:   VKICKER, KICK=0.0005;
VK4:   VKICKER, KICK:=AVK4;
KHV1:  KICKER,  HKICK=0.001,VKICK=0.0005;
KHV2:  KICKER,  HKICK:=AKHV2H,VKICK:=AKHV2V;
```

The assignment in the form of a deferred expression has the advantage that the values can be assigned and/or modified at any time (and matched !).
The straight reference system for an orbit corrector is a Cartesian coordinate system.

hansg, frs, August 28, 2003

EUROPEAN ORGANIZATION FOR NUCLEAR RESEARCH 

Beam Position Monitors

A beam monitor acts on the beam like a drift space. In addition it serves to record the beam position for closed orbit corrections. Four different types of beam position monitors are recognised:

- HMONITOR. Monitor for the horizontal beam position,
- VMONITOR. Monitor for the vertical beam position,
- MONITOR. Monitor for both horizontal and vertical beam position.
- INSTRUMENT. A place holder for any type of beam instrumentation. Optically it behaves like a drift space; it returns *no beam observation*. It represent a class of elements which is completely independent from drifts and monitors.

```
label: HMONITOR,    L=real;
label: VMONITOR,    L=real;
label: MONITOR,     L=real;
label: INSTRUMENT,  L=real;
```

A beam position monitor has one real attribute:

- L: The length of the monitor (default: 0 m). If the length is different from zero, the beam position is recorded in the centre of the monitor.

Examples:

```
MH: HMONITOR,L=1;
MV: VMONITOR;
```

The straight reference system for a monitor is a cartesian coordinate system.

hansg, June 17, 2002

Bending Magnets

Two different type keywords are recognised for bending magnets, they are distinguished only by the reference system used:

- RBEND is a rectangular bending magnet. It has parallel pole faces and is based on a curved rbend reference system; **its length is the straight length as in the Figure but internally the arc length is being used. - to define an RBEND with the arc length as length (straight line shorter than input - for compatibility with MAD8 version up to version 8.23.06 including), the option RBARC=FALSE has to be set.**

- SBEND is a sector bending magnet. Its pole faces meet at the centre of curvature of the curved sbend reference system.

They are defined by the commands:

```
SBEND,    L=real,ANGLE=real,TILT=real,K0=real,K0S=real,K1=real,E1=real,E2=real,
          FINT=real,FINTX=real,HGAP=real,K2=real,H1=real,H2=real;

RBEND,    L=real,ANGLE=real,TILT=real,K0=real,K0S=real,K1=real,E1=real,E2=real,
          FINT=real,FINTX=real,HGAP=real,K2=real,H1=real,H2=real;
```

For both types, the following first-order attributes are permitted:

- L: The length of the magnet (default: 0 m). For a rectangular magnet the length is measured along a straight line as in the Figure (internally the arc length is used), while for a sector magnet it is the arc length of the reference orbit. **To define an RBEND with the arc length (shorter straight length), the option RBARC=FALSE has to be set.**

- ANGLE: The bend angle (default: 0 rad). A positive bend angle represents a bend to the right, i.e. towards negative *x* values.

- TILT: The roll angle about the longitudinal axis (default: 0 rad, i.e. a horizontal bend). A positive angle represents a clockwise rotation. A TILT=pi/2 turns a horizontal into a vertical bend, i.e. a positive bend ANGLE denotes a deflection down. **Please note that contrary to MAD8 one has to specify the desired TILT angle, otherwise it is taken as 0 rad. This was needed to avoid the confusion in MAD8 about the actual meaning of the TILT attribute for various elements.**

- **Please take note that K_0 and K_{0s} are left in the data base but are no longer used for the MAP of the bends (but see below for what K_0 is being used), instead ANGLE and TILT are used exclusively. We believe that this will allow for a clearer and unambiguous definition, in particular in view of the upcoming integration of MAD-X with PTC which will allow a more general definition of bends. However, it is required to specify k0 to assign RELATIVE field errors to a bending magnet since k0 is used for the normalization and NOT the ANGLE. (see EFCOMP).**

- K1: The quadrupole coefficient

  K_1 = *(1 / B rho) (del* B_y */ del x).*

  The default is 0 m^{-2}. A positive quadrupole strength implies horizontal focussing of positively charged particles.
- E1: The rotation angle for the entrance pole face (default: 0 rad).
- E2: The rotation angle for the exit pole face (default: 0 rad).
- FINT: The field integral whose default value is 0.
- FINTX: Allows (FINTX > 0)to set FINT at the element exit different from its entry value. In particular useful to switch it off (FINTX=0).
- HGAP: The half gap of the magnet (default: 0 m).

The pole face rotation angles are referred to the magnet model for rectangular bend and sector bend respectively. The quantities FINT and HGAP specify the finite extent of the fringe fields as defined in [SLAC-75] There they are defined as follows:

$$\mathtt{FINT} = \int_{-\infty}^{\infty} \frac{B_y(s)(B_0 - B_y(s))}{g \cdot B_0^2} ds, \qquad g = 2 \cdot \mathtt{HGAP}.$$

The default values of zero corresponds to the hard-edge approximation, i.e. a rectangular field distribution. For other approximations, enter the correct value of the half gap, and one of the following values for FINT:

```
Linear Field drop-off                    1/6
Clamped "Rogowski" fringing field        0.4
Unclamped "Rogowski" fringing field      0.7
"Square-edged" non-saturating magnet     0.45
```

Entering the keyword FINT alone sets the integral to 0.5. This is a reasonable average of the above values. The following second-order attributes are permitted:

- K2: The sextupole coefficient K_2 = *(1 / B rho) (del*^{2} B_y */ del* x^2 *).*
- H1: The curvature of the entrance pole face (default: 0 m^{-1}).
- H2: The curvature of the exit pole face (default: 0 m^{-1}). A positive pole face curvature induces a negative sextupole component; i.e. for positive H1 and H2 the centres of curvature of the pole faces are placed inside the magnet.

Examples:

```
BR: RBEND,L=5.5,ANGLE=+0.001;              // Deflection to the right
BD: SBEND,L=5.5,K0S=+0.001/5.5;            // Deflection up
BL: SBEND,L=5.5,K0=-0.001/5.5;             // Deflection to the left
BU: SBEND,L=5.5,K0S=-0.001;                // Deflection down
```

hansg, frs, August 28, 2003

Marker.

```
label: MARKER;
```

The simplest element which can occur in a beam line is the MARKER. It has no effect on the beam, but it allows one to identify a position in the beam line, for example to apply a matching constraint.

Example:

```
m27: marker;
```

hansg, June 6, 2002

EUROPEAN ORGANIZATION FOR NUCLEAR RESEARCH 

Sign Conventions for Magnetic Fields

The MAD program uses the following Taylor expansion for the field on the mid-plane y=0, described in SLAC-75:

$$B_y(x,0) = \sum_{n=0}^{\infty} \frac{B_n x^n}{n!}$$

Note the factorial in the denominator. The field coefficients have the following meaning:

- B_0: Dipole field, with a positive value in the positive y direction; a positive field bends a positively charged particle to the right.
- B_1: Quadrupole coefficient

  B_1 = (del B_y / del x);

  a positive value corresponds to horizontal focussing of a positively charged particle.
- B_2: Sextupole coefficient

  B_2 = (del^2 B_y / del x^2).

- B_2: Octupole coefficient

  B_3 = (del^3 B_y / del x^3).

Using this expansion and the curvature h of the reference orbit, the longitudinal component of the vector potential to order 4 is:

$$\begin{aligned} A_s = &+ B_0\left(x - \tfrac{hx^2}{2(1+hx)}\right) + B_1\left(\tfrac{1}{2}(x^2-y^2) - \tfrac{h}{6}x^3 + \tfrac{h^2}{24}(4x^4-y^4) + \ldots\right) \\ &+ B_2\left(\tfrac{1}{6}(x^3-3xy^2) - \tfrac{h}{24}(x^4-y^4) + \ldots\right) + B_3\left(\tfrac{1}{24}(x^4-6x^2y^2+y^4) + \ldots\right) + \ldots \end{aligned}$$

Taking curl A in curvilinear coordinates, the field components can be computed as

$$\begin{aligned} B_x(x,y) = &+ B_1\left(y + \tfrac{h^2}{6}y^3 + \ldots\right) \\ &+ B_2\left(xy - \tfrac{h^3}{6}y^3 + \ldots\right) + B_3\left(\tfrac{1}{6}(3x^2y-y^3) + \ldots\right) + \ldots \\ B_y(x,y) = &+ B_0 + B_1\left(x - \tfrac{h}{2}y^2 + \tfrac{h^2}{2}xy^2 + \ldots\right) \\ &+ B_2\left(\tfrac{1}{2}(x^2-y^2) - \tfrac{h}{2}xy^2 + \ldots\right) + B_3\left(\tfrac{1}{6}(x^3-3xy^2) + \ldots\right) + \ldots \end{aligned}$$

It can be easily verified that both curl B and div B are zero to the order of the B_3 term. Introducing the magnetic rigidity Brho, the multipole coefficients are computed as

$K_n = e\, B_n \,/\, p_s = B_n \,/\, B$ rho.

hansg, June 17, 2002

Variables

For each variable the physical units are listed in square brackets.

Canonical Variables Describing Orbits

MAD uses the following canonical variables to describe the motion of particles:

- X: Horizontal position x of the (closed) orbit, referred to the ideal orbit [m].
- PX: Horizontal canonical momentum p_x of the (closed) orbit referred to the ideal orbit, divided by the reference momentum: PX = p_x / p_0, [1].
- Y: Vertical position y of the (closed) orbit, referred to the ideal orbit [m].
- PY: Vertical canonical momentum p_y of the (closed) orbit referred to the ideal orbit, divided by the reference momentum: PY = p_x / p_0, [1].
- T: Velocity of light times the negative time difference with respect to the reference particle: T = -$c\,t$, [m]. A positive T means that the particle arrives ahead of the reference particle.
- PT: Energy error, divided by the reference momentum times the velocity of light: PT = delta(E) / $p_s\,c$, [1]. This value is only non-zero when synchrotron motion is present. It describes the deviation of the particle from the orbit of a particle with the momentum error DELTAP.
- DELTAP: Difference of the reference momentum and the design momentum, divided by the reference momentum: DELTAP = delta(p) / p_0, [1]. This quantity is used to normalize all element strengths.

The independent variable is:

- S: Arc length s along the reference orbit, [m].

In the limit of fully relativistic particles (gamma >> 1, $v = c$, $p\,c = E$), the variables T, PT used here agree with the longitudinal variables used in [TRANSPORT]. This means that T becomes the negative path length difference, while PT becomes the fractional momentum error. The reference momentum p_s must be constant in order to keep the system canonical.

Normalised Variables and other Derived Quantities

- XN: The normalised horizontal displacement

  XN = x_n = Re($E_1{}^T\,S\,Z$), [sqrt(m)].

- PXN: The normalised horizontal transverse momentum

  PXN = x_n = Im($E_1{}^T\,S\,Z$), [sqrt(m)].

- WX: The horizontal Courant-Snyder invariant

  WX = sqrt($x_n^2 + p_{xn}^2$), [m].

- PHIX: The horizontal phase

  PHIX = - atan(p_{xn} / x_n) / 2 pi [1].

- YN: The normalised vertical displacement

  YN = x_n = Re($E_2^T S Z$), [sqrt(m)].

- PYN: The normalised vertical transverse momentum

  PYN = x_n = Im($E_2^T S Z$), [sqrt(m)].

- WY: The vertical Courant-Snyder invariant

  WY = sqrt($y_n^2 + p_{yn}^2$), [m].

- PHIY: The vertical phase

  PHIY = - atan(p_{yn} / y_n) / 2 pi [1].

- TN: The normalised longitudinal displacement

  TN = x_n = Re($E_3^T S Z$), [sqrt(m)].

- PTN: The normalised longitudinal transverse momentum

  PTN = x_n = Im($E_3^T S Z$), [sqrt(m)].

- WT: The longitudinal invariant

  WT = sqrt($t_n^2 + p_{tn}^2$), [m].

- PHIT: The longitudinal phase

  PHIT = + atan(p_{tn} / t_n) / 2 pi [1].

in the above formulas Z is the phase space vector

$Z = (x, p_x, y, p_y, t, p_t)^T$.

the matrix S is the ‘‘symplectic unit matrix’’

$$S = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & -1 & 0 \end{pmatrix},$$

and the vectors E_i are the three complex eigenvectors.

Linear Lattice Functions (Optical Functions)

Several MAD commands refer to linear lattice functions. Since MAD uses the canonical momenta (p_x, p_y) instead of the slopes (x', y'), their definitions differ slightly from those in [Courant and Snyder]. Notice that in MAD-X PT substitutes DELTAP as longitudinal variable. Dispersive and chromatic functions are hence derivatives with respects to PT. Being PT=BETA*DELTAP, where BETA is the relativistic Lorentz factor, those functions must be multiplied by BETA a number of time equal to the order of the derivative. The linear lattice functions are known to MAD under the following names:

- BETX: Amplitude function beta$_x$, [m].
- ALFX: Correlation function alpha$_x$, [1]:

  ALFX = alpha$_x$ = - 1/2 * (del beta$_x$ / del s).

- MUX: Phase function mu$_x$, [2pi]:

  MUX = mu$_x$ = integral (ds / beta$_x$).

- DX: Dispersion D_x of x, [m]:

  DX = D_x = (del x / del PT).

- DPX: Dispersion D_{px} of p_x, [1]:

  DPX = D_{px} = (del p_x / del PT) / p_s.

- BETY: Amplitude function beta$_y$, [m].
- ALFY: Correlation function alpha$_y$, [1].

  ALFY = alpha$_y$ = - 1/2 * (del beta$_y$ / del s).

- MUY: Phase function mu$_y$, [2pi].

  MUY = mu$_y$ = integral (ds / beta$_y$).

- DY: Dispersion D_y of y, [m]:

  DY = D_y = (del y / del PT).

- DPY: Dispersion D_{px} of p_x, [1]:

  DPY = D_{py} = (del p_y / del PT) / p_s.

- R11, R12, R21, R22: Coupling Matrix
- ENERGY: The total energy per particle in GeV. If given, it must be greater then the particle mass.

Chromatic Functions

Several MAD commands refer to the chromatic functions. (p_x, p_y) instead of the slopes (x', y'), their definitions differ slightly from those in [Montague]. Notice that in MAD-X PT substitutes DELTAP as longitudinal variable. Dispersive and chromatic functions are hence derivatives with respects to PT. Being PT=BETA*DELTAP, where BETA is the relativistic Lorentz factor, those functions must be multiplied by BETA a number of time equal to the order of the derivative. The chromatic functions are known to MAD under the following names:

- WX: Chromatic amplitude function W_x, [1]:

  WX = W_x = sqrt($a_x{}^2$ + $b_x{}^2$),

  a_x = (del beta$_x$ / del PT) / beta$_x$,

  b_x = (del alpha$_x$ / del PT) - (alpha$_x$ / beta$_x$) * (del beta$_x$ / del PT).

- PHIX: Chromatic phase function Phi$_x$, [2pi]:

  PHIX = Phi$_x$ = atan(a_x / b_x).

- DMUX: Chromatic derivative of phase function mu$_x$, [2pi]:

  DMUX = (del mu$_x$ / del PT).

- DDX: Chromatic derivative of dispersion D_x, [m]:

  DDX = (del^2 x / del PT^2).

- DDPX: Chromatic derivative of dispersion D_{px}, [1]:

  DDPX = (del^2 p_x / del PT^2) / p_s.

- WY: Chromatic amplitude function W_y, [1]:

  WY = W_y = sqrt($a_y{}^2$ + $b_y{}^2$),

  a_y = (del beta$_y$ / del PT) / beta$_y$,

  b_y = (del alpha$_y$ / del PT) - (alpha$_y$ / beta$_y$) * (del beta$_y$ / del PT).

- PHIY: Chromatic phase function Phi$_y$, [2pi]:

  PHIY = Phi$_y$ = atan(a_y / b_y).

- DMUY: Chromatic derivative of phase function mu$_y$, [2pi]:

  DMUY = (del mu$_y$ / del PT).

- DDY: Chromatic derivative of dispersion D_y, [m]:

  DDY = (del^2 y / del PT^2).

- DDPY: Chromatic derivative of dispersion D_{py}, [1]:

  DDPY = (del^2 p_y / del PT^2) / p_s.

Variables in the SUMM Table

After a successful TWISS command a summary table is created which contains the following variables:

- LENGTH: The length of the machine, [m].
- ORBIT5: The T (= $c\,t$, [m]) component of the closed orbit.
- ALFA: The momentum compaction alpha$_p$, [1].
- GAMMATR: The transition energy gamma$_{transition}$, [1].
- Q1: The horizontal tune Q_1 [1].
- DQ1: The horizontal chromaticity dq$_1$, [1]:

  DQ1 = dq$_1$ = (del Q_1 / del PT).

- BETXMAX: The largest horizontal beta$_x$, [m].
- DXMAX: The largest horizontal dispersion [m].
- DXRMS: The r.m.s. of the horizontal dispersion [m].
- XCOMAX: The maximum of the horizontal closed orbit deviation [m].
- XRMS: The r.m.s. of the horizontal closed orbit deviation [m].
- Q2: The vertical tune Q_2 [1].
- DQ2: The vertical chromaticity dq$_2$, [1]:

  DQ2 = dq$_2$ = (del Q_2 / del PT).

- BETYMAX: The largest vertical beta$_y$, [m].
- DYMAX: The largest vertical dispersion [m].
- DYRMS: The r.m.s. of the vertical dispersion [m].
- YCOMAX: The maximum of the vertical closed orbit deviation [m].
- YCORMS: The r.m.s. of the vertical closed orbit deviation [m].
- DELTAP: Energy difference, divided by the reference momentum times the velocity of light, [1]:

  DELTAP = delta(E) / p_s c.

Notice that in MAD-X PT substitutes DELTAP as longitudinal variable. Dispersive and chromatic functions are hence derivatives with respects to PT. Being PT=BETA*DELTAP, where BETA is the relativistic Lorentz factor, those functions must be multiplied by BETA a number of time equal to the order of the derivative.

Variables in the TRACK Table

The command RUN writes tables with the following variables:

- X: Horizontal position x of the orbit, referred to the ideal orbit [m].
- PX: Horizontal canonical momentum p_x of the orbit referred to the ideal orbit, divided by the reference momentum.
- Y: Vertical position y of the orbit, referred to the ideal orbit [m].
- PY: Vertical canonical momentum p_x of the orbit referred to the ideal orbit, divided by the reference momentum.
- T: Velocity of light times the negative time difference with respect to the reference particle, [m]. A positive T means that the particle arrives ahead of the reference particle.
- PT: Energy difference, divided by the reference momentum times the velocity of light, [1].

When tracking Lyapunov companions (not yet implemented), the TRACK table defines the following dependent expressions:

- DISTANCE: the relative Lyapunov distance between the two particles.
- LYAPUNOV: the estimated Lyapunov Exponent.
- LOGDIST: the natural logarithm of the relative distance.
- LOGTURNS: the natural logarithm of the turn number.

hansg, January 24, 1997. Revised in February 2007.

EUROPEAN ORGANIZATION FOR NUCLEAR RESEARCH 

Physical Units

Throughout the computations MAD uses international (SI, Système International) units. These units are summarised in the Units table.

Table 1: Physical Units

Length	m (metres)
Angle	rad (radians)
Quadrupole coefficient	m**(-2)
Multipole coefficient, 2n poles	m**(-n)
Electric voltage	MV (Megavolts)
Electric field strength	MV/m
Frequency	MHz (Megahertz)
Phase angles	2 pi
Particle energy	GeV
Particle mass	GeV/c**2
Particle momentum	GeV/c
Beam current	A (Amperes)
Particle charge	e (elementary charges)
Impedances	MOhm (Megohms)
Emittances	pi m mrad
RF power	MW (Megawatts)
Higher mode loss factor	V/pc **Table 1:** Physical Units

hansg, June 17, 2002

Command Format

- Statements
- Comments
- Identifiers or Label
- Command Attribute
  - Name or String Attribute
  - Logical Attribute
  - Integer Attribute
  - Real Expression, built from operator and operand.
  - A Deferred Expression is evaluated every time it is used
  - Constraint
  - Variable Name
- Wild Card Pattern

hansg, May 8, 2001

EUROPEAN ORGANIZATION FOR NUCLEAR RESEARCH 

Statements

and

Comments

Input for MAD-X follows in broad lines the new MAD-9 format, i.e. free format with commas "," as separators; however, outside {...} enclosures blanks may be used as separators. Blank input lines do not affect program execution. The input is not case sensitive except for strings enclosed in " ".

The input file consists of a sequence of commands, also known as statements. A statement may occupy any number of input lines. It must be terminated by a semicolon, except if it contains a block of statements itself, like in

```
if (a < 3) {a=b^2; b=2*b+4;}
```

Several statements may be placed on the same line. When a "!" or "//" is found on an input line, the remaining characters of the line are skipped. A line "/*" starts a comment region, it ends with a "*/" line. The general format for a command is (items enclosed in /rep/ ... /rep/ can be repeated any number of times, including zero):

```
label: keyword /rep/,attribute/rep/ ;
```

It has three parts:

- A label is required for a definition statement. It gives a name to the stored command.
- A keyword identifies the action desired.
- The attributes are normally entered in the form "attribute-name=attribute-value" and serve to define data for the command, where:
  - attribute-name selects the attribute,
  - attribute-value gives it a value.

If a value has to be assigned to an attribute, the attribute name is mandatory. For logical attributes it is sufficient to enter the name only. The attribute is then given a default value taken from the command dictionary.

Example: TILT attribute for various magnets.

The command attributes can have one of the following types:

- String attribute,
- Logical attribute,
- Integer attribute,
- Real attribute,
- Expression,

- Range selection,

Any integer or real attribute can be replaced by a real expression; expressions are normally deferred (see deferred expression), except in the definition of constants where they are evaluated immediately. When a command has a label, MAD-X keeps it in memory. This allows repeated execution of the same command by just entering EXEC label. This construct may be nested. For an exhaustive list of valid declarations of constants or variables see declarations.

hansg, May 8, 2001

EUROPEAN ORGANIZATION FOR NUCLEAR RESEARCH 

Identifiers or Labels

A keyword begins with a letter and consists of letters and digits. The MAD-X keywords are protected; using one of them as a label results in a fatal error.

hansg, May 8, 2001

String Attributes

A string attribute makes alphanumeric information available, e.g. a title or a file name. It can contain any characters, enclosed in single (’) or double (") quotes, except for quotes of the type used as its delimiter.

Examples:

```
TITLE,'This is a title for the program run "test"';
system,"ln -fns some-lengthy-directory-name local-link";
```

hansg, June 17, 2002

Real Attributes

Most attributes are of type REAL and are treated internally as double precision values. They may be set by integer values, real values, or expressions. Example:

```
ddd:drift,l=1.2345;
dddd:drift,l=ddd->l-0.3;
```

hansg, May 8, 2001

Selection Statements

The elements, or a range of elements, in a sequence can be selected for various purposes. Such selections remain valid until cleared (in difference to MAD-8); it is therefore recommended to always start with a

```
select,flag=...,clear;
```

before setting a new selection.

```
SELECT,FLAG=name,RANGE=range,CLASS=class,PATTERN=pattern[,FULL][,CLEAR];
```

where the name for FLAG can be one of ERROR, MAKETHIN, SEQEDIT or the name of a twiss table which is established for all sequence positions in general.

Selected elements have to fulfill the RANGE, CLASS, and PATTERN criteria.

Any number of SELECT commands can be issued for the same flag and are accumulated (logically ORed). In this context note the following:

```
SELECT,FLAG=name,FULL;
```

selects all positions in the sequence for this flag. This is the default for all tables and makethin, whereas for ERROR and SEQEDIT the default is "nothing selected".

SAVE: A SELECT,FLAG=SAVE statement causes the selected sequences, elements, and variables to be written into the save file. A class (only used for element selection), and a pattern can be specified. Example:

```
select,flag=save,class=variable,pattern="abc.*";
save,file=mysave;
```

will save all variables (and sequences) containing "abc" in their name, but not elements with names containing "abc" since the class "variable" does not exist (astucieux, non ?).

SECTORMAP: A SELECT,FLAG=SECTORMAP statement causes sectormaps to be written into the file "sectormap" like in MAD-8. For the file to be written, a flag SECTORMAP must be issued on the TWISS command in addition.

TWISS: A SELECT,FLAG=TWISS statement causes the selected rows and columns to be written into the Twiss TFS file (former OPTICS command in MAD-8). The column selection is done on the same select. See as well example 2. Example 1:

```
TITLE,'Test input for MAD-X';

option,rbarc=false; // use arc length of rbends
beam; ! sets the default beam for the following sequence
option,-echo;
call file=fv9.opt;  ! contains optics parameters
```

```
call file="fv9.seq"; ! contains a small sequence "fivecell"
OPTION,ECHO;
SELECT,FLAG=SECTORMAP,clear;
SELECT,FLAG=SECTORMAP,PATTERN="^m.*";
SELECT,FLAG=TWISS,clear;
SELECT,FLAG=TWISS,PATTERN="^m.*",column=name,s,betx,bety;
USE,PERIOD=FIVECELL;
twiss,file=optics,sectormap;
stop;
```

This produces a file sectormap, and a twiss output file (name = optics):

```
@ TYPE             %05s "TWISS"
@ PARTICLE         %08s "POSITRON"
@ MASS             %le          0.000510998902
@ CHARGE           %le                       1
@ E0               %le                       1
@ PC               %le           0.99999986944
@ GAMMA            %le           1956.95136738
@ KBUNCH           %le                       1
@ NPART            %le                       0
@ EX               %le                       1
@ EY               %le                       1
@ ET               %le                       0
@ LENGTH           %le                   534.6
@ ALFA             %le        0.00044339992938
@ ORBIT5           %le                      -0
@ GAMMATR          %le           47.4900022541
@ Q1               %le           1.25413071556
@ Q2               %le           1.25485338377
@ DQ1              %le           1.05329608302
@ DQ2              %le           1.04837000224
@ DXMAX            %le           2.17763211131
@ DYMAX            %le                       0
@ XCOMAX           %le                       0
@ YCOMAX           %le                       0
@ BETXMAX          %le            177.70993499
@ BETYMAX          %le           177.671582415
@ XCORMS           %le                       0
@ YCORMS           %le                       0
@ DXRMS            %le           1.66004270906
@ DYRMS            %le                       0
@ DELTAP           %le                       0
@ TITLE            %20s "Test input for MAD-X"
@ ORIGIN           %16s "MAD-X 0.20 Linux"
@ DATE             %08s "07/06/02"
@ TIME             %08s "14.25.51"
* NAME               S                 BETX              BETY
$ %s                 %le               %le               %le
 "MSCBH"             4.365             171.6688159       33.31817319
 "MB"                19.72             108.1309095       58.58680717
 "MB"                35.38             61.96499987       102.9962313
 "MB"                51.04             34.61640793       166.2227523
 "MSCBV.1"           57.825            33.34442808       171.6309057
 "MB"                73.18             58.61984637       108.0956006
 "MB"                88.84             103.0313887       61.93159422
 "MB"                104.5             166.2602486       34.58939635
 "MSCBH"             111.285           171.6688159       33.31817319
 "MB"                126.64            108.1309095       58.58680717
 "MB"                142.3             61.96499987       102.9962313
 "MB"                157.96            34.61640793       166.2227523
 "MSCBV"             164.745           33.34442808       171.6309057
```

```
"MB"                180.1               58.61984637         108.0956006
"MB"                195.76              103.0313887         61.93159422
"MB"                211.42              166.2602486         34.58939635
"MSCBH"             218.205             171.6688159         33.31817319
"MB"                233.56              108.1309095         58.58680717
"MB"                249.22              61.96499987         102.9962313
"MB"                264.88              34.61640793         166.2227523
"MSCBV"             271.665             33.34442808         171.6309057
"MB"                287.02              58.61984637         108.0956006
"MB"                302.68              103.0313887         61.93159422
"MB"                318.34              166.2602486         34.58939635
"MSCBH"             325.125             171.6688159         33.31817319
"MB"                340.48              108.1309095         58.58680717
"MB"                356.14              61.96499987         102.9962313
"MB"                371.8               34.61640793         166.2227523
"MSCBV"             378.585             33.34442808         171.6309057
"MB"                393.94              58.61984637         108.0956006
"MB"                409.6               103.0313887         61.93159422
"MB"                425.26              166.2602486         34.58939635
"MSCBH"             432.045             171.6688159         33.31817319
"MB"                447.4               108.1309095         58.58680717
"MB"                463.06              61.96499987         102.9962313
"MB"                478.72              34.61640793         166.2227523
"MSCBV"             485.505             33.34442808         171.6309057
"MB"                500.86              58.61984637         108.0956006
"MB"                516.52              103.0313887         61.93159422
"MB"                532.18              166.2602486         34.58939635
```

Example 2:

Addition of variables to (any internal) table:

```
select,flag=table,column=name,s,betx,...,var1,var2,...; or
select,flag=table,full,column=var1,var2,...; ! default col.s + new
```

will write the current value of var1 etc. into the table each time a new line is added; values from the same (current) line can be accessed by these variables, e.g.

```
var1:= sqrt(beam->ex*table(twiss,betx));
```

in the case of table above being "twiss". The plot command accepts the new variables.

Remark: this replaces the "string" variables of MAD-8.

This example demonstrates as well the usage of a user defined table.

```
beam,ex=1.e-6,ey=1.e-3;
// element definitions
mb:rbend, l=14.2, angle:=0,k0:=bang/14.2;
mq:quadrupole, l:=3.1,apertype=ellipse,aperture={1,2};
qft:mq, l:=0.31, k1:=kqf,tilt=-pi/4;
qf.1:mq, l:=3.1, k1:=kqf;
qf.2:mq, l:=3.1, k1:=kqf;
qf.3:mq, l:=3.1, k1:=kqf;
qf.4:mq, l:=3.1, k1:=kqf;
qf.5:mq, l:=3.1, k1:=kqf;
qd.1:mq, l:=3.1, k1:=kqd;
qd.2:mq, l:=3.1, k1:=kqd;
qd.3:mq, l:=3.1, k1:=kqd;
qd.4:mq, l:=3.1, k1:=kqd;
```

```
qd.5:mq, l:=3.1, k1:=kqd;
bph:hmonitor, l:=l.bpm;
bpv:vmonitor, l:=l.bpm;
cbh:hkicker;
cbv:vkicker;
cbh.1:cbh, kick:=acbh1;
cbh.2:cbh, kick:=acbh2;
cbh.3:cbh, kick:=acbh3;
cbh.4:cbh, kick:=acbh4;
cbh.5:cbh, kick:=acbh5;
cbv.1:cbv, kick:=acbv1;
cbv.2:cbv, kick:=acbv2;
cbv.3:cbv, kick:=acbv3;
cbv.4:cbv, kick:=acbv4;
cbv.5:cbv, kick:=acbv5;
!mscbh:sextupole, l:=1.1, k2:=ksf;
mscbh:multipole, knl:={0,0,0,ksf},tilt=-pi/8;
mscbv:sextupole, l:=1.1, k2:=ksd;
!mscbv:octupole, l:=1.1, k3:=ksd,tilt=-pi/8;

// sequence declaration

fivecell:sequence, refer=centre, l=534.6;
   qf.1:qf.1, at=1.550000e+00;
   qft:qft, at=3.815000e+00;
!   mscbh:mscbh, at=3.815000e+00;
   cbh.1:cbh.1, at=4.365000e+00;
   mb:mb, at=1.262000e+01;
   mb:mb, at=2.828000e+01;
   mb:mb, at=4.394000e+01;
   bpv:bpv, at=5.246000e+01;
   qd.1:qd.1, at=5.501000e+01;
   mscbv:mscbv, at=5.727500e+01;
   cbv.1:cbv.1, at=5.782500e+01;
   mb:mb, at=6.608000e+01;
   mb:mb, at=8.174000e+01;
   mb:mb, at=9.740000e+01;
   bph:bph, at=1.059200e+02;
   qf.2:qf.2, at=1.084700e+02;
   mscbh:mscbh, at=1.107350e+02;
   cbh.2:cbh.2, at=1.112850e+02;
   mb:mb, at=1.195400e+02;
   mb:mb, at=1.352000e+02;
   mb:mb, at=1.508600e+02;
   bpv:bpv, at=1.593800e+02;
   qd.2:qd.2, at=1.619300e+02;
   mscbv:mscbv, at=1.641950e+02;
   cbv.2:cbv.2, at=1.647450e+02;
   mb:mb, at=1.730000e+02;
   mb:mb, at=1.886600e+02;
   mb:mb, at=2.043200e+02;
   bph:bph, at=2.128400e+02;
   qf.3:qf.3, at=2.153900e+02;
   mscbh:mscbh, at=2.176550e+02;
   cbh.3:cbh.3, at=2.182050e+02;
   mb:mb, at=2.264600e+02;
   mb:mb, at=2.421200e+02;
   mb:mb, at=2.577800e+02;
   bpv:bpv, at=2.663000e+02;
   qd.3:qd.3, at=2.688500e+02;
   mscbv:mscbv, at=2.711150e+02;
   cbv.3:cbv.3, at=2.716650e+02;
```

```
   mb:mb, at=2.799200e+02;
   mb:mb, at=2.955800e+02;
   mb:mb, at=3.112400e+02;
   bph:bph, at=3.197600e+02;
   qf.4:qf.4, at=3.223100e+02;
   mscbh:mscbh, at=3.245750e+02;
   cbh.4:cbh.4, at=3.251250e+02;
   mb:mb, at=3.333800e+02;
   mb:mb, at=3.490400e+02;
   mb:mb, at=3.647000e+02;
   bpv:bpv, at=3.732200e+02;
   qd.4:qd.4, at=3.757700e+02;
   mscbv:mscbv, at=3.780350e+02;
   cbv.4:cbv.4, at=3.785850e+02;
   mb:mb, at=3.868400e+02;
   mb:mb, at=4.025000e+02;
   mb:mb, at=4.181600e+02;
   bph:bph, at=4.266800e+02;
   qf.5:qf.5, at=4.292300e+02;
   mscbh:mscbh, at=4.314950e+02;
   cbh.5:cbh.5, at=4.320450e+02;
   mb:mb, at=4.403000e+02;
   mb:mb, at=4.559600e+02;
   mb:mb, at=4.716200e+02;
   bpv:bpv, at=4.801400e+02;
   qd.5:qd.5, at=4.826900e+02;
   mscbv:mscbv, at=4.849550e+02;
   cbv.5:cbv.5, at=4.855050e+02;
   mb:mb, at=4.937600e+02;
   mb:mb, at=5.094200e+02;
   mb:mb, at=5.250800e+02;
   bph:bph, at=5.336000e+02;
end:marker, at=5.346000e+02;
endsequence;

// forces and other constants

l.bpm:=.3;
bang:=.509998807401e-2;
kqf:=.872651312e-2;
kqd:=-.872777242e-2;
ksf:=.0198492943;
ksd:=-.039621283;
acbv1:=1.e-4;
acbh1:=1.e-4;
!save,sequence=fivecell,file,mad8;

s := table(twiss,bpv[5],betx);
myvar := sqrt(beam->ex*table(twiss,betx));
use, period=fivecell;
select,flag=twiss,column=name,s,myvar,apertype;
twiss,file;
n = 0;
create,table=mytab,column=dp,mq1,mq2;
mq1:=table(summ,q1);
mq2:=table(summ,q2);
while ( n < 11)
{
  n = n + 1;
  dp = 1.e-4*(n-6);
  twiss,deltap=dp;
  fill,table=mytab;
```

```
}
write,table=mytab;
plot,haxis=s,vaxis=aper_1,aper_2,colour=100,range=#s/cbv.1,notitle;
stop;
```

prints the following user table on output:

```
@ NAME             %05s "MYTAB"
@ TYPE             %04s "USER"
@ TITLE            %08s "no-title"
@ ORIGIN           %16s "MAD-X 1.09 Linux"
@ DATE             %08s "10/12/02"
@ TIME             %08s "10.45.25"
* DP                 MQ1                MQ2
$ %le                %le                %le
 -0.0005            1.242535951        1.270211135
 -0.0004            1.242495534        1.270197018
 -0.0003            1.242452432        1.270185673
 -0.0002            1.242406653        1.270177093
 -0.0001            1.242358206        1.270171269
 0                  1.242307102        1.27016819
 0.0001             1.242253353        1.270167843
 0.0002             1.242196974        1.270170214
 0.0003             1.24213798         1.270175288
 0.0004             1.242076387        1.270183048
 0.0005             1.242012214        1.270193477
```

and produces a twiss file with the additional column myvar, as well as a plot file with the aperture values plotted.

Example of joing 2 tables with different length into a third table making use of the length of either table as given by table("your_table_name", tablelength) and adding names by the "_name" attribute.

```
title,   "summing of offset and alignment tables";
 set,    format="13.6f";

 readtable, table=align,  file="align.ip2.b1.tfs";   // mesured alignment
 readtable, table=offset, file="offset.ip2.b1.tfs";  // nominal offsets

 n_elem  =  table(offset, tablelength);

 create,  table=align_offset, column=_name,s_ip,x_off,dx_off,ddx_off,y_off,dy_off,ddy_off;

 calcul(elem_name) : macro = {
     x_off = table(align,  elem_name, x_ali) + x_off;
     y_off = table(align,  elem_name, y_ali) + y_off;
 }


 one_elem(j_elem) : macro = {
     setvars, table=offset, row=j_elem;
     exec,  calcul(tabstring(offset, name, j_elem));
     fill,  table=align_offset;
 }


 i_elem = 0;
 while (i_elem < n_elem) { i_elem = i_elem + 1; exec,  one_elem($i_elem); }

 write, table=align_offset, file="align_offset.tfs";


stop;
```

hansg, May 8, 2001

EUROPEAN ORGANIZATION FOR NUCLEAR RESEARCH 

Range and Class Selection Format

- RANGE: A range can be defined starting at a given element and ending at another element, both elements included. Two forms exist:

```
range=position;
range=position1/position2;
```

  In the first case, only one element is selected; in the second case, one or several elements are selected. NOTE: position1 must not be behind position2 in the sequence.

  "position" is composed of the element name and an optional occurrence count in the sequence:

```
mq.ir5.l6..1          ! no occurrence count given
mb[17]                ! occurrence count given
```

  There are two predefined MAD indices:
  - #S. The start of the beam line expanded by USE,
  - #E. The end of the beam line expanded by USE.

  If, in the USE statement, only a range is selected:

```
use,period=lhcb1,range=ir1/ir5;
```

  then "#s" and "#e" refer to the start and end of the expanded range, of course.

  Examples for ranges:

```
..,range=#s;           ! first element
..,range=#s/#e;        ! full expansion range
..,range=mb[5]/#e;     ! from mb 5 to end
..,range=mq.ir5.l6..1; ! first occurrence of element mq.ir5.l6..1
```

- CLASS: The single name of a class (no occurrence counts). A class is the name of an element (or basic type) from which other elements have been derived. Example:

```
mq:quadrupole;
q1:mq;
q2:mq;
q1..a:q1;
q2..b:q2;
```

  makes classes from mq, q1, and q2. Selection class="mq" will actually select q1, q2, q1..a, and q2..b in the above example.

  hansg, June 17, 2002

EUROPEAN ORGANIZATION FOR NUCLEAR RESEARCH 

Sectormap output

The flag "sectormap" on the Twiss command (together with an element selection via select,flag=sectormap,...) causes a file "sectormap" to be written.

For each user-selected element, it contains the user-selected coefficients of the kick vector `K` (6 values), of the first-order map `R` (6 x 6 values) and of the second-order map `T` (6 x 6 x 6 values)

The sector file is the output of a standard TFS table, which means that the set of columns of interest may be selected through a MAD-X command such as the following:

```
select,flag=my_sect_table,column=name,pos,k1,r11,r66,t111;
```

Each line of the sectormap file contains for each selected element, the set of chosen K,R,T matrix coefficients:

@ NAME	%13s	"MY_SECT_TABLE"
@ TYPE	%09s	"SECTORMAP"
@ TITLE	%08s	"no-title"
@ ORIGIN	%19s	"MAD-X 3.04.62 Linux"
@ DATE	%08s	"18/12/08"
@ TIME	%08s	"10.33.58"

* NAME	POS	K1	R11	R66	T111
$ %s	%le	%le	%le	%le	%le
"FIVECELL$START"	0	0	1	1	0
"SEQSTART"	0	0	1	1	0
"QF.1"	3.1	-1.305314637e-05	1.042224745	1	0
"DRIFT_0"	3.265	7.451656548e-21	1	1	0
"MSCBH"	4.365	-1.686090613e-15	0.9999972755	1	0.006004411526
"CBH.1"	4.365	0	1	1	0
"DRIFT_1"	5.519992305	-6.675347543e-21	1	1	0
"MB"	19.72000769	2.566889547e-18	1.000000091	1	-4.135903063e-25
"DRIFT_2"	21.17999231	-1.757758802e-20	1	1	0
"MB"	35.38000769	2.822705549e-18	1.000000091	1	-4.135903063e-25
"DRIFT_2"	36.83999231	2.480880093e-20	1	1	0
"MB"	51.04000769	3.006954115e-18	1.000000091	1	-4.135903063e-25
"DRIFT_3"	52.21	-4.886652187e-20	1	1	0
...	...	...	...	...	...
...	...	...	...	...	...
...	...	...	...	...	...

Of course, the `select` statement can be combined with additional options to filter-out the list of elements, such as in the following statement, which for instance only retains drift-type elements:

```
select,flag=my_sect_table,class=drift,column=name,pos,k1,r11,r66,t111;
```

`K` coefficients range: `K1`... `K6`

`R` coefficients range:

```
R11   ...   R61
R12   ...   R62
...   ...   ...
R61   ...   R66
```

`T` coefficients range:

```
T111   ...   T611
T121   ...   T621
...    ...   ...
T161   ...   T661
T112   ...   T612
...    ...   ...
T166   ...   T666
```

In the above notation, `Rij` stands for "effect on the `i`-th coordinate of the `j`-th coordinate in phase-space", whereas `Tijk` stands for "combined effect on the `i`-th coordinate of both the `j`-th and `k`-th coordinates in phase-space" along the lattice.

The maps are the accumulated maps between the selected elements. They contain both the alignment, and field errors present. Together with the starting value of the closed orbit (which can be obtained from the standard twiss file) this allows the user to track particles over larger sectors, rather than element per element. A typical usage therefore lies in the interface to other programs, such as TRAIN.

hansg, May 8, 2001

Variable Declarations

In the following, "=" means that the variable at the left receives the current value of the expression at right, but does not depend on it any further, whereas ":=" makes it depend on this expression, i.e. every time the expression changes, the variable is re-evaluated, except for "const" variables.

```
var = expression;
var := expression;
real var = expression;        // identical
real var := expression;       // to above
int var = expression;         // truncated if expression is real
int var := expression;
const var = expression;
const var := expression;
const real var = expression;        // identical
const real var := expression;       // to above
const int var = expression;         // truncated if expression is real
const int var := expression;
```

hansg, May 8, 2001

Identifiers or Labels

A label begins with a letter, followed by up to fifteen letters, digits, decimal points (.), or underscores (_). Characters beyond the sixteenth are dropped, but should be avoided, and the resulting sequence must be unique.

A label may refer to a keyword, an element, a beam-line, a sequence, etc. The MAD-X keywords are protected; using one of them as a label results in a fatal error.

hansg, May 8, 2001

EUROPEAN ORGANIZATION FOR NUCLEAR RESEARCH 

Command Attributes

The following types of attributes are available in MAD:

- A name or string attribute refers to an object, or a string.
- A logical attribute selects or deselects an option.
- An integer attribute is a counter, as for repetition in a beam line.
- A real expression defines a datum for a command, it may be varied in matching. An expression is built of a combination of operator and operand.
- A constraint, specifies a matching constraint.
- A variable name selects a variable to be matched.

hansg, May 8, 2001

EUROPEAN ORGANIZATION FOR NUCLEAR RESEARCH 

Name or String Attributes

A name or string attribute often selects one of a set of options:

```
use,period=lhc;    // expand the LHC sequence
```

It may also refer to a user-defined object:

```
twiss,file=optics;    // specifies the name of the OPTICS output file
```

It may also define a string:

```
title,"LHC version 6.2";
```

The case of letters is only significant if a string is enclosed in quotes, otherwise all characters are converted to lower at reading. On the other hand, strings that do not contain blanks do not need to be enclosed in quotes. Example:

```
call,file="my.file";
call,file=my.file;
call,file=MY.FILE;
call,file="MY.FILE";
call,file='MY.FILE';
```

In the first three cases, MAD-X will try to read a file my.file, in the last two it will try to read MY.FILE.

hansg, May 8, 2001

EUROPEAN ORGANIZATION FOR NUCLEAR RESEARCH 

Logical Attributes

Many commands in MAD require the setting of logical values (flags) to represent the on/off state of an option. A logical value "flag" can be set in two ways:

```
flag | flag = true
```

It can be reset by:

```
-flag | flag=false
```

Example:

```
option,-echo;  // switch off copying the input to the standard output
```

The default for a logical flag is normally false, but can be found e.g. for options by the command

```
help,option;
```

hansg, May 8, 2001

Integer Attributes

An integer attribute usually denotes a count. Example:

```
myline:line=(-3*(a,b,c));
```

In this case, a litteral integer is requested; however, in the following

```
rfc:rfcavity,harmon=12345;
```

or

```
rfc:rfcavity,harmon=num;
```

"num" may be an integer variable, a real variable, or an expression (in the two latter cases, the value is truncated).

hansg, May 8, 2001

Real Expressions

To facilitate the definition of interdependent quantities, any real value and integer value can be entered as an arithmetic expression. When a value used in an expression is redefined by the user or changed in a matching process, the expression is reevaluated. Expression definitions may be entered in any order. MAD evaluates them in the correct order before it performs any computation. At evaluation time all operands used must have values assigned.

An expression is built from a combination of operator and operand, and it may contain random generators.

Operators in Arithmetic Expressions

An expression can be formed using the following operators:

Arithmetic operators

- + Addition,
- - Subtraction,
- * Multiplication,
- / Division,
- ^ Exponentiation.

Ordinary functions

- sqrt(x) square root,
- log(x) natural logarithm,
- log10(x) logarithm base 10,
- exp(x) exponential,
- sin(x) trigonometric sine,
- cos(x) trigonometric cosine,
- tan(x) trigonometric tangent,
- asin(x) arc sine,
- acos(x) arc cosine,
- atan(x) arc tangent,
- sinh(x) hyperbolic sine,
- cosh(x) hyperbolic cosine,
- tanh(x) hyperbolic tangent,
- abs(x) absolute value;

random number generators

- 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;

in the above cases, "x" can be any expression, i.e. contain other functions, variable or constant expressions. To initialize the MAD-X random generator use the Eoption command.

table access functions

- table(x,z): accesses value of the named table column "z" of table "x"; example: table(summ,q1) returns the hor. tune of the Twiss summary table "summ".
- table(x,y,z): accesses value of the named table column "z" for element "y" (first table row with that name) of table "x"; example: table(twiss,mb.12,betx) returns the beta_x at element mb.12 from the Twiss table "twiss".

  Beware:
- Unamed Drifts are not included in the table name database, so as to speed up the search for "real" elements. Therefore, those unamed drifts cannot be found. However, named drifts can be searched for.
- Due to the importance of finding elements in the table for a proper functioning of the MAD-X runs, the programs throws a "fatal_error" if an element cannot be found in the table.

  There is a second option of this function with 3 entries
- table(x,z,N_row): accesses the value of the named table column "z" at the "N_row" number of rows of table "x" (row numbers start at 1); example: table(twiss,betx,370) returns the beta_x at row number "370" of the Twiss table "twiss". The return value is zero if "N_row" is outside of the allowed range.

  Note that "N_row" has to be a number and not a variable. However, the Macro facility in MAD-X allows one to use a variable instead.

  A typical example could look like this, in fact the square root of betx (user defined variable myvar) is added to the twiss table:

```
myvar := sqrt(table(twiss,betx));
select,flag=twiss,column=name,s,myvar,betx;
twiss,file;
```

  Or another arbitrary test case of adding k1l taken from the Twiss table:

  Define macro:

```
mycrap(xx,yy,zz): macro = {myval = table(xx,yy,zz);};
```

  Use macro in loop:

```
i = 0;
incval = 0;
while (i < 100) {
value,i;
exec,mycrap(twiss,k1l,$i);
incval = incval + myval;
value,i,myval,incval;
i = i + 1;
}
```

Features as of Version 3_03_50

- FILL,TABLE=t,ROW=n;

  fill a table row with the present variable values. If ROW is negative or missing a new row is created. If ROW is greater than the number of rows, the last row is selected without creating a new row.

- SETVARS,TABLE=t,ROW=n;

  set variables according to the column names of the given table and the values of the given row. if ROW is negative, missing or greater than the number of rows, the last row is selected.

- An example can be found at: Special Features

- The length of a table can be determined by using the attribute "tablelength" via table("your_table_name", tablelength). This is useful when creating a table from existing ones. See an example at: user table II

Operands in Arithmetic Expressions

An expression may contain the following operands:

Literal constants

Numerical values are entered like FORTRAN constants. Real values are accepted in INTEGER or REAL format. The use of a decimal exponent, marked by the letter D or E, is permitted.

Examples:

```
1, 10.35, 5E3, 314.1592E-2
```

Symbolic constants

MAD recognises some mathematical and physical constants. Their names must not be used for user-defined labels.

Additional symbolic constants may be defined to simplify their repeated use in statements and expressions.

```
CONST name=constant-expression;
```

defines a real constant with the name given. An existing symbolic constant can be redefined, but it cannot change in a matching procedure.

Example:

```
const in = 0.0254;
```

mad name	symbol	value used	unit
pi	pi	4 * atan(1)	1
twopi	2 pi	2 * pi	1
degrad	180/pi	180 / pi	deg/rad
raddeg	pi/180	180 / pi	rad/deg
e	e	exp(1)	1
emass	m_e	.510998902*10(-3)	GeV
pmass	m_p	.938271998	GeV
mumass	m_mu	.1056583568	GeV
clight	c	2.99792458*10**8	m/s
qelect	elem. charge	1.602176462e-19	As

Parameter labels

Often a set of numerical values depends on a common variable parameter. Such a parameter must be defined as a global parameter. A global parameter always has a current value; however, this value may be re-evaluated or not, depending on the parameter definition:

```
x = a;
```

x is set to the current value of a and not changed, even if a changes. This makes assignments such as

```
x = x + 1;
```

perfectly valid (this replaces the old SET instruction). The definition of the deferred expression

```
x := a;
```

assign the current value of a to x every time x is used, i.e. it is re-evaluated using the latest value of a; therefore,

```
x := x + 1;
```

results in an infinite loop (!) when x is used (error abort). Of course, the following definitions are equivalent:

```
x = 0.1;
x := 0.1;
```

When such a parameter is used in an expression, MAD uses the current value of the parameter if the expression is deferred:

Example:

```
x:=1.0;
d1: drift,l = x;
d2: drift,l := 2.0 - x;
```

When the value of X is changed, the length of the drift d1 remains unchanged, that of d2 is recalculated.

Element or command attributes

In arithmetic expressions the attributes of physical elements or commands can occur as operands. They are named respectively by

```
element-name->attribute-name
command-name->attribute-name
```

Values are assigned to attributes in element definitions or commands.

Example:

```
D1: DRIFT,L=1.0;
D2: DRIFT,L=2.0-D1->L;
```

D1->L refers to the length L of the drift space D1.

Expressions and Random Values

The definition of random machine imperfections requires evaluation of expressions containing random functions. These are evaluated like any other expression when the expression is used, i.e. only once if a "=" assignment refers to it, or every time if the assignment is ":="; this latter case is used by the error generation routines.

Example:

```
a := 3*ranf();
```

Every time a is used, it gets a random value assigned from a uniform distribution between 0 and 3.

```
error:  ealign,range,dx=sigma*gauss();
```

All elements in range are assigned independent random displacements sampled from a Gaussian distribution with standard deviation sigma.

hansg, May 8, 2001

EOPTION: Set Error Options

The random generator for MAD is taken from [Knuth]. The error option command specifies different seeds for random values:

```
EOPTION,SEED=real,ADD=logical;
```

- SEED: Selects a particular sequence of random values. A SEED value is an integer in the range [0...999999999] (default: 123456789). SEED alone continues with the current sequence See also: Random values. SEED may be an expression.
- ADD: If this logical flag is set, an EALIGN or EFCOMP, causes the errors to be added on top of existing ones. If it is not set, new errors overwrite any previous definitions. The default values is TRUE.
  Please note a recent modification: the default value for the ADD option is only applied as long as the ADD option has not been set explicitly. Once it was set with EOPTION, it is NOT reset to the default when the ADD option is omitted in subsequent calls to EOPTION.

Example:

```
EOPTION,SEED=987456321;
```

Werner Herr 18.6.2002

Program Flow Statements

- ## IF

  ```
  if (logical_expression) {statement 1; statement 2; ...; statement n; }
  ```

  where "logical_expression" is one of

  ```
  expr1 oper expr2
  expr11 oper1 expr12 && expr21 oper2 expr22
  expr11 oper1 expr12 || expr21 oper2 expr22
  ```

  and oper one of

  ```
  ==          ! equal
  <>          ! not equal
  <           ! less than
  >           ! greater than
  <=          ! less than or equal
  >=          ! greater than or equal
  ```

  The expressions are arithmetic expressions of type real. The statements in the curly brackets are executed if the logical expression is true.

- ## ELSEIF

  ```
  elseif (logical_expression) {statement 1; statement 2; ...; statement n; }
  ```

  Only possible (in any number) behind an IF, or another ELSEIF; is executed if logical_expression is true, and if none of the preceding IF or ELSEIF logical conditions was true.

- ## ELSE

  ```
  else {statement 1; statement 2; ...; statement n; }
  ```

  Only possible (once) behind an IF, or an ELSEIF; is executed if logical_expression is true, and if none of the preceding IF or ELSEIF logical conditions was true.

  For a real life example, see ELSE example.

-

WHILE

```
while (logical_condition){statement 1; statement 2; ...; statement n; }
```

executes the statements in curly brackets while the logical_expression is true. A simple example (in case you have forgotten the first ten factorials) would be

```
option,-info;   ! otherwise you get redifiniton warnings
n=1; m=1;
while (n <= 10)
{
  m = m * n;  value, m;
  n = n + 1;
};
```

For a real life example, see WHILE example.

-

MACRO

```
label: macro = {statement 1; statement 2; ...; statement n; };
label(arg1,...,argn): macro = {statement 1; statement 2; ...; statement n; };
```

The first form allows the execution of a group of statements via a single command:

```
exec, label;
```

will execute the statements in curly brackets exactly once. This command can be issued any number of times.

The second form allows to replace strings anywhere inside the statements in curly brackets by other strings, or integer numbers prior to execution. This is a powerful construct and should be handled with care.

Simple example:

```
option,-echo,-info;  ! otherwise the output is somewhat confusing
simple(xx,yy): macro = { xx = yy^2 + xx; value, xx;};
a = 3;
b = 5;
exec, simple(a,b);
```

Somewhat more tricky (a "$" in front of an argument means that the truncated integer value of this argument is used for replacement, rather than the argument string itself).

```
tricky(xx,yy,zz): macro = {mzz.yy: xx, l = 1.yy, kzz = k.yy;};
n=0;
while (n < 3)
{
  n = n+1;
  exec,tricky(quadrupole,$n,1);
  exec,tricky(sextupole,$n,2);
};
```

Whereas the actual use of the preceding example is NOT recommended, a real life example, showing the full power (!) of macros is to be found under macro usage for the usage, and under macro definition for the definition.

Beware of the following rules:

- Generally speaking: *special constructs* like IF, WHILE, DO, MACRO will only allow one level of inclusion of another *special construct* .
- Macros must not be called with numbers, but with strings (i.e. variable names in case of numerical values), i.e.

NOT

```
exec,thismacro($99,$129);
```

BUT

```
n1=99; n2=219;
exec,thismacro($n1,$n2);
```

hansg, June 17, 2002

Real life example for IF statements, and MACRO usage

```
! Creates a footprint for head-on + parasitic collisions at IP1+5
! of lhc.6.5; both lhcb1 (for tracking) and lhcb2 (to define the
! beam-beam elements, i.e. weak-strong) are used; there are flags to
! select head-on, left, and right parasitic separately at all IPs.
! The bunch spacing can be given in nanosec and automatically creates
! the beam-beam interaction points at the correct positions.
! It is important to set the correct BEAM parameters, i.e. number
! of particles, emittances, bunch length, energy.

!--- For completeness, all files needed by this job are copied
!    to the local directory ldb. The links to the the originals
!    in offdb (official database) are commented out.

Option,  warn,info,echo;
!System,
"ln -fns /afs/cern.ch/eng/sl/MAD-X/dev/test_suite/foot/V3.01.01 ldb";
!system,"ln -fns /afs/cern.ch/eng/lhc/optics/V6.4 offdb";
Option, -echo,-info,warn;
SU=1.0;
call, file = "ldb/V6.5.seq";
call,file="ldb/slice_new.madx";
Option, echo,info,warn;

!+++++++++++++++++++++++++ Step 1 ++++++++++++++++++++++
!       define beam constants
!+++++++++++++++++++++++++++++++++++++++++++++++++++++++

b_t_dist = 25.e-9;                  !--- bunch distance in [sec]
b_h_dist = clight * b_t_dist / 2 ;  !--- bunch half-distance in [m]
ip1_range = 58.;                     ! range for parasitic collisions
ip5_range = ip1_range;
ip2_range = 60.;
ip8_range = ip2_range;

npara_1 = ip1_range / b_h_dist;     ! # parasitic either side
npara_2 = ip2_range / b_h_dist;
npara_5 = ip5_range / b_h_dist;
npara_8 = ip8_range / b_h_dist;

value,npara_1,npara_2,npara_5,npara_8;

 eg   =  7000;
 bg   =  eg/pmass;
 en   = 3.75e-06;
 epsx = en/bg;
 epsy = en/bg;

Beam, particle = proton, sequence=lhcb1, energy = eg,
          sigt=      0.077     ,
          bv = +1, NPART=1.1E11, sige=      1.1e-4,
          ex=epsx,   ey=epsy;
```

```
Beam, particle = proton, sequence=lhcb2, energy = eg,
          sigt=      0.077     ,
          bv = -1, NPART=1.1E11, sige=      1.1e-4,
          ex=epsx,   ey=epsy;

beamx = beam%lhcb1->ex;   beamy%lhcb1 = beam->ey;
sigz  = beam%lhcb1->sigt; sige = beam%lhcb1->sige;

!--- split5, 4d
long_a= 0.53 * sigz/2;
long_b= 1.40 * sigz/2;
value,long_a,long_b;

ho_charge = 0.2;

!++++++++++++++++++++++++ Step 2 ++++++++++++++++++++++
!       slice, flatten sequence, and cycle start to ip3
!++++++++++++++++++++++++++++++++++++++++++++++++++++++

use,sequence=lhcb1;
makethin,sequence=lhcb1;
!save,sequence=lhcb1,file=lhcb1_thin_new_seq;
use,sequence=lhcb2;
makethin,sequence=lhcb2;
!save,sequence=lhcb2,file=lhcb2_thin_new_seq;
!stop;

option,-warn,-echo,-info;
call,file="ldb/V6.5.thin.coll.str";
option,warn,echo,info;

! keep sextupoles
ksf0=ksf; ksd0=ksd;
use,period=lhcb1;
select,flag=twiss.1,column=name,x,y,betx,bety;
twiss,file;
plot,haxis=s,vaxis=x,y,colour=100,noline;

use,period=lhcb2;
select,flag=twiss.2,column=name,x,y,betx,bety;
twiss,file;
plot,haxis=s,vaxis=x,y,colour=100,noline;
seqedit,sequence=lhcb1;
flatten;
endedit;

seqedit,sequence=lhcb1;
cycle,start=ip3.b1;
endedit;

seqedit,sequence=lhcb2;
flatten;
endedit;

seqedit,sequence=lhcb2;
cycle,start=ip3.b2;
endedit;

bbmarker: marker;  /* for subsequent remove */


!++++++++++++++++++++++++ Step 3 ++++++++++++++++++++++
```

```
!       define the beam-beam elements
!+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
!
!=============================================================
! read macro definitions
option,-echo;
call,file="ldb/bb.macros";
option,echo;

!
!=============================================================
!   this sets CHARGE in the head-on beam-beam elements.
!   set +1 * ho_charge   for parasitic on, 0 for off

 on_ho1  = +1 * ho_charge; ! ho_charge depends on split
 on_ho2  = +0 * ho_charge; ! because of the "by hand" splitting
 on_ho5  = +1 * ho_charge;
 on_ho8  = +0 * ho_charge;

!
!=============================================================
!   set CHARGE in the parasitic beam-beam elements.
!   set +1 for parasitic on, 0 for off
 on_lr1l = +1;
 on_lr1r = +1;
 on_lr2l = +0;
 on_lr2r = +0;
 on_lr5l = +1;
 on_lr5r = +1;
 on_lr8l = +0;
 on_lr8r = +0;

!
!=============================================================
!   define markers and savebetas
assign,echo=temp.bb.install;
!--- ip1
if (on_ho1 <> 0)
{
  exec, mkho(1);
  exec, sbhomk(1);
}
if (on_lr1l <> 0 || on_lr1r <> 0)
{
  n=1; ! counter
  while (n < npara_1)
  {
   exec, mkl(1,$n);
   exec, sbl(1,$n);
   n=n+1;
  };
}
if (on_lr1r <> 0 || on_lr1l <> 0)
{
  n=1; ! counter
  while (n < npara_1)
  {
   exec, mkr(1,$n);
   exec, sbr(1,$n);
   n=n+1;
  };
}
```

```
!--- ip5
if (on_ho5 <> 0)
{
  exec, mkho(5);
  exec, sbhomk(5);
}
if (on_lr5l <> 0 || on_lr5r <> 0)
{
  n=1; ! counter
  while (n < npara_5)
  {
   exec, mkl(5,$n);
   exec, sbl(5,$n);
   n=n+1;
  };
}
if (on_lr5r <> 0 || on_lr5l <> 0)
{
  n=1; ! counter
  while (n < npara_5)
  {
   exec, mkr(5,$n);
   exec, sbr(5,$n);
   n=n+1;
  };
}
!--- ip2
if (on_ho2 <> 0)
{
  exec, mkho(2);
  exec, sbhomk(2);
}
if (on_lr2l <> 0 || on_lr2r <> 0)
{
  n=1; ! counter
  while (n < npara_2)
  {
   exec, mkl(2,$n);
   exec, sbl(2,$n);
   n=n+1;
  };
}
if (on_lr2r <> 0 || on_lr2l <> 0)
{
  n=1; ! counter
  while (n < npara_2)
  {
   exec, mkr(2,$n);
   exec, sbr(2,$n);
   n=n+1;
  };
}
!--- ip8
if (on_ho8 <> 0)
{
  exec, mkho(8);
  exec, sbhomk(8);
}
if (on_lr8l <> 0 || on_lr8r <> 0)
{
  n=1; ! counter
  while (n < npara_8)
```

```
  {
   exec, mkl(8,$n);
   exec, sbl(8,$n);
   n=n+1;
  };
}
if (on_lr8r <> 0 || on_lr8l <> 0)
{
  n=1; ! counter
  while (n < npara_8)
  {
   exec, mkr(8,$n);
   exec, sbr(8,$n);
   n=n+1;
  };
}
assign,echo=terminal;
call,file=temp.bb.install;
system, "rm temp.bb.install";
!
!============================================================
!   install bb markers
assign,echo=temp.bb.install;
!--- ip1
if (on_ho1 <> 0)
{
exec, inho(mk,1);
}
if (on_lr1l <> 0 || on_lr1r <> 0)
{
  n=1; ! counter
  while (n < npara_1)
  {
   exec, inl(mk,1,$n);
   n=n+1;
  };
}
if (on_lr1r <> 0 || on_lr1l <> 0)
{
  n=1; ! counter
  while (n < npara_1)
  {
   exec, inr(mk,1,$n);
   n=n+1;
  };
}
!--- ip5
if (on_ho5 <> 0)
{
exec, inho(mk,5);
}
if (on_lr5l <> 0 || on_lr5r <> 0)
{
  n=1; ! counter
  while (n < npara_5)
  {
   exec, inl(mk,5,$n);
   n=n+1;
  };
}
if (on_lr5r <> 0 || on_lr5l <> 0)
{
```

```
  n=1; ! counter
  while (n < npara_5)
  {
   exec, inr(mk,5,$n);
   n=n+1;
  };
}
!--- ip2
if (on_ho2 <> 0)
{
exec, inho(mk,2);
}
if (on_lr2l <> 0 || on_lr2r <> 0)
{
  n=1; ! counter
  while (n < npara_2)
  {
   exec, inl(mk,2,$n);
   n=n+1;
  };
}
if (on_lr2r <> 0 || on_lr2l <> 0)
{
  n=1; ! counter
  while (n < npara_2)
  {
   exec, inr(mk,2,$n);
   n=n+1;
  };
}
!--- ip8
if (on_ho8 <> 0)
{
exec, inho(mk,8);
}
if (on_lr8l <> 0 || on_lr8r <> 0)
{
  n=1; ! counter
  while (n < npara_8)
  {
   exec, inl(mk,8,$n);
   n=n+1;
  };
}
if (on_lr8r <> 0 || on_lr8l <> 0)
{
  n=1; ! counter
  while (n < npara_8)
  {
   exec, inr(mk,8,$n);
   n=n+1;
  };
}
assign,echo=terminal;
seqedit,sequence=lhcb2;
call,file=temp.bb.install;
endedit;
system, "rm temp.bb.install";

!
!=============================================================
!--- get beta functions at bb in all four IPs
```

```
use,period=lhcb2;
!select,flag=twiss,class=bbmarker,column=name,s,x,y;
twiss, sequence=lhcb2; !,file;
!--- separation for halo collisions at IP2
on_sep2 = 2.118 * sqrt(epsx * r2ip2->betx) / 0.0007999979093;
value,on_sep2;
!============================================================
!   define bb elements
assign,echo=temp.bb.install;
!--- ip1
if (on_ho1 <> 0)
{
exec, bbho(1);
}
if (on_lr1l <> 0)
{
  n=1; ! counter
  while (n < npara_1)
  {
   exec, bbl(1,$n);
   n=n+1;
  };
}
if (on_lr1r <> 0)
{
  n=1; ! counter
  while (n < npara_1)
  {
   exec, bbr(1,$n);
   n=n+1;
  };
}
!--- ip5
if (on_ho5 <> 0)
{
exec, bbho(5);
}
if (on_lr5l <> 0)
{
  n=1; ! counter
  while (n < npara_5)
  {
   exec, bbl(5,$n);
   n=n+1;
  };
}
if (on_lr5r <> 0)
{
  n=1; ! counter
  while (n < npara_5)
  {
   exec, bbr(5,$n);
   n=n+1;
  };
}
!--- ip2
if (on_ho2 <> 0)
{
exec, bbho(2);
}
if (on_lr2l <> 0)
{
```

```
  n=1; ! counter
  while (n < npara_2)
  {
   exec, bbl(2,$n);
   n=n+1;
  };
}
if (on_lr2r <> 0)
{
  n=1; ! counter
  while (n < npara_2)
  {
   exec, bbr(2,$n);
   n=n+1;
  };
}
!--- ip8
if (on_ho8 <> 0)
{
exec, bbho(8);
}
if (on_lr8l <> 0)
{
  n=1; ! counter
  while (n < npara_8)
  {
   exec, bbl(8,$n);
   n=n+1;
  };
}
if (on_lr8r <> 0)
{
  n=1; ! counter
  while (n < npara_8)
  {
   exec, bbr(8,$n);
   n=n+1;
  };
}
assign,echo=terminal;
call,file=temp.bb.install;
system, "rm temp.bb.install";
!
!============================================================
!   install bb elements
assign,echo=temp.bb.install;
!--- ip1
if (on_ho1 <> 0)
{
exec, inho(bb,1);
}
if (on_lr1l <> 0)
{
  n=1; ! counter
  while (n < npara_1)
  {
   exec, inl(bb,1,$n);
   n=n+1;
  };
}
if (on_lr1r <> 0)
{
```

```
  n=1; ! counter
  while (n < npara_1)
  {
   exec, inr(bb,1,$n);
   n=n+1;
  };
}
!--- ip5
if (on_ho5 <> 0)
{
exec, inho(bb,5);
}
if (on_lr5l <> 0)
{
  n=1; ! counter
  while (n < npara_5)
  {
   exec, inl(bb,5,$n);
   n=n+1;
  };
}
if (on_lr5r <> 0)
{
  n=1; ! counter
  while (n < npara_5)
  {
   exec, inr(bb,5,$n);
   n=n+1;
  };
}
!--- ip2
if (on_ho2 <> 0)
{
exec, inho(bb,2);
}
if (on_lr2l <> 0)
{
  n=1; ! counter
  while (n < npara_2)
  {
   exec, inl(bb,2,$n);
   n=n+1;
  };
}
if (on_lr2r <> 0)
{
  n=1; ! counter
  while (n < npara_2)
  {
   exec, inr(bb,2,$n);
   n=n+1;
  };
}
!--- ip8
if (on_ho8 <> 0)
{
exec, inho(bb,8);
}
if (on_lr8l <> 0)
{
  n=1; ! counter
  while (n < npara_8)
```

```
  {
   exec, inl(bb,8,$n);
   n=n+1;
  };
}
if (on_lr8r <> 0)
{
  n=1; ! counter
  while (n < npara_8)
  {
   exec, inr(bb,8,$n);
   n=n+1;
  };
}
assign,echo=terminal;
select,flag=seqedit,class=bbmarker;
seqedit,sequence=lhcb2;
remove,element=selected;
endedit;
select,flag=seqedit,clear;

seqedit,sequence=lhcb1;
call,file=temp.bb.install;
endedit;

!--- Now the beam-beam element installation is complete

system, "rm temp.bb.install";

seqedit,sequence=lhcb1;
cycle,start=ip1;
endedit;

use,period=lhcb1;
!twiss, sequence=lhcb1;
!
!  make footprint
!

option,trace;
small=0.05;
big=sqrt(1.-small^2);
track;
xs=small; ys=small;
value,xs,ys;
start,fx=xs,fy=ys;  // zero amplitude
nsigmax=6;
n=1; // sigma multiplier
m=0; // angle multiplier
while (n <= nsigmax)
{
  angle = 15*m*pi/180;
  if (m == 0) {xs=n*big; ys=n*small;}
  elseif (m == 6) {xs=n*small; ys=n*big;}
  else
  {
    xs=n*cos(angle);
    ys=n*sin(angle);
  }
  value,xs,ys;
  start,fx=xs,fy=ys;
  m=m+1;
```

```
  if (m == 7) { m=0; n=n+1;}
};
dynap,fastune,turns=1024;
endtrack;
write,table=dynap,file;
write,table=dynaptune,file;
system,"foot < dynaptune > footprint";
stop;
```

Real life example of MACRO definitions

```
bbho(nn): macro = {
!--- macro defining head-on beam-beam elements; nn = IP number
print, text="bbipnnl2: beambeam, sigx=sqrt(rnnipnnl2->betx*epsx),";
print, text="          sigy=sqrt(rnnipnnl2->bety*epsy),";
print, text="          xma=rnnipnnl2->x,yma=rnnipnnl2->y,";
print, text="          charge:=on_honn;";
print, text="bbipnnl1: beambeam, sigx=sqrt(rnnipnnl1->betx*epsx),";
print, text="          sigy=sqrt(rnnipnnl1->bety*epsy),";
print, text="          xma=rnnipnnl1->x,yma=rnnipnnl1->y,";
print, text="          charge:=on_honn;";
print, text="bbipnn:   beambeam, sigx=sqrt(rnnipnn->betx*epsx),";
print, text="          sigy=sqrt(rnnipnn->bety*epsy),";
print, text="          xma=rnnipnn->x,yma=rnnipnn->y,";
print, text="          charge:=on_honn;";
print, text="bbipnnr1: beambeam, sigx=sqrt(rnnipnnr1->betx*epsx),";
print, text="          sigy=sqrt(rnnipnnr1->bety*epsy),";
print, text="          xma=rnnipnnr1->x,yma=rnnipnnr1->y,";
print, text="          charge:=on_honn;";
print, text="bbipnnr2: beambeam, sigx=sqrt(rnnipnnr2->betx*epsx),";
print, text="          sigy=sqrt(rnnipnnr2->bety*epsy),";
print, text="          xma=rnnipnnr2->x,yma=rnnipnnr2->y,";
print, text="          charge:=on_honn;";
};

mkho(nn): macro = {
!--- macro defining head-on markers; nn = IP number
print, text="mkipnnl2: bbmarker;";
print, text="mkipnnl1: bbmarker;";
print, text="mkipnn:   bbmarker;";
print, text="mkipnnr1: bbmarker;";
print, text="mkipnnr2: bbmarker;";
};

inho(xx,nn): macro = {
!--- macro installing bb or markers for head-on beam-beam (split into 5)
print, text="install, element= xxipnnl2, at=-long_b, from=ipnn;";
print, text="install, element= xxipnnl1, at=-long_a, from=ipnn;";
print, text="install, element= xxipnn,   at=1.e-9,   from=ipnn;";
print, text="install, element= xxipnnr1, at=+long_a, from=ipnn;";
print, text="install, element= xxipnnr2, at=+long_b, from=ipnn;";
};

sbhomk(nn): macro = {
!--- macro to create savebetas for ho markers
print, text="savebeta, label=rnnipnnl2, place=mkipnnl2;";
print, text="savebeta, label=rnnipnnl1, place=mkipnnl1;";
print, text="savebeta, label=rnnipnn,   place=mkipnn;";
print, text="savebeta, label=rnnipnnr1, place=mkipnnr1;";
print, text="savebeta, label=rnnipnnr2, place=mkipnnr2;";
};
```

```
mkl(nn,cc): macro = {
!--- macro to create parasitic bb marker on left side of ip nn; cc = count
print, text="mkipnnplcc: bbmarker;";
};

mkr(nn,cc): macro = {
!--- macro to create parasitic bb marker on right side of ip nn; cc = count
print, text="mkipnnprcc: bbmarker;";
};

sbl(nn,cc): macro = {
!--- macro to create savebetas for left parasitic
print, text="savebeta, label=rnnipnnplcc, place=mkipnnplcc;";
};

sbr(nn,cc): macro = {
!--- macro to create savebetas for right parasitic
print, text="savebeta, label=rnnipnnprcc, place=mkipnnprcc;";
};

inl(xx,nn,cc): macro = {
!--- macro installing bb and markers for left side parasitic beam-beam
print, text="install, element= xxipnnplcc, at=-cc*b_h_dist, from=ipnn;";
};

inr(xx,nn,cc): macro = {
!--- macro installing bb and markers for right side parasitic beam-beam
print, text="install, element= xxipnnprcc, at=cc*b_h_dist, from=ipnn;";
};

bbl(nn,cc): macro = {
!--- macro defining parasitic beam-beam elements; nn = IP number
print, text="bbipnnplcc: beambeam, sigx=sqrt(rnnipnnplcc->betx*epsx),";
print, text="          sigy=sqrt(rnnipnnplcc->bety*epsy),";
print, text="          xma=rnnipnnplcc->x,yma=rnnipnnplcc->y,";
print, text="          charge:=on_lrnnl;";
};

bbr(nn,cc): macro = {
!--- macro defining parasitic beam-beam elements; nn = IP number
print, text="bbipnnprcc: beambeam, sigx=sqrt(rnnipnnprcc->betx*epsx),";
print, text="          sigy=sqrt(rnnipnnprcc->bety*epsy),";
print, text="          xma=rnnipnnprcc->x,yma=rnnipnnprcc->y,";
print, text="          charge:=on_lrnnr;";
};
```

hansg, June 17, 2002

Parameter Statements

Relations between Variable Parameters

A relation is established between variables by the statement

```
parameter-name = expression;
```

or

```
parameter-name := expression;
```

The first form evaluates the expression at the right immediately and assigns its value to the parameter. The second form assigns the value by evaluating the expression at right every time the parameter is actually used. This holds as well for element parameters - beware ! If you want to modify e.g. the strength of a quadrupole later (e.g. in a match, or by entering a new value for a parameter on which it depends) then the defition has to be

```
qd:quadrupole,k1:= ak1;
```

and not

```
qd:quadrupole,k1 = ak1;
```

In the latter case, k1 will be set to the current value of ak1, and will not change when ak1 changes.

Parameters not yet defined have the value zero.

Example:

```
gev = 100;
beam,energy=gev;
```

the parameter on the left may appear on the right as well:

```
x = x+1;
```

Increases the value of x by 1. As a result, the SET statement of MAD-8 is no longer necessary and is not implemented.

```
Circular definitions are allowed in the first form:

a = b + 2;
b = a * b;

However, circular definitions in the second form are forbidden:
```

```
a := b + 2;
b := a * b;
```

will result in an error.

VALUE: Output of Parameters

The VALUE statement

```
VALUE = expression;
```

or

```
VALUE = expression1, expression2, ...;
```

evaluates the current value of "expression" resp. "expression1" etc. and prints the result on the standard output file.

Example:

```
p1 = 5;
p2 = 7;
value,p1*p2-3;
```

After echoing the command, this prints:

```
p1*p2-3 = 32       ;
```

Another example:

```
option,-warn;
while (x < 100) {x = x + 1;}
value,x,x^2,log10(x);
```

After echoing the command, this prints:

```
x = 100        ;
x^2 = 10000      ;
log10(x) = 2      ;
```

hansg 11.9.2000

Constraints

In matching it is desired to specify equality constraints, as well as lower and upper limits for a quantity. MAD accepts the following forms of constraints:

```
! equality constraint:
name=expression

! upper limit:
name<expression

! lower limit:
name>expression

! both upper and lower limit for the same name:
name<expression,name>expression
```

hansg, May 8, 2001

Variable Names

A variable name can have one of the formats:

```
1. parameter-name
2. element-name->attribute-name
3. command-name->attribute-name
4. beam%sequence-name->attribute-name
5. table(table-name,..,..)
```

The first format refers to the value of the global parameter "parameter-name", the second and third formats refer to the real attribute "attribute-name" of the element "element-name", or the command "command-name". NUmber four is specific to beams belonging to a particular sequence (for details see sequences and beams). Number five allows extraction of variables from existing tables, as specified in table access.

hansg, May 8, 2001

EUROPEAN ORGANIZATION FOR NUCLEAR RESEARCH 

Regular Expressions

Some commands allow selection of items via "regular expression" strings. Such a pattern string **must** be enclosed in single or double quotes. MAD-X follows regexp (Unix regular expression patterns) for matching. The following features are implemented:

A "search string" below is the string containing the pattern, a "target string" is the string being searched for a possible match with the pattern.

- "^" at the start of the search string: Match following search string at the start of the target string; otherwise the search string can start anywhere in the target string. To search for a genuine "^" anywhere, use "\^".
- "$" at the end of the search string: Match preceding search string at the end of the target string; otherwise the search string can end anywhere in the target string. To search for a genuine "$" anywhere, use "\$".
- ".": Stands for an arbitrary character; to search for a genuine ".", use "\."
- "[xyz]": Stands for one character belonging to the string contained in brackets (example: "[abc]" means one of a, b, c).
- "[a-ex-z]": Stands for ranges of characters (example: "[a-zA-Z]" means any letter).
- "[^xyz]" (i.e. a "^" as first character in a square bracket): Stands for exclusion of all characters in the list, i.e. "[^a-z]" means "any character but a lower case letter".
- "*": Allows zero or more repetitions of the preceding character, either specified directly, or from a list. (examples: "a*" means zero or more occurrences of "a", "[A-Z]*" means zero or more upper-case letters).
- "backslash-c" (e.g. "\."): Removes the special meaning of character c.

All other characters stand for themselves. Example:

```
select,flag=twiss,pattern="^d..$" ;
select,flag=twiss,pattern="^k.*qd.*\.r1$" ;
```

The first command selects all elements whose names have exactly three characters and begin with the letter "D". The second command selects elements beginning with the letter "K", containing the string "QD", and ending with the string ".R1". The two occurrences of ".*" each stand for an arbitrary number (including zero) of any character, and the occurrence "\." stands for a literal period.

hansg, May 8, 2001

EUROPEAN ORGANIZATION FOR NUCLEAR RESEARCH 

Control Statements

MAD-X consists of a core program, and modules for specific tasks such as twiss parameter calculation, matching, thin lens tracking, and so on.

The statements listed here are those executed by the program core. They deal with the I/O, element and sequence declaration, sequence manipulation, statement flow control (e.g. IF, WHILE), MACRO declaration, saving sequences onto files in MAD-X or MAD-8 format, and so on.

- ## Program flow control
  - IF
  - ELSEIF
  - ELSE
  - WHILE
  - MACRO
- ## General control
  - ASSIGN
  - CALL
  - COGUESS
  - CREATE
  - DUMPSEQU
  - EXEC
  - EXIT
  - FILL
  - HELP
  - OPTION
  - PRINT
  - QUIT
  - READTABLE
  - RETURN
  - SAVE
  - SAVEBETA
  - SELECT
  - SET
  - SHOW
  - STOP

- SYSTEM
- TABSTRING
- TITLE
- USE
- VALUE
- WRITE

Beam specification

- BEAM
- RESBEAM

PLOT

- PLOT
- RESPLOT
- SETPLOT

Sequence editing

- SEQEDIT
- FLATTEN
- INSTALL
- MOVE
- REMOVE
- CYCLE
- REFLECT
- ENDEDIT

hansg, June 17, 2002

General Control Statements

- ## ASSIGN

  ```
  assign, echo = file_name;
  ```

  where "file_name" is the name of an output file, or "terminal". This allows switching the echo stream to a file or back, but only for the commands value, show, and print. Allows easy composition of future MAD-X input files. A real life example (always the same) is to be found under footprint example.

- ## CALL

  ```
  call, file = file_name;
  ```

  where "file_name" is the name of an input file. This file will be read until a "return;" statement, or until end_of_file; it may contain any number of calls itself, and so on to any depth.

- ## COGUESS

  ```
  coguess,tolerance=double,x=double,
         px=double,y=double,py=double,t=double,pt=double;
  ```

  sets the required convergence precision in the closed orbit search ("tolerance", see as well Twiss command tolerance.

  The other parameters define a first guess for all future closed orbit searches in case they are different from zero.

- ## CREATE

  ```
  create,table=table,column=var1,var2,_name,...;
  ```

  creates a table with the specified variables as columns. This table can then be filled, and finally one can write it in TFS format. The attribute "_name" adds the element name to the table at the specified column, this replaces the undocumented "withname" attribute that was not always working properly.

See the user table I example;

or an example of joining 2 tables of different length in one table including the element name: user table II

- ## DELETE

  ```
  delete,sequence=s_name,table=t_name;
  ```

  deletes a sequence with name "s_name" or a table with name "t_name" from memory. The sequence deletion is done without influence on other sequences that may have elements that were in the deleted sequence.

- ## DUMPSEQU

  ```
  dumpsequ, sequence = s_name, level = integer;
  ```

  Actually a debug statement, but it may come handy at certain occasions. Here "s_name" is the name of an expanded (i.e. USEd) sequence. The amount of detail is controlled by "level":

  ```
  level = 0:    print only the cumulative node length = sequence length
        > 0:    print all node (element) names except drifts
        > 2:    print all nodes with their attached parameters
        > 3:    print all nodes, and their elements with all parameters
  ```

- ## EXEC

  ```
  exec, label;
  ```

  Each statement may be preceded by a label; it is then stored and can be executed again with "exec, label;" any number of times; the executed statement may be another "exec", etc.; however, the major usage of this statement is the execution of a macro.

- ## EXIT

  ```
  exit;
  ```

  ends the program execution.

- ## FILL

  Every command

  ```
  fill,table=table;
  ```

  adds a new line with the current values of all column variables into the user table created beforehand. This table one can then write in TFS format. See as well the user table example.

- ## HELP

  ```
  help,statement_name;
  ```

  prints all parameters, and their defaults of the statement given; this includes basic element types.

- ## OPTION

  ```
  option, flag { = true | false };
  option, flag | -flag;
  ```

  sets an option as given in "flag"; the part in curly brackets is optional: if only the name of the option is given, then the option will be set true (see second line); a "-" sign preceding the name sets it to "false".

  Example:

  ```
  option,echo=true;
  option,echo;
  ```

  are identical, ditto

  ```
  option,echo=false;
  option,-echo;
  ```

  The available options are:

  ```
  name           default meaning if true
  ====           ======= ===============
  bborbit         false  the closed orbit is modified by beam-beam kicks
  sympl           false  all element matrices are symplectified in Twiss
  echo            true   echoes the input on the standard output file
  trace           false  prints the system time after each command
  verify          false  issues a warning if an undefined variable is used
  warn            true   issues warnings
  info            true   issues informations
  tell            false  prints the current value of all options
  reset           false  resets all options to their defaults
  rbarc           true   converts the RBEND straight length into the arc length
  thin_foc        true   if false suppresses the 1(rho**2) focusing of thin dipoles
  no_fatal_stop   false  Prevents madx from stopping in case of a fatal error. Use at your own risk.
  ```

  The option "rbarc" is implemented for backwards compatibility with MAD-8 up to version 8.23.06 included; in this version, the RBEND length was just taken as the arc length of an SBEND with inclined pole faces, contrary to the MAD-8 manual.

- ## PRINT

  ```
  print,text="...";
  ```

  prints the text to the current output file (see ASSIGN above). The text can be edited with the help of a macro statement. For more details, see there.

- ## QUIT

  ```
  quit;
  ```

  ends the program execution.

- ## READTABLE

  ```
  readtable,file=filename;
  ```

  reads a TFS file containing a MAD-X table back into memory. This table can then be manipulated as any other table, i.e. its values can be accessed, it can be plotted, written out again etc.

- ## READMYTABLE

  ```
  readmytable,file=filename,table=internalname;
  ```

  reads a TFS file containing a MAD-X table back into memory. This table can then be manipulated as any other table, i.e. its values can be accessed, it can be plotted, written out again etc. An internal name for the table can be freely assigned while for the command READTABLE it is taken from the information section of the table itself. This feature allows to store multiple tables of the same type in memory without overwriting existing ones.

- ## RESBEAM

  ```
  resbeam,sequence=s_name;
  ```

  resets the default values of the beam belonging to sequence s_name, or of the default beam if no sequence is given.

- ## RETURN

  ```
  return;
  ```

  ends reading from a "called" file; if encountered in the standard input file, it ends the program execution.

- ## SAVE

  ```
  save,beam,sequence=sequ1,sequ2,...,file=filename,beam,bare;
  ```

  saves the sequence(s) specified with all variables and elements needed for their expansion, onto the file "filename". If quotes are used for the "filename" capital and low characters are kept as specified, if they are omitted the "filename" will have lower characters only. The optional flag

can have the value "mad8" (without the quotes), in which case the sequence(s) is/are saved in MAD-8 input format.

The flag "beam" is optional; when given, all beams belonging to the sequences specified are saved at the top of the save file.

The parameter "sequence" is optional; when omitted, all sequences are saved.

However, it is not advisable to use "save" without the "sequence" option unless you know what you are doing. This practice will avoid spurious saved entries. Any number of "select,flag=save" commands may precede the SAVE command. In that case, the names of elements, variables, and sequences must match the pattern(s) if given, and in addition the elements must be of the class(es) specified. See here for a SAVE with SELECT example.

It is important to note that the precision of the output of the save command depends on the output precision. Details about default precisions and how to adjust those precisions can be found at the SET Format instruction page.

The Attribute ’bare’ allows to save just the sequence without the element definitions nor beam information. This allows to re-read in a sequence with might otherwise create a stop of the program. This is particularly useful to turn a line into a sequence to seqedit it. Example:

```
tl3:line=(ldl6,qtl301,mqn,qtl301,ldl7,qtl302,mqn,qtl302,ldl8,ison);
DLTL3 : LINE=(delay, tl3);
use, period=dltl3;

save,sequence=dltl3,file=t1,bare; // new parameter "bare": only sequ. saved
call,file=t1; // sequence is read in and is now a "real" sequence
// if the two preceding lines are suppressed, seqedit will print a warning
// and else do nothing
use, period=dltl3;
twiss, save, betx=bxa, alfx=alfxa, bety=bya, alfy=alfya;
plot, vaxis=betx, bety, haxis=s, colour:=100;
SEQEDIT, SEQUENCE=dltl3;
  remove,element=cx.bhe0330;
  remove,element=cd.bhe0330;
ENDEDIT;

use, period=dltl3;
twiss, save, betx=bxa, alfx=alfxa, bety=bya, alfy=alfya;
```

-

SAVEBETA

```
savebeta, label=label,place=place,sequence=s_name;
```

marks a place "place" in an expanded sequence "s_name"; at the next TWISS command execution, a beta0 block will be saved at that place with the label "label". This is done only once; in order to get a new beta0 block there, one has to re-issue the command. The contents of the beta0 block can then be used in other commands, e.g. TWISS and MATCH.

Example (after sequence expansion):

```
savebeta,label=sb1,place=mb[5],sequence=fivecell;
twiss;
show,sb1;
```

will save and show the beta0 block parameters at the end (!) of the fifth element mb in the sequence.

-

SELECT

```
select, flag=flag,range=range,class=class,pattern=pattern,
        slice=integer,column=s1,s2,s3,..,sn,sequence=s_name,
        full,clear;
```

selects one or several elements for special treatment in a subsequent command. All selections for a given command remain valid until "clear" is specified; the selection criteria on the same command are logically ANDed, on different SELECT statements logically ORed.

Example:

```
select,flag=error,class=quadrupole,range=mb[1]/mb[5];
select,flag=error,pattern="^mqw.*";
```

selects all quadrupoles in the range mb[1] to mb[5], and all elements (in the whole sequence) the name of which starts with "mqw" for treatment by the error module.

"flag" can be one of the following::

- seqedit: selection of elements for the seqedit module.
- error: selection of elements for the error assignment module.
- makethin: selection of elements for the makethin module that converts the sequence into one with thin elements only.
- sectormap: selection of elements for the sectormap output file from the Twiss module.
- table: here "table" is a table name such as twiss, track etc., and the rows and columns to be written are selected.

For the RANGE, CLASS, PATTERN, FULL, and CLEAR parameters see SELECT.

"slice" is only used by makethin and prescribes the number of slices into which the selected elements have to be cut (default = 1).

"column" is only valid for tables and decides the selection of columns to be written into the TFS file. The "name" argument is special in that it refers to the actual name of the selected element. For an example, see SELECT.

-

SHOW

```
show,command;
```

prints the "command" (typically "beam", "beam%sequ", or an element name), with the actual value of all its parameters.

-

STOP

```
stop;
```

ends the program execution.

-

SYSTEM

```
system,"...";
```

transfers the string in quotes to the system for execution.

Example:

```
system,"ln -s /afs/cern.ch/user/u/user/public/some/directory short";
```

-

TABSTRING

```
tabstring(arg1,arg2,arg3)
```

The"string function" tabstring(arg1,arg2,arg3) with exactly three arguments; arg1 is a table name (string), arg2 is a column name (string), arg3 is a row number (integer), count starts at 0. The function can be used in any context where a string appears; in case there is no match, it returns _void_.

-

TITLE

```
title,"...";
```

inserts the string in quotes as title in various tables and plots.

-

USE

```
use,period=s_name,range=range;
```

expands the sequence with name "s_name", or a part of it as specified in the range.

-

VALUE

```
value,exp1,exp2,...;
```

prints the actual values of the expressions given.

Example:

```
a=clight/1000.;
value,a,pmass,exp(sqrt(2));
```

results in

```
a = 299792.458          ;
pmass = 0.938271998         ;
exp(sqrt(2)) = 4.113250379         ;
```

-

WRITE

```
write,table=table,file=file_name;
```

writes the table "table" onto the file "file_name"; only the rows and columns of a preceding select,flag=table,...; are written. If no select has been issued for this table, the file will only contain the header. If the FILE argument is omitted, the table is written to standard output.

hansg, June 17, 2002

Set Statements

```
set,format="...", sequence="...";
```

The set command allows 2 actions:

1) Format

The first command lets you vary the output precision.

```
parameter: format = s1, s2, s3
```

(up to) three strings defining the integer, floating, and string output format for the save, show, value, and table output. The formats can be given in any order and stay valid until replaced. The defaults are:

```
"10d","18.10g","-18s".
```

They follow the C convention. The quotes are mandatory. The allowed formats are:

```
"nd" for integer with n = field width.
```

```
"m.nf" or "m.ng" or "m.ne" for floating, m field width, n precision.
```

```
"ns" for string output.
```

The default is "right adjusted", a "-" changes it to "left adjusted". Example:

```
set,format="22.14e";
```

changes the current floating point format to 22.14e; the other formats remain untouched.

```
set,format="s","d","g";
```

sets all formats to automatic adjustment according to C conventions.

2) Sequence

The second command lets you choose the current sequence without having to use the "USE" command, which would bring you back to a bare lattice without errors. The command only works if the chosen sequence had been activated before with the "USE" command, otherwise a warning will be issued and MAD-X will continue with the unmodified current sequence. This command is particularly useful for commands that do not have the sequence as an argument like "EMIT" or "IBS".

hansg, frs, June 18, 2003

EUROPEAN ORGANIZATION FOR NUCLEAR RESEARCH 

RESBEAM: reset beam defaults

```
label: RESBEAM,SEQUENCE=name;
```

If the sequence name is omitted, the default beam is reset.

```
Default BEAM Data:
PARTICLE        POSITRON
ENERGY          1 GeV
EX              1 rad m
EY              1 rad m
ET              1 rad m
KBUNCH          1
NPART           0
BCURRENT        0 A
BUNCHED         .TRUE.
RADIATE         .FALSE.
```

hansg, January 24, 1997

EUROPEAN ORGANIZATION FOR NUCLEAR RESEARCH 

Edit a Beam Line Sequence

With the help of the commands explained below, a sequence may be modified in many ways: the starting point can be moved to another place; the order of elements can be inverted; elements can be inserted one by one, or as a whole group with one single command; single elements, or classes thereof can be removed; elements can be replaced by others; finally, the sequence can be "flattened", i.e. all inserted sequences are replaced by their actual elements, such that a flattened sequence contains only elements. It is good practice to add a *flatten;* statement at the end of a *seqedit operation* to ensure a fully operational sequence. And this is particularly useful for the *save* command to properly save *shared sequences* and to write out in *MAD-8* format.

- ## SEQEDIT

  ```
  seqedit, sequence=s_name;
  ```

  selects the sequence named for editing. The editing is performed on the non-expanded sequence; after having finished the editing, one has to re-expand the sequence if necessary.

- ## EXTRACT

  ```
  extract,sequence=s_name,from=MARKER_1,to=MARKER_2,newname=p;
  ```

  From the sequence named "s_name" is extracted a new sequence with name "p" starting from MARKER_1 and ending at MARKER_2. The new sequence "p" can be USEd as any other sequence. It is declared as "shared" and can therefore be combined E.G. into the cycled original sequence.

- ## FLATTEN

  ```
  flatten;
  ```

  This command includes all sequences in the sequence being edited, if any. The resulting sequence contains only elements.

- ## INSTALL

  ```
  install, element=name,class=class_name,at=real,from=place|selected;
  ```

  where the parameters have the following meaning:
  - element: name of the (new) element to be inserted (mandatory)

- class: class of the new element to be inserted (mandatory)
- at: position where the element is to be inserted; if no "from" is given,this is relative to the start of the sequence. If "from" is given, it is relative to the position specified there.
- from:either a place (i.e. the name(+occurrence count) of an element already existing in the sequence, e.g. mb[15], or mq.a..i1..4 etc.; or the string "selected"; in this latter case an element of the type specified will be inserted behind all elements in the sequence that are currently selected by one or several SELECT commands of the type

  ```
  select,flag=seqedit,class=.., pattern=.., range=..;
  ```

- *Attention: no element definitions inside seqedit.*

MOVE

```
move, element=name|selected,by=real,to=real,from=place;
```

- element: name of the existing element to be moved, or "selected", in which case all elements from existing SELECT commands will be moved; in the latter case, "by" must be given.
- by: amount by which the element(s) is/are to be moved; no "to" nor "from" in this case.
- to: position to which the element has to be moved; if no from, then this is relative to the start of the sequence; otherwise, it is relative to the place given in "from".
- from: place in the sequence with respect to which the element is to be positioned.

REMOVE

```
remove, element=name|selected;
```

- element: name of the existing element to be removed, or "selected", in which case all elements from existing SELECT commands will be removed.

CYCLE

```
cycle,start=place;
```

This makes the sequence start at the place given, which must be a marker.

In the case there is a shared sequence in the used sequence, the command FLATTEN has to be used before the command CYCLE. Example:

flatten ; cycle,start=place;

REFLECT

```
reflect;
```

This inverts the order of element in the sequence, starting from the last element.

If there are shared sequences inside the USEd sequence, the command FLATTEN must be used before the command REFLECT. Alternatively each shared sequence must first be reflected. Example:

flatten ; reflect;

- ## REPLACE

  ```
  replace,element=name1|selected,by=name2;
  ```

  Element with name1 is replaced by element with name2. If name1 is "selected", then all elements selected by SELECT commands will be replaced by the element name2.

- ## ENDEDIT

  ```
  endedit;
  ```

  terminates the sequence editing process. The nodes in the sequence are renumbered according to their occurrence which might have changed during editing.

  hansg, June 17, 2002

EUROPEAN ORGANIZATION FOR NUCLEAR RESEARCH 

Elements and Markers

- Element Input Format
- Aperture, Geometric
- MARKER: Marker Definition
- DRIFT: Drift Space
- Bending Magnet
  - RBEND: Rectangular Bending Magnet
  - SBEND: Sector Bending Magnet
  - Dipedge Element
- QUADRUPOLE
- SEXTUPOLE
- OCTUPOLE
- MULTIPOLE
- SOLENOID
- Closed Orbit Corrector
  - HKICKER: Horizontal Orbit Corrector
  - VKICKER: Vertical Orbit Corrector
  - KICKER: Combined Orbit Corrector
- RFCAVITY
- ELSEPARATOR: Electrostatic Separator
- Beam Position Monitor
  - HMONITOR: Horizontal Monitor
  - VMONITOR: Vertical Monitor
  - MONITOR: Combined Monitor
  - INSTRUMENT: Other Beam Instrumentation
- Collimators
  - RCOLLIMATOR:Rectangular Collimator
  - ECOLLIMATOR:Elliptic Collimator
- Coordinate Transformations
  - YROTATION: Rotation About the Vertical Axis
  - SROTATION: Rotation Around the Longitudinal Axis
- BEAMBEAM: Beam-Beam Interaction
- MATRIX: Arbitrary Element
- Editing Element Definitions
- Element Class

hansg, January 24, 1997

EUROPEAN ORGANIZATION FOR NUCLEAR RESEARCH 

Element Input Format

All physical elements are defined by statements of the form

```
label: keyword {,attribute};
```

Example:

```
QF: QUADRUPOLE,L=1.8,K1=0.015832;
```

where

- label is a name to be given to the element (in the example QF),
- keyword is an element type keyword (in the example QUADRUPOLE).
- attribute normally has the form "attribute-name=attribute-value" or "attribute-name:=attribute-value" (except for multipoles).
- attribute-name selects the attribute, as defined for the element type keyword (in the example L and K1).
- attribute-value gives it a value (in the example 1.8 and 0.015832). The value may be specified by an expression. The "=" assigns the value on the right to the attribute at the time of definition, regardless of any further change of the right hand side; the ":=" re-evaluates the expression at the right every time the attribute is being used. For constant right hand sides, "=" and ":=" are of course equivalent.

Omitted attributes are assigned a default value, normally zero.

A special format is used for a multipole:

```
m:multipole, kn= {kn0, kn1, kn2, ..., knmax},
             ks= {ks0, ks1, ks2, ..., ksmax};
```

where kn and ks give the integrated normal and skew strengths, respectively. The commas are mandatory, each strength can be an expression; their position defines the order. example:

```
m:multipole, kn={0,0,0,myoct*lrad}, ks={0,0,0,0,-1.e-5};
```

defines a multipole with a normal octupole, and a skew decapole component.

To know the current maximum order, enter the command

```
help,multipole;
```

and count.

hansg, January 24, 1997

EUROPEAN ORGANIZATION FOR NUCLEAR RESEARCH 

Dipedge Element

A thin element describing the edge focusing of a dipole has been introduced in order to make it possible to track trajectories in the presence of dipoles with pole face angles. Only linear terms are considered since the higher order terms would the tracking non-symplectic. The transformation of the machine elements into thin lenses leaves dipedge untouched and splits correctly the SBENDS's.

It does not make sense to use it alone. It can be specified at the entrance and the exit of a SBEND. They are defined by the commands:

```
label : dipedge, h=real, e1=real, fint=real, hgap=real, tilt=real;
```

It has zero length and five attributes.

- H: Is angle/length or 1/rho (default: 0 m^{-1} - for the default the dipedge element has no effect). (must be equal to that of the associated SBEND)
- E1: The rotation angle for the pole face.The sign convention is as for a SBEND Bending Magnet. Note that it is different for an entrance and an exit. (default: 0 rad).
- FINT: field integral as for SBEND sector bend. Note that each dipedge has its own fint, so fintx is no longer necessary.
- HGAP: half gap height of the associated SBEND Bending Magnet.
- TILT: The roll angle about the longitudinal axis (default: 0 rad, i.e. a horizontal bend). A positive angle represents a clockwise rotation.

frs, February 27, 2005

MULTIPOLE: General Thin Multipole

```
label:   MULTIPOLE,LRAD=real,TILT=real,
         KNL:={..,..,..}, KSL:={..,..,..};
```

A MULTIPOLE is a thin-lens magnet of arbitrary order, including a dipole:

- LRAD: A fictitious length, which was originally just used to compute synchrotron radiation effects.
  A non-zero **LRAD** in conjunction with the OPTION **thin_foc** set to a **true** logical value takes into account of the weak focussing of bending magnets.
- TILT: The roll angle about the longitudinal axis (default: 0 rad). A positive angle represents a clockwise rotation of the multipole element.

  Please note that contrary to MAD8 one has to specify the desired TILT angle, otherwise it is taken as 0 rad. We believe that the MAD8 concept of having individual TILT values for each component and on top with default values led to considerable confusion and allowed for excessive and unphysical freedom. Instead, in MAD-X the KNL/KSL components can be considered as the normal or skew multipole components of the magnet on the bench, while the TILT attribute can be considered as an tilt alignment error in the machine.
- KNL: The normal multipole coefficients from order zero to the maximum; the parameters are positional, therefore zeros must be filled in for components that do not exist. Example of a thin-lens sextupole:

  ```
  ms:multipole, knl:={0, 0, k2l};
  ```

- KSL: The skew multipole coefficients from order zero to the maximum; the parameters are positional, therefore zeros must be filled in for components that do not exist. Example of a thin-lens skew octupole:

  ```
  ms:multipole, ksl:={0, 0, 0, k3sl};
  ```

Both KNL and KSL may be specified for the same multipole.

A multipole with no dipole component has no effect on the reference orbit, i.e. the reference system at its exit is the same as at its entrance. If it includes a dipole component, it has the same effect on the reference orbit as a dipole with zero length and deflection angle K0L, being the first component of KNL above.

hansg, Frank.Schmidt, August 28, 2003

EUROPEAN ORGANIZATION FOR NUCLEAR RESEARCH 

Collimators

Two types of collimators are defined:

- ECOLLIMATOR. Elliptic aperture,
- RCOLLIMATOR. Rectangular aperture.

```
label: ECOLLIMATOR,TYPE=name,L=real,XSIZE=real,YSIZE=real;
label: RCOLLIMATOR,TYPE=name,L=real,XSIZE=real,YSIZE=real;
```

Either type has three real attributes:

- L: The collimator length (default: 0 m).
- XSIZE: The horizontal half-aperture (default: unlimited).
- YSIZE: The vertical half-aperture (default: unlimited).

For elliptic apertures, XSIZE and YSIZE denote the half-axes respectively, for rectangular apertures they denote the half-width of the rectangle. Optically a collimator behaves like a drift space, but during tracking, it also introduces an aperture limit. The aperture is checked at the entrance. If the length is not zero, the aperture is also checked at the exit.

Example:

```
COLLIM: ECOLLIMATOR,L=0.5,XSIZE=0.01,YSIZE=0.005;
```

The straight reference system for a collimator is a cartesian coordinate system.

NOTE: When a collimator is displaced transversally in order to model an asymmetric collimator, particle losses in tracking are reported with respect to the **displaced** reference system, not with respect to the surrounding beam line.

hansg, January 24, 1997

Coordinate Transformations

YROTATION: Rotation About the Vertical Axis

```
label: YROTATION,TYPE=name,ANGLE=real;
```

The element YROTATION rotates the straight reference system about the vertical (y) axis. YROTATION has no effect on the beam, but it causes the beam to be referred to the new coordinate system

$x_2 = x_1 \cos(\text{theta}) - s_1 \sin(\text{theta})$, $y_2 = x_1 \sin(\text{theta}) + s_1 \cos(\text{theta})$,

It has one real attribute:

- ANGLE: The rotation angle theta (default: 0 rad). It must be a *small* angle, i.e. an angle comparable to the transverse angles of the orbit.

A positive angle means that the new reference system is rotated clockwise about the local y-axis with respect to the old system.

Example:

```
KINK: YROTATION,ANGLE=0.0001;
```

SROTATION: Rotation Around the Longitudinal Axis

```
label: SROTATION,ANGLE=real;
```

The element SROTATION rotates the straight reference system about the longitudinal (s) axis. SROTATION has no effect on the beam, but it causes the beam to be referred to the new coordinate system

$x_2 = x_1 \cos(\text{psi}) - y_1 \sin(\text{psi})$, $y_2 = x_1 \sin(\text{psi}) + y_1 \cos(\text{psi})$,

It has one real attribute:

- ANGLE: The rotation angle psi (default: 0 rad)

A positive angle means that the new reference system is rotated clockwise about the s-axis with respect to the old system.

Example:

```
ROLL1: SROTATION,ANGLE=PI/2.;
ROLL2: SROTATION,ANGLE=-PI/2.;
HBEND: SBEND,L=6.0,ANGLE=0.01;
VBEND: LINE=(ROLL1,HBEND,ROLL2);
```

The above is a way to represent a bend down in the vertical plane, it could be defined more simply by

```
VBEND: SBEND,L=6.0,K0S=0.01/6;
```

hansg, June 17, 2002

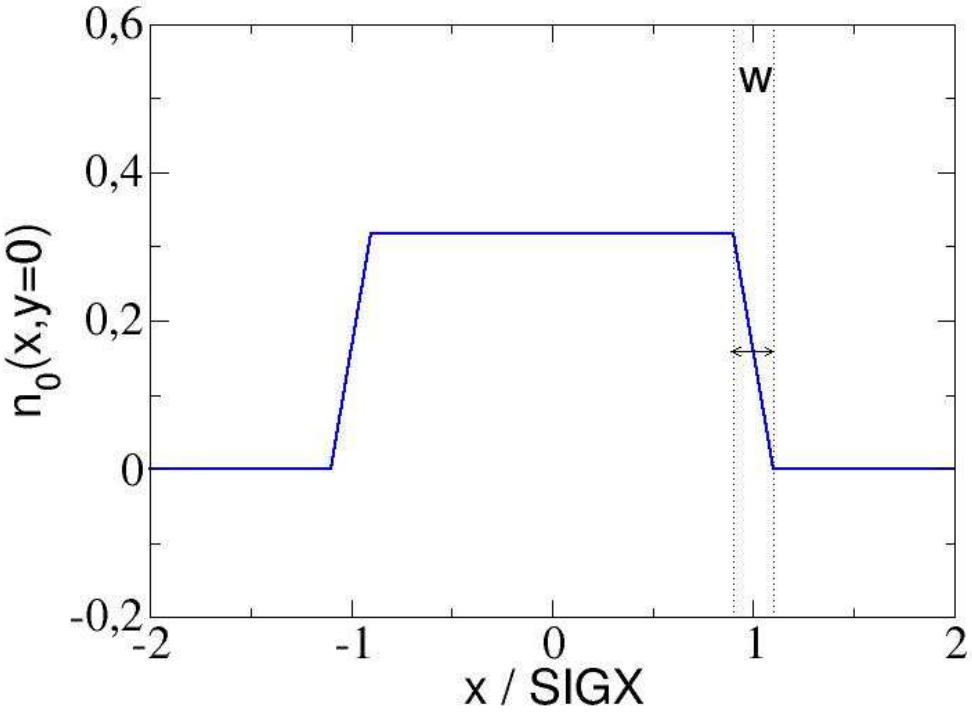
BEAMBEAM: Beam-beam Interaction

The command BEAMBEAM may be inserted in a beam line to simulate a beam-beam interaction point:

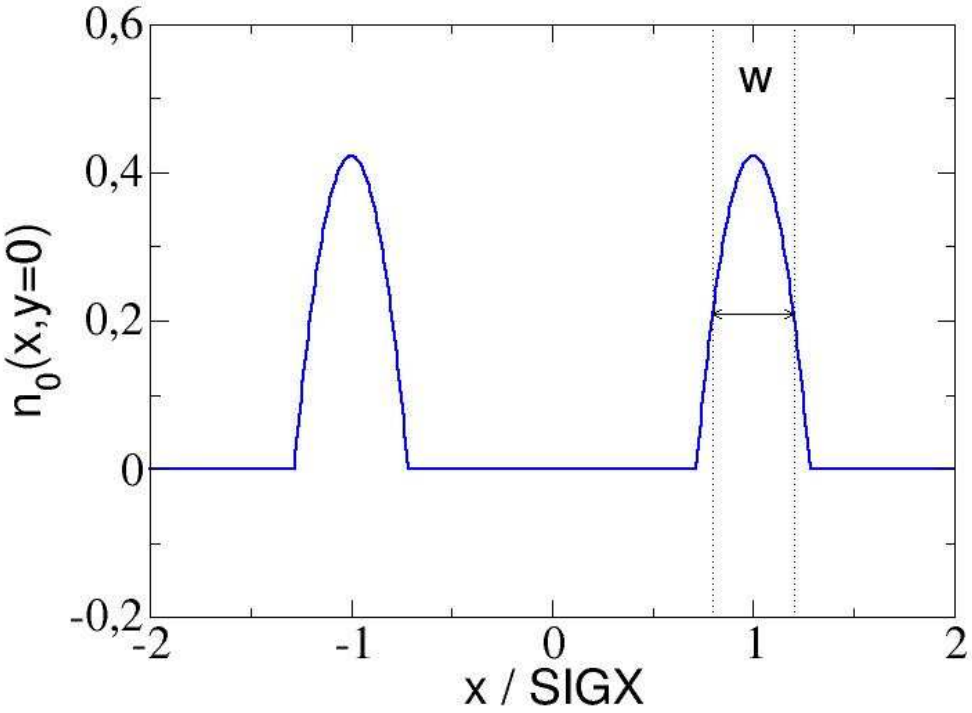
```
label: BEAMBEAM, SIGX=real,SIGY=real,
                 XMA=real,YMA=real,CHARGE=real
                 BBSHAPE=int,WIDTH=real,BBDIR=int;
```

The beam-beam interaction is represented by a four-dimensional interaction with a thin element, i.e. horizontal and vertical non-linear kicks. The code for this element has been contributed by J.M. Veuillen (1987) and extended by S. Sorge (2007).

- SIGX: The horizontal extent of the opposite beam (default: 1 m). Meaning depends on parameter BBSHAPE.
- SIGY: The vertical extent of the opposite beam (default: 1 m). Meaning depends on parameter BBSHAPE.
- XMA: The horizontal displacement of the opposite beam with respect to the ideal orbit (default: 0 m).
- YMA: The vertical displacement of the opposite beam with respect to the ideal orbit (default: 0 m).
- CHARGE: The charge of particles in the opposite beam in elementary charges. It is set by default CHARGE=1. So, if you want to describe collisions between beams containing the same particles having a charge different from 1, you have to set CHARGE explicitly in BEAM and in BEAMBEAM.
- BBSHAPE: The parameter to choose the radial density shape of the opposite beam (default: 1)
  - BBSHAPE=1: Gaussian shape (default), SIGX/SIGY: standard deviation in vertical/horizontal direction.
  - BBSHAPE=2: trapezoidal shape, SIGX/SIGY: half width of density profile, i.e. distance from the centre to half edge region with linear decrease of density in horizontal/vertical direction. Still only circular opposite beam possible, i.e. in the calculations SIGX'=SIGY'=(SIGX+SIGY)/2 is used, if SIGX and SIGY have different values



- BBSHAPE=3: hollow-parabolic shape, SIGX/SIGY: distance from the centre to the maximum of the parabolic density profile in vertical/horizontal direction. Still only circular opposite beam possible, i.e. in the calculations SIGX'=SIGY'=(SIGX+SIGY)/2 is used, if SIGX and SIGY have different values



The restriction to circular opposite beams in the cases BBSHAPE=2,3 appears to be sufficient, because such beam profiles are more important for the description of the interaction between the particle beam and an electron beam of an electron cooler, which are usually circular.

- WIDTH: The relative extent of the edge region, absolute value is given by WIDTH*SIGX and WIDTH*SIGY vertical and horizontal direction, respectively. For
  - BBSHAPE=1, WIDTH is meaningless and will be ignored.
  - BBSHAPE=2, WIDTH denotes the full width of the edge region in units of SIGX (or SIGX' and SIGY', respectively, if SIGX and SIGY are not equal), i.e. if WIDTH=0.01 and SIGX=5 mm, the edge region has a full width of 0.05 mm. It must be WIDTH < 2.0.
  - BBSHAPE=3, WIDTH denotes the full width at half maximum of the parabolic density profile in units of SIGX (or SIGX' and SIGY', respectively, if SIGX SIGY are not equal. It must be WIDTH < SQRT(2.0).
- BBDIR: The parameter to choose the direction of motion of the opposite beam relative to the beam considered. It determines the sign of the Lorenz force between the both beams (default: -1):

  >
  - BBDIR=-1: Beams move in the opposite direction as in a collider. Therefore, the Lorenz force enhances the beam-beam interaction.
  - BBDIR=0: Opposite beam does not move, Lorenz force is neglected
  - BBDIR=1: Beams move in the same direction as in an electron cooler. So, the Lorenz force reduces the beam-beam interaction. <

ul> Note:

>
- The particles in the beam considered may have a momentum deviation given by DELTAP defined in the TRACK command.
- The opposite beam is assumed to have the velocity according to the unperturbed energy o the particles in the beam considered. So, only the direction of motion can be chosen.

- In the case of motion in the opposite direction (BBDIR=-1), the time of interaction between the beams is given by tau = length

(2*beta*c_light), where length is the length of a bunch in the opposite beam. In the case of motion in the same direction (BBDIR=1) as in an electron cooler, this time is given by tau = length/(beta*c_light), where length is the length of the cooler. So, the factor 1/2 is inserted only for BBDIR=-1 to calculate correct results.

A beam-beam element requires the particle energy (ENERGY) and the particle charge (CHARGE) as well as the number of particles per bunch (NPART) to be set by a BEAM command before any calculations are performed.

Examples of a four-dimensional beam-beam element definition:

Collider regime example:

```
beam,   particle=positron,npart=1.e12,energy=50.0;
bb:     beambeam,sigx=1.e-3,sigy=5.e-4,charge=1.;
```

Electron cooler example:

```
gamma0=1.032;                           ! relativistic factors
beta0=sqrt(1.0-1.0/gamm0/gamma0);

i_e=0.2;                                ! electron current
re_cool=0.01;                           ! electron beam radius
l_cool=5.0;                             ! cooling length
nelect=i_e*l_cool/beta0/clight/qelect;  ! electron number in e-cooler

beam,particle=antiproton,gamma=gamma0,npart=nelect;
bb_ecool:beambeam,sigx=re_cool,sigy=re_cool,bbshape=2,width=0.01,charge=-1,bbdir=1;
```

For the definition of the LHC head-on and parasitic beam-beam elements see beam-beam element examples.

hansg, ssorge, July 13, 2007

MATRIX: Arbitrary Element

```
label: MATRIX,TYPE=name,L=real,KICK1=real,...,KICK6=real,
               RM11=real,...,RM66=real,
               TM111=real,...,TM666=real;
```

The MATRIX permits the definition of an arbitrary transfer matrix. It has four real array attributes:

- L: Length of the element, which may be zero.
- KICKi: Defines the kick of the element acting on the six phase space coordinates.
- RMik: Defines the linear transfer matrix (6*6) of the element.
- TMikl: Defines the second-order terms (6*6*6) of the element.

Data values not entered are taken from the identity transformation, kick and second order terms are zero as default. In the thin-lens tracking module the length of an arbitrary matrix is accepted, however no second order are allowed to avoid non symplectic tracking runs. In the latter case the tracking run will be aborted.

frs, June 25, 2003

Editing Element Definitions

An element definition can be changed in two ways:

- **Entering a new definition:** The element will be replaced in the main beam line expansion.
- **Entering the element name together with new attributes:** The element will be updated in place, and the new attribute values will replace the old ones.

This example shows two ways to change the strength of a quadrupole:

```
QF: QUADRUPOLE,L=1,K1=0.01;     ! Original definition of QF
QF: QUADRUPOLE,L=1,K1=0.02;     ! Replace whole definition of QF
QF,K1=0.02;                     ! Replace value of K1
```

When the type of the element remains the same, replacement of an attribute is the more efficient way.

Element definitions can be edited freely. The changes do not affect already defined objects which belong to its element class.

hansg, January 24, 1997

Element Classes

The concept of element classes solves the problem of addressing instances of elements in the accelerator in a convenient manner. It will first be explained by an example. All the quadrupoles in the accelerator form a class QUADRUPOLE. Let us define three subclasses for the focussing quadrupoles, the defocussing quadrupoles, and the skewed quadrupoles:

```
MQF: QUADRUPOLE,L=LQM,K1=KQD;     ! Focussing quadrupoles
MQD: QUADRUPOLE,L=LQM,K1=KQF;     ! Defocussing quadrupoles
MQT: QUADRUPOLE,L=LQT;            ! Skewed quadrupoles
```

These classes can be thought of as new keywords which define elements with specified default attributes. We now use theses classes to define the real quadrupoles:

```
QD1: MQD;           ! Defocussing quadrupoles
QD2: MQD;
QD3: MQD;
 ...
QF1: MQF;           ! Focussing quadrupoles
QF2: MQF;
QF3: MQF;
 ...
QT1: MQT,K1S=KQT1;   ! Skewed quadrupoles
QT2: MQT,K1S=KQT2;
 ...
```

These quadrupoles inherit all unspecified attributes from their class. This allows to build up a hierarchy of objects with a rather economic input structure.

The full power of the class concept is revealed when object classes are used to select instances of elements for various purposes. Example:

```
select,flag=twiss,class=QUADRUPOLE; ! Select all quadrupoles for the
                                    ! Twiss TFS file
```

More formally, for each element keyword MAD maintains a class of elements with the same name. A defined element becomes itself a class which can be used to define new objects, which will become members of this class. A new object inherits all attributes from its class; but its definition may override some of those values by new ones. All class and object names can be used in range selections, providing a powerful mechanism to specify positions for matching constraints and printing.

hansg, January 24, 1997

EUROPEAN ORGANIZATION FOR NUCLEAR RESEARCH 

Beam Line Sequences

MAD-X accepts two forms of an accelerator definition: sequences and lines. However, the sequence definition is the only one used internally; lines are converted into sequences when they are USEd. Consequently, only sequences can be saved (written onto a file) by MAD-X.

The corresponding sequence of statements defining a sequence is

```
name: SEQUENCE,REFER=keyword,REFPOS=name,LENGTH=real
label: class,AT=real{,attributes} | class,AT=real | sequ_name, AT=real
    ...
ENDSEQUENCE
```

where "real" means a real number, variable, or expression.

The first line gives the sequence name, a REFER flag (entry, centre, or exit) which specifies at which part of the element its position along the beam line will be given (default: centre), a REFPOS argument used for sequence insertion, and the total length.

Inside the sequence ... endsequence bracket three types of commands may be placed:

- an element declaration as usual, with an additional "at" attribute giving the element position relative to the start of the sequence; CAUTION: an existing definition for an element with the same name will be replaced, however, defining the same element twice inside the same sequence results in a fatal error, since a unique object (this element) would be placed at two different positions.
- a class name with a position; this causes an instance of the class to be placed at the position given. For uses inside ranges, instances of the same class can be accessed with an occurrence count.
- a sequence name with a position; this causes the sequence with that name to be placed at the position indicated. The entry, centre, or exit of the inserted sequence are placed at the position given, UNLESS a "refpos" (the name of an element in the inserted sequence) is given, in which case the sequence is inserted such that the refpos element is at the insertion point.

When the sequence is expanded in a USE command, MAD generates the missing drift spaces. At this moment, overlapping elements will cause "negative drift length" errors.

For efficiency reasons MAD-X imposes an **important restriction** on element lengths and positions: once a sequence is expanded, the element positions and lengths are considered as fixed; in order to vary a position or element length, a re-expansion of the sequence becomes necessary. The MATCH command contains a special flag "vlength" to match element lengths.

Example:

```
! define a default beam (otherwise fatal error)
beam;
! Define element classes for a simple cell:
b:     sbend,l=35.09, angle = 0.011306116;
qf:    quadrupole,l=1.6,k1=-0.02268553;
qd:    quadrupole,l=1.6,k1=0.022683642;
sf:    sextupole,l=0.4,k2=-0.13129;
sd:    sextupole,l=0.76,k2=0.26328;
! define the cell as a sequence:
sequ:  sequence,l=79;
   b1:    b,      at=19.115;
   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;
```

hansg, June 17, 2002

Beam Lines

The accelerator to be studied is known to MAD-X either as a sequence of physical elements called sequence, or as a hierarchically structured list of elements called a *beam line*. A beam line is built from simpler beam lines whose definitions can be nested to any level. A powerful syntax allows to repeat, to reflect, or to replace pieces of beam lines. However, internally MAD-X knows only sequences, and lines as shown below are converted into flat sequences with the same name when they are expanded. Consequently, only sequences can be SAVEd onto a file (see save).

Formally a beam line is defined by a LINE command:

```
label(arg{,arg}): LINE=(member{,member});
```

Label gives a name to the beam line for later reference.

The formal argument list (arg{,arg}) is optional (see below). Each "member" may be one of the following:

- Element label,
- Beam line label,
- Sub-line, enclosed in parentheses,
- Formal argument name,
- Replacement list label.

Beam lines may be nested to any level.

Simple Beam Lines

The simplest beam line consists of single elements:

```
label: LINE=(member{,member});
```

Example:

```
l:      line=(a,b,c,d,a,d);
        use,period=l;
```

The USE command tells MAD to perform all subsequent calculations on the sequence

```
a,b,c,d,a,d
```

Sub-lines

Instead of referring to an element, a beam line member can refer to another beam line defined in a separate command. This provides a shorthand notation for sub-lines which occur several times in a beam line. Lines and sub-lines can be entered in any order, but when a line is expanded, all its sub-lines must be known.

Example:

```
l:      line=(a,b,s,b,a,s,a,b);
s:      line=(c,d,e);
        use,period=l;
```

this example produces the following expansion steps:

1. replace sub-line s:

```
(a,b,(c,d,e),b,a,(c,d,e),a,b)
```

2. omit parentheses:

```
a,b,c,d,e,b,a,c,d,e,a,b
```

Reflection and Repetition

An unsigned repetition count and an asterisk indicate repetition of a beam line member. A minus prefix causes reflection, i.e. all elements in the subsequence are taken in reverse order. Sub-lines of reflected lines are also reflected, but physical elements are not. If both reflection and repetition are desired, the minus sign must precede the repetition count.

Example:

```
r:      line=(g,h);
s:      line=(c,r,d);
t:      line=(2*s,2*(e,f),-s,-(a,b));
        use,period=t;
Attention: the repetition "2*s" will only work if
"s" is itself a line. In case "s" is an element replace by
"2*(s)".
```

Proceeding step by step, this example produces

1. Replace sub-line S:

```
((c,r,d),(c,r,d),(e,f),(e,f),(d,-r,c),(b,a))
```

2. replace sub-line r:

```
((c,(g,h),d),(c,(g,h),d),(e,f),(e,f),(d,(h,g),c),(b,a))
```

3. omit parentheses:

```
c,g,h,d,c,g,h,d,e,f,e,f,d,h,g,c,b,a
```

Note that the inner sub-line R is reflected together with the outer sub-line S.

Replaceable Arguments

A beam line definition may contain a formal argument list, consisting of labels separated by commas and enclosed in parentheses. Such a line can be expanded for different values of its arguments. When this line is referred to, its label must be followed by a list of actual arguments separated by commas and enclosed in parentheses. These arguments must be beam line, or element names. The number of actual arguments must agree with the number of formal arguments. All occurrences of a formal argument on the right-hand side of the line definition are replaced by the corresponding actual

argument.

Example:

```
s:        line=(a,b,c);
l(x,y):   line=(d,x,e,3*y);
l4f:      line=(4*f);
lm2s:     line=(-2*s);
res:      line=l(l4f,lm2s);
```

Proceeding step by step, this example generates the expansion

```
d,f,f,f,f,e,c,b,a,c,b,a,c,b,a,c,b,a,c,b,a,c,b,a
```

Second example:

```
cel(sf,sd):     line=(qf,d,sf,d,b,d,qd,d,sd,d,b,d);
arc:            line=(cel(sf1,sd1),cel(sf2,sd2),cel(sf1,sd1));
                use,period=arc;
```

This example generates the expansion

1. Replace the line CEL and its formal arguments:

```
((qf,d,(sf1),d,b,d,qd,d,(sd1),d,b,d)
 (qf,d,(sf2),d,b,d,qd,d,(sd2),d,b,d)
 (qf,d,(sf1),d,b,d,qd,d,(sd1),d,b,d))
```

2. Omit parentheses:

```
qf,d,sf1,d,b,d,qd,d,sd1,d,b,d
qf,d,sf2,d,b,d,qd,d,sd2,d,b,d
qf,d,sf1,d,b,d,qd,d,sd1,d,b,d
```

Warning: Line Depreciation

MADX has been devolopped using sequences, in fact internally the code works with sequences only. Consequently, there may exist some inconveniences when only lines are used. It is recommended to convert as soon as possible lines into sequences (by means of the save command) in a design phase and to use only sequences for a finalised machine.

Limits of Construction of Lines

Since Lines are in fact depreciated there are some limits of how they can be constructed. Please find below a running MADX run which shows an example of OK (valid) and WRONG (invalid) cases.

```
!-------------------------------------------------------------------
beam, PARTICLE=electron, energy=1;

qf: QUADRUPOLE, L:=1,K1:=1;
qd: QUADRUPOLE, L:=1,K1:=-1;
d: DRIFT, l=1;
m: MARKER;

rpl(a,b): LINE=(a,b);
sl: LINE=(qf,d,qd);
test0: LINE=(rpl(sl,sl));          !OK
test1: LINE=(rpl((sl),(sl)));      !OK
```

```
test2: LINE=(rpl((sl),(-sl)));     !OK
test3: LINE=(sl,-sl);              !OK
test4: LINE=(rpl((3*sl),(3*sl)));  ! WRONG
test5: LINE=(3*sl,3*sl);           !OK
test6: LINE=(rpl((3*sl),(-3*sl))); ! WRONG
test7: LINE=(3*sl,-3*sl);          !OK

use, period=test0;
twiss,BETX=1,bety=1;

use, period=test1;
twiss,BETX=1,bety=1;

use, period=test2;
twiss,BETX=1,bety=1;

use, period=test3;
twiss,BETX=1,bety=1;

use, period=test4;
twiss,BETX=1,bety=1;

use, period=test5;
twiss,BETX=1,bety=1;

use, period=test6;
twiss,BETX=1,bety=1;

use, period=test7;
twiss,BETX=1,bety=1;
!---------------------------------------------------------------------
```

hansg, June 17, 2002

EUROPEAN ORGANIZATION FOR NUCLEAR RESEARCH 

Defining aperture in MAD-X

A new feature of MAD-X is the ability to set an aperture for a particular element, or parent of a set of elements. This removes the need of placing a collimator next to every element to do aperture tracking. The aperture of any elements can be specified (excepts drifts) by the use of the following parameters:

- `APERTYPE` This can have seven text values: CIRCLE, RECTANGLE, ELLIPSE, LHCSCREEN (a superposition of a CIRCLE and a RECTANGLE), MARGUERITE (two LHCSCREENS, one rotated by 90 degrees), RECTELLIPSE (a superposition of an ELLIPSE and a RECTANGLE) and RACETRACK.
- `APERTURE` This is an array of values, the number and meaning of which depends on the APERTYPE:

APERTYPE	# of parameters	meaning of parameters
CIRCLE	1	radius
ELLIPSE	2	horizontal half axis, vertical half axis
RECTANGLE	2	half width and half height
LHCSCREEN	3	half width, half height (of rect.) and radius (of circ.)
MARGUERITE	3	half width, half height (of rect.) and radius (of circ.)
RECTELLIPSE	4	half width, half height (of rectangle), horizontal half axis, vertical half axis (of ellipse)
RACETRACK	3	radius, horizontal shift, vertical shift
FILENAME	0	where the file contains a list of x and y coordinates outlining the shape. This option is only supported by the aperture module, see below.

Here is an example for setting an ELLIPTICAL aperture for the main dipoles for the LHC.

```
MB : SBEND, L := l.MB, APERTYPE=ELLIPSE, APERTURE={0.02202,0.02202};
```

And an example for setting a FILENAME aperture for another magnet. Notice that no aperture parameters are needed.

```
MB: SBEND, L := 5, APERTYPE=myfile;
```

The syntax of myfile should be like this:

```
x0   y0
xi   yi
...
xn   yn
```

Notes concerning the use of aperture:

- There is some inconsistency in the parameter definition for the different APERTYPE. This is historical and has to be kept for backwards compatibility. Pay some attention to the parameters you introduce!
- When MAKETHIN is called all the thin slices inherit the aperture from their original thick lens version.
- When the SIXTRACK command is called (see the SixTrack converter module C6T) the apertures are ignored by default. To convert the apertures as well the APERTURE flag has to be set.
- Aperture parameters are like all parameters and are inherited by offspring. Like other parameters they can also be overridden by the offspring elements if necessary.

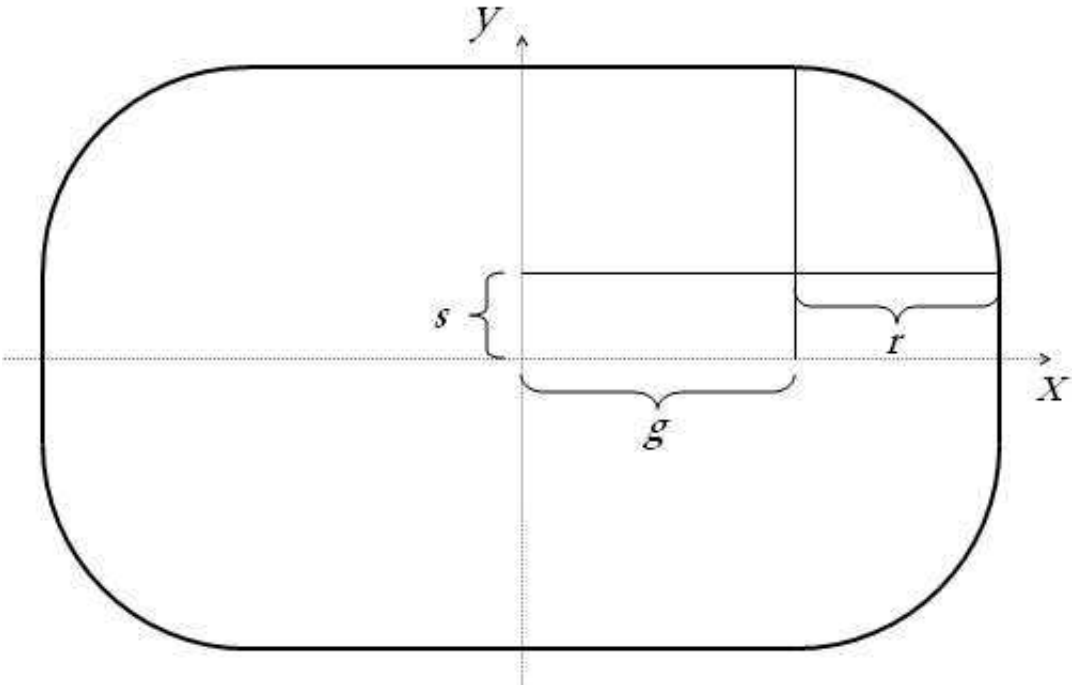
The APERTYPE and the APERTUREs themselves can be conveniently added to the TWISS table (see Twiss Module) by using the SELECT command. E.G. the command:

```
select,flag=twiss,clear;
select,flag=twiss,column=name,s,betx,alfx,mux,bety,alfy,muy,apertype,aper_1,aper_2;
```

and a subsequent TWISS command will put the aperture information together with the specified TWISS parameters into the TWISS table.

Defining tolerances in MAD-X

A parameter closely connected to the aperture is the sum of the mechanical and alignment tolerances. The mechanical tolerance is the maximal error margin of errors in the element body which causes a decrease of aperture, and the alignment tolerance is a mislignment of the element in the accelerator, which also causes a decrease of aperture. The tolerance is given in the transverse plane as a racetrack, like in the picture below.



A tolerance can be assigned to each element in a MAD-X sequence as a vector:

```
Syntax: APER_TOL = {r, g, s};

MB : SBEND, L := l.MB, APER_TOL={1.5, 1.1, 0};
```

APERTURE MODULE

Computes the n1 values for a piece of machine. Each element is sliced into thick subelements at given intervals, and the available aperture is computed at the end of each slice. The computation is based on the last Twiss table, so it is important to run the Twiss and aperture commands on the same period or sequence, see the aperture example below. Also showed in the example is how n1 values can be plotted.

The minimum n1 for each element is written to the last Twiss table, to allow for matching by aperture.

- ### Aperture,

```
file=filename,
halofile=filename,
pipefile=filename,
range=range,
exn=real,
eyn=real,
dqf=real,
betaqfx=real,
dp=real,
dparx=real,
dpary=real,
cor=r,
bbeat=real,
nco=integer,
```

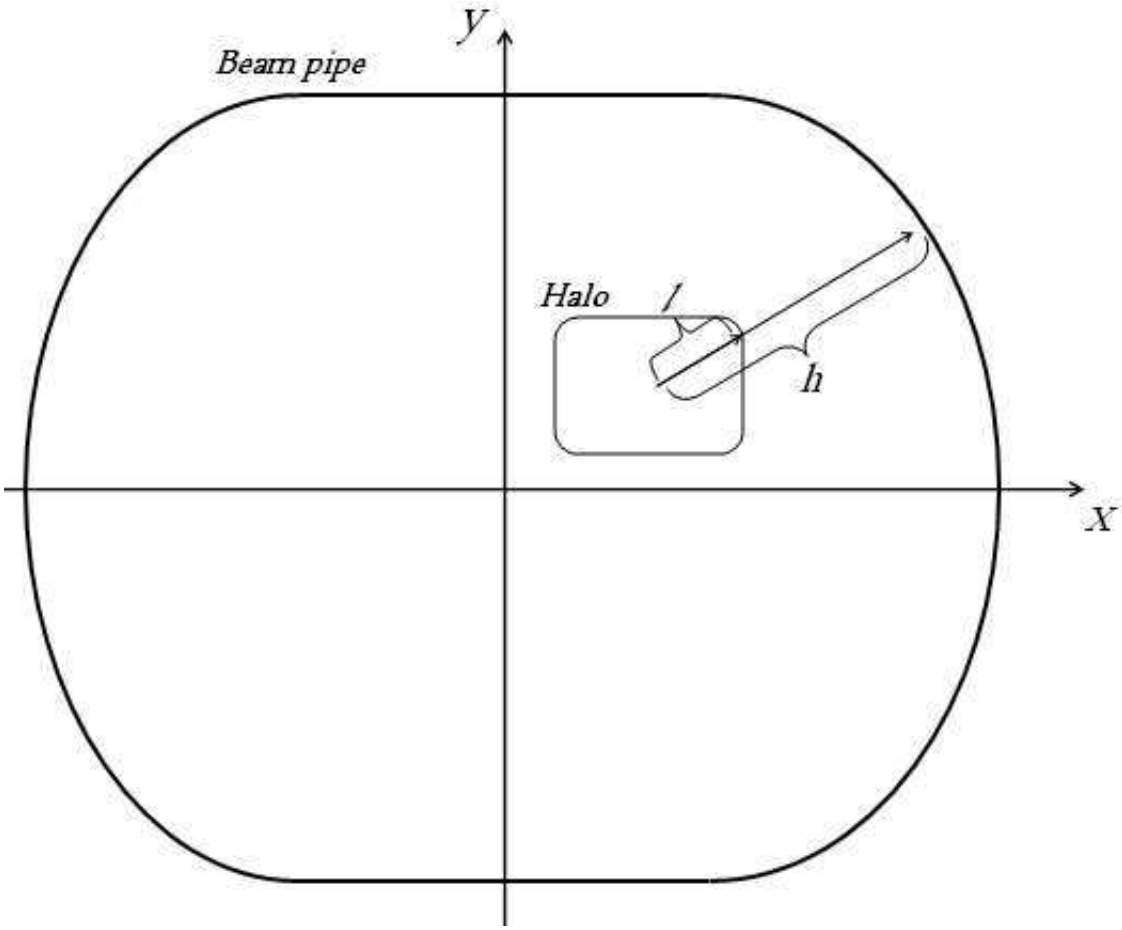
```
halo={real,real,real,real},
interval=real
spec=real,
notsimple=logical,
trueprofile=filename,
offsetelem=filename;
```

where the parameters have the following meaning:

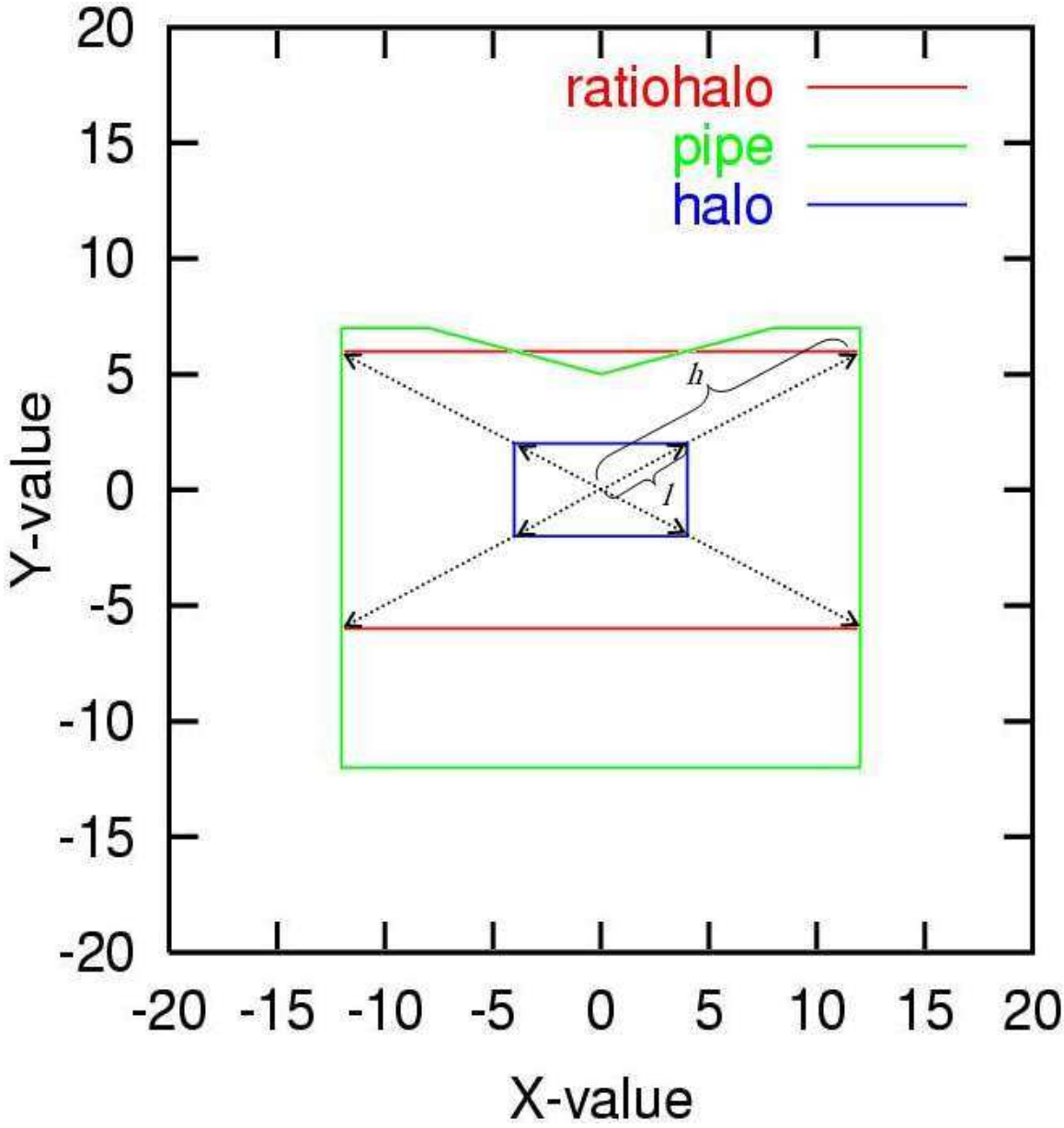
- file: Output file with aperture table. Default = none
- halofile: Input file with halo polygon coordinates. Will suppress an eventual halo parameter. Default = none
- range: Range given by elements. Default = #s/#e
- exn: Normalised horizontal emittance. Default = 3.75*e-6
- eyn: Normalised vertical emittance. Default = 3.75*e-6
- dqf: Peak linear dispersion [m]. Default = 2.086
- betaqfx: Beta x in standard qf [m]. Default = 170.25
- dp: Bucket edge at the current beam energy. Default = 0.0015
- dparx: Fractional horizontal parasitic dispersion. Default = 0.273
- dpary: Fractional vertical parasitic dispersion. Default = 0.273
- cor: Maximum radial closed orbit uncertainty [m]. Default = 0.004
- bbeat: Beta beating coefficient applying to beam size. Default = 1.1
- nco: Number of azimuth for radial scan. Default = 5
- halo: Halo parameters: {n, r, h, v}. n is the radius of the primary halo, r is the radial part of the secondary halo, h and v is the horizontal and vertical cuts in the secondary halo. Default = {6, 8.4, 7.3, 7.3}
- interval: Approximate length in meters between measurements. Actual value: nslice = nodelength/interval, nslice is rounded down to closest integer, interval = nodelength/nslice. Default = 1.0
- spec: Aperture spec, for plotting only. Gives the spec line in the plot. Default = 0.0
- notsimple: Use only if one or more beamscreens in the range are considered not to be a "simply connex". Since all MAD-X apertypes are simply connex, this is only possible if an input file with beam screen coordinates are given. See below for a graphical example. Default = false.
- trueprofile: A file containing a list of magnets, and for each magnet a list of horizontal and vertical deviations from the ideal magnet axis. These values may come from measurements done on the magnet. See below for example. Default = none.
- offsetelem: A file containing a reference point in the machine, and a list of magnets with their offsets from this point described as a parabola. See below for example. Default = none.

Not simply connex beam pipes

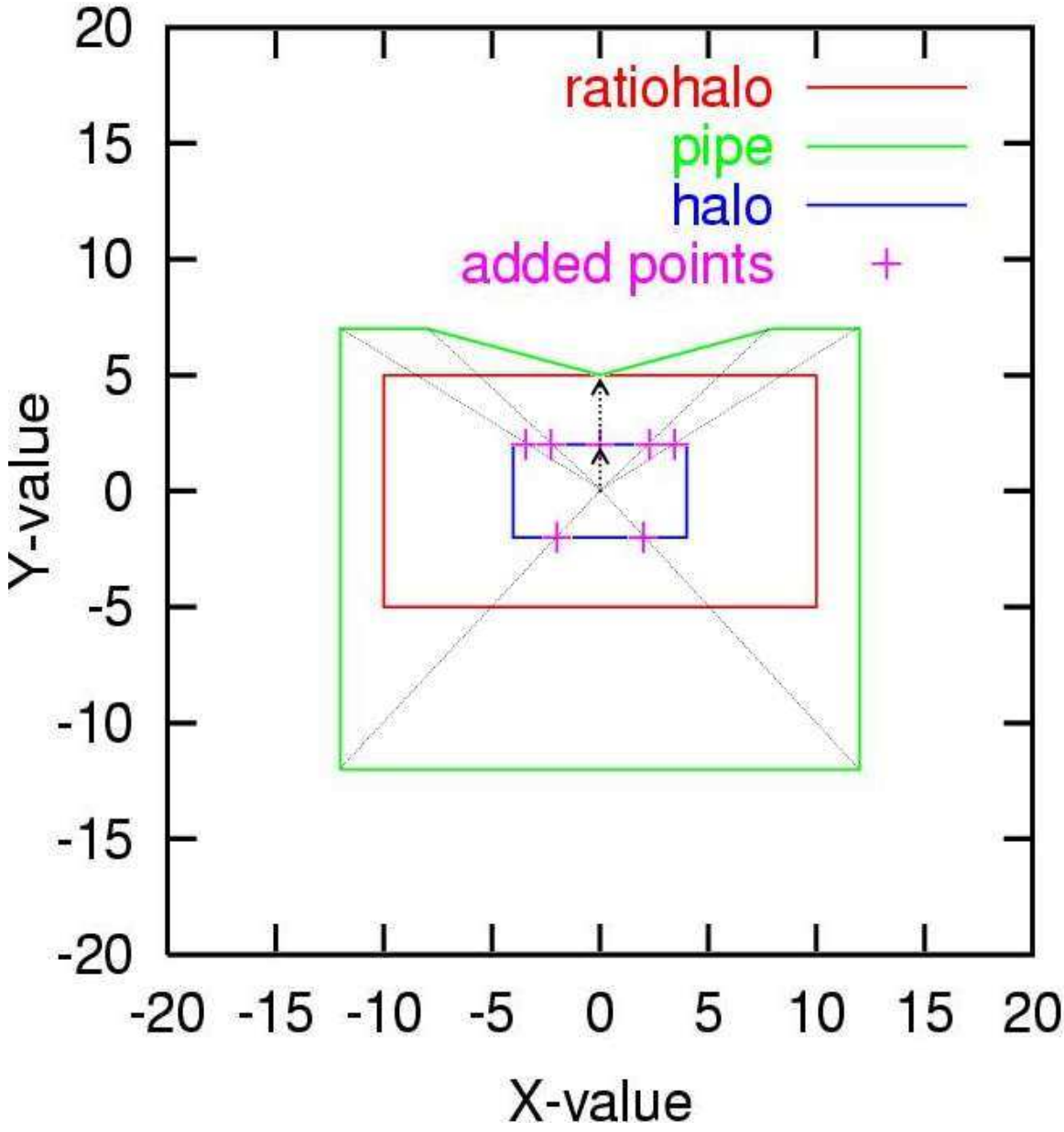
Methodically, the algorithm for finding the largest possible halo is fairly simple. The distance from halo centre to the first apex (i = 0) in the halo is calculated (l_i), and the equation for a line going through these points is derived. This line is then compared with all lines making the pipe polygon to find their respective intersection coordinates. The distance h_i between halo centre and intersection are then divided by l_i, to find the maximal ratio of enlargement, as seen below. This procedure is then repeated for all apexes i in the halo polygon, and the smallest ratio of all apexes is the maximal enlargement ratio for this halo to just touch the pipe at this particular longitudinal position.



There is one complication to this solution; polygons which are not simple connexes. (Geometrical definition of ‘‘simply connex’’: A figure in which any two points can be connected by a line segment, with all points on the segment inside the figure.) The figure below shows what happens when a beam pipe polygon is not a simple connex. The halo is expanded in such a way that it overlaps the external polygon in the area where the latter is dented inwards.



To make the module able to treat all kinds of polygons, *notsimple* must be activated. With this option activated, apexes are strategically added to the halo polygon wherever the beam pipe polygon might have an inward dent. This is done by drawing a line from halo centre to each apex on the pipe polygon. An apex with its coordinates on the intersection point line-halo is added to a table of halo polygon apexes. The result is that the halo polygon has a few ‘‘excessive’’ points on straight sections, but the algorithm used for expansion will now never miss a dent in the beam pipe. The use of the notsimple option significantly increases computation time.



Trueprofile file syntax

This file contains magnet names, and their associated displacements of the axis for an arbitrary number of S, where So is the start of the magnet and Sn the end. The interval between each S must be regular, and X and Y must be given in meters. The magnet name must be identical to how it appears in the sequence. The displacements are only valid for this particular magnet, and cannot be assigned to a family of magnets. n1 is calculated for a number of slices determinated by the number of Si.

Layout of file:

```
magnet.name1
So    X    Y
Si    X    Y
Si    X    Y
Sn    X    Y
```

```
magnet.name2
So   X   Y
Si   X   Y
Si   X   Y
Sn   X   Y

etc.
```

Example of file:

```
!This is the start of the file.
!Comments are made with exclamation marks.

mb.a14r1.b1
0        0.0002        0.000004
7.15     1.4e-5        0.3e-3
14.3     0.0000000032  4e-6

!further comments can of course be added

mq.22r1.b1
0      0.3e-5     1.332e-4
1.033  0.00034    0.3e-9
2.066  0          0.00e-2
3.1    4.232e-4   0.00000003

!This is the end of the file.
```

Offsetelem file syntax

This file contains coordinates describing how certain elements are displaced w.r.t. a given reference point in the machine. It might be used with elements in insertions, or other special-purpose elements that has a magnet axis which does not coincide with the reference trajectory. We operate with two coordinate system, s,x and s,y, where the reference point is the origin and the actual element axis is described as a parabola with coefficients A, B and C. For each element we give two sets of coefficients, one for horizontal displacement and one for vertical:

```
X_offs(s) = Ax*s^2 + Bx*s + Cx
```

and

```
Y_offs(s) = Ay*s^2 + By*s + Cy
```

. The coordinate systems are in meters.

Layout of file:

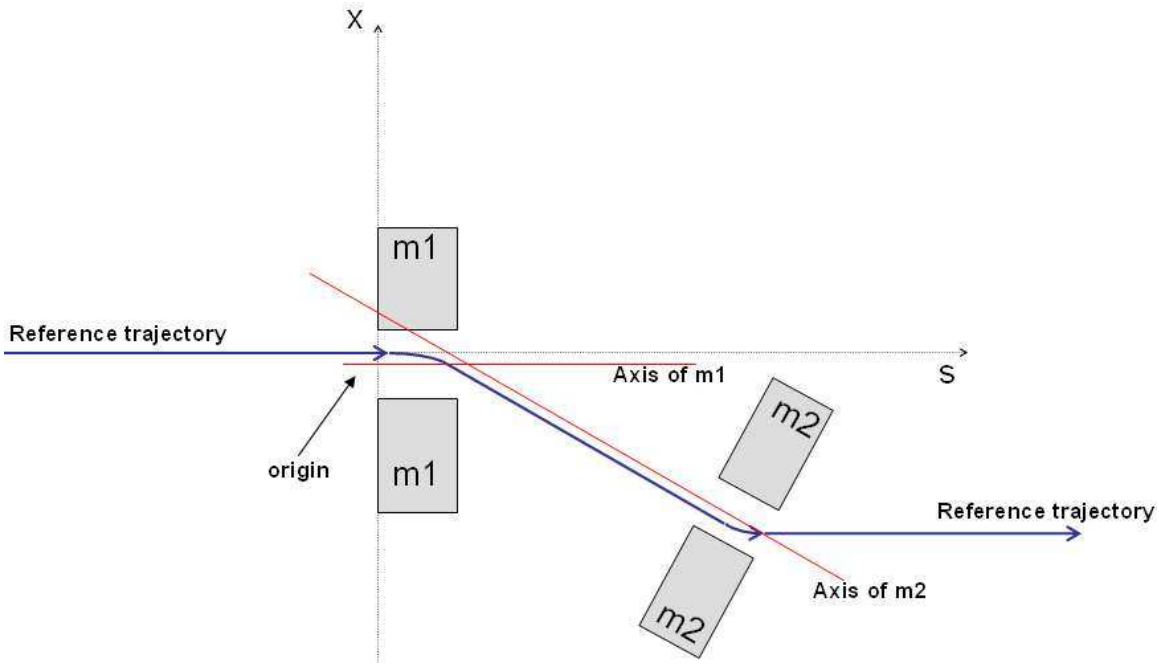
```
reference.point

magnet.name1
Ax   Bx   Cx
Ay   By   Cy

magnet.name2
```

```
Ax   Bx   Cx
Ay   By   Cy

etc.
```

Example of file:

```
!This is the start of the file.
!First we give a reference point. The origin of the
!coordinate system will be at the START of this element.

mq.12r1.b1

!Then we give a list of elements and their displacement
!w.r.t. the reference point.

mcbxa.3l2
0   -2.56545   -3
0   -2.3443666  0

!The next nodes use the same reference point.
!Elements offset w.r.t. another point must be given in another file,
!together with the new reference point.

mcbxa.3r2
0.3323  32.443355 -0.84
0.2522  32.554363 0.0

!This is the end of the file.
```

As an example we see in the picture below how the horizontal axes of elements m1 and m2 does not coincide with the reference trajectory.



The X_ref(s) and Y_ref(s) of the reference trajectory are calculated via an internal call to the Survey module. X_offs(s) and Y_offs(s) are derived from the coefficients given in the file. The resulting

```
X_tot(s) = X_ref(s) - X_offs(s)
```

and

```
Y_tot(s) = Y_ref(s) - Y_offs(s)
```

are taken into account in the aperture calculations.

Aperture command example

The aperture module needs a Twiss table to operate on. It is important not to USE another period or sequence between the Twiss and aperture module calls, else aperture looses its table. One can choose the ranges for Twiss and aperture freely, they need not be the same.

```
use, period=lhcb1;
select, flag=twiss,range=mb.a14r1.b1/mb.a17r1.b1,column=keyword,name,parent,k0l,k1l,s,betx,bety,n1;
twiss, file=twiss.b1.data, betx=beta.ip1, bety=beta.ip1, x=+x.ip1, y=+y.ip1, py=+py.ip1;
plot,haxis=s,vaxis=betx,bety,colour=100;

select, flag=aperture, column=name,n1,x,dy;
aperture, range=mb.b14r1.b1/mb.a17r1.b1, spec=5.235;
plot,table=aperture,noline,vmin=0,vmax=10,haxis=s,vaxis=n1,spec,on_elem,style=100;
```

The select command can be used to choose which columns to print in the output file.
Column names: name, n1, n1x_m, n1y_m, apertype, aper_1, aper_2, aper_3, aper_4, rtol, xtol, ytol, s, betx, bety, dx, dy, x, y, on_ap, on_elem, spec

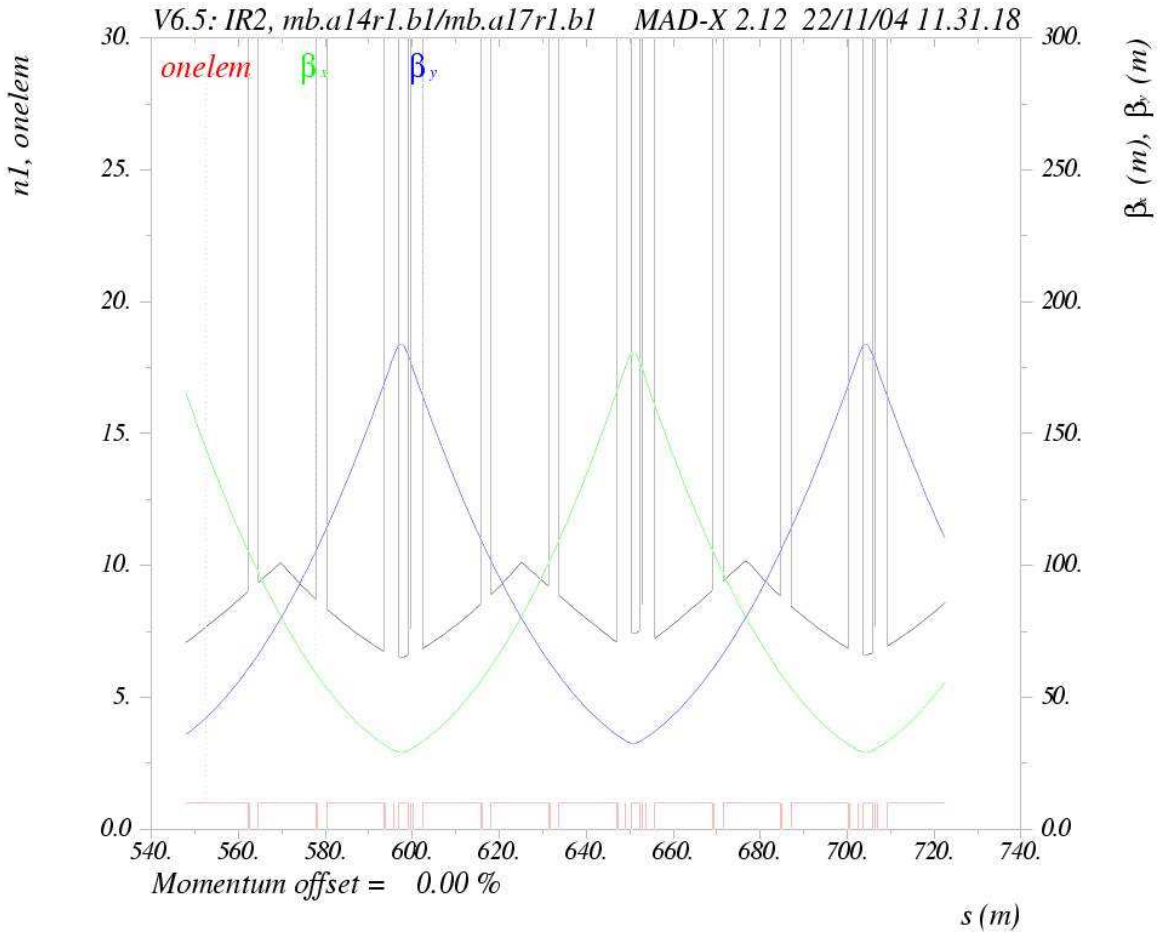
n1 is the maximum beam size in sigma, while n1x_m and n1y_m is the n1 values in si-units in the x- and y-direction.

aper_# means for all apertypes but racetrack:
aper_1 = half width rectangle
aper_2 = half heigth rectangle
aper_3 = half horizontal axis ellipse (or radius if circle)
aper_4 = half vertical axis ellipse

For racetrack, the aperture parameters will have the same meaning as the tolerances:
aper_1 and xtol = horizontal displacement of radial part
aper_2 and ytol = vertical displacement of radial part
aper_3 and rtol = radius
aper_4 = not used

On_elem indicates whether the node is an element or a drift, and on_ap whether it has a valid aperture. The Twiss parameters are the interpolated values used for aperture computation.

When one wants to plot the aperture, the table=aperture parameter is necessary. The normal line of hardware symbols along the top is not compatible with the aperture table, so it is best to include noline. Plot instead the column on_elem along the vaxis to have a simple picture of the hardware. Spec can be used for giving a limit value for n1, to have something to compare with on the plot. This example provides a plot,



where we see the n1, beta functions and the hardware symbolized by on_elem.

Ivar Waarum, 24.02.05 - Mark Hayes, 19.06.02

EUROPEAN ORGANIZATION FOR NUCLEAR RESEARCH 

SixTrack: Produce input files for tracking in SixTrack

In dynamic aperture studies [SixTrack] is often used because of its speed and controllability. However the input files are notoriously difficult to produce by hand. This command may be used to produce SixTrack files from a sequence in MAD-X's memory.

N.B.: The files contain all information concerning optics, field errors and misalignments. Hence these should all be set and a

```
TWISS,SAVE;
```

command should always be issued beforehand.

The generation of the SixTrack input files is then done by the command:

```
SIXTRACK, CAVALL,
          MULT_AUTO_OFF,
          MAX_MULT_ORD,
          SPLIT,
          APERTURE,
          RADIUS = ref. radius of magnets;
```

The parameters are defined as:

- CAVALL - (optional flag) This puts a cavity element (SixTrack identifier 12) with Volt, Harmonic Number and Lag attributes at each location in the machine. Since for large hadron machines the cavities are typically all located at one particular spot in the machine and since many cavities slow down the tracking simulations considerably all cavities are lumped into one and located at the first appearance of a cavity. This default is enforced by omitting this flag.
- MULT_AUTO_OFF - (optional flag) If .TRUE. (DEFAULT) code does not process zero value multipoles; if .FALSE. process up to order *MAX_MULT_ORD*.
- MAX_MULT_ORD - (optional parameter) Process up to this order for *mult_auto_off* = *.FALSE.*.
- SPLIT - (optional flag) OBSOLETE. This splits all the elements in two. This is for backwards compatibilty only. The user should now use the MAKETHIN command instead.
- APERTURE - (optional flag) Set this to convert the apertures from MAD-X to SixTrack, so SixTrack will track with aperture.
- RADIUS - (optional, default value is 1m). This sets the reference radius for the magnets. This argument is optional but should normally be set.
- *Note: the bv flag is presently ignored*
- *WARNING: SixTrack and c6t are presently set up for names of a maximum of 16 characters!!!!! Therefore, it is mandatory to respect this limit for MAD-X names.*

The command will then always produce the following file:

- fc.2 - contains the basic structure of the lattice.

and may produce any or all of the following files, depending on the sequence:

- fc.3 - contains the multipole mask(s).
- fc.3.aux - contains various beam parameters.
- fc.8 - contains the misalignments and tilts.
- fc.16 - contains the field errors and/or combined multipole kicks.
- fc.34 - file with various optics parameters at various locations (not needed by SixTrack but may be used as input to [SODD].)

For a full description of these files see [SixTrack] and for information on running SixTrack see [Run Environment].

Mark Hayes 20.06.02

EUROPEAN ORGANIZATION FOR NUCLEAR RESEARCH 

MAKETHIN: Slice a sequence into thin lenses

This module converts a sequence with thick elements into one composed entirely of thin elemtens as required by the default MAD-X tracking.

Slicing is done by the MAKETHIN command:

```
MAKETHIN,SEQUENCE=sequence name,
         STYLE=slicing style;
```

The parameters are defined as:

- SEQUENCE chooses the sequence you wish to slice.
- STYLE (optional) chooses the slicing style. The options are:
  - SIMPLE : this is a simplified slicing algorithm which produces any number of equal strength slices at equidistant positions with the kick in the middle of each slice.
  - COLLIM : this is the default slicing for collimators. If only one slice is chosen it is placed in the middle of the old element. If two slices are chosen they are placed at either end. Three slices or more are treated as one slice.
  - [TEAPOT] (default): this is the standard slicing as used by MAD9. N.B. This has a maximum of four slices for any one object.

By default all elements are converted to one thin element positioned at the center of the thick element. To get a greater slicing for certain elements use a standard SELECT command with FLAG=MAKETHIN and CLASS, RANGE or PATTERN:

```
SELECT,FLAG=MAKETHIN,
       CLASS=class,RANGE=range,
       SLICE=no of slices;
```

The created thin lens sequence has the following properties:

- The created sequence has the same name as the original. The original is therefore no longer available and has to be reloaded if it is needed again.
- The slicer also slices any inserted sequence used in the main sequence. These are also given the same names as the originals.
- Any component changed into a single thin lens has the same name as the original.
- If a component is sliced into more than one slice, the individual slices have the same name as the original component and a suffix `..1`, `..2`, etc... and a marker will be placed at the center with the original name of the component.

Hints:

- See the examples for makethin.
- Compare the optics before and after slicing with makethin. Consider to increase the number of slices and rematch after makethin to reach the required accuracy.

- Consider to replace rbend by sbend + thin quads taking into account the edge focusing before slicing with makethin.
- The selection works on the current sequence. Consider to insert a "USE,SEQUENCE=.." before SELECT

Helmut Burkhardt, September 2005

EUROPEAN ORGANIZATION FOR NUCLEAR RESEARCH 

DYNAP: Tunes, Tune Footprints, Smear and Lyapunov Exponent

DYNAP can be called instead of RUN inside a TRACK command:

```
DYNAP,TURNS=real, FASTUNE=logical,LYAPUNOV=real,MAXAPER:={..,..,..,..,..,..},ORBIT=logical;
```

For each previously entered start command, DYNAP tracks two close-by particles over a selected number of turns, from which it obtains the betatron tunes with error, the action smear, and an estimate of the lyapunov exponent. Many such companion particle-pairs can be tracked at the same time, which speeds up the calculation. The *smear* is defined as $2.0\,(wxy_{max} - wxy_{min})\,/\,(wxy_{max} + wxy_{min})$, where the $wxy_{min,max}$ refer to the miminum and maximum values of the sum of the transverse betatron invariants *wx+wy* during the tracking. The tunes are computed by using an FFT and either formula (18) or formula (25) of CERN SL/95-84 (AP), depending on whether the number of turns is less-equal or larger than 64.

The arguments have the following meaning:

- TURNS: The number of turns to be tracked (default: 64, present maximum: 1024).
- FASTUNE: A logical flag. If set, the tunes are computed (default: false).
- MAXAPER: If the particle phase-space coordinates exceed certain *maximum* values, the particle is considered lost. The maximum aperture is a vector of 6 real numbers (default: (0.1, 0.01, 0.1,0.01, 1.0, 0.1)).
- LYAPUNOV: The launch distance between two companion particles added to the *x* coordinate (default: 1.e-7 m).
- ORBIT: A logical flag. If set, the flag *orbit* is true during the tracking and its initialization (default: true). **This flag should be set to be true, if normalized coordinates are to be entered.**

Example:

```
BEAM,PARTICLE=ELECTRON,ENERGY=50,EX=1.E-6,EY=1.E-8,ET=0.002,SIGT=1.E-2;
...
USE,PERIOD=FODO;
...
TRACK;
START,X=0.0010,Y=0.0017,PT=0.0003;
DYNAP,FASTUNE,TURNS=1024,LYAPUNOV=1.e-7;
ENDTRACK;
...
```

The first command defines the beam parameters. It is essential that the longitudinal emittance *ET* is set. The command *use* selects the beam line or sequence. The *track* activates the tracking module, *start* enters the starting coordinates (more than one particle can be defined), *dynap* finally tracks two nearby particles with an initial distance *lyapunov* for each *start* definition over *turns* revolutions, and *endtrack* terminates the execution of the tracking module.

The results are stored in the *DYNAP* and *DUNAPTUNE* tables, and can be obtained by the commands

```
value,table(dynap,smear);
```

resp.

```
value,(dynaptune,tunx),(dynaptune,tuny),(dynaptune,dtune);
```

More generally, all results can be printed to a file, using the commands

```
write,table=dynap,file;
write,table=dynaptune,file;
```

The output file 'lyapunov.data' lists the turn number and phase distance between the two Lyapunov partners, respectively, allowing for visual inspection of chaoticity.

frankz 20.03.2006

Fully Coupled Motion and Radiation

EMIT: Equilibrium Emittances

The command

```
EMIT,DELTAP=real,TOL=tolerance;
```

adjusts the RF frequencies such as to obtain the specified average energy error. More precisely, the revolution frequency f_0 is determined for a fictitious particle with constant momentum error

DELTAP = delta$_s$ = delta(E) / p_s c

which travels along the design orbit. The RF frequencies are then set to

$f_{RF} = h f_0$.

If the machine contains at least one RF cavity, and if synchrotron radiation is on, the EMIT command computes the equilibrium emittances and other electron beam parameters using the method of [Chao]. In this calculation the effects of quadrupoles, sextupoles, and octupoles along the closed orbit is also considered. Thin multipoles are used only if they have a fictitious length LRAD different from zero.

If the machine contains no RF cavity, if synchrotron radiation is off, or if the longitudinal motion is not stable, it only computes the parameters which are not related to radiation.

The tolerance is for the distinction static/dynamic: if for the eigenvalues of the one-turn matrix, |e_val_5| < tol and |e-val_6| < tol, then the longitudinal motion is not considered, otherwise it is. The default for TOL is 1.000001.

In the current implementation, the BEAM values of the emittances and beam sizes are only updated for deltap = zero. Example:

```
RFC: RFCAVITY,HARMON...,VOLT=...;
     BEAM,ENERGY=100.0,RADIATE;
     EMIT,DELTAP=0.01;
```

Remark: This module assumes nearly constant lattice functions inside elements. This assumption works for many machines, like LEP (see example), but it fails when the lattice funcionts largely vary inside single elements. In the later case it is advised to slice the elements as shown in ALBA.

R. Tomás **Last updated:**

Error Definitions

This chapter describes the commands which provide error assignment and output of errors assigned to elements. It is possible to assign alignment errors and field errors to single beam elements or to ranges or classes of beam elements.
Elements, classes or ranges of elements are selected by the SELECT command.
ATTENTION: since errors can only be assigned to machine elements, it is necessary to FLATTEN a sequence if it includes other sequences.
Errors can be specified both with a constant or random values. Error definitions consist of four types of statements listed below. They may be entered after having selected a beam line by means of a USE command.
WARNING: any further USE command will destroy the assigned errors. Use the ESAVE option to save and reload errors.

- EALIGN: Define Misalignments
- Field Errors
  - EFCOMP: Components
- EOPTION: Set Error Options
- EPRINT: List Machine Imperfections
- ESAVE: Save Machine Imperfections and read back from file

Werner Herr 18.6.2002

EALIGN: Define Misalignments

Alignment errors are defined by the EALIGN command. The misalignments refer to the local reference system for a perfectly aligned machine. Possible misalignments are displacements along the three coordinate axes, and rotations about the coordinate axes. Alignment errors can be assigned to all beam elements except drift spaces. The effect of misalignments is treated in a linear approximation. A Beam position monitor can be given read errors in both horizontal and vertical planes. Monitor errors (MREX, MREY, MSCALX and MSCALY) are ignored for all other elements. Each new EALIGN statement replaces the misalignment errors for all elements in its range, unless EOPTION,ADD=TRUE has been entered.

Alignment errors are defined by the statement

```
SELECT,FLAG=ERROR,RANGE=range,CLASS=name,PATTERN=string;
EALIGN, DX=real,DY=real,DS=real,
        DPHI=real,DTHETA=real,DPSI=real,
        MREX=real,MREY=real,
        MSCALX=real,MSCALY=real,
        AREX=real,AREY=real;
```

and elements are now selected by the SELECT command. The attributes are:

DX: The misalignment in the *x*-direction for the entry of the beam element (default: 0 m).
DX>0 displaces the element in the positive *x*-direction

DY: The misalignment in the *y*-direction for the entry of the beam element (default: 0 m).
DY>0 displaces the element in the positive *y*-direction

DS: The misalignment in the *s*-direction for the entry of the beam element (default: 0 m).
DS>0 displaces the element in the positive *s*-direction

DPHI: The rotation around the *x*-axis.
A positive angle gives a greater *x*-coordinate for the exit than for the entry (default: 0 rad).

DTHETA: The rotation around the *y*-axis according to the right hand rule (default: 0 rad).

DPSI: The rotation around the *s*-axis according to the right hand rule (default: 0 rad).

MREX: The horizontal read error for a monitor. This is ignored if the element is not a monitor
If MREX>0 the reading for *x* is too high (default: 0 m).

MREY: The vertical read error for a monitor. This is ignored if the element is not a monitor
If MREY>0, the reading for *y* is too high (default: 0 m).

AREX: The misalignment in the *x*-direction for the entry of an aperture limit (default: 0 m).
AREX>0 displaces the element in the positive *x*-direction

AREY: The misalignment in the *y*-direction for the entry of an aperture limit (default: 0 m).
AREY>0 displaces the element in the positive *y*-direction

MSCALX: The relative horizontal scaling error for a monitor. This is ignored if the element is not a monitor.
If MSCALX>0 the reading for *x* is too high (default: 0). A value of 0.5 implies the actual reading is multiplied by 1.5.

MSCALY: The relative vertical scaling error for a monitor. This is ignored if the element is not a monitor.
If MSCALY>0 the reading for *y* is too high (default: 0). A value of -0.3 implies the actual reading is multiplied by 0.7.
Example:

```
SELECT,FLAG=ERROR,CLASS=MQ;
EALIGN,DX=0.002,DY=0.0004*RANF(),DPHI=0.0002*GAUSS();
```

Assigns alignment errors to all elements of class MQ.

```
SELECT,FLAG=ERROR,PATTERN="QF.*";
EALIGN,DX=0.001*TGAUSS(2.5),DY=0.0001*RANF();
```

Assigns alignment errors to all elements starting with "QF". TGAUSS(2.5) means a Gaussian distribution cut at 2.5 sigma.

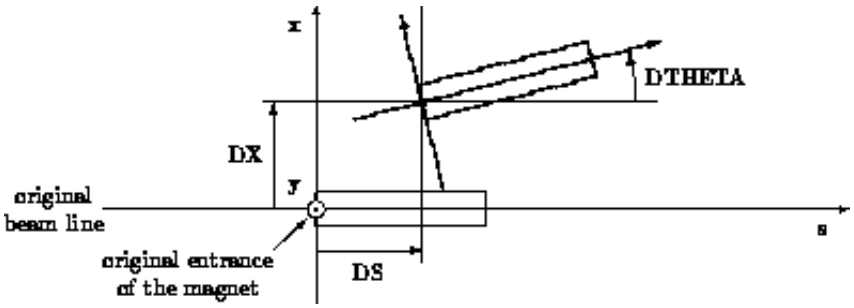


Figure 1: Example of Misplacement in the (*x, s*)-plane.

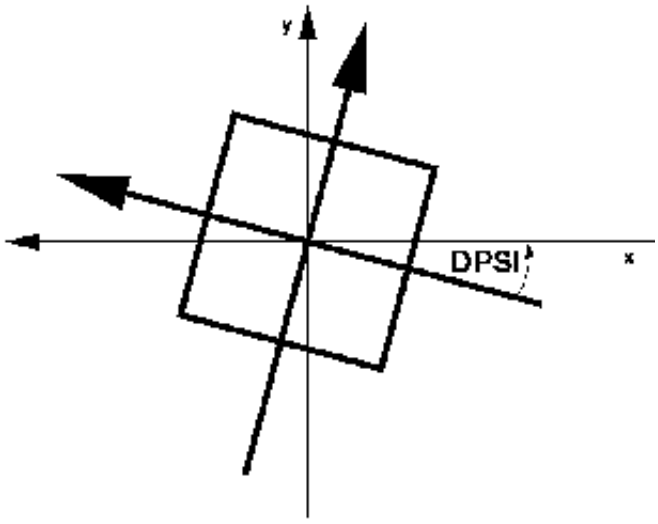


Figure 2: Example of Misplacement in the (*x, y*)-plane.

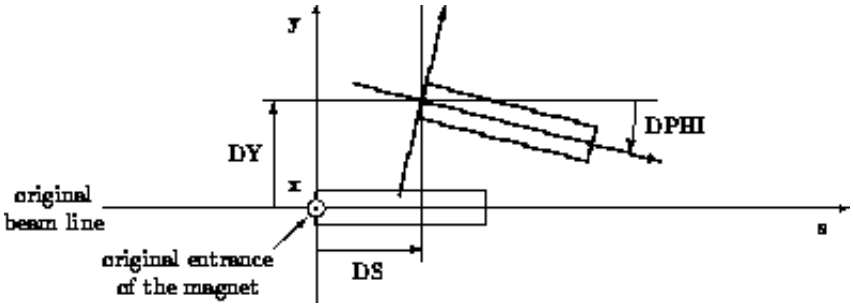


Figure 3: Example of Misplacement in the (*y, s*)-plane.

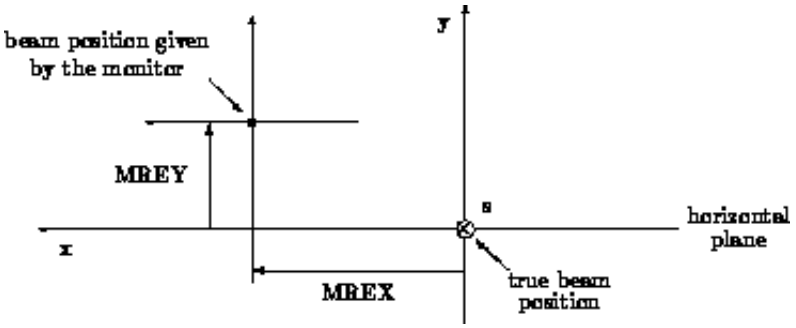


Figure 4: Example of Read Errors in a monitor

Last updated: 02.9.2002

Werner Herr 18.6.2002

Field Errors

Field errors can be entered as relative or absolute errors. Different multipole components can be specified with different kinds of errors (relative or absolute). Relations between absolute and relative field errors are listed below.

In MAD8 two commands were used for that purpose: EFIELD and EFCOMP. Only EFCOMP was implemented in MAD-X since it provides the full functionality of EFIELD and there was no need for duplication.

All field errors are specified as the integrated value int(*K*ds*) of the field components along the magnet axis in m^{-i}. There is no provision to specify a global relative excitation error affecting all field components in a combined function magnet. Such an error may only be entered by defining the same relative error for all field components.

Field errors can be specified for all magnetic elements by the statement

```
SELECT,FLAG=ERROR,RANGE=range,CLASS=name,PATTERN=string;
EFCOMP, ORDER:=integer,RADIUS:=real,
        DKN:={dkn(0),dkn(1),dkn(2),...},
        DKS:={dks(0),dks(1),dks(2),...},
        DKNR:={dknr(0),dknr(1),dknr(2),...},
        DKSR:={dksr(0),dksr(1),dksr(2),...};
```

and elements are now selected by the SELECT command. Each new EFCOMP statement replaces the field errors for all elements in its range (s). Any old field errors present in the range are discarded or incremented depending on the setting of EOPTION,ADD. EFCOMP defines them in terms of relative or absolute components.

The attributes are:

ORDER: If relative errors are entered for multipoles, this defines the order of the base component to which the relative errors refer. This reference strength k_{ref} always refers to the normal component. To use a skew component as the reference the reference radius should be specified as a negative number. The default is zero.
Please note that this implies to specify k_0 to assign relative field errors to a bending magnet since k_0 is used for the normalization and NOT the ANGLE.

RADIUS: Radius *R* were dknr(i) or dksr(i) are specified for 0...i...20 (default 1 m). This attribute is required if dknr(i) or dksr(i) are specified. If *R* is negativ, the skew component is used for the reference strength.

DKN(i): Absolute error for the normal multipole strength with (2i+2) poles (default: 0 m^{-i}).

DKS(i): Absolute error for the skewed multipole strength with (2i+2) poles (default: 0 m^{-i}).

DKNR(i): Relative error for the normal multipole strength with (2i+2) poles (default: 0 m^{-i}).

DKSR(i): Relative error for the skewed multipole strength with (2i+2) poles (default: 0 m^{-i}).

Time memory effects:
The relative errors can be corrected for possible time memory effects. A correction term is computed and added to the relative error.

The correction term is parametrized as a 3rd order polynomial in the reference strength k_{ref} according to:

$$Delta = sum (c_i * k^i_{ref}) \qquad i = 0,..3$$

The coefficients c_i for the polynominal must be supplied in the command.
Two additional parameters and options are required:

HYSTER: if it is set to 1 applies the correction term derived from the reference strength and the coefficients.

HCOEFFN and HCOEFFS: coefficients (normal and skew) for the computation of the correction term. The 4 coefficients are specified in increasing order, starting with the 0th order. Each group of four coefficients is valid for one order of the field errors. Trailing zeros can be omitted, preceding zeros must be given.
Examples:
Example 1 (assign relative errors to quadrupoles);

```
select, flag=error, pattern="q.*";
efcomp, order:=1, radius:=0.010,
dknr:={0,4e-1,1e-1,2e-3,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
dksr:={0,4e-1,1e-1,2e-3,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
```

Example 2 (add time memory effect to relative errors):

```
select, flag=error, pattern="^q.*";
efcomp, order=1, radius=0.020, hyster=1,
hcoeffn:={0.000,0.000,0.000,0.000,   // coefficients multipole order 0
          0.001,0.000,0.000,0.000,   // coefficients multipole order 1
          0.000,0.000,0.002,0.000},  // coefficients multipole order 2
dknr:={0,1e-2,2e-4,4e-5,1e-5,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
dksr:={0,1e-2,2e-4,4e-5,1e-5,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
```

See also: Random values and deferred expressions.

Werner Herr 6.12.2004

EPRINT: List Machine Imperfections

This command prints a table of errors assigned to elements. The range for these elements has to be specified. Field errors are printed as absolute errors, because all relative errors are transformed to the corresponding absolute error at definition time. An error print is requested by the statement

```
SELECT,FLAG=ERROR,RANGE=range,CLASS=name,PATTERN=string;
EPRINT;
```

and elements are now selected by the SELECT command.
A listing for ALL elements, i.e. not only the selected, can be obtained with the command

```
EPRINT,FULL=TRUE;
```

In that case, the SELECT command has no effect.

Werner Herr 18.6.2002

ESAVE: Save Machine Imperfections and read back from file

Writing errors to a file:
This command saves a table of errors assigned to elements on a file, using a format which can be read in again to obtain the same results. This allows dumping the errors and reloading them after a new USE command. The range for these elements has to be specified. An error save is requested by the statement

```
ESAVE,FILE=string;

Example:

SELECT,FLAG=ERROR,RANGE=range,CLASS=name,PATTERN=string;
ESAVE,FILE=err.file;
```

and elements selected by the SELECT command are saved to the file.
To save the errors of all elements to a file, one can use:

```
SELECT,FLAG=ERROR,FULL;
ESAVE,FILE=err.file;
```

Please note: in case of field errors, the absolute errors are saved and not relative errors.
Setting errors from a table or file:
To assign errors from a file is not a priori straightforward. It may be required to re-assign existing errors after a **USE** command was executed (which deletes all errors attached to a sequence).
Errors stored in the form of an internal table (*errtab*) can be directly attached to the appropriate positions in the sequence with the command:

```
SETERR,TABLE=errtab;
```

The table *errtab* can be generated internally or from an external file (*errfile*) with the generic command READMYTABLE.
The command sequence:

```
READMYTABLE,file=errfile,table=errtab;
SETERR,TABLE=errtab;
```

reads the file *errfile* into the table *errtab* and the command SETERR attaches the errors to the elements in the active sequence.
The file *errfile* can be produced by a preceding ESAVE command or any other utility. It should follow the format of a file generated with ESAVE (see example program).

Werner Herr 18.6.2002

The Intra-Beam Scattering Module (IBS)

As emphasized by its name, the Intra-Beam Scattering module (IBS) computes the contribution to emittance growth rates due to Coulomb scattering of particles within relativistic beams. The formalism used in this module is that derived by J.D. Bjorken and S.K.Mtingwa [[Bjorken and Mtingwa]] in 1982. Contrary to other IBS-routines, the Bjorken-Mtingwa formalism takes into account the variation of the lattice parameters around the machine, rather than using average values. Consequently, the knowledge of the optical functions of the machine is required. In MAD-X, this is achieved with the ‘‘**twiss**’’ command.

It is well known that the intra-beam scattering growth times behave like:

$$\frac{1}{\tau_i} = C_i \times \frac{N}{\gamma \epsilon_x \epsilon_y \epsilon_s} \qquad (i = x, y, s)$$

where C_i accounts for some constants and the integrals for the scattering functions, N is the number of particles in the bunch, γ is the relativistic factor and ϵ_i are the normalized emittances in the horizontal, vertical and longitudinal plane respectively. It thus follows that the second required input is a description of the beam parameters, which is achieved via the ‘‘**beam**’’ command (see below).

Once the optical functions and the beam parameters have been defined, the evaluation of the scattering growth times follows via the ‘‘**ibs**’’ command. The logical follow-up of the MAD-X commands is illustrated in the two examples provided with the IBS-module.

Input of the beam parameters

This section briefly describes the parameters which have to be present in the ‘‘**beam**’’ command in order to run the IBS-module:

Type of particle

The parameter ‘‘**particle=**’’ is mandatory. It can take one of the following **three** values: **proton, electron or ion**. For proton and electron, the parameter ‘‘particle’’ is the only one to be defined. In case **ion** is used, two additional parameters have to be defined, namely ‘‘**mass=**’’, which is typically the number of nucleons for the corresponding ion multiplied by **nmass** the unified atomic mass unit [0.931494013 GeV/(c**2)] , and ‘‘**charge=**’’ for the number of charges.

The energy

The definition of the energy (total, kinetic, total energy of the ions or energy per nucleon) is a difficult one. In the present approach, the energy is the **total** energy of the particle. For ions, the expected input is the **proton equivalent** energy, i.e. the total energy a proton would have when circulating in the defined machine. As an illustration, in the LHC, protons will be injected with an energy of 450 GeV. Consequently, to evaluate the growth times for Lead ions at injection in the LHC, one has to input **energy=450*charge**. Therefore the above example of Lead at the LHC injection energy may look as follows in the MAD-X input language:

nucleon=208; charge=82;
beam,particle=ion,charge=charge,energy=450*charge,mass=nucleon*nmass;

An important check for the correctness of the input is the printed value of the relativistic factor γ . The latter should correspond to:

$$\gamma_{ion} = \gamma_{proton} \times \frac{charge}{nucleon}$$

The number of particles

The number of particles (or number of ions) is defined with the parameter ‘‘**npart=**’’.

Beam sizes - Emittances

This part of the input is used to define the normalized emittances (horizontal, vertical and longitudinal). The required parameters are the **physical** transverse emittances (**ex=** and **ey=** [π m]), the bunch length (**sigt=** [m]) and the relative energy spread (**sige=**).

File Attribute

If FILE="file_name" appears MAD-X produces a table and writes on a file for each element of the machine: ELEMENT NAME, Position S [m], DELS [m] (Length Difference of consecutive Elements in the Table), TLI (Longitudinal growth time), TXI (Horizontal growth time), TYI (Vertical growth time).

Examples

The two examples provided for the module Intra-Beam Scattering illustrate the commands required to run the module. The two examples have been selected such as to highlight the differences between a computation for protons and that for ions. Both examples compute the IBS growth times at injection into the LHC. The examples are located at http://frs.home.cern.ch/frs/Xdoc/mad-X.html.

Frank Schmidt 2003-05-23

EUROPEAN ORGANIZATION FOR NUCLEAR RESEARCH 

Matching Module

Before a match operation at least one sequence must be selected by means of a USE command. Matching is then initiated by the MATCH command. The matching module can act on more than one sequence simultaneously by specifying more than one sequence when INITIATING the matching mode. From this command to the corresponding ENDMATCH command MAD accepts the matching commands listed below. For a mathematical description of the minimisation procedures see [James]. In particular one may do the following:

- Define the sequence(s) the matching module will work on
- Set initial conditions for transfer line matching
- Define constraints
- Define the parameters to be varied
- Match by different methods.

The matching commands are described in detail below. Some other commands can also be issued during matching.

- Enter and Leave Matching Mode
  - MATCH: Initiating the Matching Mode
  - ENDMATCH: Leave Matching Mode
- Define Variable Parameter
  - VARY: Set Parameter to Vary
- Constraint
  - CONSTRAINT: Simple Constraint
  - CONSTRAINT: User Defined Variables
  - WEIGHT: Matching Weights
  - GLOBAL: Global Constraints
  - GWEIGHT: Weights for Global Constraints
- Matching Methods
  - LMDIF: Fast Gradient Minimisation
  - MIGRAD: Gradient Minimisation
  - SIMPLEX: Simplex Minimisation
  - JACOBIAN: Newton Minimisation
- Expression Matching with USE_MACRO

- Matching Examples

Oliver Brüning, June, 2002; Riccardo de Maria, February, 2006.

Enter and Leave Matching Mode

Before matching at least one SEQUENCE must be selected by means of a USE command. The matching module can act on more than one sequence simultaneously by specifying more than one sequence when INITIATING the matching mode:

Initiating the Matching Module

The 'match' command can be either used for matching a periodic cell or for matching an insertion to another part of the machine. Both matching modes are initiated by the MATCH command.

- Cell matching:

  In the first mode the matching routine is initiated only with one attribute specifying the sequence(s) the matching module will work on. In this matching mode the periodicity of the optics functions is enforced at the beginning and end of the selected range.

  MATCH, SEQUENCE='name1', 'name2',..,nema-n';

- Insertion matching:

  In the second mode, called insertion matching, the matching routine is initiated with two attributes: one specifying the sequence(s) the matching module will work on and one specifying the initial conditions of the optic functions for each sequence. In this case the initial values are assumed as exact.
  - Specification of Initial Values: The initial values of the optical functions for the insertion matching can either be specified in form of a SAVEBETA command or by explicitly stating the individual optic function values after the 'MATCH' command. The two options can be entered as

    ```
    MATCH,sequence='name1','name2',..,'name-n',BETA0='beta01','beta02',..,'beta0n';
    ```

    or

    ```
    MATCH,SEQUENCE='sequence-name', BETX=real,ALFX=real,MUX=real,
                                    BETY=real,ALFY=real,MUY=real,
                                    X=real,PX=real,Y=real,PY=real,
                                    DX=real,DY=real,DPX=real,DPY=real,
                                    DELTAP=real;
    ```

    > Examples:
    - Example 1:

```
CELL: SEQUENCE=(...) ;
INSERT: SEQUENCE=(...) ;
USE,PERIOD=cell;
SAVEBETA,LABEL=bini,place=#e;
TWISS,SEQUENCE=cell;
USE,PERIOD=insert;
MATCH,SEQUENCE=insert,BETA0=bini;
CONSTRAINT,SEQUENCE=insert,RANGE=#e,MUX=9.345,MUY=9.876;
```

This matches the sequence 'INSERT' with initial conditions to a new phase advance. The initial conditions are given by the periodic solution for the sequence CELL1.

- Example 2:

```
USE,PERIOD=INSERT;
MATCH,SEQUENCE=insert;
CONSTRAINT,SEQUENCE=insert,RANGE=#e,MUX=9.345,MUY=9.876;
```

This matches the beam line 'INSERT' with periodic boundary conditions to a new phase advance.

The initial conditions can also be transmitted by a combination of a SAVEBETA command and explicit optic function specifications:

```
USE,CELL1;
SAVEBETA,LABEL=bini,PLACE=#E;
TWISS,SEQUENCE=CELL1;
USE,PERIOD=LINE1;
MATCH,SEQUENCE=LINE1,BETA0=bini,MUX=1.234,MUY=4.567;
```

This example transmits all values of the SAVEBETA array 'bini' as initial values to the MATCH command and overrides the initial phase values by the given values.

An additional CONSTRAINT may be imposed in other places, i.e. intermediate or end values of the optics functions at the transition point.

- More than one active sequence:

  The matching module can act on more than one sequence simultaneously by specifying more than one sequence after the MATCH command:

```
MATCH,SEQUENCE=LINE1,CELL1,BETA0=bini1,bini2;
```

  This example initiates the matching mode for the 'LINE1' and the 'CELL1' sequence. The Twiss module function of the two sequences are calculated with fixed initial conditions. The SAVEBETA array 'bini1' is used for calculating the optics functions of sequence 'LINE1' and the SAVEBETA array 'bini2' for calculating the optics functions of sequence 'CELL1'. Without the initial conditions the matching module will work in the CELL mode.

Further attributes of the TWISS statements are:

- RMATRIX: If this flag is used the one-turn map at the location of every element is calculated and prepared for storage in the TWISS table.
  Target values for the matrix elements at certain positions in the sequence can be specified with the help of the CONSTRAINT command and the keywords: **RE, RE11...RE16...RE61...RE66**,

where **REij** refers to the "ij" matrix component.

> Examples:

```
Example 1:
MATCH,RMATRIX,SEQUENCE='name',BETA0='beta-block-name';
CONSTRAINT,SEQUENCE=insert,RANGE=#e,RE11=-2.808058321,re22=2.748111197;
VARY,NAME=kqf,STEP=1.0e-6;
VARY,NAME=kqd,STEP=1.0e-6;
```

This matches the sequence 'name' with initial conditions to new values for the matrix elements 'RE11' and 'RE22' by varying the strength of the main quadrupole circuits.

- CHROM: If this flag is used the chromatic functions at the location of every element are calculated and prepared for storage in the TWISS table.
  Target values for the chromatic functions at certain positions in the sequence can be specified with the help of the CONSTRAINT command and the keywords WX, PHIX, WY, PHIY,....

Leave Matching Mode

The ENDMATCH command terminates the matching section and deletes all tables related to the matching run.

```
ENDMATCH;
```

Oliver Brüning, October, 2003; Riccardo de Maria, January, 2008.

References

MAD Home Page, MAD-8 User Guide,

1

The Graphical Kernel System (GKS). ISO, Geneva, July 1985. International Standard ISO 7942.

2

B. Autin and Y. Marti. *Closed Orbit Correction of Alternating Gradient Machines using a small Number of Magnets*. CERN/ISR-MA/73-17, CERN, 1973.

3

D.P. Barber, K. Heinemann, H. Mais and G. Ripken, *A Fokker--Planck Treatment of Stochastic Particle Motion within the Framework of a Fully Coupled 6-dimensional Formalism for Electron-Positron Storage Rings including Classical Spin Motion in Linear Approximation*, DESY report 91-146, 1991.

4

R. Bartolini, A. Bazzani, M. Giovannozzi, W. Scandale and E. Todesco, *Tune evaluation in simulations and experiments*, CERN SL/95-84 (AP) (1995).

5

J. D. Bjorken and S. K. Mtingwa. *Particle Accelerators* **13**, pg. 115.

6

E. M. Bollt and J. D. Meiss, *Targeting chaotic orbits to the Moon through recurrence*, Phys. Lett. A 204,373 (1995).

7

P. Bramham and H. Henke. private communication and LEP Note LEP-70/107, CERN.

8

Karl L. Brown. *A First-and Second-Order Matrix Theory for the Design of Beam Transport Systems and Charged Particle Spectrometers*. SLAC 75, Revision 3, SLAC, 1972.

9

Karl L. Brown, D. C. Carey, Ch. Iselin, and F. Rothacker. *TRANSPORT - A Computer Program for Designing Charged Particle Beam Transport Systems*. CERN 73-16, revised as CERN 80-4, CERN, 1980.

10

A. Chao. *Evaluation of beam distribution parameters in an electron storage ring*. Journal of Applied Physics, 50:595-598, 1979.

11

A. W. Chao and M. J. Lee. *SPEAR II Touschek lifetime*. SPEAR-181, SLAC, October 1974.

12

M. Conte and M. Martini. *Particle Accelerators* **17**, 1 (1985).

13

E. D. Courant and H. S. Snyder. *Theory of the alternating gradient synchrotron*. Annals of Physics, 3:1-48, 1958.

14

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

15

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

16

A. Dragt. *Lectures on Nonlinear Orbit Dynamics*, 1981 Summer School on High Energy Particle Accelerators, Fermi National Accelerator Laboratory, July 1981. American Institute of Physics, 1982.

17

D. A. Edwards and L. C. Teng. *Parametrisation of linear coupled motion in periodic systems*. IEEE Trans. on Nucl. Sc., 20:885, 1973.

18

M. Giovannozzi, *Analysis of the stability domain of planar symplectic maps using invariant manifolds*, CERN/PS 96-05 (PA) (1996).

19

H. Grote. *GXPLOT User's Guide and Reference Manual*. LEP TH Note 57, CERN, 1988.

20

LEP Design Group. *Design Study of a 22 to 130 GeV electron-positron Colliding Beam Machine (LEP)*. CERN/ISR-LEP/79-33, CERN, 1979.

M. Hanney, J. M. Jowett, and E. Keil. *BEAMPARAM - A program for computing beam dynamics and performance of electron-positron storage rings*. CERN/LEP-TH/88-2, CERN, 1988.

22

R. H. Helm, M. J. Lee, P. L. Morton, and M. Sands. *Evaluation of synchrotron radiation integrals*. IEEE Trans. Nucl. Sc., NS-20, 1973.

23

F. James. *MINUIT, A package of programs to minimise a function of n variables, compute the covariance matrix, and find the true errors*. program library code D507, CERN, 1978.

24

E. Keil. *Synchrotron radiation from a large electron-positron storage ring*. CERN/ISR-LTD/76-23, CERN, 1976.

25

D. E. Knuth. *The Art of Computer Programming*. Volume 2, Addison-Wesley, second edition, 1981. Semi-numerical Algorithms.

26

J. Laskar, C. Froeschle and A. Celletti, *The measure of chaos by the numerical analysis of the fundamental frequencies. Application to the standard mapping*, Physica D 56, 253 (1992).

27

H. Mais and G. Ripken, *Theory of Coupled Synchro-Betatron Oscillations*. DESY internal Report, DESY M-82-05, 1982.

28

M. Meddahi, *Chromaticity correction for the 108/60 degree lattice*, CERN SL/Note 96-19 (AP) (1996).

29

J. Milutinovic and S. Ruggiero. *Comparison of Accelerator Codes for a RHIC Lattice*. AD/AP/TN-9, BNL, 1988.

30

B. W. Montague. *Linear Optics for Improved Chromaticity Correction*. LEP Note 165, CERN, 1979.

31

Gerhard Ripken, *Untersuchungen zur Strahlführung und Stabilität der Teilchenbewegung in Beschleunigern und Storage-Ringen unter strenger Berücksichtigung einer Kopplung der Betatronschwingungen*. DESY internal Report R1-70/4, 1970.

32

F. Ruggiero, *Dynamic Aperture for LEP 2 with various optics and tunes*, Proc. Sixth Workshop on LEP Performance, Chamonix, 1996, ed. J. Poole (CERN SL/96-05 (DI),1996), pp. 132--136.

33

L. C. Teng. *Concerning n-Dimensional Coupled Motion*. FN 229, FNAL, 1971.

34

U. Völkel. *Particle loss by Touschek effect in a storage ring*. DESY 67-5, DESY, 1967.

35

R. P. Walker. *Calculation of the Touschek lifetime in electron storage rings*. 1987. Also SERC Daresbury Laboratory preprint, DL/SCI/P542A.

36

P. B. Wilson. *Proc. 8th Int. Conf. on High-Energy Accelerators*. Stanford, 1974.

37

A. Wrülich and H. Meyer. *Life time due to the beam-beam bremsstrahlung effect*. PET-75-2, DESY, 1975.

MAD Home Page, MAD-8 User Guide,

---

fci, January 23, 1997

EUROPEAN ORGANIZATION FOR NUCLEAR RESEARCH 

Define Variable Parameter

VARY: Define Variable Parameter

A parameter to be varied is specified by the command

```
VARY,NAME=variable,STEP=real,LOWER=real,UPPER=real;
```

It has four attributes:

- NAME: The name of the parameter or attribute to be varied,
- STEP: The approximate initial step size for varying the parameter. If the step is not entered, MAD tries to find a reasonable step, but this may not always work.
- LOWER: Lower limit for the parameter (optional),
- UPPER: Upper limit for the parameter (optional).
- SLOPE: allowed change rate (optional, available only using JACOBIAN routine). Limit the parameter to increase (SLOPE=1) decrease (SLOPE=-1) only.
- OPT: optimal value for the parameter (optional, available only using JACOBIAN routine).

Examples:

```
VARY,NAME=PAR1,STEP=1.0E-4;                          ! vary global parameter PAR1
VARY,NAME=QL11->K1,STEP=1.0E-6;                      ! vary attribute K1 of the QL11
VARY,NAME=Q15->K1,STEP=0.0001,LOWER=0.0,UPPER=0.08; ! vary with limits
```

If the upper limit is smaller than the lower limit, the two limits are interchanged. If the current value is outside the range defined by the limits, it is brought back to range. If a parameter comes outside the limits during the matching process the matching module resets the parameter to a value inside the limits and informs the user with a message. If such a 'rescaling' occurs more than 20 times the matching process terminates. The user should either eliminate the corresponding parameters from the list of varied parameters or change the corresponding upper and lower limits before restarting the matching process. After a matching operation all varied attributes retain their value after the last successful matching iteration. Using JACOBIAN routine, STRATEGY=3, in case the number of parameters is greater the the number of constraint, if a parameter comes outside the limits, it is excluded automatically from the set of variables and a new solution is searched.

Oliver Brüning, June, 2002. Riccardo de Maria, February, 2006.

Constraints

CONSTRAINT: Simple Constraint

Simple constraints are imposed by the CONSTRAINT command. The CONSTRAINT command has three attributes:

- the SEQUENCE entry specifies the sequence for which the constraint applies.
- the RANGE entry specifies the position where the constraint must be satisfied. The RANGE can either be the name of a single element in the sequence or a range between two elements. In the later case the two element names must be separated by a '/': RANGE=nam1/name2
- the optics functions to be constrained

The optic functions can be constraint in four different ways:

- lower limits: 'BETX > value' -> type1
- upper limits: 'BETX < value' -> type2
- lower and upper limits: 'BETX < value1,BETX > value2' -> type3
- target value: BETX=value -> type 4

In case one element is affected my more than one constraint command the last CONSTRAINT will be chosen. For example, one can specify the maximum acceptable beta function over a range of the sequence and specify the target beta function for one element that lies inside this range. In this case one must first specify the constraint that affects the whole range and then the constraint for the single element. This way the constraint of the target value overrides the previous constraint on the upper limit for the selected element. For example, the following constraint statements limit the maximum horizontal beta function between 'marker1' and 'marker2' to 200 meter and require a horizontal beta function of 180 meter at element 'name1':

```
CONSTRAINT,SEQUENCE=sequence-name,RANGE='marker1'/'marker2',BETX<200.0;
CONSTRAINT,SEQUENCE=sequence-name,RANGE='name1'/'marker2',BETX=100.0;
```

When the two constraint statements are interchanged the horizontal beta function at element 'name1' will only be limited to less than 200 meter and NOT constrained to 100 meter!

The CONSTRAINTS can either be specified with explicit values for the constraints of the optic functions or via a pre-calculated SAVEBETA module. The first options has the form:

```
CONSTRAINT,SEQUENCE=sequence-name,RANGE=position,BETX=real,ALFX=real,MUX=real,
                                                 BETY=real,ALFY=real,MUY=real,
                                                 X=real,PX=real,Y=real,PY=real,
                                                 DX=real,DY=real,DPX=real,DPY=real;
```

Here all linear lattice functions (BETX, BETY, ALFX, ALFY, MUX, MUY, DX, DY, DPX, DPY) or chromatic lattice functions (WX, XY, PHIX, PHIY, DMUX, DUMY, DDX, DDY, DDPX, DDPY) are constrained at the selected range to the corresponding values.

The second form of the CONSTRAINT command has the form

```
CONSTRAINT,SEQUENCE=sequence-name,RANGE=position,BETA0=beta0-name,MUX=real,MUY=real
```

Here all of (BETX, BETY, ALFX, ALFY, MUX, MUY, DX, DY, DPX, DPY) are constrained in the selected points to the corresponding values of a pre-calculated SAVEBETA module. In the above example the phases (MUX, MUY) are overridden by the numerical values specified via 'MUX=real' and 'MUY=real'. Normally ''RANGE'' refers to a single position.

User Defined Matching Constraints

In addition to the nominal TWISS variables the user can define a limited set of 'user-defined' variables in the constrint statement. This allows, for example, the matching of the nromalized dispersion or the mechanical aperture. The MATCH module allows four user defined variables called: mvar1, mvar2, mvar3 and mvar4. The variables can be defined according to the general variable declaration rules of deferred exressions. For example, in order to match the normalized dispersion at a certain location in the sequence one would first define a variable:

```
mvar1 := table(twiss,dx)/sqrt(table(twiss,betx));
```

After that the user has to select the variable for output in the TWISS statement (see TWISS module and SELECT for more details on the TWISS module and SELECTION statements):

```
select, flag=twiss, clear;
select, flag=twiss, column=keyword,, name, s, betx, dx, mvar1;
twiss, sequence='sequence-name',file=twiss.file;
```

The variable can now be referenced like any other TWISS variable in the constraint command:

```
constraint, sequence='sequence-name',range='location',mvar1='target-value';
```

Matching Weights

The matching procedures try to fulfil the constraints in a least square sense. A penalty function is constructed which is the sum of the squares of all errors, each multiplied by the specified weight. The larger the weight, the more important a constraint becomes. The weights are taken from a table of current values. These are initially set to weight default values, and may be reset at any time to different values. Values set in this way remain valid until changed again. The command

```
WEIGHT, BETX=real,ALFX=real,MUX=real,
        BETY=real,ALFY=real,MUY=real,
        X=real,PX=real,Y=real,PY=real,
        DX=real,DPX=real,DY=real,DPY=real;
```

changes the weights for subsequent constraints. The weights are entered with the same name as the linear lattice functions orbit coordinate to which the weight applies. Frequently the matching weights serve to select a restricted set of functions to be matched.

Default Matching Weights

name	weight	name	weight	name	weight	name	weight	name	weight	name	weight
BETX	1.0	ALFX	10.0	MUX	10.0	BETY	1.0	ALFY	10.0	MUY	10.0
X	10.0	PX	100.0	Y	10.0	PY	100.0	T	0.0	PT	0.0
DX	10.0	DPX	100.0	DY	10.0	DPY	100.0	-			
WX	10.0	PHIX	10.0	DMUX	100.0	WY	10.0	PHIY	10.0	DMUY	100.0
DDX	10.0	DDPX	100.0	DDY	10.0	DDPY	100.0	-			
MVAR1	10.0	MVAR2	10.0	MVAR3	10.0	MVAR4	10.0	-			

GLOBAL: Global Matching Constraints

In addition to conventional matching constraints that specify the optics functions at a certain position in the sequence the user can also constrain global optics parameters such as, for example, the overall machine tune and the machine chromaticity. Such global optics parameters can be constraint via the GLOBAL command, having the following syntax:

```
GLOBAL,SEQUENCE=sequence-name,Q1=real,Q2=real,dQ1=real,dQ2=real,&
                              ddQ1=real,ddQ2=real;
```

Global matching weights can be (re)set by the new GWEIGHT command, having attributes identical to those of GLOBAL. All the attributes are optional and have the following meaning:

Q1, Q2, dQ1, dQ2
: enable a correction of tunes and chromaticities in presence of magnetic imperfections or misalignments,

ddQ1, ddQ2
: enable a correction of nonlinear chromaticities

Oliver Brüning, June, 2002

Matching Methods

MADX currently supports four different matching algorithms:

- ## LMDIF: Fast Gradient Minimisation

  The LMDIF command minimises the sum of squares of the constraint functions using their numerical derivatives:

  ```
  LMDIF,CALLS=integer,TOLERANCE=real;
  ```

  It is the fastest minimisation method available in MAD. The command has two attributes:
  - CALLS: The maximum number of calls to the penalty function (default: 1000).
  - TOLERANCE: The desired tolerance for the minimum (default: 10**(-6)).

  Example:

  ```
  LMDIF,CALLS=2000,TOLERANCE=1.0E-8;
  ```

- ## MIGRAD: Gradient Minimisation

  The MIGRAD command minimises the penalty function using the numerical derivatives of the sum of squares:

  ```
  MIGRAD,CALLS=integer,TOLERANCE=real,STRATEGY=1;
  ```

  The command has three attributes:
  - CALLS: The maximum number of calls to the penalty function (default: 1000).
  - TOLERANCE: The desired tolerance for the minimum (default: 10**(-6)).
  - STRATEGY: A code for the strategy to be used (default: 1). Details are given in [James].

  Example:

  ```
  MIGRAD,CALLS=2000,TOLERANCE=1.0E-8;
  ```

- ## SIMPLEX: Simplex Minimisation

  The SIMPLEX command minimises the penalty function by the simplex method:

  ```
  SIMPLEX,CALLS=integer,TOLERANCE=real;
  ```

  Details are given in [James]. The command has two attributes:
  - CALLS: The maximum number of calls to the penalty function (default: 1000).

- TOLERANCE: The desired tolerance for the minimum (default: 10**(-6)).

Example:

```
SIMPLEX,CALLS=2000,TOLERANCE=1.0E-8;
```

-

JACOBIAN: Newton Minimisation

The JACOBIAN command minimises the penalty function calculating the Jacobian and solving the linear problem. A QR or LQ decomposition is performed when the system is over or under-determined. Before starting the matching routine two optional transformations (COOL and RANDOM) are performed.

```
JACOBIAN,CALLS=integer,TOLERANCE=real,REPEAT=integer,STRATEGY=integer,COOL=real,BALANCE=real, random=real;
```

The command has the attributes:

- CALLS: The maximum number of calls to the penalty function (default: 30).
- TOLERANCE: The desired tolerance for the minimum (default: 10**(-6)).
- REPEAT: The number of call of the JACOBIAN routine (default: 1).
- BISEC: Selects the maximum number of iteratation used to determin the step length which reduces the penalty function during the main iteration. A large number (i.e. 6) reduce the probability to diverge from the solution, but increase the one for being trapped in a local minum.
- STRATEGY: A code for the strategy to be used (default: 3). If STRATEGY=1 the routine resets the values of the variables which exceeds the limits. If STRATEGY=2 the routine print the Jacobian and exit without matching. If STRATEGY=3 the routine disables the variables which exceeds the limits keeping however the number of variables greater or equal to the number of the constraints.
- COOL, BALANCE: The factors which specify the following transformation:

  ```
  if "balance" >=0
     newval=(1-cool)*oldval+cool*( (1-balance)*maxval+balance*minval )
  else
     newval=(1-cool)*oldval+cool* optval
  ```

  where `newval` is the new value after the transformation, `oldval` is the previous value, `maxval, minval, optval` are the maximum value, minimum value, optimal value of the variable specified in the VARY command.
- RANDOM: The factors which specify the following transformation:

  ```
  newval= (1+ random * rand() ) * oldval
  ```

  where `newval` is the new value after the transformation, `oldval` is the previous value, `rand()` is a stochastic variable with a uniform (-0.5,0.5) distribution.

Example:

```
JACOBIAN,CALLS=20,TOLERANCE=1.0E-8,STRATEGY=3,COOL=0.1,BALANCE=0.5,RANDOM=0.01;
```

Oliver Brüning, June, 2002. Riccardo de Maria, February, 2006.

Introduction

It is possible to match user defined expressions with the USE_MACRO keyword. The general input structure for a match command is the following:

```
MATCH,USE_MACRO;
... VARY statements ...
USE_MACRO, NAME=macro1;
     or
macro1: MACRO={ ... madx statements};
CONSTRAINT, expr=  "lhs1 < | = | > rhs1";
CONSTRAINT, expr=  "lhs2 < | = | > rhs2";
...  CONSTRAINT statements ...
MACRO 2 definition
... CONSTRAINT statements ...
MACRO n definition
... CONSTRAINT statements ...
... METHODS statements ...
ENDMATCH;
```

The algorithm for evaluating the penalty function is the following:

- execute the first macro,
- evaluate and compute the difference between the lhs and the rhs the first set of expressions,
- in case of other macros, evaluates in order the macro and the expressions
- the set of differences are minimized by the selected method using the variables defined in the VARY statements.

Initiating the Matching Module with USE_MACRO

With:

```
MATCH,USE_MACRO;
```

the ’match’ command can be used for matching any expression which can be defined through expression. It requires a slightly different syntax.

VARY statements

In the USE_MACRO mode the vary statement follows the same rules of the other modes explained in the section Define Variable Parameter

Macro definitions

The macro to be used in the matching routine can be defined in two ways:

- using USE_MACRO statement:

  ```
  USE_MACRO, NAME=macro1;
  ```

  defining a new macro on the fly using the usual syntax for macros.

After a macro definition is necessary to define a set of constraints exclusively with the following syntax:

```
CONSTRAINT, expr=  "lhs = rhs";
```

or

```
CONSTRAINT, expr=  "lhs < rhs";
```

or

```
CONSTRAINT, expr=  "lhs > rhs";
```

where "lhs" and "rhs" are well defined MadX expressions. Other set of macro and constraints can be defined afterwards.

Examples

The following example the USE_MACRO mode can emulate a matching script which uses the normal syntax.
Normal syntax:

```
MATCH,SEQUENCE=LHCB1,LHCB2;
    VARY, NAME=KSF.B1, STEP=0.00001;
    VARY, NAME=KSD.B1, STEP=0.00001;
    VARY, NAME=KSF.B2, STEP=0.00001;
    VARY, NAME=KSD.B2, STEP=0.00001;
    GLOBAL,SEQUENCE=LHCB1,DQ1=QPRIME;
    GLOBAL,SEQUENCE=LHCB1,DQ2=QPRIME;
    GLOBAL,SEQUENCE=LHCB2,DQ1=QPRIME;
    GLOBAL,SEQUENCE=LHCB2,DQ2=QPRIME;
    LMDIF, CALLS=10, TOLERANCE=1.0E-21;
ENDMATCH;
```

USE_MACRO syntax:

```
MATCH,USE_MACRO;
    VARY, NAME=KSF.B1, STEP=0.00001;
    VARY, NAME=KSD.B1, STEP=0.00001;
    VARY, NAME=KSF.B2, STEP=0.00001;
    VARY, NAME=KSD.B2, STEP=0.00001;
    M1: MACRO={ TWISS,SEQUENCE=LHCB1; };
    CONSTRAINT, EXPR= "TABLE(SUMM,DQ1)=QPRIME";
    CONSTRAINT, EXPR= "TABLE(SUMM,DQ2)=QPRIME";
    M2: MACRO={ TWISS,SEQUENCE=LHCB2; };
```

```
    CONSTRAINT, EXPR= "TABLE(SUMM,DQ1)=QPRIME";
    CONSTRAINT, EXPR= "TABLE(SUMM,DQ2)=QPRIME";
    LMDIF, CALLS=10, TOLERANCE=1.0E-21;
ENDMATCH;
```

Oliver Brüning, October, 2003; Riccardo de Maria, February, 2006.

EUROPEAN ORGANIZATION FOR NUCLEAR RESEARCH 

Matching Examples

All matching examples and the related files for executing the MADX sample jobs can be found on the 'afs' directory under:

/afs/cern.ch/group/si/slap/share/mad-X/test_suite/match/V3.02.03.

- ## Simple Periodic Cell

  Match a simple cell to given phase advances:

  ```
  FIVE-CELL
  ```

- ## Simple Periodic Cell

  Match the matrix elements of the linear transfer matrix at the end of a sequence 5 periodic cells:

  ```
  RMATRIX
  ```

- ## Transfer line with initial conditions

  Match a sequence of 5 periodic cells with initial conditions to given beta-functions at the end of the sequence:

  ```
  Transfer line
  ```

- ## Global tune matching in a sequence of 5 periodic cells

  Match the global tune of a sequence of 5 periodic cells:

  ```
  Global tune
  ```

Global tune matching for the LHC

Match the global tune for beam1 of the LHC:

```
Global tune for the LHC
```

-

Global chromaticity matching for the LHC

Match the global chromaticity for beam1 of the LHC:

```
Global chromaticity for the LHC
```

-

Global chromaticity matching for both beams of the LHC

Match the global chromaticity for beam1 and beam2 of the LHC:

```
Global chromaticity for both beams of the LHC
```

-

IR8 insertion matching for beam1 of the LHC

Match the insertion IR8 with initial conditions to given values of the optics functions at the IP and the end of the insertion:

```
IR8 insertion matching for beam1 of the LHC
```

-

IR8 insertion matching for beam1 of the LHC with upper limits on the optics functions

Match the insertion IR8 with initial conditions to given values of the optics functions at the IP and the end of the insertion while limiting the maximum acceptable beta functions over the whole insertion:

```
IR8 insertion matching for beam1 of the LHC with upper limits for all beta functions inside the insertion
```

-

Simultaneous orbit matching at IP8 for beam1 and beam2 of the LHC

Match simultaneously the orbit of beam1 and beam of the LHC at IP8 with initial conditions to the same given values at the IP:

```
Orbit matching at IP8 for beam1 and beam2 of the LHC
```

-

IR8 beta squeeze for beam1 using JACOBIAN matching routine

Try to find a beta squeeze for IR8 starting from 10 meters.

```
Beta squeeze for IR8
```

-

Mathching first and second order chromaticity of the LHC using USE_MACRO option.

Match simultaneously the first and second order chromaticity by defining macros which compute them using the TWISS command or PTC.

```
Second order chromaticity
```

-

Mathching s position using VLENGTH flag.

match the positions of elements and the total sequence length for a simple sample sequence.

```
s position matching
```

-

Mathching s position using USE_MACRO.

match the positions of elements and the total sequence length for a simple sample sequence using USE_MACRO.

```
s position matching
```

Oliver Brüning, June, 2002; Riccardo de Maria, August, 2007.

Orbit Correction

This chapter describes the commands which can be used to correct the closed orbit or a trajectory. The distorted orbit is taken from an internal or external TFS table.
Purpose of this Module:
The purpose of this orbit module is to provide some basic tools to assess the performance of an orbit correction system of a machine in the design phase.
Although some interface is available, it cannot and does not provide the full functionality expected from a dedicated online orbit correction and steering program.

- CORRECT: Correction commands and parameters
- Activate/Deactivate correctors and monitors
- READ/WRITE corrector settings
- COPTION: Global Correction Options

Werner Herr 22.10.2008

CORRECT: Orbit Correction

The CORRECT statement makes a complete closed orbit or trajectory correction using the **computed** values at the monitors from the Twiss table.
The CORRECT command has the following format (not all possible options included, some options are valid only for special algorithms):

```
CORRECT, ORBIT=myorbit,MODEL=mymodel,TARGET=mytarget,
         FLAG=ring,MODE=lsq,
         MONERROR=integer,MONON=real,MONSCALE=real,
         PLANE=x,COND=integer,RESOUT=integer,
         CLIST=file1,MLIST=file2;
```

The command CORRECT is set up with defaults which should allow a reasonable correction for most cases with a minimum of required options (see Example 1 below).
The orbit correction must always be preceded by TWISS commands which generate Twiss tables. The most recent Twiss table is assumed to contain the optical parameters and the distorted orbits.
The options used in the CORRECT command are:

- FLAG: FLAG can be "ring" or "line", either a circular machine or a trajectory is corrected.
  Default flag is "ring".
- MODE: MODE defines the method to be used for corrections.
  Available modes are LSQ, MICADO and SVD. The first performs a least squares minimization using all available correctors. The mode SVD uses a Singular Value Decomposition to compute a correction using all available correctors. The latter can also be used to condition the response matrix for the modes LSQ or MICADO (using COND=1). It is highly recommended to precede a LSQ correction by a SVD conditioning (set COND=1).
  The mode MICADO is a "best kick" algorithm. Naive use or using it with a large number of correctors (see option NCORR) can give unexpected results. To avoid the creation of local bumps, it is recommended to precede a MICADO correction by a SVD conditioning (set COND=1).
  Default mode is MICADO.
- PLANE: If this attribute is x, only the horizontal correction is made; if it is y, only the vertical correction is made. (This differs from the MAD8 implementation).
  Default plane is horizontal.
- COND: When COND is 1, a Singular Value Decomposition is performed and the response matrix CONDitioned to avoid linearly dependent correctors. This can be used to avoid creation of artificial bumps during a LSQ or MICADO correction (requires some computing time). Please note: this option is not robust since it depends on parameters which control the determination of singular values and redundant correctors. These can be set with the commands SNGVAL and SNGCUT. Both parameters depend on the machine and may need adjustment. Default values are adjusted to large machines and "reasonable" performance for smaller machines.
- NCORR: Only used by the MICADO algorithm. Defines the number of correctors to be used, unless set to 0 in which case all available correctors are used.
  Default is 0 (all available correctors).
- SNGVAL: Used to set the threshold for finding singular values with the COND command. (Hint: smaller number finds fewer singular values).
  Use with care !
  Default is 2.0
- SNGCUT: Used to set the threshold for finding redundant correctors with the COND command. (Hint: larger number finds fewer redundant correctors).

Use with extreme care !
Default is 50.0

- MONERROR: When MONERROR is 1, the alignment errors on monitors assigned by EALIGN MREX and MREY are taken into account, otherwise they are ignored.
  Default is 0.
- MONSCALE: When MONSCALE is 1, the scaling errors on monitors assigned by EALIGN MSCALX and MSCALY are taken into account, otherwise they are ignored.
  Default is 0.
- MONON: MONON takes a real number between 0.0 and 1.0. It determines the number of available monitors. If the command is given, each monitor is considered valid with a probability MONON. In the average a fraction (1.0 - MONON) of the monitors will be disabled for the correction, i.e. they are considered not existing. This allows to study the effect of missing monitors.
  Default is 1.0 (100 %).
- CORRLIM: A limit on the maximum corrector strength can be given and a WARNING is issued if it is exceeded by one or more correctors. Please note: the strengths computed by the correction algorithms are NOT limited, only a warning is printed !
  Default is 1.0 mrad.
  Normally the last active table provides the orbit to be corrected and the model for the correction. This can be overwritten by the appropriate options. Optionally, these tables can be given names like in: TWISS, TABLE=name; (see documentation on TWISS command). To use these named tables, one of the following optional parameters must be used:
- ORBIT: When this parameter is given, the orbit to be corrected is taken from a named table. The default is the last (named or unnamed) Twiss table.
- MODEL: When this parameter is given, the model for the correction is taken from a named Twiss table. The default is the last (named or unnamed) Twiss table.
- TARGET: When this parameter is given, the correction is made to a named target orbit, pre-computed with a TWISS command. Default is correction to the zero orbit.

Two attributes affect the printing of tables and results:

- CLIST=**file**: Corrector settings (in units of rad) before and after correction printed to **file**
- MLIST=**file**: Monitor readings (in units of m) before and after correction printed to **file**
- RESOUT: This command outputs the results for all monitors and all correctors in a computer readable format if its integer argument is larger than 0. The argument is added to the output. Useful to analyze runs with loops to produce large statistics.
  ATTENTION: May produce gigantic outputs for large machines.
- TWISSUM: If the argument of twissum is larger than 0, it prints maximum orbit and r.m.s. for both planes taken from the Twiss summary table in computer readable form. Allows to analyze orbits etc. at elements that are not monitors or correctors. The argument is added to the output. Only for output: no correction is made, all other commands are ignored.

Obsolete commands or options:

```
ITERATE, ITERMAX             /* Done with loop feature in MAD commands */
THREADER, THRTOL, WRORBIT    /* Not part of orbit correction module */
M1LIST, M2LIST               /* Replaced by MLIST */
C1LIST, C2LIST               /* Replaced by CLIST */
GETORBIT, PUTORBIT           /* Replaced by generic TFS access */
GETKICK, PUTKICK             /* Replaced by generic TFS access */
```

EXAMPLES (for complete MAD input files see section on examples):
Example 1 (correct orbit in horizontal plane, taken from most recent Twiss table, using default

algorithm (MICADO)):
CORRECT,PLANE=x;
Example 2 (no correction, only output of Twiss summary):
CORRECT,TWISSUM=1;
Example 3 (correct orbit in horizontal plane, corrector and monitor output on table):
CORRECT,PLANE=x,MODE=lsq,CLIST=corr.out,MLIST=mon.out;
Example 4 (correct orbit in horizontal plane, use alignment and scaling errors, 15% of orbit correctors faulty): CORRECT,PLANE=x,MONERROR=1,MONSCALE=1,MONON=0.85;

Last updated: 22.10.2008

Werner Herr 14.06.2006

Activate/Deactivate Correctors or Monitors

To provide more flexibility with orbit correction two commands are provided:

```
USEMONITOR, STATUS=flag,
            [,SEQUENCE=sequence][,RANGE=range][,CLASS=class][,PATTERN=regex]
USEKICK,    STATUS=flag,
            [,SEQUENCE=sequence][,RANGE=range][,CLASS=class][,PATTERN=regex]
```

The purpose of the two commands is:

- USEMONITOR: Activates or deactivates a selection of beam position monitors. This command affects elements of types MONITOR, HMONITOR, or VMONITOR.
- USEKICK: Activates or deactivates a selection of orbit correctors. This command affects elements of types KICKER, HKICKER, or VKICKER.

Both commands have the same attributes:

- STATUS: If this flag is true (on), the selected elements are activated. Active orbit monitor readings will be considered, and active correctors can change their strengths in subsequent correction commands. Inactive elements will be ignored subsequently.
- SEQUENCE: The sequence can be specified, otherwise the currect sequence is used for this operation.
- RANGE, CLASS, PATTERN: The usual selection commands are used to identify the elements for this operation.

Example:

```
USE,...                    ! set working beam line
...                        ! define imperfections
USEKICK,RANGE=..., OFF;    ! deactivate selected correctors
USEMONITOR,RANGE=..., OFF; ! deactivate selected monitors
CORRECT,NCORR=32;          ! uses different set of correctors
USEKICK,RANGE=..., OFF;    ! deactivate different set of correctors
CORRECT,NCORR=32;          ! uses different set of correctors
```

Werner Herr 18.6.2002

This page is under construction, options presently only available in MADX development version.

CSAVE: Write orbit correctior settings to file

SETCORR: Set orbit correctior settings

Werner Herr 18.6.2002

COPTION: Set Orbit Correction Options

The random generator for MAD is taken from [Knuth].
In the orbit program monitors can be randomly disabled and
the correct option command specifies different seeds for random values:

```
COPTION,SEED=integer,PRINT=2
```

- SEED: Selects a particular sequence of random values.
  A SEED value is an integer in the range [0...999999999] (default: 123456789).
  SEED alone continues with the current sequence
  See also: Random values.
  SEED may be an expression.
- PRINT: This flag can take integer values and controls the printout.
  In general: the higher its value the more printout is produced.
  For PRINT=0 no output is produced.
  The default value is 1 (Correction summary is given).

Example:

```
COPTION,SEED=987456321,PRINT=2;
```

Werner Herr 18.6.2002

PLOT

Values contained in MAD-X tables can be plotted in the form column versus column, with up to four differently scaled vertical axes; furthermore, if the horizontal axis is the position "s" of the elements in a sequence, then the symbolic machine can be plotted above the curves as well. In certain conditions True interpolation inside the element is available (through calls to the Twiss module for each slice) . The "environment" (interpolation, line thickness, annotation size, PostScript format) can be set with the setplot command.

-

PLOT

```
plot,
vaxis=vname1,vname2,..,vnamen,
vaxis1=vname1,vname2,..,vnamen,
vaxis2=vname1,vname2,..,vnamen,
vaxis3=vname1,vname2,..,vnamen,
vaxis4=vname1,vname2,..,vnamen,
haxis=vname,
hmin=real,hmax=real,
vmin=reals,vmax=reals,
bars=integer,
style=integer,
colour=integer,
symbol=integer,
noversion=logical,
interpolate=logical,
noline=logical,
notitle=logical,
table=table_name,
particle=particle1,particle2,..,particlen,
multiple= logical,
title=string,
range=range,
file=file_name_start,
ptc=logical,
ptc_table=table_name,
trackfile=table_name;
```

where the parameters have the following meaning:

- vaxis: one or several variables from the table to be plotted against the (only) vertical axis.
- vaxis1: one or several variables from the table to be plotted against the vertical axis number 1 (out of 4 possible ones).
- vaxis2: one or several variables from the table to be plotted against the vertical axis number 2 (out of 4 possible ones).
- vaxis3: one or several variables from the table to be plotted against the vertical axis number 3 (out of 4 possible ones).
- vaxis4: one or several variables from the table to be plotted against the vertical axis number

4 (out of 4 possible ones).

- *Important: vaxis and vaxisI are exclusive in their application!*
- haxis: name of the horizontal variable
- hmin: lower horizontal edge
- hmax: upper horizontal edge; to be used, both hmin and hmax must be given.
- vmin:lower edges of vertical axes, up to four numbers
- vmax:upper edges of vertical axes, up to four numbers; both vmin and vmax must be given for an axis to be effective.
- bars: 0 (default) or 1 - in the latter case, all curve points coming from the table are connected with the horizontal axis by vertical bars.
- style: 1 (default), 2, 3, or 4: curve style, being solid, dashed, dotted, and dot-dashed; a value of 100 makes MAD-X use these four styles in turn for successive curves in the same plot. If style is 0 no curve is printed between points. N.B. If symbol and style are null at the same time, style is forced to its default value (= 1).
- colour: 1 (default), 2, 3, , or 5: colour, being black, red, green, blue, and magenta; a value of 100 makes MAD-X use these five colours in turn for successive curves.
- symbol: 0 (default), 1, 2, 3, 4, or 5: none, dot, "+", "*", circle, and "x". These symbols are potted at all curve points; there size may have to be adapted (see below).
- noversion: logical, default=false. If set true, the information concerning the madx version and the date are suppressed from the title. This option frees more space for the user's title.
- interpolate: logical, default=false. Normally the curve points from the table are connected by straight lines; if "interpolate" is requested, then on-momentum Twiss parameters such as beta, alfa, and dispersion are interpolated with calls to the Twiss module inside each element, for all other variables splines are used to smooth the curves.
- noline: logical, default=false. If s is the horizontal variable, then the machine will be plotted in symbolic form above the curve plot (except for tables having been read back into MAD-X). This may result in a thick black block if the horizontal scale is too large. "noline" allows the user to suppress the machine plotting.
- notitle: logical, default=false. If true, suppresses the title line.
- table: name of the table to be plotted from (default: twiss). If it is *track*, the data to be plotted are taken from the tracking files generated for each required particle as defined by the attribute *particle*. The name of this file has the following format: file name as defined by the attribute *trackfile*, the observation point fixed to 1 and the particle number, e.g. *testtrack.obs0001.p0003*. If the required file has not been generated by the previous MAD-X command track, no plot is done for that particle. The plot is obtained through the *gnuplot* package and is available as a postscript file whose name is defined by the attribute *file_name*. N.B. the previous track command should contain the attributes *onepass* and *dump*. The plot routine for tracking plots, in opposition to the optic ones, appends the plots to the existing files having the same names instead of creating new ones.
- particle: one or several numbers associated to the tracked particles for which the specified plot has to be displayed.
- multiple: logical, default=false. If true all the curves generated for each tracked particle are put on one plot. Otherwise there will be one plot for each particle.
- title: plot title string; if absent, the last overall title is used; if no such overall title as well, the sequence name is used.
- range: horizontal plot range given by elements.
- file_name: start of the file name for the Postscript file(s). Only the first occurrence of such a name will be used. Default is "madx" or "madx_track" if the *table* attribute is track. Depending on the format (.ps or .eps, see below) the plots will either all be written into one file

file_name.ps, or one per plot into file_name01.eps, file_name02.eps, etc.

- ptc: logical, default=false. If set true, the data to be plotted are taken from the table defined by the attribute *ptc_table* which is expected to be generated previously by the ptc package. The data belong to the column identified by one of the names set in the definition of the ptc twiss table. Interpolation is not available and the attribute *interpolate* has no effect.
- ptc_table: name of the ptc twiss table to be plotted from (default: ptc_twiss)
- trackfile: first part of the name of the files containing tracking data for each particle (default: track)

SETPLOT

```
setplot, post=integer,font=integer,
lwidth=real,xsize=real,ysize=real,
ascale=real, lscale=real, sscale=real, rscale=real;
```

where the parameters have the following meaning:

- post: default = 1. If =1, makes one PostScript file (.ps) with all plots; if =2, makes one Encapsulated PostSscript file (.eps) per plot.
- font: there are two defaults: 1 for screen plotting: this uses characters made from polygons; -1 for PostScript files; this is Times-Italic. There are various fonts available for positive and negative integers, best to be tried out, since they will look different on different systems anyway. GhostView will show strange vertical axis annotations, but the printed versions are normally OK.
- lwidth: default = 1. Allows the user to set the curve line width. Depends on the system as well, so to be tried out.
- xsize: bounding box size for PostScript, default=27 cm.
- ysize: bounding box size for PostScript, default=19 cm.
- ascale: annotation character height scale factor, default=1.
- lscale: axis label character height scale factor, default=1.
- sscale: curve symbol (see above) scale factor, default=1.
- rscale: axis text character height scale factor, default=1.

RESPLOT

```
resplot;
```

resets all defaults for the setplot command.

hansg, June 17, 2002, rdemaria rdemaria, September 2007.

EUROPEAN ORGANIZATION FOR NUCLEAR RESEARCH 

SODD

This command will execute the Second Order Detuning and Distortion as described in the paper of J. Bengtsson and J. Irwin "Analytical Calculation of Smear and Tune Shift " (SSC-232, February 1990), on the beam line defined by the last USE command followed by a TWISS command. It is based on the stand-alone program written by Frank Schmidt in November 1998 - January 1999 who also extended the analytical computation to the second order distortion (cfr. Beam Physics Note 60 F. Schmidt "SODD: A physics Guide"). It consists of three parts:

○

Subroutine detune (launched by the attribute detune)

It calculates the detuning function terms in first and second order in the strength of the multipoles. If the attribute print_at_end has been set, the following two files (and the corresponding madx tables) are created :

detune_1_end containing five columns :

1) 'multipole order', 2) '(hor., ver. plane => (1/2)', 3) 'hor. or ver. detuning', 4) 'order of horizontal invariant', 5) 'order of vertical invariant'.

detune_2_end containing five columns :

1) 'first multipole order', 2) 'second multipole order', 3) 'horizontal detuning', 4) 'order of horizontal invariant', 5)'order of vertical invariant'.

If the attribute print_all has been set, the following two files (and the corresponding madx tables) are created :

detune_1_all containing five columns :

1) 'multipole order', 2) '(hor., ver. plane => (1/2)', 3) 'hor. or ver. detuning', 4) 'order of horizontal invariant', 5)'order of vertical invariant'.

detune_2_all containing five columns :

1) 'first multipole order', 2) 'second multipole order', 3) 'horizontal detuning', 4) 'order of horizontal invariant', 5) 'order of vertical invariant'.

○

Subroutine distort1 (launched by the attribute distort1)

It calculates the distortion function and the Hamiltonian terms in first order in the strength of the multipoles. If the attribute print_at_end has been set, the two files (and the corresponding madx tables) are created :

distort_1_F_end containing eight columns :

> 1) 'multipole order', 2) 'cosine part of distortion', 3) 'sine part of distortion', 4) 'amplitude of distortion', 5) 'j', 6) 'k', 7) 'l', 8) 'm'.

distort_1_H_end containing eight columns :

> 1) 'multipole order', 2) 'cosine part of Hamiltonian', 3) 'sine part of Hamiltonian', 4) 'amplitude of Hamiltonian', 5) 'j', 6) 'k', 7) 'l', 8) 'm'.

If the attribute print_all has been set, the following two files (and the corresponding madx tables) are created :

distort_1_F_all containing eleven columns :

> 1) 'multipole order', 2) 'appearance number in position range', 3) 'number of resonance', 4) 'position', 5) 'cosine part of distortion', 6) 'sine part of distortion', 7) 'amplitude of distortion', 8) 'j', 9) 'k', 10) 'l', 11) 'm'.

distort_1_H_all containing eleven columns :

> 1) 'multipole order', 2) 'appearance number in position range, 3) 'number of resonance', 4) 'position', 5) 'cosine part of Hamiltonian', 6) 'sine part of Hamiltonian', 7) 'amplitude of Hamiltonian', 8) 'j', 9) 'k', 10) 'l', 11) 'm'.

○

Subroutine distort2 (launched by the attribute distort2)

It calculates the distortion function and Hamiltonian terms in second order in the strength of the multipoles. If the attribute print_at_end has been set, the following two files (and the corresponding madx tables) are created :

distort_2_F_end containing nine columns :

> 1) 'first multipole order',2) 'second multipole order', 3) 'cosine part of distortion', 4) 'sine part of distortion', 5) 'amplitude of distortion', 6) 'j', 7) 'k', 8) 'l', 9) 'm'.

distort_2_H_end containing nine columns :

> 1) 'first multipole order', 2) 'second multipole order', 3) 'cosine part of Hamiltonian', 4) 'sine part of Hamiltonian', 5) 'amplitude of Hamiltonian', 6) 'j', 7) 'k', 8) 'l', 9) 'm'.

N. B. The first row of every file is a header containing the names of the columns. This row is absent in the internal tables.

SODD

```
sodd,
detune=logical,
distort1=logical,
distort2=logical,
start_stop = start,stop
multipole_order_range = fist,last
noprint = logical
print_all = logical
print_at_end = logical
nosixtrack  = logical
```

where the parameters have the following meaning:

- detune : logical, default=false. If true, the detune subroutine is executed.
- distort1 : logical, default=false. If true, the distort1 subroutine is executed.
- distort2 : logical, default=false. If true, the distort2 subroutine is executed.
- start_stop : longitudinal interval of the beam line (in m). start and stop should be given as real numbers.
- multipole_order_range : the lowest and the largest multipole order which will be taken in account. first and last should be given as integers.
- noprint : logical, default=false. If true, no file or internal table will be created to keep the results. In this case the attributes print_all or print_at_end have no effect.
- print_all : logical, default=false. If true, the files and internal tables containing results at each multipole will be generated.
- print_at_end : logical, default=false. If true, the files and internal tables containing results at the end of the position range will be generated.
- nosixtrack : logical, default=false. If true, the input file fc.34 will not be generated internally by invoking the conversion routine of sixtrack and the user should provide it before the execution of the sodd command.

- A more detailed description can be found in

  AB-note-2004-069

  damico, September 10, 2004

EUROPEAN ORGANIZATION FOR NUCLEAR RESEARCH 

GEOMETRIC LAYOUT

The SURVEY command computes the coordinates of all machine elements in a global reference system. These coordinates can be used for installation. In order to produce coordinates in a particular system, the initial coordinates and angles can be specified. The computation results are written on an internal table (survey) and can be written on an external file. Each line contains the coordinates at the end of the element.
The last "USEd" sequence is used except if another one is specified.

WARNING : in the case a machine geometry is constructed with thick lenses, the circumference will change if the structure is converted into thin lenses (via the makethin command). This is an unavoidable feature. ONLY the structure with thick lenses must be used for practical purposes.
INFORMATION : The skew dipole component of a MULTIPOLE element (MULTIPOLE, KSL={FLOAT}) is NOT taken into account in the survey calculation. You should use a tilted normal MULTIPOLE or BEND instead.

The survey calculation is launched by a single command line with the following syntax :

SURVEY, x0=double, y0=double, z0=double, theta0=double, phi0=double, psi0=double,
file=string, table=string, sequence=string;

parameter	meaning	default value
x0	initial horizontal transverse coordinate	0.0
y0	initial vertical transverse coordinate	0.0
z0	initial longitudinal coordinate	0.0
theta0	initial horizontal angle	0.0
phi0	initial vertical angle	0.0
psi0	initial transverse tilt	0.0
file	name of external file	null (default name survey)
table	name of internal table	null (default name survey)
sequence	name of sequence to be surveyed	last used sequence

Example : average LHC ring with CERN coordinates.

```
REAL CONST R0 = 1.0;              !  to obtain the average ring
OPTION, -echo, -info;
CALL, file="V6.4.seq.070602";     !  follow this link for the file
OPTION, echo;
```

```
BEAM, particle=proton, energy=450, sequence=lhcb1;
USE, period=lhcb1;
! SELECT, flag=survey,clear;            !  uncomment if the optional select below is used
! optional SELECT to specify a class and the output columns
! SELECT, flag=survey, class=marker, column=name,s,psi;

SURVEY, x0=-2202.21027, z0=2710.63882, y0=2359.00656, theta0=-4.315508007,
phi0=0.0124279564, psi0=-0.0065309236, file=survey.lhcb1;
WRITE, table=survey;                    !   to display the results immediately
STOP;
!**********      The external file "survey.lhcb1" can now be read **********
F.Tecker, March 2006
```

SXF file input and output

The command

```
SXFWRITE,FILE=filename;
```

writes the currently (i.e. last) USEd sequence with all alignment and field errors in [SXF] format onto the file specified. This then represents one "instance" of the sequence, where all parameters are given by numbers rather than expressions; the file can be read by other programs to get a complete picture of the sequence.

The command

```
SXFREAD,FILE=filename;
```

reads a file in SXF format, stores the sequence away and USEs it(!) in order to keep the existing errors. The following does therefore work:

Example:

```
job 1:

! define sequence MYSEQU

use,mysequ;

! add alignment errors and field errors

sxfwrite,file=file;
stop;

job 2:

sxfread,file=file;
twiss;
stop;
```

hansg, January 24, 1997

EUROPEAN ORGANIZATION FOR NUCLEAR RESEARCH 

TFS File Format

[TFS] files (Table File System) have been used in the LEP control system. The MAD program knows only coded TFS files. The TFS format has been chosen for all table output of MAD-X. TFS formatted tables can be read back into MAD-X, and may then be further processed.

- Descriptor Lines
- Column Formats
- TFS file example

hansg, June 17, 2002

EUROPEAN ORGANIZATION FOR NUCLEAR RESEARCH 

Descriptor Lines

MAD-X writes the following descriptors on all tables:

- COMMENT: The current title string from the most recent TITLE command.
- ORIGIN: The version of MAD-X used.
- DATE: The date of the MAD-X run.
- TIME: The wall clock time of the MAD-X run.
- TYPE: The type of the table: e.g. TWISS

Additional descriptors exist in the Twiss table, as well as the Track tables.

hansg, June 17, 2002

EUROPEAN ORGANIZATION FOR NUCLEAR RESEARCH 

Twiss TFS file header

The format of the twiss table is best illustrated with an TFS file example.

It should be mentioned that MAD-X allows to access parameters from twiss and other tables using the table access function.

hansg, June 17, 2002

Column Formats

The column formats used are listed in the TFS columns table.

Table: 1 Column Formats used in TFS Tables

C format	Meaning	C format
%hd	Short integer	(%8d)
%le	Long float	(%-18.10g)
%ks	String of length k	("\"-18s\"") **Table: 1**Column Formats used in TFS Tables

Control lines begin with the TFS control character, followed by a blank. Data lines begin with two blanks. Columns are also separated by one blank character. The column width is chosen such as to accommodate the large of the column name and the data values of the column.

hansg, June 17, 2002

EUROPEAN ORGANIZATION FOR NUCLEAR RESEARCH 

TOUSCHEK: Touschek Lifetime and Scattering Rates

The TOUSCHEK module computes the Touschek lifetime and the scattering rates around a lepton or hadron storage ring, based on the formalism of Piwinski [A. Piwinski, "The Touschek Effect in Strong Focusing Storage Rings," DESY-98-179; see also Piwinski's article on Touschek lifetime in the Handbook of Accelerator Physics and Engineering (A. Chao, M. Tigner, eds.), World Scientific, 1999] .

The syntax of the TOUSCHEK command is:

```
TOUSCHEK, FILE;
```

TOUSCHEK should be called after a TWISS command. One or several cavities with rf voltages should be defined prior to calling TWISS and TOUSCHEK. [Warning: Calling EMIT between the TWISS and TOUSCHEK commands leads to TOUSCHEK using wrong beam parameters, even if the BEAM command is reiterated.]

The momentum acceptance is taken from the bucket size taking into account the energy loss per turn *U0* from synchrotron radiation. The value of *U0* is computed from the second synchrotron radiation integral *synch_2* in the TWISS summ table (*synch_2* is calculated only when the TWISS option 'chrom' is invoked), using Eq. (3.61) in Matt Sands' report SLAC-121, which was generalized to the case of several harmonic rf systems. If *synch_2=0*, not defined, or not calculated, zero energy loss is assumed.

In the case of several rf systems with nonzero voltages, it is assumed that the lowest frequency system defines the phase of the outer point on the separatrix when calculating the momentum acceptance, and that all higher-harmonic systems are either in phase or in anti-phase to the lowest frequency system. (Note: if a storage rings really uses a different rf scheme, one would need to change the acceptance function in the routine *cavtousch* for that ring.)

The arguments have the following meaning:

- FILE: The name of the output file (default: 'touschek')

Example:

```
BEAM,PARTICLE=PROTON,ENERGY=450,NPART=1.15e11,EX=7.82E-9,EY=7.82E-9,ET=5.302e-5,SIGE=7.164e-4,SIGT=0.1124,RADIATE=TRUE;

...

USE,PERIOD=FODO;

...
```

```
VRF=400;

...

SELECT,FLAG=TWISS,CLEAR;
TWISS,CHROM,TABLE,FILE;

TOUSCHEK,FILE;

...
```

The first command defines the beam parameters. It is essential that the longitudinal emittances and bunch length are set. The command *use* selects the beam line or sequence. The next command assign a value to the cavity rf voltage vrf (example name). The *select* clear previous assignments to the *twiss* module, *twiss* calculates and saves the values of all twiss parameters for all elements in the ring; the *touschek* command computes the Touschek lifetime and writes it to the file 'touschek' (default name).

The results are stored in the *TOUSCHEK* tables, and can be written to a file (with the default name 'touschek' in the example above), or values can be extracted from the table using the value command as follows

value,table(touschek,name),table(touschek,s),table(touschek,tli),table(touschek,tliw),table(touschek,tlitot);

where 'name' denotes the name of a beamline element, *s* the position of the center of the element, *tli* the instanteneous Touschek loss rate within the element, and *tliw* the instantaneous rate weighted by the length of the element divided by the circumference (its contribution to the total loss rate), and *tlitot* the accumulated loss rate adding the rates over all beamline elements through the present position. The value of *tlitot* at the end of the beamline is the inverse of the Touschek lifetime in units of 1/s.

Also, all results can be printed to a file using the command

write,table=touschek,file;

The MADX Touschek module was developed by Catia Milardi and Frank Zimmermann .

frankz 11.03.2008

EUROPEAN ORGANIZATION FOR NUCLEAR RESEARCH 

Twiss Module

The TWISS command causes computation of the [Courant and Snyder] linear lattice functions, and optionally of the chromatic functions. The coupled functions are calculated in the sense of [Edwards and Teng]. For the uncoupled cases they reduce to the C and S functions. It operates on the working beam line defined in the latest USE command. One can also specify either a SEQUENCE="sequence_name" or a LINE="line_name" on the TWISS command. Moreover, one can restrict the TWISS calculation to a desired RANGE.

The relative energy error DELTAP may be entered in one of the 2 forms

```
DELTAP=real{,real}DELTAP=initial:final:step
```

The first form lists several numbers, which may be general expressions, separated by commas. The second form specifies an initial value, a final value, and a step, which must be constant expressions, separated by colons.

Examples:

```
DELTAP=0.001                ! a single valueDELTAP=0.001,0.005          ! two valuesDELTAP=0.001:0.007:0.002       ! four values
```

If DELTAP is missing, MAD-X uses the value 0.0.

Further attributes of the TWISS statements are:

- CHROM: A logical flag. If set, MAD-X also computes the chromatic functions.
- FILE: If FILE="file_name" appears MAD-X writes a full TFS Twiss table Example TFS Twiss table on the disk file "file_name". FILE alone is equivalent to FILE="twiss":
- TABLE (overrides SAVE): MAD-X creates a full Twiss table in memory and gives it the name TWISS, unless TABLE="table_name" appears on the command, then it is called table_name. This table includes linear lattice functions as well as the chromatic functions for all positions. An important new feature of MAD-X is the possibility to access entries of tables and in particular the twiss table (see table access).
- SUMM: After a successful TWISS run MAD-X creates a table of summary parameters including tunes, chromaticities etc versus the selected values of DELTAP. Notice that in MAD-X DELTAP is converted in PT, which is used as longitudinal variable. Dispersive and chromatic functions are hence derivatives with respects to PT(see table). These parameters can later be accessed via the table access function using "summ" as table name.
- CENTRE: This flag enforces the calculation of the linear lattice functions at the center of the element instead of the end of it.
- RMATRIX: If this flag is used the the one-turn map at the location of every element is calculated and prepared for storage in the TWISS table. Using the SELECT command and using the column RE, RE11...RE16...RE61...RE66 these components will be added to the TWISS table, i.e. with "column, RE" and "column, REij" one gets all or the component "ij" respectively.
- SECTORMAP: This flag initiates the calculation of a sector map as described at: SECTORMAP.
- SECTORFILE: Used to write SECTORMAPs to the file SECTORFILE="file_name", if missing

the output of SECTORMAP will go to the file "sectormap" with the format as found in SECTORMAP.

- KEEPORBIT: The keeporbit attribute (with an optional name, keeporbit="name") stores the orbit under this name at the start, and at all monitors.
- USEORBIT: The useorbit attribute (with an optional name, useorbit="name") uses the start value provided for the closed orbit search; the values at the monitors are used by the threader.
- *COUPLE (obsolete)* : This MAD8 option can no longer be set since TWISS in MAD-X is always calculated in coupled mode. MAD-X computes the coupled functions in the sense of [Edwards and Teng]. For the uncoupled cases they reduce to the C and S functions.
- *Twiss calculation is 4D only!* : The Twiss command will calculate an approximate 6D closed orbit when the accelerator structure includes an active cavity. However, the calcuation of the Twiss parameters are 4D only. This may result in apparently non-closure of the beta values in the plane with non-zero dispersion. The full 6D Twiss parameters can be calculated with the ptc_twiss command.

The tables are suitable for plot.

Twiss Parameters for a Period

The simplest form of the TWISS command is

```
TWISS, DELTAP=real{,value},CHROM,       TABLE=table_name;
```

It computes the periodic solution for the specified beam line for all values of DELTAP entered (or for DELTAP = 0, if none is entered).

Example:

```
USE,period=OCT;TWISS,DELTAP=0.001,CHROM;
```

This example computes the periodic solution for the linear lattice and chromatic functions for the beam line OCT. The DELTAP value used is 0.001. Apart from saving computing time, it is equivalent to the command sequence

```
RING: LINE=(4*(OCT,-OCT));      USE,period=RING;      TWISS,DELTAP=0.001,CHROM;
```

Initial Values from a Periodic Line

It is possible to track the lattice functions starting with the periodic solution for another beam line. If this is desired the TWISS command takes the form

```
TWISS, DELTAP=real{,value},LINE=beam-line,       MUX=real,MUY=real,       TABLE=table_name;
```

No other attributes should appear in the command. For each value of DELTAP MAD-X first searches for the periodic solution for the beam line mentioned in LINE=beam-line: The result is used as an initial condition for the lattice function tracking.

Example:

```
CELL:   LINE=(...);INSERT: LINE=(...);       USE,period=INSERT;       TWISS,LINE=CELL,DELTAP=0.0:0.003:0.001,CHROM,FILE;
```

For four values of DELTAP the following happens: First MAD-X finds the periodic solution for the beam line CELL: Then it uses this solution as initial conditions for tracking the lattice functions of the beam line CELL: Output is also written on the file TWISS:

If any of the beam lines was defined with formal arguments, actual arguments must be provided:

```
CELL(SF,SD): LINE=(...);INSERT(X):   LINE=(...);            USE,period=INSERT;            TWISS,LINE=CELL(SF1,SD1);
```

Given Initial Values

Initial values for linear lattice functions and chromatic functions may also be numerical. Initial values can be specified on the TWISS command:

```
TWISS,  BETX=real,ALFX=real,MUX=real,     BETY=real,ALFY=real,MUY=real,     DX=real,DPX=real,DY=real,DPY=real,     X=real,PX=real,Y=real,PY=real,     T=real,PT=real,     WX=real,PHIX=real,DMUX=real,     WY=real,PHIY=real,DMUY=real,     DDX=real,DDPX=real,DDPY=real,     R11=real,R12=real,R21=real,R22=real,  !coupling matrix     TABLE=table_name,     TOLERANCE=real,     DELTAP=real:=real;
```

All initial values for linear lattice functions and chromatic functions are permitted, but BETX and BETY are required. Moreover, a beta0 block can be added as filled by the savebeta command or see below. The lattice parameters are taken from this block, but will be overwritten by explicitly stated lattice parameters. As entered, the initial conditions cannot depend on DELTAP, and can thus be correct only for one such value.

Tolerance

This value defines the maximum closed orbit error of all six orbit components during the closed orbit search. The default value is 1.e-6. The value is only valid for the current twiss command; a permanent value can be entered via the COGUESS command.

SAVEBETA: Save Lattice Parameters

Initial lattice parameters can be transfered for later commands, in particular for twiss or the match module, by using the savebeta command sequence.

It should be mentioned that parameters can be also accessed from tables using the table access function.

```
USE,period=...;SAVEBETA,LABEL=name,PLACE=place,SEQUENCE=s_name;TWISS,...;
```

When reaching the place in the sequence "s_name" during execution of TWISS, MAD-X will save a beta0 block with the label name: This block is filled with the values of all lattice parameters in place. Example 1:

```
USE,period=CELL;SAVEBETA,LABEL=END,PLACE=#E,SEQUENCE=CELL;TWISS;USE,period=INSERT;TWISS,BETA0=END;
```

This first example calculates the periodic solution of the line CELL, and then track lattice parameters through INSERT, using all end conditions (including orbit) in CELL to start.

Example 2:

```
USE,period=CELL;SAVEBETA,LABEL=END,PLACE=#E,SEQUENCE=CELL;TWISS;USE,period=INSERT;TWISS,BETX=END->BETY,BETY=END->BETX;

This is similar to the first example,but the beta functions are interchanged (overwritten).
```

---

frs, 06-Apr-2003. Revised in February 2007.

EUROPEAN ORGANIZATION FOR NUCLEAR RESEARCH 

PTC Set-up Parameters

The E. Forest s Polymorphic Tracking Code (PTC) is a kick code, allowing a symplectic integration through all accelerator elements giving the user full control over the precision (number of steps and integration type) and exactness (full or extended Hamiltonian) of the results. The degree of exactness is determined by the user and the speed of his computer. The main advantage is that the code is inherently based on the map formalism and provides users with all associated tools.

The PTC code is actually a library that can be used in many different ways to create an actual module that calculates some property of interest. Several modules using the PTC code have been presently implemented in MAD-X. These MADX-PTC modules [b] are executed by the following commands: ptc_twiss, ptc_normal, ptc_track, ptc_track_line. To perform calculations with these MADX-PTC commands, the PTC environment must be initialized, handled and turned off by the special commands within the MAD-X input script.

Synopsis

```
PTC_CREATE_UNIVERSE, sector_nmul_max=integer,
sector_nmul=integer;
PTC_CREATE_LAYOUT, model=integer, method=integer, nst=integer,
[exact], offset_deltap=double;
.......................
PTC_MOVE_TO_LAYOUT;
.......................
PTC_ALIGN;
.......................
SELECT_PTC_NORMAL, .....;
PTC_NORMAL,.....;
.......................
PTC_TWISS, .....;
.......................
PTC_TRACK, .....;
PTC_TRACK_END;
.......................
PTC_TRACKCAVS;
.......................
PTC_END;
```

Commands

***PTC_CREATE_UNIVERSE*;**

sector_nmul_max=integer, **sector_nmul**=integer

Description

The *"PTC_CREATE_UNIVERSE"* command is needed to set-up the PTC environment.

Options

Option	Meaning	Default Value	Value Type
SECTOR_NMUL_MAX	Global variable in PTC needed for exact sector bends defining up to which order Maxwell's equation are solved [a, page 76-77]. The value of SECTOR_NMUL_MAX must not be smaller than SECTOR_NMUL otherwise MAD-X stops with an error.	10	integer
SECTOR_NMUL	Global variable in PTC needed for exact sector bends defining up to which order the multipole are included in solving Maxwell's equation up to order SECTOR_NMUL_MAX. Multipoles of order N with N > SECTOR_NMUL and N ≤ SECTOR_NMUL_MAX are treated a la SixTrack.	10	integer

PTC_CREATE_LAYOUT,
time=logical, **model=**integer, **method=**integer,
nst=integer, **exact**, **offset_deltap=**double;

Description

The *"PTC_CREATE_LAYOUT"* command creates the PTC-layout according to the specified integration method and fills it with the current MAD-X sequence defined in the latest USE command.

The logical input variable time controls the coordinate system that is being used.

Options

Option	Meaning		Default Value	Value Type										
TIME	5D	"time=true": fifth coordinate is PT, p_t=$\Delta E/p_0 c$;	.TRUE.	logical										
		"time=false": fifth coordinate is DELTAP, δ_p=$\Delta p/p_0$												
	6D	"time=true": MAD-X coordinate system {-ct, p_t}												
		"time=false": second PTC coordinate system {-pathlength, δ_p}												
MODEL	Type of element: 1, 2,or 3.		1	integer										
METHOD	Integration order (2, 4, 6) [a, Chapter K]		2	integer										
NST	Number of integration steps: 1, 2, 3, .		1	integer										
EXACT	Switch to turn on calculations with an exact Hamiltonian, otherwise the expanded Hamiltonian is used.		.FALSE.	logical										
OFFSET_DELTAP	Beware: Expert flag! The relative momentum deviation of the reference particle (6D case ONLY). This option implies "totalpath=true".		0.0	double										

ERRORS_IN	Flag to read-in multipolar errors in Efcomp table format via:
	readmytable, file=Your_Error_File, table=errors_read;
	The table "errors_read" can be conveniently filled with files create in a preceding MAD-X run using "ERRORS_OUT" (see below). This "ERRORS_IN" flag has precedence over the "ERRORS_OUT" flag.
		.FALSE.	logical	ERRORS_OUT	Flag to write-out multipolar errors in Efcomp table format. Two tables are filled "errors_field" and "errors_total". In the first case only field errors are written out and in the second one also desired field components are added. The latter is useful e.g. to include the strength of correctors. The choice of magnets is defined by the "magnet_name" attribute (see below). As usual the tables can be written to files for later use for read-in via the "ERRORS_IN" flag:
					write, table=errors_field,file=Your_Errors_Field_File;
					write, table=errors_total,file=Your_Errors_Total_File;
					The "ERRORS_IN" flag has precedence over this "ERRORS_OUT" flag.
						.FALSE.	logical	MAGNET_NAME	Simple selection for the names of magnet to be used for an error write-out using the "ERRORS_OUT" flag (see above). In fact, the errors are recorded for all magnets with their name starting with the exact string of "MAGNET_NAME".	NULL	string

Remarks

TIME: at small energy ($\beta_0 << 1$), momentum-dependent variables like dispersion will depend strongly on the choice of the logical input variable "time". In fact, the derivative $(\partial/\partial\delta_p)$ and $(\partial/\partial p_t)$ are different by the factor β_0. One would therefore typically choose the option "time=false", which sets the fifth variable to the relative momentum deviation δ_p.

MODEL: 1 for "Drift-Kick-Drift"; 2 for "Matrix-Kick-Matrix"; 3 for "Delta-Matrix-Kick-Matrix" (SixTrack-code model).

NST: sets the same value for all "thick" elements ($l > 0$) of a beam-line. Please note, that each individual element may have its own NST value (see below).

PTC_MOVE_TO_LAYOUT,
index=integer;

Description

Several PTC layouts can be created within a one PTC-"universe". The layouts are automatically numbered with sequential integers by the MAD-X code. The *"PTC_MOVE_TO_LAYOUT"* is used for an activation of a requested layout and the next PTC commands will be applied to this active PTC layout until a new PTC layout will be created or activated.

Option

Option	Meaning	Default Value	Value Type
INDEX	Number of the PTC layout to be activated.	1	integer

PTC_ALIGN;

Description

The *"PTC_ALIGN"* command is used to apply the MAD-X alignment errors to the current PTC layout.

PTC_END;

Description

The *"PTC_END"* command is turning off the PTC environment, which releases all memory back to the MAD-X world proper;

Additional Options for Physical Elements

[SBEND | RBEND | QUADRUPOLE | SEXTUPOLE | OCTUPOLE | SOLENOID],
l=double**, ...,tilt**=double**, ..., nst**=integer**, ...,**
knl:={0, double, double,..}, **ksl**:={0, double, double,..}**;**

Description

1. The full range of normal and skew multipole components on the bench can be specified for the following physical elements: *sbend, rbend, quadrupole, sextupole, octupole and solenoid.* Multipole coefficients are specified as the integrated value "$\int K\,ds$" of the field components along the magnet axis (see the table below). These multipole components in PTC are spread over a whole element, if $l > 0$. This is a considerable advantage of PTC input compare to MAD-X which allows only thin multipoles.
2. To preserve the reference orbit of straight elements, dipole components for those elements are ignored, knl(0)=0, ksl(0)=0.
3. Individual NST values for a particular "thick" element ($l > 0$) can be specified. For example, in MAD-X any RF cavity is represented by a single kick, while PTC splits the RF cavity into (global) NST segments. In this way, PTC considers properly transit-time effects of the cavity. In case, one wants to reproduce the approximate results of MAD-X, one can use NST=1 for RF cavity in PTC.

Multipoles on Bench (PTC only)

Option	Meaning	Default Value	Value Type
KNL	The normal multipole coefficient	0 [m^{-1}]	double array
KSL	The skew multipole coefficient	0 [m^{-1}]	double array

Remarks

Length l: Bending magnets (*sbend, rbend*) are treated as "markers", if $l = 0$.
Additional Field Errors: A full range of multipole field errors can be additionally specified with EFCOMP command. Errors are added to the above multipole fields on the bench.

Caution

A user has to understand that PTC exists inside of MAD-X as a library. MAD-X offers the interface to PTC, i.e. the MAD-X input file is used as input for PTC. Internally, both PTC and MAD-X have their own independent databases which are linked via the interface.
With the *"PTC_CREATE_LAYOUT"* command, only numerical numbers are transferred from the MAD-X database to the PTC database.
Any modification to the MAD-X database is ignored in PTC until the next call to *"PTC_CREATE_LAYOUT"*.
For example, a deferred expression of MAD-X after a *"PTC_CREATE_LAYOUT"* command is ignored within PTC.

Examples

Examples for any MADX-PTC module contain the above PTC set-up commands.

References

a) E. Forest, F. Schmidt and E. McIntosh, Introduction to the Polymorphic Tracking Code , CERN-SL-2002-044-AP, KEK report 2002-3, July 2002.
b) F. Schmidt, "‘MAD-X PTC Integration’’, Proc. of the 2005 PAC Conference in Knoxville, USA, pp.1272.

See Also

ptc_twiss, ptc_normal, ptc_track, ptc_track_line.

V. Kapin (ITEP) and F. Schmidt, March 2006

EUROPEAN ORGANIZATION FOR NUCLEAR RESEARCH 

Overview of MAD-X Tracking Modules

A number of particles with given initial conditions can be tracked through a beam-line or a ring. The particles can be tracked either for a single passage or for many turns.

While MAD-X [a] is keeping most of the functionality of its predecessor MAD-8, the trajectory tracking in MAD-X is considerably modified comparing to MAD-8 . The reason is that in MAD8 the thick lens tracking is inherently not symplectic, which implies that the phase space volume is not preserved during the tracking, i.e. contrary to the real particle the tracked particle amplitude is either growing or decreasing.

The non-symplectic tracking as in MAD-8 has been completely excluded from MAD-X by taking out the thick lens part from the tracking modules. Instead two types of tracking modules (both symplectic) are implemented into MAD-X.

The first part of this design decision is the thin-lens tracking module (*thintrack*) which tracks symplecticly through drifts and kicks and by replacing the end effects by their symplectic part in form of an additional kick on either end of the element. This method demands a preliminary conversion of a sequence with thick elements into one composed entirely of thin elements (see the *MAKETHIN* command). The details of its usage are given on the page "thintrack".

The second part of this design decision is to produce a thick lens tracking module based on the PTC code [b] that allows a symplectic treatment of all accelerator elements giving the user full control over the precision (number of steps and integration type) and exactness (full or extended Hamiltonian) of the results.

The first PTC thick-lens tracking module is named ptc_track. It has the same features as the thin-lens tracking code (*thintrack*) except it treats thick-lenses in a symplectic manner.

There is a second PTC tracking module called the line tracking module (ptc_track_line). It is meant for tracking particles in CLIC, in fact it treats beam-lines containing traveling-wave cavities and includes a beam acceleration.

References

a) F. Schmidt, "‘MAD-X PTC Integration’’, Proc. of the 2005 PAC Conference in Knoxville, USA, pp.1272.
b) E. Forest, F. Schmidt and E. McIntosh, Introduction to the Polymorphic Tracking Code, CERN-SL-2002-044-AP, KEK report 2002-3, July 2002.

See Also

PTC Set-up Parameters

V. Kapin (ITEP) and F. Schmidt, March 2006

EUROPEAN ORGANIZATION FOR NUCLEAR RESEARCH 

Thin-Lens Tracking Module (thintrack)

The **thin-lens tracking module** of MAD-X performs element per element tracking of (one to many) particle trajectories in the last *use*d sequence. Only thin elements are allowed (apart from the element *drift*), which guarantees the symplecticity of the coordinate transformation. Any lattice can be converted into a "thin element" lattice by invoking the *makethin* command.

Synopsis

```
TRACK, onepass, deltap= double, dump;
  START, x= double, px= double, y= double, py= double, t= double,
pt= double;
  RUN, turns= integer;
ENDTRACK;
```

Commands

***TRACK*, *deltap*= double, *onepass*, *dump*, *onetable*, *file*= string**; (MAD-X version 1)
***TRACK*, *deltap*= double, *onepass*, *damp*, *quantum*, *dump*, *aperture*, *onetable*, *file*= string**; (MAD-X version 2)
***TRACK*, *deltap*= double, *onepass*, *damp*, *quantum*, *dump*, *aperture*, *onetable*, *recloss*, *file*= string**; (MAD-X version 3)
[commands];
ENDTRACK;

Description

The *TRACK* command initiates trajectory tracking by entering the thin-lens tracking module. Several options can be specified, the most important being *dump*, *deltap* and *aperture*.
Inside the block *TRACK-ENDTRACK* a series of initial trajectory coordinates can be specified by the *START* command (as many commands as trajectories). This will be usually done in a *while*-loop. **Note** that the coordinates are either **canonical** coordinates or **action-angle** variables!

- For usual tracking (single/multi-turn), all coordinates are specified with respect to the actual closed orbit (possibly off-momentum, with magnet errors) and **NOT** with respect to the reference orbit.
- If the option *onepass* is used, the coordinates are specified with respect to the reference orbit. The name "onepass" might be misleading: Still tracking can be single- or multi-turn!

The tracking is actually started with the *RUN* command, where the option *turns* defines for how many turns the particles will be tracked in the given sequence.
If the option *dump* is used, the particle coordinates are written to files at each turn. The output files are named automatically. The name given by the user is followed by .obsnnnn(observation point), followed by .pnnnn(particle number). Hence filenames look like `track.obs0001.p0001`.
Tracking is terminated by the command *ENDTRACK*.

Options

Option	Meaning	Default Value	Value Type
DELTAP	relative momentum offset for reference closed orbit (switched off for onepass)	0.0	double
ONEPASS	the sequence is treated as transfer line (no stability test, ie. no closed-orbit search)	.FALSE.= closed-orbit search	logical
DAMP	introduce synchrotron damping (needs RF cavity, RADIATE in BEAM)	.FALSE.= no damping	logical
QUANTUM	introduce quantum excitation via random number generator and tables for photon emission	.FALSE.= no excitation	logical
DUMP	write the particle coordinates in files (names generated automatically)	.FALSE.= no file generated	logical
APERTURE	particle is lost if its trajectory is outside the aperture of the current element. Notes.	.FALSE.= no aperture check	logical
ONETABLE	write all particle coordinates in a single file	.FALSE.= one file per particle	logical
RECLOSS	create a table named "trackloss" in memory with lost particles' coordinates	.FALSE.= no table	logical
FILE	name for the track table	"track", "trackone"	string

Remarks

IMPORTANT: If an RF cavity has a no zero voltage, synchrotron oscillations are automatically included. If tracking with constant momentum is desired, then the voltage of the RF cavities has to be set to zero. If an RF cavity has a no zero voltage and DELTAP is non zero, tracking is done with synchrotron oscillations around an off-momentum closed orbit.

DELTAP

Defining a non-zero *deltap* results in a change of the beam momentum/energy without changing the magnetic properties in the sequence. This leads to a new closed orbit, the off-momentum closed orbit. Particle coordinates are then given with respect to this new closed orbit, unless the option *onepass* is used!

ONEPASS

If the option *onepass* is used, no closed orbit is searched, which also means that no stability test is done. Use this option if you want to get the particles' coordinates with respect to the reference orbit rather than the closed orbit. Unfortunately the name is misleading, but for backwards compatibility it is kept. "onepass" does **NOT** restrict the

tracking to one turn only!

APERTURE

- If the *aperture* option is applied, the *apertype* and *aperture* information of each element in the sequence is used to check whether the particle is lost or not. For further information on the definition of apertures and different aperture types, see the documentation of the *APERTURE* module.
- In case no aperture information was specified for an element, the following procedure will currently take place:
  → No aperture definition for element → Default apertype/aperture assigned (currently this is `apertype= circle, aperture = {0}`)
  → If tracking with *aperture* is used and an element with `apertype= circle` AND `aperture= {0}` is encountered, then the first value of the *maxaper* vector is assigned as the circle's radius (no permanent assignment!). See option *maxaper* for the default values.
  ⇒ Hence even if no aperture information is specified by the user for certain elements, default values will be used!

RECLOSS

Traditionally, when a particle is lost on the aperture, this information is written to stdout. To allow more flexible tracking studies, the lost particles' coordinates and further information can also be saved in a table in memory. Usually one would save this table to a file using the *WRITE* command after the tracking run has finished. The following information is available in the TFS table "trackloss":

- Particle ID (number)
- Turn number
- Particle coordinates (x,px,y,py,t,pt)
- Longitudinal position in the machine (s)
- Beam energy
- Element name, where the particle is lost

***START*, *x*= double, *px*= double, *y*= double, *py*= double, *t*= double, *pt*= double;**
***START*, *fx*= double, *phix*= double, *fy*= double, *phiy*= double, *ft*= double, *phit*= double;**

Description

After the *TRACK* command, a series of initial trajectory coordinates has to be given by means of a *START* command (as many commands as trajectories). The coordinates can be either **canonical** coordinates,
START, X= double, PX= double, Y= double, PY= double, T= double, PT= double;
or **action-angle** coordinates,
START, FX= double, PHIX= double, FY= double, PHIY= double, FT= double, PHIT= double;
For this case the normalised amplitudes are expressed in number of r.m.s. beam size F_X, F_Y, F_T (the actions being computed with the emittances in the *BEAM* command) **in each mode plane**. The phases are PHI_X, PHI_Y and PHI_T expressed in radian. In the uncoupled case, we have in the plane mode labelled z,
$Z = F_z\ sqrt(E_z)\ cos(PHI_z)$, $P_z = F_z\ sqrt(E_z)\ sin(PHI_z)$,
where E_z is the r.m.s. emittance in the plane Z.

Options

Option	Meaning	Default Value	Value Type	Unit
X, PX, Y, PY, T, PT	canonical coordinates	0.0	double	m
FX, PHIX, FY, PHIY, FT, PHIT	action-angle coordinates	0.0	double	rad

Remarks

- For usual tracking (single/multi-turn), all coordinates are specified with respect to the actual closed orbit (possibly off-momentum, with magnet errors) and **NOT** with respect to the reference orbit.
- If the option *onepass* is used, the coordinates are specified with respect to the reference orbit. The name "onepass" might be misleading: Still tracking can be single- or multi-turn!

***OBSERVE*, *place*= string;**

Description

Coordinates can be recorded at places that have names. Such observation points are specified by the command *OBSERVE* (as many commands as places). The output files are named automatically. The name given by the user is followed by .obsnnnn(observation point), followed by .pnnnn(particle number). Hence filenames look like `track.obs0001.p0001`.

Options

Option	Meaning	Default Value	Value Type
PLACE	name of the observation point		string

Remarks

If no *OBSERVE* command is given, but the *dump* option in the *TRACK* command is used, the particles trajectory coordinates are still recorded. The observation point is then the starting point of the sequence.

***RUN*, *maxaper*= double array, *turns*= integer, *ffile*= integer;**

Description

The actual tracking itself is launched by the *RUN* command. Via the option *turns* the user can specify how many turns will be tracked.

Options

Option	Meaning	Default Value	Value Type
MAXAPER	upper limits for the six coordinates	{0.1, 0.01, 0.1, 0.01, 1.0, 0.1}	double array
TURNS	number of turns	1	interger
FFILE	periodicity for printing coordinates	1	interger

Remarks

The limits defined by the *maxaper* option are only being taken into account if the *aperture* option of the *TRACK* command is used.

Remarks

- Plotting is possible in MAD-X, however it can also be done externally by using the files created by *TRACK*.
- The following internal tables are created while tracking:
  `tracksumm`, `trackloss`, and `trackone` or `track.obs$$$$.p$$$$` (depending on option *onetable*).
  These internal tables can be accessed via the *table*-access functions.

See Also

APERTURE, *MAKETHIN*

A. Koschik, February 2007

EUROPEAN ORGANIZATION FOR NUCLEAR RESEARCH 

Thick-Lens Tracking Module (PTC-TRACK Module)

The **PTC-TRACK module** [a] is the symplectic thick-lens tracking facility in MAD-X [b]. It is based on PTC library written by E.Forest [c]. The commands of this module are described below, optional parameters are denoted by square brackets ([ ]). Prior to using this module the active beam line must be selected by means of a USE command. The general PTC environment must also be initialized.

Synopsis

```
PTC_CREATE_UNIVERSE;
PTC_CREATE_LAYOUT, model=integer,method=integer, nst=integer,
[exact];
.........................
PTC_START, .....;
.........................
PTC_OBSERVE,....;
.........................
PTC_TRACK, .....;
.........................
PTC_TRACK_END;
.............................
PTC_END;
```

Commands

PTC_START,

x=double, **px**=double, **y**=double, **py**=double, **t**=double, **pt**=double,
fx=double, **phix**=double, **fy**=double, **phiy**=double, **ft**=double, **phit**=double ;

Description

To start particle tracking, a series of initial trajectory coordinates has to be given by means of *PTC_START* command (as many commands as trajectories). It must be done before the *PTC_TRACK* command. The coordinates can be either canonical coordinates (**x**, **px**, **y**, **py**, **t**, **pt**) or action-angle coordinates (**fx**, **phix**, **fy**, **phiy**, **ft**, **phit**), which are expressed by the normalized amplitude, F_z and the phase, Φ_z for the z-th mode plane (z={x,y,t}). The actions are computed with the values of the emittances, F_z, which must be specified in the preceding BEAM command. F_z are expressed in number of r.m.s. beam sizes and Φ_z are expressed in radians.

Options

Option	Meaning	Default Value	Value Type
X, PX, Y, PY, T, PT	canonical coordinates	0.0	double
FX, PHIX, FY, PHIY, FT, PHIT	action-angle coordinates	0.0	double

Remarks

1. If the option *closed_orbit* in the *PTC_TRACK* command is active (see below), all coordinates are specified with respect to the actual closed orbit (possibly off-momentum with magnet errors) and NOT with respect to the reference orbit. If the option *closed_orbit* is absent, then coordinates are specified with respect to the reference orbit.
2. In the uncoupled case, the canonical and the action-angle variables are related with equations
$z = F_z(E_z)^{1/2}\cos(\Phi_z),\ p_z = F_z(E_z)^{1/2}\sin(\Phi_z).$

3. The use of the action-angle coordinates requires the option *closed_orbit* in the *PTC_TRACK* command.

4. If both the canonical and the action-angle coordinates are given in the *PTC_START* command, they are summed after conversion of the action-angle coordinates to the canonical ones.

PTC_OBSERVE,
place=string;

Description

Besides of the beginning of the beam-line, one can define an additional observation points along the machine. Subsequent *PTC_TRACK* command will then record the tracking data on all these observation points.

Option

Option	Meaning	Value Type
PLACE	name of observation point (markers are very much preferred)	string

Remarks

1. The first observation point at the beginning of the beam-line is marked as **"start"**.
2. It is recommended to use labels of markers in order to avoid usage observations at the ends of thick elements.
3. The data at the observation points other than at **"start"** can be produced by two different means:
a) traditional (MADX) element-by-element tracking (use option element_by_element);
b) coordinate transformation from **"start"** to the respective observation point using high-order PTC transfer maps
(required option *closed_orbit*; turned off options radiation and element_by_element).

PTC_TRACK,
deltap=double**, icase=**integer**, closed_orbit, element_by_element, turns=**integer**,**
dump, onetable, maxaper=double array, norm=integer**, norm_out,**
file[=string**], extension=**string**, ffile=**integer**,**
radiation, radiation_model1, radiation_energy_loss, radiation_quadr,
beam_envelope, space_charge;

Description

The *PTC_TRACK* command initiates trajectory tracking by entering the thick-lens tracking module. Several options can be specified, the most important are presented in table "Basic Options". There are also switches to use special modules for particular tasks. They are presented in the table "Special Switches".

The tracking can be done element-by-element using the option element-by-element, or "turn-by-turn" (default) with coordinate transformations over the whole turn. Tracking is done in parallel, i.e. the coordinates of all particles are transformed through each beam element (option element-by-element) or over full turns.

The particle is lost if its trajectory is outside the boundaries as specified by maxaper option. In PTC, there is a continuous check, if the particle trajectories stays within the aperture limits.

The Normal Form calculations (required option closed_orbit) is controlled by norm_no and norm_out are used.

Basic Options

Option	Meaning	Default Value	Value Type
ICASE	user-defined dimensionality of the phase-space (4, 5 or 6).	4	integer
DELTAP	relative momentum offset for reference closed orbit (used for 5D case ONLY).	0.0	double
CLOSED_ORBIT	switch to turn on the closed orbit calculation	.FALSE.	logical
ELEMENT_BY_ELEMENT	switch from the default turn-by-turn tracking to the element-by-element tracking.	.FALSE.	logical
TURNS	number of turns to be tracked.	1	integer
DUMP	enforces writing of particle coordinates to formatted text files	.FALSE.	logical

ONETABLE	writing all particle coordinates to a single file		.FALSE.	logical
MAXAPER	upper limits for the particle coordinates.		{0.1,0.01, 0.1, 0.01, 1.0,0.1}	double, array (1:6)
NORM_NO	order of the Normal Form		1	integer
NORM_OUT	switch to transform canonical variables to action-angle variables		.FALSE.	logical
FILE	omitted	no output written to a file		
	present	file name for printing the track tables.	track	string
EXTENSION	the extension of filename for the track table, e.g., txt, doc etc		none	logical
FFILE	printing coordinates after every FFILE turns		1	integer

Remarks

ICASE: has a highest priority over other options:

a) RF cavity with non-zero voltage will be ignored for icase=4, 5;

b) A non-zero deltap will be ignored for icase=4, 6.

However, if RF cavity has the voltage set to zero and for icase=6, the code sets icase=4.

DELTAP: is ignored for icase=6, but the option offset_deltap of the command *PTC_CREATE_LAYOUT* may be used, if

the reference particle should have an momentum off-set as specified by offset_deltap.

CLOSED_ORBIT : It must be used for closed rings only. This option allows to switch ON

the Normal Form analysis, if required. If CLOSED_ORBIT is off, the sequence is treated as a transfer line.

NORM_NO=1: makes the Normal Form linear (always true for MAD8/X).
FILE: The output file endings are: .obsnnnn(observation point), followed by .pnnnn (particle number),
if the onetable option is not used.

Special Switches

Option	Meaning	Default Value	Value Type
RADIATION	turn on the synchrotron radiation calculated by an internal procedure of PTC	.FALSE.	logical
RADIATION_MODEL1	switch to turn on the radiation according to the method given in the Ref. [d]	.FALSE.	logical
RADIATION_ENERGY_LOSS	adds the energy loss for radiation_model1	.FALSE.	logical
RADIATION_QUADR	adds the radiation in quadrupoles. It supplements either radiation, radiation_model1	.FALSE.	logical
BEAM_ENVELOPE	turn on the calculations of the beam envelope with PTC	.FALSE.	logical
SPACE_CHARGE (under construction)	turn on the simulations of the space charge forces between particles.	.FALSE.	logical

Remarks

1. RADIATION: Has precedence radiation model1.
2. RADIATION_MODEL1: Additional module by F. Zimmermann. The model simulates quantum excitation via a random number generator and tables for photon emission. It can be used only with the element-by-element tracking (option element-by-element).
3. RADIATION_ENERGY_LOSS: Of use for radiation_model1.
4. BEAM_ENVELOPE: It requires the options radiation and icase=6.
5. SPACE_CHARGE: This option is under construction and is reserved for future use.

PTC_TRACK_END;

Description

The *PTC_TRACK_END* command terminate the command lines related to the PTC_TRACK module.

TRACKSUMM table

The starting and final canonical coordinates are collected in the internal table "tracksumm" (printed to the file with WRITE command).

Examples

Several examples are found on the here.

The typical tasks

The following table facilitates the choice of the correct options for a number of tasks.

Option	1	2	3	4	5
CLOSED_ORBIT	-	-	+	+	+
ELEMENT_BY_ELEMENT	-	+	-	+	-
PTC_START, X, PX, ...	+	+	+	+	+
PTC_START, FX, PHIX,	-	-	+	+	+
NORM_NO	-	-	>1	>1	>1
NORM_OUT	-	-	+	-	+
PTC_OBSERVE	-	+	+	+	-
RADIATION	-	-	-	-	+
RADIATION_MODEL1	-	-	-	-	-
RADIATION_ENERGY_LOSS	-	-	-	-	-
RADIATION_QUAD	-	-	-	-	+/-
BEAM_ENVELOPE	-	-	-	-	-
SPACE_CHARGE	-	-	-	-	-

1) The tracking of a beam-line with default parameters.
2) As 1), but with element-by-element tracking and an output at observation points.
3) Tracking in a closed ring with closed orbit search and the Normal Forms calculations. Both canonical and action-angle input/output coordinates are possible. Output at observation points is produced via PTC maps.
4) Similar to "3)" except that output at observation points is created by element-by-element tracking.
5) The with PTC radiation.

References for PTC-TRACK

a) V. Kapin and F. Schmidt, PTC modules for MAD-X code, to be published as CERN internal note by the end of 2006
b) F. Schmidt, "‘MAD-X PTC Integration’’, Proc. of the 2005 PAC Conference in Knoxville, USA, pp.1272.

c) E. Forest, F. Schmidt and E. McIntosh, Introduction to the Polymorphic Tracking Code, KEK report 2002-3, July 2002
d) G.J. Roy, A new method for the simulation of synchrotron radiation in particle tracking codes, Nuclear Instruments & Methods in Phys. Res., Vol. A298, 1990, pp. 128-133.

See Also

Overview of MAD-X Tracking Modules, PTC Set-up Parameters, thintrack, PTC-TRACK Examples.

V. Kapin (ITEP) and F. Schmidt, July 2005; revised in April, 2006

EUROPEAN ORGANIZATION FOR NUCLEAR RESEARCH 

PTC_TRACKLINE

PERFORMS A PARTICLE TRAJECTORY TRACKING WITH ACCELERATION USING PTC

USER MANUAL

SYNOPSIS

```
PTC_TRACKLINE,
turns          [integer, 1, 0 ] ,
onetable       [logical, false, true ],
everystep      [logical, false, true],
tableallsteps [logical, false, true],
gcs            [logical, false, true],
file           [string, "track", "track" ],
rootntuple     [logical, false, true],
extension      [string, "", ""];
```

Parameter Name	Type	Default value		Description
		Not present	Present, but value not specified	
turns	integer	1	-	Number of turns
onetable	logical	false	true	If false, tracking data are written to a single table for each track for each observation point. Table names follow the naming *filename*.obsMMMM.pNNNN, where *filename* is settable prefix with **file** parameter (see below), MMMM is observation point number and NNNN is track number If true, all data are written to single table called onetable
file	string	"track"	"track"	Name of file where track parameters are written, see description of onetable switch above
rootntuple	logical	false	true	Stores data to ROOT file as ntuple. Accessible only if RPLOT plugin is available. i.e. only if madxp is dynamically linked and RPLOT plugin is present
everystep	logical	false	true	Switches on track parameters recording every integration step. Normally tracking data are stored only at the end of each element. Everystep mode allows the user to get finer data points. It implies usage of the so called node (thin) layout. Track parameters are stored for each step in thintracking_ptc.txt file. Storage of parameters in a table for each step might be very memory consuming. To switch it off use **tableallsteps** Collective effects can be taken to the account only using this mode (this feature of PTC is not interfaced into MAD-X).
gcs	logical	false	true	Instructs the code to store track parameters in Global Coordinate System - normally it starts at the entrance phase of the first element.

Description

This MAD-X command performs ray tracking that takes to the account acceleration in traveling wave cavities. It must be invoked in the scope of correctly initialized PTC environment, i.e. after PTC_CREATE_UNIVERSE and PTC_CREATE_LAYOUT commands and before corresponding PTC_STOP . All tracks that are spawned with PTC_START commands beforehand

PTC_TRACKLINE command is issued are tracked. Track parameters are dumped at every defined observation point (see PTC_OBSERVE command). Please note that MAD-X always creates observation point at the end of a sequence. Depending on value of onetable switch, all output information is stored in one table (and also file), or in one table per track per observation point is written if the switch is false. The user must note that track parameters plotting (see PLOT command) is only possible if onetable switch is set to false (status as for Feb. 2006). This unfortunate solution is the legacy of the regular MAD-X track command, that is designed for circular machines where the user usually tracks a few particles for many turns rather then many particles for one turn each.

Tracks that do not fit in aperture are immediately stopped.

Behavior of PTC calculations can be adapted with PTC_SETSWITCH command and with appropriate switches of PTC_CREATE_LAYOUT command.

Command parameters and switches

turns
integer, default value 1, no default value if value explicitly not specified

Number of turns around sequence. If layout is not closed then its value is enforced to 1.

onetable
boolean, default value false, if value explicitly not specified then true

If true then only one table is created and one file is written to disk. If false one file per track per observation point is written. File format is filename.obsNNNN.pMMMM, where NNNN and MMMM are numbers of observation point and track, respectively. Filename is defined by the switch described below.

file
character string, default is "track"

name of file where track parameters are written, see description of onetable switch above.

PROGRAMMERS MANUAL

The routine PTC_TRACKLINE is implemented in file madx_ptc_trackcavs.f90 Its single parameter is the number of observation points.

The call sequence from MAD-X interpreter is the following
exec_command in madxp.c;
pro_ptc_trackline in madxn.c; This routine creates appropriates tables where the track parameters are stored, and after execution of the Fortran routine dumps filled table(s) to files.
w_ptc_trackline_ in wrap.f90; Just interface to the appropriate Fortran module
ptc_trackline in madx_ptc_trackline.f90

The key routine that enables appropriate calculation of beam and track parameters in the presence of traveling wave cavities is setcavities.

Firstly, the ptc_trackline routine finds out which are the observation points. For this purpose array of integers observedelements is allocated. Its length is equal to the number of elements in the sequence. All elements are zero by default. If an element with an index n is an observation point then

observedelements[n] is equal to 1. This solution enables fast checking if track parameters should be sent to a table after a given element.

Further setcavities subroutine is called if it was not executed yet before.

PTC_TRACKLINE reads the track initial parameters from the table with the help of gettrack function (implemented in C in file madxn.c). For the performance reasons gettrack creates a two dimensional array and buffers there all the initial track parameters upon first call. The array is destroyed with a call of deletetrackstrarpositions function that is performed at the very end of ptc_trackline subroutine.

Tracking itself is implemented in a doubly nested loop. The external one goes over all initiated tracks, and the internal one performs tracking of a given track element by element. The key PTC routine is called TRACK. It propagates a track described by an array of 6 real numbers, denoted as X in equations below. The important issue is that they are the canonical variables. In order to follow the standard MAD-X representation the values that are written to tables and files are scaled appropriately to the reference momentum for a given element. In the general case it changes along the line if traveling wave cavities are present. Hence, momenta xp, yp and zp are

```
zp=sqrt((1+x(5))**2 - x(2)**2 - x(4)**2)
xp = x(2)/zp
yp = x(4)/zp
```

where array x containing 6 elements is the track position in the PTC representation, i.e. x(1) is horizontal spacial coordinate, x(2) - horizontal momentum, x(3) - vertical spacial coordinate, x(4) - vertical momentum, x(5) - $\delta p/p_0 c$, x(6) - longitudinal coordinate (caution, the exact meaning depends on the PTC settings, see PTC_SETSWITCH command).

EUROPEAN ORGANIZATION FOR NUCLEAR RESEARCH 

PTC_SETSWITCH

routine that sets the internal PTC switches

USER MANUAL

SYNOPSIS

```
PTC_SETSWITCH,
debuglevel = [i,0],
maxacceleration = [l, true, true],
exact_mis = [l, true, true],
totalpath = [l, true, true],
radiation = [l false, true],
fringe = [l, false, true],
time = [l, true, true];
```

Description

Using this command the user can set switches of PTC and the MAD-X-PTC interface, adapting this way the program behavior to his needs.

Command parameters and switches

debuglevel
integer, default value 1, no default value if value explicitly not specified

Sets the level of debugging printout 0 none, 4 everything

maxacceleration
logical, default true, if value explicitly not specified then true

Switch saying to set cavities phases so the reference orbit is always on the crest, i.e. gains max energy

exact_mis
logical, default true, if value explicitly not specified then true

Switch ensures exact misalignment treatment.

totalpath
logical, default true, if value explicitly not specified then true

If true, the 6th variable of PTC, i.e. 5th of MAD-X is the total path. If false it is deviation from the reference particle, which is normally the closed orbit for closed layouts.

radiation

logical, default false, if value explicitly not specified then true:

Sets the radiation switch/internal state of PTC.

fringe

logical, default true, if value explicitly not specified then true:

Sets the fringe switch/internal state of PTC. If true the influence of the fringe fields is evaluated for all the elements.

Please note that currently fringe fields are always taken to the account for some elements (f.g. traveling wave cavities) even if this flag is set to false. The more detailed list of the elements will be provided later, when the situation in this matter will be definitely settled.

time

logical, default true, if value explicitly not specified then true :

If true, Selects time of flight rather than path length. (cT to be precise) as the 6th variable of PTC, i.e. 5th of MAD-X.

PROGRAMMERS MANUAL

Values of the switches are stored in Fortran 90 module mad_ptc_intstate (mad_ptc_intstate.f90). The command is processed by pro_ptc_setswitch C function in file madxn.c It call an appropriate routines of the Fortran module to set each of the switches:

- ptc_setdebuglevel
- ptc_setaccel_method
- ptc_setexactmis
- ptc_setradiation
- ptc_settotalpath
- ptc_settime
- ptc_setfringe

EUROPEAN ORGANIZATION FOR NUCLEAR RESEARCH 

PTC_SetCavities

f90 routine that adjusts cavities and sets appropriate reference momenta for a layout containing traveling wave cavities

PROGRAMMERS MANUAL

CAUTION For the time being cavities MUST not be placed one after another, and at least a marker must be inserted between two neighboring accelerating structures. Otherwise, program will stop with the error message.

Description

This routine sets up the properties of a layout and traveling wave cavities. The main goal is to update reference beam energy for the elements that follow a traveling wave cavity. It traces the synchronous particle, i.e. one that has all its parameters set to zero at the beginning of the layout under study. At the point it arrives to a cavity, the parameters of the latter one are adjusted according to the switches defined by the user. There are 2 cases

1. **Leaves all parameters untouched**
2. **Phase of cavity is adjusted so it gives the maximum acceleration** Afterwards to the calculated phase the lag is added. This setting is acquired using set_switch command, setting maxaccel parameter to true.

Afterwards, the synchronous particle is tracked through traveling wave cavity and its energy gain is known. This energy becomes the reference one for all the elements downstream of the cavity. The particle is tracked further to the next cavity, for which the procedure described above is repeated.

Parameters of the cavities are dumped to the file named twcavsettings.txt.

At the end patches at the ends of the cavities are set, so the parameters after them are calculated taking to the account reference energy increase.

The exact program behavior depends on the PTC switches settings.

Please note that in PTC phase velocity of a cavities wave is always equal to speed of light. Hence, if PTC internal state TIME is TRUE, what is the most correct setting, then voltage seen by a particle is varying along the structure. If TIME is FALSE, track is assumed to fly with speed of light and in such case a particle moves together with the wave front.

EUROPEAN ORGANIZATION FOR NUCLEAR RESEARCH 

PTC_TWISS Module
(Ripken Optics Parameters)

The **PTC_TWISS module** of MAD-X [a] is based on the PTC code. It is a supplementary to the TWISS module. The Twiss parameters are calculated in Ripken's style (invented by G. Ripken in 1968 [31] and most accessible in Ref. [b]). These parameters were available in MAD8 using the TWISS3 command. This module is a typical example of the advantages when using PTC and its Normal Form technique (and of course the object-oriented Fortran90 coding): once the rather modest programming has been performed the Twiss calculation will always be automatically correct for all machine conditions like closed orbit, coupling or after a new element has been introduced into the code. In a traditional coding like in MAD8 this depends on reprogramming and modifying the code at various places which is inherently error-prone.

The PTC_TWISS tracks a special representation of the beam in three degrees of freedom. It works on the coupled lattice functions defined in Ref.[b], which are essentially the projections of the lattice functions for the eigen-modes on the three planes. The PTC_TWISS lists the projections of the ellipses of motion onto the three planes (x, p_x), (y, p_y), (t, p_t) expressed them via the Ripken's parameters $b_{k,j}$, $a_{k,j}$, $g_{k,j}$ along with the phase advances m_j in selected positions, where index k=1...3 refers to the plane (x, y,...), and the index j=1...3 denotes the eigen-mode. The PTC_TWISS also calculates the dispersion values D_1,...,D_4. In the MAD-X commands and tables, these parameters are denoted as `beta11,...,beta33, alfa11,...,alfa33, gama11,...,gama33, mu1,...,mu3, disp1,...,disp4`, respectively.

PTC_TWISS can also compute the deltap/p-dependency of the Twiss parameters. The column names `beta11p,...,beta33p, alfa11p...,alfa33p, gama11p,...,gama33p` denote the derivatives of the optics parameters w.r.t. deltap/p. If one is interested in evaluating deltap/p-dependency of the Twiss parameters, one must ensure that the order (no) of the map is set to 2 at least.

For clarification: in the 4-D case, there is the following correspondence between MAD-X and the Ripken's notations: `beta11`®$b_{x\mathrm{I}}$, `beta12`®$b_{x\mathrm{II}}$, `beta21`®$b_{y\mathrm{I}}$, `beta22`®$b_{y\mathrm{II}}$, while in the uncoupled 4-D case `beta11` is the same as the classical b_x (`betx`) and `beta22` is b_y (`bety`), while `beta12` and `beta21` are zero. When there is coupling all `betaNN` are non-zero and `beta11`, `beta22` are distinctively different from b_x, b_y, respectively.

PTC_TWISS also tracks the eigenvectors and prints them to Twiss table according to the SELECT command (flag=ptc_twiss). Either all 36 components or particular components of the eigenvectors can be selected with eign or eign*ij*, respectively (j = number of eigenvector, i = number of coordinate $\{x, p_x, y, p_y, t, p_t\}$).

For ring lattices, PTC_TWISS computes momentum compaction, transition energy, as well as other one-turn characteristics such as the tunes and chromaticities.

Synopsis

```
PTC_CREATE_UNIVERSE;
PTC_CREATE_LAYOUT, model=integer,method=integer, nst=integer,
[exact];
.............................
SELECT, flag=ptc_twiss, clear;
SELECT, flag=ptc_twiss, column=name, s,
beta11,...,beta33,alfa11,..., alfa33,gama11,...,gama33,
beta11p,...,beta33p,alfa11p,...,alfa33p,gama11p,...,gama33p,
mu1,...,mu3, disp1,...,disp4,
[eign], eign11, ...,eign16,...,eign61,...,eign66;
.............................
PTC_TWISS;
.............................
PTC_END;
```

Commands

PTC_TWISS,
***icase=integer, deltap*=double, closed_orbit, slice_magnets,**
range=string, file[=string], table[=string],
initial_matrix_table, initial_matrix_manual, beta0=string,
betx=double, alfx=double, mux=double,
bety=double, alfy=double, muy=double,
dx=double, dpx=double, dy=double, dpy=double,
***x*=double, *px*=double, *y*=double, *py*=double, *t*=double, *pt*=double,**
re11=double, re12=double, ... ,re16=double,
.....
re61=double, re62=double, ... ,re66=double;

Description

The *PTC_TWISS* command causes computation of the Twiss parameters in the Ripken's style. It operates on the working beam line defined in the latest USE command. Several options can be specified, the most important being *icase, deltap*, *closed_orbit, slice_magnets, no,* and *file, table*. (see the table below). Other options should be specified for particular tasks. Applications for the *PTC_TWISS* command are similar to the *TWISS*-command. The PTC_TWISS can be applied to two basic tasks. It can calculate either a periodical solution or a solution with initial conditions.

Options

Option	Meaning	Default Value	Value Type
ICASE	the user-defined dimensionality of the phase-space (4, 5 or 6).	4	integer

NO	the order of the map, which is often supplied as 1 but internally set to 2 to retrieve chromaticities for instance. For evaluating the Twiss parameteres derivatives w.r.t. deltap/p, one must pass order 2 at least.		1	integer
DELTAP	relative momentum offset for reference closed orbit.		0.0	double
CLOSED_ORBIT	the switch to turn on the closed orbit calculation (periodical solution ONLY).		.FALSE.	logical
SLICE_MAGNETS	the switch to turn on the evaluation of Twiss parameters at each thin slice inside successive magnets, instead of at the middle of each magnet. Slices are located at the so-called 'integration nodes' determined by the number of steps (nst) selected when creating the PTC layout.		.FALSE.	logical
FILE	omitted	no output written to a file		
	present	the name of the file for printing the PTC_TWISS output.	ptc_twiss	string
TABLE	omitted	no output written to a internal table		
	present	the name of the internal PTC_TWISS table	ptc_twiss	string
SUMMARY_FILE	omitted	no output written to a file		
	present	the name of the file for printing the PTC_TWISS_SUMMARY table output.	ptc_twiss_summary	string
SUMMARY_TABLE	omitted	no output written to a internal table		
	present	the name of the internal PTC_TWISS_SUMMARY table	ptc_twiss_summary	string
RANGE	specifies a segment of beam-line for the PTC_TWISS calculation		#s/#e	string

INITIAL_MATRIX_TABLE	Reading the transfer matrix from the map-table in memory (preceding *PTC_NORMAL* command needed).	.FALSE.	logical
INITIAL_MATRIX_MANUAL	Using the input variables RE11,...,RE66 (see next entry) as the transfer matrix.	.FALSE.	logical
RE11,..., RE66	Values of the 6x6 transfer matrix.	unit matrix	double
BETA0	The name of the *BETA0*-block, which contains the Twiss parameters for the input.	beta0	string
betx, alfx, mux, bety, alfy, muy, dx, dpx, dy, dpy	Twiss and dispersion parameters $b_{x,y}, a_{x,y}, g_{x,y}, D_x, D_{px}, D_y, D_{py}$ (Edwards and Teng).	0	double
x, px, y, py, t, pt	The **canonical** coordinates of the initial orbit.	0	double

Remarks

ICASE: It can be internally corrected by the code. For example, if RF cavity has the voltage set to zero and for icase=6, the code sets icase=4.

Periodical Solution

PTC_TWISS,
***icase=integer, deltap*=double, closed_orbit,**
range=string, file[=string], table[=string];

Description

This is the simplest form of the *PTC_TWISS* command, which computes the periodic solution for a specified beam line. It may accept all basic options described in the above table.

Evaluation of Twiss parameters inside magnets

PTC_TWISS,
***icase=integer, deltap*=double, closed_orbit, slice_magnets**
range=string, file[=string], table[=string];

Description

This computes the periodic solution for a specified beam line and evaluates the Twiss parameters at each thin-slice (a.k.a "integration-node") inside magnets. The number of such integration-nodes is given by the number of steps (nst) selected when creating the PTC layout. All other basic options described in the above table may be selected .

Example

An example is found in the *PTC_TWISS* Examples' repository. .

Solution with Initial Conditions

Code Logic:

```
IF ("initial_matrix_table"=ON .AND.
&   {the map-table exists})   THEN
  (from a Map-Table)

ELSEIF("initial_matrix_manual"=ON) THEN
  (from a Given Matrix)

ELSEIF(BETA0 block =ON) THEN
  (from Twiss Parameters via BETA0-block)

ELSE
  (from Given Twiss Parameters)

ENDIF
```

Initial Values from a Map-Table
(obtainable by a preceding** PTC_NORMAL **command):

PTC_TWISS,
***icase=integer, deltap*=double, closed_orbit,**
range=string, file[=string], table[=string],
initial_matrix_table;

Description

PTC_TWISS calculates a solution with initial conditions given as a map-table of preceding ring or beam-line. It requires the input option *initial_matrix_table* and an existence of the map-table in memory, which was generated by a preceding *PTC_NORMAL* command.

Example

An example is found in the *PTC_TWISS* Examples in the folder "Example3".

Initial Values from a Given Matrix:

PTC_TWISS,
***icase=integer, deltap*=double, closed_orbit,**
range=string, file=string, table=string,
initial_matrix_manual,
re11=double, re12=double, ... ,re16=double,
.....
re61=double, re62=double, ... ,re66=double;

Description

PTC_TWISS calculates a solution with initial conditions given by the matrix, which is "manually" entered on the command-line. It requires the option *initial_matrix_manual*. MAD-X expects a symplectic 6x6 transfer matrix as input.

Example

An example is found in the *PTC_TWISS* Examples in the folder "Example4".

Initial Values from Twiss Parameters via BETA0-block:

PTC_TWISS,
***icase=integer, deltap*=double, closed_orbit,**
range=string, file[=string], table[=string],
beta0=string;

Description

PTC_TWISS calculates a solution with initial conditions given by Twiss parameters, which are transferred from the *BETA0*-block. The data in the the *BETA0*-block have to be filled by a combination of the SAVEBETA and TWISS commands of a preceding ring or beam-line. Note, that this case is limited to uncoupled motion of the preceding machine.

Example

An example is found in the *PTC_TWISS* Examples in the folder "Example1".

Initial Values from the Given Twiss Parameters:

PTC_TWISS,
***icase=integer, deltap*=double, closed_orbit,**
range=string, file[=string], table[=string],
betx=double, alfx=double, mux=double,
bety=double, alfy=double, muy=double,
dx=double, dpx=double, dy=double, dpy=double,
***x*=double, *px*=double, *y*=double, *py*=double,**
***t*=double, *pt*=double;**

Description

PTC_TWISS calculates a solution with initial conditions given by the Twiss parameters, which are explicitly typed on the command line. Note, that this case is also limited to uncoupled motion of the preceding ring or beam-line.

Example

An example is found in the *PTC_TWISS* Examples in the folder "Example2".

References for PTC_TWISS

a) F. Schmidt, "'MAD-X PTC Integration'', Proc. of the 2005 PAC Conference in Knoxville, USA, pp.1272.
b) G. Ripken and F. Willeke, "Methods of Beam Optics", DESY 88114, 1988.
c) K. Zhang, "PTC twiss with initial TWISS parameters", MAD-X Meeting 13 (04.07.2005), slides in ppt.

See Also

TWISS, *PTC_TWISS* Examples.

V. Kapin (ITEP) and F. Schmidt, March 2006

EUROPEAN ORGANIZATION FOR NUCLEAR RESEARCH 

PTC_NORMAL Module
(Non-Linear Machine Parameters)

The **PTC_NORMAL module** of MAD-X [a,b] is based on PTC code. This module takes full advantage of the PTC Normal Form analysis which is a considerable upgrade of what was available with the Lie Algebra technique used in MAD8. It allows to calculate dispersions, chromaticities, anharmonicities and Hamiltonian terms to very high order. In fact, the order is only limited by the RAM memory of your computer and your patience to wait for the results.

The number of terms per order increases with some power law. The internal MAD-X tables are not adequate to keep such large amounts of data. On the other hand, only a reduced set of this data is actually needed by the user. Thus a much easier and flexible solution is to gather the users requirements with a series of special MAD-X command called *SELECT_PTC_NORMAL*. A special MAD-X table is dynamically built using just those commands and it will be filled by the next call to the *PTC_NORMAL*-command.

Another essential advantage of this table is the fact that it is structured to facilitate exchange of Normal Form (including Hamiltonian terms of high order) between MAD-X modules. The immediate goal is to use this table to allow non-linear matching inside the present MAD-X MATCHING module.

Synopsis

```
PTC_CREATE_UNIVERSE;
PTC_CREATE_LAYOUT, model=integer,method=integer, nst=integer,
[exact];
.............................
SELECT_PTC_NORMAL, dx,..., gnfu;
.............................
PTC_NORMAL;
WRITE, table=normal_results, file=normal_results;
.............................
PTC_END;
```

Commands

SELECT_PTC_NORMAL,
dx=integer, dpx=integer, dy=integer, dpy=integer,
q1=0, dq1=integer, q2=0, dq2=integer,
anhx=integer array, anhy=integer array,
gnfu=integer,0,0, haml=integer,0,0,
eign=integer, integer;

Description

The *SELECT_PTC_NORMAL* command selects parameters to be calculated by the next *PTC_NORMAL* command. The dispersion and tune parameters are defined by a name

and an integer number specifying their order. For example, the notation "dx=2" means the horizontal dispersion to second order $D_x{}^{(2)}=\partial^{(2)}x_{co}/\partial\delta_p{}^{(2)}$, where "co" is abbreviation of "closed orbit". The anharmonisities are defined by a name and three integer numbers: the first is the order of ε_1, the second is the order of ε_2, the third is the order of δ_p. For example, the notation "anhx=2,0,0" means second order in ε_1: $\partial^{(2)}q_1/\partial\varepsilon_1{}^{(2)}$.
Components of the eigenvectors at the end of the structure can be specified by two integers: the first integer defines the eigenvector number, the second integer defines the coordinate $\{x, p_x, y, p_y, t, p_t\}$.
The Generating Function can be specified by $\{n, 0, 0\}$. The positive and negative values of n define the order of upright or skew resonances, respectively. The integers n_2 and n_3 are reserved for a future upgrade. For example, "gnfu=-5, 0, 0" will calculate all Generating Function terms for skew decapoles. In the output table, one finds the cosine, sine and amplitude coefficients as denoted by "GNFC", "GNFS", and "GNFA", respectively.
Similarily, the Hamiltonian terms can be specified by $\{n, 0, 0\}$. The positive and negative values of n define the order of upright or skew resonances, respectively. For example, "haml=3, 0, 0" will calculate all Hamiltonian terms for upright sextupoles. In the output table, one finds the cosine, sine and amplitude coefficients as denoted by "HAMC", "HAMS", and "HAMA", respectively.
Parameters

Notation	Meaning	Value
DX, DPX, DY,DPY	dispersions, $D_x{}^{(n)}$, $D_{px}{}^{(n)}$, $D_y{}^{(n)}$, $D_{py}{}^{(n)}$	n
Q1, Q2	horizontal and vertical tunes $q_1{}^{(0)}$, $q_2{}^{(0)}$	0
DQ1, DQ2	derivatives of horizontal and vertical tunes $\partial^{(n)}q_1/\partial\delta_p{}^{(n)}$, $\partial^{(n)}q_2/\partial\delta_p{}^{(n)}$	n
ANHX, ANHY	Anharmonicities	$n(\varepsilon_1)$, $n(\varepsilon_2)$, $n(\delta_p)$
GNFU	Generating Function	n, 0, 0
HAML	Hamiltonian	n, 0, 0
EIGN	Eigenvector (the n_2-th component of the n_1-th eigenvector)	n_1, n_2

PTC_NORMAL,
***icase=integer, normal*, closed_orbit,**
no=integer, map_table, deltap=double;
Description
The calculation of the parameters specified by the preceding *SELECT_PTC_NORMAL* commands is initiated by the *PTC_NORMAL* command, which operates on the working beam line defined in the latest USE command. The options for *PTC_NORMAL* command are described in the table below.

Options

Option	Meaning	Default Value	Value Type
ICASE	the user-defined dimensionality of the phase-space (4, 5 or 6)	4	integer
NO	the order of the map.	1	integer
CLOSED_ORBIT	the switch to turn on the closed orbit calculation.	.FALSE.	logical
DELTAP	relative momentum offset for reference closed orbit	0.0	double
MAPTABLE	turn on the map-table in memory	.FALSE.	logical
NORMAL	turn on the calculation of the Normal Form	.FALSE.	logical

Remarks

MAPTABLE: (requires no=1) creates the one-turn matrix which can be used by the next PTC_TWISS command.

Example

The simple example is located on the Web-page for the *PTC_NORMAL* example.

References for PTC_NORMAL

a) F. Schmidt, "‘MAD-X PTC Integration’’, Proc. of the 2005 PAC Conference in Knoxville, USA, pp.1272.
b) E.T. d Amico, "Nonlinear parameters from PTC", MAD-X Meeting 7 (29.11.2004), notes (doc-file).
c) A. Schoch, "Theory of linear and non-linear perturbations of betatron oscillations in alternating-gradient synchrotrons ", CERN-27-21, 1958.

.See Also

PTC_NORMAL example, PTC Set-up Parameters.

V. Kapin (ITEP) and F. Schmidt, March 2006

EUROPEAN ORGANIZATION FOR NUCLEAR RESEARCH 

MAD-X-PTC interface documentation - Auxiliaries

Available documents

- PTC_Knob.html
- PTC_SetKnobValue.html
- PTC_PrintParametric.html
- PTC_PrintFrames.html
- PTC_Select.html
- PTC_SelectMoment.html
- PTC_SetSwitch.html
- PTC_DumpMaps.html
- PTC_SetCavities.html
- PTC_EPlacement.html
- ptc_general.html
- ptc_track.html
- ptc_track_line.html
- ptc_twiss.html
- ptc_normal.html
- Under Construction Match_WithPTCKnobs.html
- Under Construction PTC_Moments.html
- this document

EUROPEAN ORGANIZATION FOR NUCLEAR RESEARCH 

PTC_KNOB

USER MANUAL

SYNOPSIS

```
PTC_KNOB,
elementname = [s, none] ,
kn     = [i, {-1}],
ks     = [i, {-1}],
exactmatch = [l, true, true] ;
```

Description

Sets knobs in PTC calculations (currently ony in PTC_TWISS, PTC_NORMAL will follow). Knobs appear as the additional parameters of the phase space. Twiss functions are then obtained as functions of these parameters (Taylor series). Also map elements might be stored as functions of knobs, see ptc_select command description to lear how to request given element to be stored as a Taylor series. The parametric results can be further

1. written to a file with ptc_printparametric.
2. plotted and studied using rviewer command (rplot plugin).
3. used to obtain very quickly approximate values of lattice functions for given values of knobs (ptc_setknobvalue). This feature is the foundation of a fast matching alorithm with PTC.

Example

dog leg chicane : Dipolar components of both rbends and dipolar and quadrupolar components of the focusing quads set as knobs. Some first and second order map coefficients set to be stored as parametric results. ptc_twiss command is performed and the parametric results are written to files in two formats. dog leg chicane : Knobs values are matched to get requested lattice functions.

Command parameters and switches

elementname
string in range format,

Specifies name of the element containing the knob(s) to be set.

kn,ks
list of integers,

Defines which order

exactmatch

logical, default true, if value explicitly not specified then true

Normally a knob is a property of a single element in a layout. The specified name must match 1:1 to an element name. This is the case when exactmatch is true.

Knobs might be alos set to all family of elements. In such case the exactmatch switch must be false. A given order field component of all the elements that name starts with the name specified by the user become a single knob.

initial

–

EUROPEAN ORGANIZATION FOR NUCLEAR RESEARCH 

PTC_SETKNOBVALUE

USER MANUAL

SYNOPSIS

```
PTC_SETKNOBVALUE,
elementname = [s, none] ,
kn    = [i, {-1}],
ks    = [i, {-1}],
value = [r] ;
```

Description

With this command the user set a given knob value. In its effect all the values in

- the twiss table used by the last ptc_twiss command
- the columns specified with ptc_select, parametric=true;

are reevaluated using the buffered parametric results.

The parameters of the command basically contains the fields that allow to identify uniquely the knob and the value to be set.

Example

dog leg chicane : strength of dipol field component in quads is matched to obtain required R56 value.

Command parameters and switches

elementname
string in range format,

Specifies name of the element containing the knob to be set.

kn,ks
list of integers,

Defines the knob

value
real, default 0, if value explicitly not specified then 0

Specifies the value the knob is set to.

EUROPEAN ORGANIZATION FOR NUCLEAR RESEARCH 

PTC_KNOB

USER MANUAL

SYNOPSIS

```
PTC_KNOB,
elementname = [s, none] ,
kn     = [i, {-1}],
ks     = [i, {-1}],
exactmatch = [l, true, true] ;
```

Description

Sets knobs in PTC calculations (currently ony in PTC_TWISS, PTC_NORMAL will follow). Knobs appear as the additional parameters of the phase space. Twiss functions are then obtained as functions of these parameters (taylor series). Also map elements might

Example

Not yet ready : postion of quads is matched to obtain required R566 value.

Command parameters and switches

elementname
string in range format,

Specifies name of the element containing the knob(s) to be set.

kn,ks
list of integers,

Defines which order

exactmatch
logical, default true, if value explicitly not specified then true

Normally a knob is a property of a single element in a layout. The specified name must match 1:1 to an element name. This is the case when exactmatch is true.

Knobs might be alos set to all family of elements. In such case the exactmatch switch must be false. Filed components of all the elements that name starts with the name specified by the user become a single knob.

—

EUROPEAN ORGANIZATION FOR NUCLEAR RESEARCH 

PTC_PRINTFRAMES

USER MANUAL

SYNOPSIS

```
PTC_PRINTFRAMES,
file = [s, none] ,
format = [s, text] ;
```

Description

Print to a specified file PTC geometry of a layout.

Example

Dog leg chicane with some elements displaced with help of ptc_eplacement:

Command parameters and switches

file

string,

Specifies name of the file.

format

string, default "text"

Format of geometry. Currently two formats are accepted:
text

Prints a simple text file.

rootmacro

Creates root macro that produces 3D display of the geometry.

———

EUROPEAN ORGANIZATION FOR NUCLEAR RESEARCH 

PTC_SELECT

USER MANUAL

SYNOPSIS

```
PTC_SELECT,
table      = [s, none, none],
column     = [s, none, none],
polynomial = [i, none] ,
monomial   = [s, none] ,
parametric = [l, false, true],
quantity   = [s, none] ; "
```

Description

Selects map elements to be:

a) **Stored in a user specified table and column.** *Table* and *column* must be specified than, and such table with such column must exists.

b) **Stored as a function (taylor series) of knobs, if any is defined.** Than, *parametric* should be set to true. Both a) and b) can be joined in one command.

Examples

dog leg chicane : strength of quads is matched to obtain required T112 value.

dog leg chicane : postion of quads is matched to obtain required T566 value.

dog leg chicane : dipols and quads strengths are matched with the help of knobs to obtain required momentum compaction and Twiss functions.

Command parameters and switches

table
string,

Specifies name of the table where values should be stored.

column
string,

Specifies name of the table where values should be stored.

polynomial

integer,

Specifies row of the map.

monomial

string composed of digits

Defines monomial of the polynomial in PTC nomenclature. Its lengh should be equal to number of variables. Each of digits corresponds to the exponent of a variable. Monomial 'ijklmn' defines $x^i p_x{}^j y^k p_y{}^l \Delta T^m (\Delta p/p)^n$. For example, element=2 and monomial=1000000 defines coefficient of the second polynomial (that defines p_x) close to x, in the other words it is R21.

parametric

logical, default false, if value explicitly not specified then true

If it is true, and any knobs are defined the map element is stored as the parametric result.

PROGRAMMERS MANUAL

The command is implemented pro_ptc_select function in madxn.c and by subroutine addpush in madx_ptc_knobs.f90, that is part of madx_ptc_knobs_module

On the very beginning the existance of the table and within column is checked. In the case of failure, error message is printed and the function is abandoned.

The command parameters are passed as the arguments of addpush Fortran routine. A selection is stored in a type called tablepush_poly defined madx_ptc_knobs.inc. A newly created object is added to array named pushes.

More then one element might be stored in a single table, so the module must assure that each of tables is augmented only ones for each magnet (or integration slice). For that purpose array of tables to be augmented (named tables) is stored separately and we assure that a table is listed here only ones. This is simply done by checking if a table name is not already listed before adding a new element to the array.

In case the user requested an element to be stored in the paramteric format, and column in the array of parametric results is reserved and the index of the column is remembered in index field of tablepush_poly type is filled. In the other case this field is equal to zero.

The routine ptc_twiss (defined in file madx_ptc_twiss.f90), after tracking each of magnets in the sequence, calls putusertable routine. This routine loops over selected elemetns defined in the pushes table. For each of them it extracts the requested element from the map using .sub. operator of PTC and stores it in the defined table and column. If index field is not zero and any knob is defined, it extracts the polynomial using .par. operator, and stores it in the 2D array called results, in the row corresponding to the number of the magnet (or integration step) and column defined by the index field.

EUROPEAN ORGANIZATION FOR NUCLEAR RESEARCH 

PTC_SELECT_MOMENT

USER MANUAL

SYNOPSIS

```
PTC_SELECT_MOMENT,
table      = [s, none, none],
column     = [s, none, none],
moments   = [s, none] ,
moments   = [i, {0}] ,
parametric = [l, false, true],
```

Description

Selects a moment to be:

a) **Stored in a user specified table and column.**

b) **Stored as a function (taylor series) of knobs, if any is defined.** Than, *parametric* switch should be set to true.

Both a) and b) can be joined in one command.

Examples

ATF2

Command parameters and switches

moment_s

list of coma separated strings composed of up to 6 digits

Defines moment of the polynomial in PTC nomenclature. String 'ijklmn' (i,j,k,l,m,n are digits) defines $<x^i p_x{}^j y^k p_y{}^l \Delta T^m (\Delta p/p)^n>$. For example, moments=1000000 defines $<x^1>$

Note that for input we always use MAD-X notation where dp/p is always the 6th coordinate. (Internaly PTC dp/p is the 5th coordinate. We perform automatic conversion that is transparent for the user.) As the consequence RMS in dp/p is always defined as 000002, even in 5D case.

This notations allows to define more then one moment with one command. In this case, the corresponding column names are as the passed strings with "mu" prefix. However, they are always extended to 6 digits, i.e. the trailing 0 are automatically added. For example, if specified moments=2, the column name is mu200000.

This method does not allow to pass bigger numbers then 9. If you need to define such a moment, use the command switch below.

moment

list of up to 6 coma separated integers,

Defines a moment. For example: moment=2 defines $\langle x^2 \rangle$, moment=0,0,2 : $\langle y^2 \rangle$, moment=0,14,0,2 : $\langle px^{14} py^2 \rangle$, etc.

table

string, default "moments"

Specifies name of the table where the calculated moments should be stored.

column

string

Ignored if *moments* is specified. Defines name of the column where values should be stored. If not specified then it is automatically generated from moment the definition $\langle x^i p_x{}^j y^k p_y{}^l \Delta T^m (\Delta p/p)^n \rangle$ => mu_i_j_k_l_m_n (numbers separated with underscores).

parametric

logical, default false, if value explicitly not specified then true

If it is true, and any knobs are defined the map element is stored as the parametric result.

EUROPEAN ORGANIZATION FOR NUCLEAR RESEARCH 

PTC_DUMPMAPS

USER MANUAL

SYNOPSIS

```
PTC_DUMPMAPS,
file     = [s, ptcmaps, ptcmaps];
```

implemented by subroutine ptc_dumpmaps() in madx_ptc_module.f90

Description

PTC_DUMPMAPS dumps linear part of the map for each element of the layout into specified file.

Command parameters and switches

file

string, default value "ptcmaps", default value if value explicitly not specified is "ptcmaps"

Specifies name of the file to which the matrices are dumped to.

PROGRAMMERS MANUAL

The command is implemented by subroutine ptc_dumpmaps() in madx_ptc_module.f90. The matrix for a single element is obtained by tracking identity map through an element, that is initialized for each element by adding identity map to the reference particle. For the elements that change reference momentum (i.e. traveling wave cavity) it is tracked to the end of the following marker, that has updated reference momentum. Hence, each cavity must be followed by a marker. If it is not, setcavities subroutine detects error and stops the program.

EUROPEAN ORGANIZATION FOR NUCLEAR RESEARCH 

PTC_EPLACEMENT

USER MANUAL

SYNOPSIS

```
PTC_EPLACEMENT,
range = [s, none],
x         = [r, 0], y         = [r, 0], z         = [r, 0],
phi       = [r, 0],
theta     = [r, 0],
onlyposition     = [l, false, true] ,
onlyorientation = [l, false, true] ,
autoplacedownstream = [l, true, true] ,
refframe = [s, gcs] ;
```

Description

Places a given element at required position and orientation. All rotations are made around the front face of the element.

Example

Dog leg chicane : postion of quads is matched to obtain required R566 value.

Command parameters and switches

range
string in range format,

Specifies name of the element to be moved.

x,y,z
real,

Coordinate of the front face of the magnet.

phi, theta
real,

polar (in xz plane, around z axis) and azimuthal (around x axis) angles, respectively.

refframe
string, default gcs

Defines the coordinate system with respect to which coordinates and angles are specified. Possible values are:

gcs
: global coordinate system

current
: current position

previouselement
: end face of the previous element

onlyposition

logical, default false, if value explicitly not specified then true

If true, only translation are performed and orientation of the element is not changed.

onlyorientation

logical, default false, if value explicitly not specified then true

If true, only rotations are performed and position of the element is not changed.

autoplacedownstream

logical, default true,

if true all the elements downstream are placed at default positions in respect to the moved element, if false the rest of the layout stays untouched.

surveyall

logical, default true, if value explicitly not specified then true

If true, survey of all the line is performed after element placement at new position and orientation. It is implemented mainly for the software debugging purposes. If patching was performed correctly, the global survey should not change anything.

–

PROGRAMMERS MANUAL

The command is implemented pro_ptc_eplacement function in madxn.c and by subroutine ptc_eplacement() in madx_ptc_eplacement.f90.

Sopecified range is resolved with help of get_range command. Number of the element in the current sequence is resolved and passed as the parameter to the fortran routine. It allows to resolve uniquely the corresponding element in the PTC layout.

TRANSLATE_Fibre and ROTATE_Fibre routines of ptc are employed to place and orient an element in space. These commands adds rotation and translation from the current position. Hence, if the specified reference frame is other then "current", the element firstly needs to be placed at the center of the reference frame and then it is moved about the user specified coordinates.

After element placement at new position and orientation patch needs to be recomputed. If autoplacedownstream is false then patch to the next element is also recomputed. Otherwise, the layout is surveyed from the next element on, what places all the elements downstream with default position with respect to the moved element.

At the end all the layout is surveyed, if surveyall flag is true, what normally should always take place.

EUROPEAN ORGANIZATION FOR NUCLEAR RESEARCH 

Matching with PTC knobs

USER MANUAL

This matching procedure takes advantage of the parametric results that are accessible with PTC. Namely, parameters occuring in the matching constrains are obtained as functions (polynomials) of the matching variables. In other words, each variable is a knob in PTC calculation. Evaluation of the polynomials is relatively fast comparing to the regular PTC calculation what makes findining the minimum with the parametrized constraints very fast.

However, the algorithm is not faster in a general case:

1. The calculation time dramatically increases with number of parameters and at some point penalty rising from this overcomes the gain we get from the fast polynomial evaluation.
2. A parametric result is an approximation that is valid only around the nominal parameter values.

The algorithm:

1. Buffer the key commands (ptc_varyknob, constraint, ptc_setswitch, ptc_twiss or ptc_normal, etc) appearing between match, useptcknobs=true; and any of matching actions calls (migrad,lmdif,jacobian, etc)
2. When matching action appears,

   a) set "The Current Variables Values" (TCVV) to zero
   b) perform THE LOOP, i.e. points 3-17

3. Prepare PTC environment (ptc_createuniverse, ptc_createlayout)
4. Set the user defined knobs (with ptc_knob).
5. Set TCVV using ptc_setfieldcomp command
6. Run a PTC command (twiss or normal)
7. Run a runtime created script that performs a standard matching; all the user defined knobs are variables of this matching.
8. Evaluate constraints expressions to get the matching function vector (I)
9. Add the matched values to TCVV
10. End PTC session (run ptc_end)
11. If the matched values are not close enough to zeroes then goto 3
12. Prepare PTC environment (ptc_createuniverse, ptc_createlayout)
13. Set TCVV using ptc_setfieldcomp command
(--- please note that knobs are not set in this case --)
14. Run a PTC command (twiss or normal)
15. Evaluate constraints expressions to get the matching function vector (II)
16. Evaluate a penalty function that compares matching function vectors (I) and (II) See points 7 and 14
17 If the matching function vectors are not similar to each other within requested precision then goto 3

18. Print TCVV, which are the matched values.

SYNOPSIS

```
MATCH, use_ptcknobs=true;

PTC_VARYKNOB:
  initial = [s, none] ,
  element = [s, none] ,
  kn    = [i, -1],
  ks    = [i, -1],
  exactmatch = [l, true, true],
  trustrange    = [r, 0.1],
  step     = [r, 0.0],
  lower    = [r, -1.e20],
  upper    = [r,  1.e20];

END_MATCH;
```

For the user convenience the limits are specified in the MAD-X units (k1,k2, etc). This also applies to dipolar field where the user must specify limits of k0=angle/path_lengh. This guarantees concistency in treatment of normal and skew dipol components.

Important: Note that inside the code skew magnets are represented only by normal component and tilt, so the nominal skew component is always zero. Inside PTC tilt can not become a knob, while skew component can. Remember about this fact when setting the limits of skew components in the matching. When the final results are exported back to MAD-X, they are converted back to the "normal" state, so the nominal skew compoment is zero and tilt and normal component are modified accordingly.

trustrange - defines the range the expansion is trusted

Description

Example

dog leg chicane :

EUROPEAN ORGANIZATION FOR NUCLEAR RESEARCH 

PTC_MOMENTS

USER MANUAL

SYNOPSIS

```
PTC_MOMENTS,
no = [i, 1],
xdistr   = [s, gauss, gauss],
ydistr   = [s, gauss, gauss],
zdistr   = [s, gauss, gauss],
```

Description

Calculates moments previously selected with the ptc_select_moment command. It uses maps saved by the ptc_twiss command, hence, the savemaps switch of ptc_twiss must be set to true (default) to be able to calculate moments.

Examples

ATF2

Command parameters and switches

no

integer

order of the calculation, maximally twise the order of the last twiss

xdistr, ydistr, zdistr

string defining type of distribution for x, y, z dimension, respectively,

1. ***gauss*** *- Gaussian*
2. ***flat5*** *- flat distribution in the first of variables (dp over p) of a given dimension and Delta Dirac in the second one (T)*
3. ***flat56*** *- flat rectangular distribution*

EUROPEAN ORGANIZATION FOR NUCLEAR RESEARCH 

Known Differences to Other Programs

Definitions

MAD uses full 6-by6-matrices to allow coupling effects to be treated, and the canonical variable set (x, p_x / p_0), (y, p_y / p_0), (-ct, delta(E) / p_0 c), as opposed to other programs most of which use the set (x, x'), (y, y'), (-delta(s), delta(p)/p_0). Like [Dragt], MAD uses the relative energy error p_y / p_0, which is equal the relative momentum error delta = delta(p)/p_0 multiplied by beta = v/c.

As from Version 8.13, MAD8 uses an additional **constant** momentum error delta_s in all optical calculations. The transfer maps contain the **exact** dependence upon this value; therefore the tunes for large deviations can be computed with high accuracy as opposed to previous versions.

The choice of canonical variables in MAD still leads to slightly different definitions of the lattice functions. In MAD the Courant-Snyder invariants in [Courant and Snyder] take the form

$$W_x = \text{gamma}_x \, x^2 - 2 \, \text{alpha}_x \, x \, p_x + \text{beta}_x \, p_x^2$$

Comparison to the original form

$$W_x = \text{gamma}_x \, x^2 - 2 \, \text{alpha}_x \, x \, x' + \text{beta}_x \, x'^2$$

shows that the orbit functions cannot be the same. A more detailed analysis, using

$$x' = p_x / (1 + \text{delta})$$

shows that all formulas can be made consistent by defining the MAD orbit functions as

$$\text{beta}_{xM} = \text{beta}_{xC} * (1 + \text{delta}), \; \text{alpha}_{xM} = \text{alpha}_{xC}, \; \text{gammaa}_{xM} = \text{gamma}_{xC} / (1 + \text{delta}),$$

For constant delta_s along the beam line and delta = 0, the lattice functions are the same. In a machine where delta varies along the circumference, e.g. in a linear accelerator or in an electron-positron storage ring, the definition of the Courant-Snyder invariants must be generalised. The MAD invariants have the advantage that they remain invariants along the beam line even for variable delta.

With the new method this problem occurs in Twiss module only for non-constant delta.

Treatment of Energy Error in TWISS

It has been noted in [Milutinovic and Ruggiero] that MAD returned tunes which are too low for non-zero delta. The difference was found to be quadratic in delta with a negative coefficient. This problem has been eliminated thanks to the new treatment of momentum errors from MAD8 Version 8.13 onwards.

hansg, January 24, 1997

Index

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

- Quadrupole
- QUIT

r

- RADIATION (PTC)
- Random Generators
- Range selection
- RBEND: Rectangular Bending Magnet
- rbend reference system
- RCOLLIMATOR:Rectangular Collimator
- READTABLE
- real attribute
- real expression
- References
- REFLECT
- REMOVE
- REPLACE
- RESBEAM
- RESPLOT
- RETURN
- RFCAVITY
- Ripken Optics Parameters (PTC_TWISS)
- RPLOT

s

- save
- savebeta
- SAVE with SELECT
- sbend reference system
- SBEND: Sector Bending Magnet
- sector bend
- sectormap
- select
- seqedit
- SEQEDIT
- Sequence editing
- Sequences
- SET
- SETCAVITIES (PTC)
- setplot
- Sextupole
- SHOW

t

u

v

w

- WEIGHT: Matching Weights
- WHILE
- WHILE example
- Wild Card Pattern
- write
- WX, PHIX, WY, PHIY,...

y

- YROTATION: Rotation About the Vertical Axis

EUROPEAN ORGANIZATION FOR NUCLEAR RESEARCH 

Effect of the bv flag in MAD-X

Introduction

When inverting the direction ("v") of a particle in a magnetic field ("b") while keeping its charge constant, the resulting force v x B changes sign. This is equivalent to flipping the field, but not the direction. If, in addition, like in the case of two rings in MAD-X, the coordinates transform like

```
x_2 = x_1
y_2 = y_1
s_2 = L - s_1
```

then the x- and y-displacements caused by this change sign. Furthermore, all derivatives of B with respect to x and/or y change sign. This can be handled by inverting the forces. Since the derivatives with respect to s do not change sign, the solenoid is not inverted (a particle with positive p_x is deflected to the same side either way).

The dipole force, however, is a special case. A normal arc dipole (SBEND or RBEND) does not change x and y (the coordinate system follows the curved ideal orbit). The dispersion must remain positive when a positive delta_p deflects the orbit towards positive x values. Therefore, the dipole field in arc bends must not be inverted. By decision of the AP optics team, kicker strength in both rings will be defined such that a positive kick deflects towards positive x. Kickers must not see their forces inverted, too. An exception are the kickers shared between both rings for which a positive force reflects towards positive x in ring 1. This force must be inverted in ring 2, obviously. Finally, there are dog-leg dipoles that are shared, and their forces must therefore be inverted in ring 2. This results in the following:

The flag "bv" can be added to two types of commands: the beam command, and magnetic element descriptions. The first will be called case A, the second case B. Case A concerns all forces except kicker and dipole forces, case B is especially for shared dipoles and kickers. The table below gives the condition under which combination of flags the magnetic force changes sign. All elements not mentioned are not affected. Please note that case B alone has no effect.

Further more detailed explanations of the underlying mathematics can be found in Stephane Fartoukh's slides.

Table: BV flag effects

BEND	QUAD	SEXT	OCT	K0L MULTIPOLE	KnL MULTIPOLE	KICKER
A and B	A	A	A	A and B	A	A and B

EUROPEAN ORGANIZATION FOR NUCLEAR RESEARCH 

MAD-X News

Hans Grote and Frank Schmidt

MAD-X News

This is a loose collection of new features, new sample jobs, and other possibly interesting remarks concerning MAD-X.

First MAD-X LHC commissioning version 3.04: 09.07.2007

```
1) Allowing reading and writing of SDDS data sets for
communication with the LHC control system.
2) A large amount of medium size and smaller code changes
and bug fixes.
3) Full blown development for CLIC purposes.
4) New Jacobian matching method.
5) Non-linear and parametric matching.
```

All changes for each file:

```
--------------------------------------------------------------------------------

MAD-X proper
------------

C Files:
--------
- minor C inconsistencies and some clean-up
- Uninitiatized variables

aperture.c
- moved aperture code from madxn.c to new file aperture.c
- corrections:
- more apex in halo polygon
- corrected the construction of rectellipse in the general case
- secured potentially dangerous division by zero

c6t.c
- The brute force quick and dirty fix of frs has been reverted for a proper
fixing at the source of the problem, IE assigning different names for
different multipoles. It has been shown that SixTrack gives identical
results for the "brute force" and the "proper solution".
Courtesy Hans Grote (honorable ABP group member)
- Fixing the problem of using the same name for very different multipoles.
This is a quick fix and a more rigorous solution is needed.
- R(E)COLLIMATORs were treated as thick elements in the single element list
but later treated as thin elements leading to skrewed up linear optics
in SixTrack. They are kept as distinct elements and are not joined with
surrounding drifts.

gxx11c.c, gxx11psc.c
- Fix the month number in the ps files

madxc.c
- Added commands to allow reading of external orbit files
- Correct read_my_table for long data elements

madxe.c
- compiler warnings removed
- Changed default behaviour of ADD option

madxn.c, madxp.c, madxu.c
- moved aperture code from madxn.c to new file aperture.c
corrections:
- more apex in halo polygon
- corrected the construction of rectellipse in the general case
- secured potentially dangerous division by zero
- Avoid divisions by zero in the aperture module.
- minor C inconsistencies and some clean-up
- mad-X_3_66  bugs found by EK
- Array overflow reported from valgrind removed
- Special non-existing End marker has been dropped. courtesy HG
- mpar made compiling, three functions moved from matchc2.c to madxu.c
- current in table_list initialized to 0
- add warning when use_macro option is used with too many variables
- Introducing aptol_(1,2,3) "rtol", "xtol", "ytol" to be available in MAD-X
input and e.g. to be added to a TWISS table.       Courtesy H.G.
Now table2(x,y) or stable(a,b,c) or things like that will not be modified.
- First step for node layout tracking in ptc_trackline
- fix a bug when you use a macro which define another macro for matching
- madX-3_03_48: "Final" fix of the TILT saga! Tilt is calculated exclusively
in twiss.F following the strategy:
0) These changes concern quad, sext, oct, elec separator;
```

   but NOT dipole or multipole
1) TILT input is the external tilt (+k ==> +ks for tilt < 0)
2) k & ks represent an internal tilt
3) at each element the total tilt & sqrt(k**2 + ks**2) is calculated
   including field errors, i.e. the correct way which might cause
   differences with MAD8
4) PTC has been adjusted appropriate
5) Possible effects on "survey" and "emit" will be tested
6) Many Thanks for HG for his help!
- Bug corrected, replaced abs with fabs
- Updated match with knobs
- Constraints for ranges with match,use_macro; implemented
- Debugging
a) madxp: Wrong sign of TILT when calculated from k & KS
b) madx_ptc_module: Total rewrite of TILT stuff!
- tilt clean-up coutesy HG
- Moments calculation fully imlemented; map buffering in ptc_twiss
- New ptc_twiss, so A_ is tracked. This makes possible tracking of moments
(to be completed).
- Introducing aptol_(1,2,3) "rtol", "xtol", "ytol" to be available in MAD-X
input and e.g. to be added to a TWISS table.   Courtesy H.G.
- Adding gino command following madx_ptc_script_module
Problems:
1) pt not displayed in table.
2) deltap not in ptc_twiss header unless a twiss command was done before
   ptc_twiss!
Solution:
1) Set up y properly
2) Put "deltap" into header of the ptc_twiss table permanently
- Fix routine readrematrix by exchanging x(5) and x(6)
- Fixing the plot crash: the header is not read if it does not exist
(courtesy HG).
-
1) Stefan Sorge new module keeper!!!!
2) Second order detuning with proper "hor" and "ver" names. In fact,
vertical terms were missing.
-
1) Bad memory bug found by RdM. Piotr found using valgrind the solution:
in mymalloc("read_table", strlen(aux_buff->c)+1) the "+1" is essential
because the character string has a "\0" at the end.
2) In plot there is still the crash when reading the header - temporary
solution applied.
- Fix the headvalue routine which stumbled over a blank line. (courtesy HG)
- First step for node layout tracking in ptc_trackline
- Adding a C routine to read headers of TFS tables (courtesy HG)
- Taking deltap from table header both for TWISS and ptc_twiss (but
  NOT in case of a SUMM table!) and place on the plot.
- Move the set_variable routine from madxn.c to madxu.c. Needed to compile
mpars.
- new setvar command; fixed readtable bug
- Removed redundant debug printout
- Exact name matching implemented, now passing name with :x
- VORNAME assigned, the same as name of node in MADX but with capital letters
- Updated match with knobs
- Constraints for ranges with match,use_macro; implemented
- Adding "e1", "e2", "h1", "h2", "hgap", "fint", "fintx" to the twiss table
- Removed charge setting to the my_ring layout to make ptc_twiss running,
redundant printouts removed
- Moments calculation fully imlemented; map buffering in ptc_twiss
- Few bugs corrected (f.g. map initialized to nd2 instead of npara when
initial twiss provided). Moments seem to work (to be tested yet)
- New ptc_twiss, so A_ is tracked. This makes possible tracking of moments
(to be completed).
-Not enough memory for buffers ==> FODO: LINE=(36*(CELL)); failed when cell
was a line itself
- clean-up of gino
- Adding gino command following madx_ptc_script_module
- PTC knobs (pol_blocks) almost completely interfaced to MAD-X.
User sets a knob with ptc_knob command.
Twiss parameters and user specified (with ptc_select) map components are
buffered in memory after every element in form of taylor series.
 The problem with table (and tabstring) replacement was correctly
stated, only in this case did simple string replacement take place (bad
implememtation, my mistake). It has been corrected, the files concerned
are madxd.h, madxp.c, and madxu.h in ~hansg/public/tmp
Now table2(x,y) or stable(a,b,c) or things like that will not be modified.
- In new_command_parameter_list pointer array of parameters initized with NULLs
- Write only long in TFS tables
- "string function" tabstring count start at 1 courtesy Hans Grote
- Now alos ptc_normal accepted in matching with ptcknobs; bug corrections
- Implemented:
1. ptc_setfieldcomp that set any order field strengh
to requested value. It enables matching of higher order field components.
2. Special matching mode use_ptcknob. It implements kind of macro
that emplys parametric PTC calculations to perform matching in a faster manner.
For further details see the comments at the top of matchptcknobs.c file.
3. Minor corrections and protections against segmentation vilation.
- Implemented:
- Problem:
[1] Macro names and clashes with internal names
! OK
mycrap(xx,yy,zz): macro = {ingvar = table(xx,yy,zz);};
somenamesarelong(xx,yy,zz): macro = {ingvar = table(xx,yy,zz);};
! not OK
table2(xx,yy,zz): macro = {ingvar = table(xx,yy,zz);};
redtableclothing(xx,yy,zz): macro = {ingvar = table(xx,yy,zz);};
readtable,
file="/afs/cern.ch/user/h/hagen/public/MAD/ALL-emfq-0001.tfs";
Solution:courtesy Hans Grote
- PTC knobs (pol_blocks) almost completely interfaced to MAD-X.
User sets a knob with ptc_knob command.
Twiss parameters and user specified (with ptc_select) map components are
buffered in memory after every element in form of taylor series.
They can be dumped to text file in two formats with ptc_printparametric command.
They can be also visualized and further studied with rviewer from rplot plugin.
Further, user can set numeric values of knobs with ptc_setknobvalue what
updates all numeric values of the parameters in the tables.
This way knobs can be used in matching.
- New tracking feature by Andres Gomez Alonso:
Using flag "recloss" in the tracking command creates a table called
"trackloss", which keeps a record of lost particles. It can be saved
using the "write, table= trackloss" command.
- Changes for SDDS and online model
- knobs implemented with PTC with pol_blocks; command to dump parametric
results to file or stdout; content of ptc_madx_tablepush.f90 moved to
ptc_madx_knobs.f90, the former one removed
- Correct bug in read_table: can read long integers now
- Sceleton for knobs and arbitrary element placement implemented. Lattice
visualization via ROOT macro. Printing detailed lattice geometry in PTC.
Several small bug corrections and some code cosmetics.
- Sodd table names usinf small letters only
- closing unit 34 to allow multiple SODD runs
- changing table entries to more logical names
- print   out clean up

- Proper initialization of pointers to NULL's, added function for deleting
command_list structures
- "string function" tabstring count start at 1 courtesy Hans Grote
- New function "exist" courtesy Hans Grote
- Add a ';' to the error message for a not found variable in the 'show' command
- Error flag implemented that signals that error code occured
- If a user table is used by ptc_select then org_cols=num_cols.
Otherwise all values are overwritten by add_vars_to_table function.
- PTC_Enforce6D implemented

madxsdds.c
- Add SDDS module

makethin.c
- Fixing TILT in multipole kick and make TILT proper in thick octupole.
Courtesy HG
- Changing the conflicting "ksl" for the integrated solenoid strength to
"ksi". This name is reserved for the vector of the integrated skew
multipoles "ksl={};". Thick solenoid can now have normal "knl" and skew
"ksl" multipole errors in PTC, ignored in madx proper. Thin solenoids
are presently not considered in PTC.

matchc.c, matchc2.c
- knobs: better definition
- knobs file defines with ":="
- Experimental knob file generation
- fix a bug when you use a macro which define another macro for matching
- Comment corrected
- Syntax error at 405: code lines must come after declarations at least for
Windows "CL".
- Printouts redirected to prt_file
- mpar made compiling, three functions moved from matchc2.c to madxu.c
- Updated match with knobs
- Constraints for ranges with match,use_macro; implemented
- fix compilation warnings
- add warning when use_macro option is used with too many variables
- New jacobian routine with svd. Option COND added for controlling the SVD.
Increased number of constraints
- Corrected bug in match use_macro in match2_evaluate_exressions
- change in the print out of match summary when USE_MACRO
- Redundant printf removed
- Implemented:
1. ptc_setfieldcomp that set any order field strengh
to requested value. It enables matching of higher order field components.
2. Special matching mode use_ptcknob. It implements kind of macro
that emplys parametric PTC calculations to perform matching in a faster manner.
For further details see the comments at the top of matchptcknobs.c file.
3. Minor corrections and protections against segmentation vilation.
- Error flag is monitored in mtcond so if an error occured during macro
execution it is handled appropriately.

matchptcknobs.c
- Functions defined elsewhere should be defined with extern so linker does not
complain about multiple definitions
- Algorithm made more stable
- Typo corrected
- introduced correct treatment of magnet families
- Protection against deletion of NULL pointer added
- Parametric matching of initial conditions works now, final tests and debugging
to be done
- Updated match with knobs
- New ptc_twiss, so A_ is tracked. This makes possible tracking of moments
(to be completed).
- Corrected bug in match use_macro in match2_evaluate_exressions
- Now alos ptc_normal accepted in matching with ptcknobs; bug corrections
- Implemented:
1. ptc_setfieldcomp that set any order field strengh
to requested value. It enables matching of higher order field components.
2. Special matching mode use_ptcknob. It implements kind of macro
that emplys parametric PTC calculations to perform matching in a faster manner.
For further details see the comments at the top of matchptcknobs.c file.
3. Minor corrections and protections against segmentation vilation.

rplot.c
- On some systems it is needed to load manually all needed ROOT libraries
- Bug correction
- Now alos ptc_normal accepted in matching with ptcknobs; bug corrections
- Code cosmetics
- Turn number added in rplot
- rviewer plugin intefaced; from now on rplot is a plugin instead of compiled
in optional code
- Sever bug in knobs corrected: attempt to delete not properly assocoated taylors
in case no knobs are set by the user

C Header Files:
---

madx.h
- Add parameter (units) for orbit correction
- Added: Enumeration type for matching mode; protection against multiple
inclusion.

madxl.h
- Definitions for tables used by Slice Tracking with PTC
- Momentum compaction "alfa" included into TWISS table for matching.
- Add "polarity" parameter to the twiss table
- Updated match with knobs
- Adding "e1", "e2", "h1", "h2", "hgap", "fint", "fintx" to the twiss table
- Suppressing "imax" in favor of "calib" - request by Thys Risselada
- Adding node value "kmax" (maximum K value) and "imax" (maximum Current value)
- New tracking feature by Andres Gomez Alonso:
Using flag "recloss" in the tracking command creates a table called
"trackloss", which keeps a record of lost particles. It can be saved
using the "write, table= trackloss" command.
- Added madX data types , mainly used in SDDS module
- New function "exist" courtesy Hans Grote
- Sodd table names usinf small letters only
- closing unit 34 to allow multiple SODD runs
- changing table entries to more logical names
- print   out clean up

madxdict.h
- Experimental knob file generation
- Added option that tells to free memory at the end of the program execution.
Option for ptc_trackline added that switches on/off track parameters storage
in memory for every slice
- Fix traditional matching of alfa
- First step for node layout tracking in ptc_trackline
- Added ptc names of twiss functions to constraint, hence one can set
constraint in a range
- new setvar command; fixed readtable bug
- Add "polarity" parameter to the twiss table
- Clean-up
- wrap.f90

- Fixing the crash for sbend + exact + multipoles larger than 10. This set-up
requires to solve Maxwell's equation up to SECTOR_NMUL_MAX. The default is
set to 10 to avoid excessive computing time. This is now safeguarded in
madxp. To this end the parameters SECTOR_NMUL and SECTOR_NMUL_MAX are
transfered from "ptc_create_layout" to "ptc_create_universe" such that
these global parameters can be set early enough. Internally in PTC the
parameter "lda_used" is incremented where needed from 1500 to 3000 and set
back. Moreover Etienne has done the following modifications to make this
possible:
"The modification I made in the new PTC I sent you are as follows:
You first select SECTOR_NMUL and SECTOR_NMUL_MAX. For all
multipole <= SECTOR_NMUL then maxwell's is solved to order SECTOR_NMUL_MAX.
For multipole above SECTOR_NMUL , they are treated a la Sixtrack.
So for example, it you have errors to order 20, you may bother with maxwells
only to order nmul=4 and nmul_max=10 as far as Maxwell's is concerned.
Multipole higher will be sixtrack multipoles.
- Updated match with knobs
-fix bug for select_ptc_normal
- Added commands to allow reading external orbit files
- Moments calculation fully imlemented; map buffering in ptc_twiss
- Few bugs corrected (f.g. map initialized to nd2 instead of npara when
initial twiss provided). Moments seem to work (to be tested yet)
- New ptc_twiss, so A_ is tracked. This makes possible tracking of moments
(to be completed).
- New jacobian routine with svd. Option COND added for controlling the SVD.
Increased number of constraints
- Adding gino command following madx_ptc_script_module
- Suppressing "imax" in favor of "calib" - request by Thys Risselada
- Adding node value "kmax" (maximum K value) and "imax" (maximum Current value)
- drop useless lcavity since ywcavity has same mad-8 element code
- Implemented:
1. ptc_setfieldcomp that set any order field strengh
to requested value. It enables matching of higher order field components.
2. Special matching mode use_ptcknob. It implements kind of macro
that emplys parametric PTC calculations to perform matching in a faster manner.
For further details see the comments at the top of matchptcknobs.c file.
3. Minor corrections and protections against segmentation vilation.
- PTC knobs (pol_blocks) almost completely interfaced to MAD-X.
User sets a knob with ptc_knob command.
Twiss parameters and user specified (with ptc_select) map components are
buffered in memory after every element in form of taylor series.
They can be dumped to text file in two formats with ptc_printparametric command.
They can be also visualized and further studied with rviewer from rplot plugin.
Further, user can set numeric values of knobs with ptc_setknobvalue what
updates all numeric values of the parameters in the tables.
This way knobs can be used in matching.
- New tracking feature by Andres Gomez Alonso:
Using flag "recloss" in the tracking command creates a table called
"trackloss", which keeps a record of lost particles. It can be saved
using the "write, table= trackloss" command.
- change in beambeam command: usage of scattering beam
with different radial shapes is possible:
parameters:
bbshape: 1 (default) Gaussian, standard as before
         2 flattop (or trapezoidal)
         3 hollow-parabolic
width: for bbshape=2: fractional width of edge region
         for bbshape=3: fractional width of the parabolic part
- knobs implemented with PTC with pol_blocks; command to dump parametric
results to file or stdout; content of ptc_madx_tablepush.f90 moved to
ptc_madx_knobs.f90, the former one removed
- Comment removed
- Element placement options added
- Commands for SDDS read and write
- Sceleton for knobs and arbitrary element placement implemented. Lattice
visualization via ROOT macro. Printing detailed lattice geometry in PTC.
Several small bug corrections and some code cosmetics.
- Introducing TRUERBEND and WEDGRBEND in PTC. To this end 2 flags have been
introduced in the NAD-X dictionary madxdict.h:
1) ptcrbend: if true it uses a PTC type RBEND
2) truerbend: if true it uses TRUERBEND; if false it uses WEDGRBEND
- Option for ptc_trackline position given in global coordinate system added.
- Changing the conflicting "ksl" for the integrated solenoid strength to
"ksi". This name is reserved for the vector of the integrated skew
multipoles "ksl={};". Thick solenoid can now have normal "knl" and skew
"ksl" multipole errors in PTC, ignored in madx proper. Thin solenoids
are presently not considered in PTC.
- PTC_Enforce6D implemented
- defaultlevel default has been 0 now 1 as planned originally

matchptcknobs.h
- Updated match with knobs
- Now alos ptc_normal accepted in matching with ptcknobs; bug corrections
- Implemented:
1. ptc_setfieldcomp that set any order field strengh
to requested value. It enables matching of higher order field components.
2. Special matching mode use_ptcknob. It implements kind of macro
that emplys parametric PTC calculations to perform matching in a faster manner.
For further details see the comments at the top of matchptcknobs.c file.
3. Minor corrections and protections against segmentation vilation.

rplot.h
- Code cosmetics
- Turn number added in rplot
- rviewer plugin intefaced; from now on rplot is a plugin instead of compiled in
optional code
- Sever bug in knobs corrected: attempt to delete not properly assocoated taylors
in case no
knobs are set by the user

sdds.h
- New files for MAD-X On-Line Modeling Version

FORTRAN Files:

Changes:
----

emit.F and all Fortran Files
- Clean-up of unused variables
- Fortran Clean-Up

match.F
- tentative new output for matching var

matchjc.F
- better ending jacobian
- Penalty function printed with twise larger precision
- New jacobian routine with svd. Option COND added for controlling the SVD.
Increased number of constraints
- change in calls behavior for JACOBIAN

matchlib.F

```
1) Just the routines needed ==> 5 times smaller: 33% ==> 10%
2) Fortran90 Clean-up
- Changes needed to compile routine dlamc1 without optimization in extra file
matchlib2.F. Otherwise madx gets stuck in matching procedures. In all Linux
Makefiles matchlib2.F is compiled when using g77. For the Fortran90 compilers
lf95, g95, f95(NAG) and gfortran an optimized routine is used as provided by
Andy Vaught, the "g95" maintainer. For Windows the special compile flag:
-lfe "-D_G95" was needed (special undocumented Fujitsu compile flag of
the Lahey lf95) to compile this special Fortran90 version of dlamc1.
- Replacing the DLAMC1 by an improved Fortran90 one as proposed by
Andy Vaught
- remove Makefile_nag offending code
- New jacobian routine with svd. Option COND added for controlling the SVD.
Increased number of constraints

matchlib2.F
- Needed to compile routine dlamc1 without optimization. Otherwise madx
gets stuck in matching procedures.

orbf.F
- no blanks between & position 6 and the start of the code.

plot.F
- Add energy label "E" in routine "pegetn", I.E. 69th item of svanno(69)=E
- Safeguard uncontrolled access to unsupported labels like Energy
- Fixing the plot crash: the header is not read if it does not exist
(courtesy HG).
-
1) Bad memory bug found by RdM. Piotr found using valgrind the solution:
in mymalloc("read_table", strlen(aux_buff->c)+1) the "+1" is essential
because the character string has a "\0" at the end.
2) In plot there is still the crash when reading the header - temporary
solution applied.
- Fix the headvalue routine which stumbled over a blank line. (courtesy HG)
- Preliminary fix of the crash when the aperture table is used
- Adding a C routine to read headers of TFS tables (courtesy  HG)
- Taking deltap from table header both for TWISS and ptc_twiss (but
  NOT in case of a SUMM table!) and place on the plot.
- Blanking out buffer if ptc_flag = true
- deltap plot entry taken from PTC plots

ptc_dummy.F
- First step for node layout tracking in ptc_trackline
- Missing dummy definitions added
- Moments calculation fully imlemented; map buffering in ptc_twiss
- New ptc_twiss, so A_ is tracked. This makes possible tracking of moments
(to be completed).
- Adding gino command following madx_ptc_script_module
- Implemented:
1. ptc_setfieldcomp that set any order field strengh
to requested value. It enables matching of higher order field components.
2. Special matching mode use_ptcknob. It implements kind of macro
that emplys parametric PTC calculations to perform matching in a faster manner.
For further details see the comments at the top of matchptcknobs.c file.
3. Minor corrections and protections against segmentation vilation.
- PTC knobs (pol_blocks) almost completely interfaced to MAD-X.
User sets a knob with ptc_knob command.
Twiss parameters and user specified (with ptc_select) map components are
buffered in memory after every element in form of taylor series.
They can be dumped to text file in two formats with ptc_printparametric command.
They can be also visualized and further studied with rviewer from rplot plugin.
Further, user can set numeric values of knobs with ptc_setknobvalue what
updates all numeric values of the parameters in the tables.
This way knobs can be used in matching.
- knobs implemented with PTC with pol_blocks; command to dump parametric results
to file or stdout; content of ptc_madx_tablepush.f90 moved to ptc_madx_knobs.f90,
the former one removed
- Sceleton for knobs and arbitrary element placement implemented. Lattice
visualization via ROOT macro. Printing detailed lattice geometry in PTC. Several
small bug corrections and some code cosmetics.
- PTC_Enforce6D implemented

sodd.F
- Suppression of excessive printing (courtesy SS)
-
1) Stefan Sorge new module keeper!!!!
2) Second order detuning with proper "hor" and "ver" names. In fact,
vertical terms were missing.
- Close unit 34 even on error output
- Sodd table names usinf small letters only
- closing unit 34 to allow multiple SODD runs
- changing table entries to more logical names
- print   out clean up

trrun.F
- The argument "el" was removed from argument list of subroutine tmarb called
here (line 532 in trrun.F), because it did not coincide with the argument
list of the real subroutine tmarb in twiss.F.
- Bug found by Stefan Sorge in trrun/trinicmd: Variables 'bet0' and 'bet0i' were
undefined, leading to erronous results. Corrected.
- Fixing: Unwanted changes commented.
- The aperture of the collimators was checked before undergoing the (possible)
rotation at the entry of the element. Corrected now: If collimator has a roll
angle ('tilt'), coordinates are transformed to the rotated local coordinate
system and only after that apertures are checked.
- Add BV flag to the solenoids
- Some unnecessary changes taken out again by Andres Gomez Alonso.
- Inconsistent variable declarations of z0 encountered by Piotr when using
make -f Makefile_nag. Fixed.
- New tracking feature by Andres Gomez Alonso:
Using flag "recloss" in the tracking command creates a table called
"trackloss", which keeps a record of lost particles. It can be saved
using the "write, table= trackloss" command.
- safeguard faulty input
- change in beambeam command: usage of scattering beam
with different radial shapes is possible:
parameters:
bbshape: 1 (default) Gaussian, standard as before
         2 flattop (or trapezoidal)
         3 hollow-parabolic
width:   for bbshape=2: fractional width of edge region
         for bbshape=3: fractional width of the parabolic part
- Changing the conflicting "ksl" for the integrated solenoid strength to
"ksi".  This name is reserved for the vector of the integrated skew
multipoles "ksl={};". Thick solenoid can now have normal "knl" and skew
"ksl" multipole errors in PTC, ignored in madx proper. Thin solenoids
are presently not considered in PTC.

twiss.F
- Closed orbit implemented in the maps for the beambeam element with
flattop and hollow parabolic radial density profile, i.e. in the
subtoutines tmbb_flattop and tmbb_hollowparabolic in twiss.F
- Momentum compaction "alfa" included into TWISS table for matching.
- Add "polarity" parameter to the twiss table
```

- madX-3_03_48: "Final" fix of the TILT saga! Tilt is calculated exclusively
in twiss.F following the strategy:
0) These changes concern quad, sext, oct, elec separator;
   but NOT dipole or multipole
1) TILT input is the external tilt (+k ==> +ks for tilt < 0)
2) k & ks represent an internal tilt
3) at each element the total tilt & sqrt(k**2 + ks**2) is calculated
   including field errors, i.e. the correct way which might cause
   differences with MAD8
4) PTC has been adjusted appropriate
5) Possible effects on "survey" and "emit" will be tested
6) Many Thanks for HG for his help!
- safeguard atan2 against both arguments equal to zero
- Fixing TILT in multipole kick and make TILT proper in thick octupole.
Courtesy HG
- tilt clean-up coutesy HG
- Suppressing "imax" in favor of "calib" - request by Thys Risselada
- Adding node value "kmax" (maximum K value) and "imax" (maximum Current value)
- Add BV flag to the solenoids
- Correction of an error occurring in subroutine tmbb_flattop for ftrk=.false.
- safeguard faulty input
- change in beambeam command: usage of scattering beam
with different radial shapes is possible:
parameters:
bbshape: 1 (default) Gaussian, standard as before
         2 flattop (or trapezoidal)
         3 hollow-parabolic
width: for bbshape=2: fractional width of edge region
         for bbshape=3: fractional width of the parabolic part
- Changing the conflicting "ksl" for the integrated solenoid strength to
"ksi". This name is reserved for the vector of the integrated skew
multipoles "ksl={};". Thick solenoid can now have normal "knl" and skew
"ksl" multipole errors in PTC, ignored in madx proper. Thin solenoids
are presently not considered in PTC.

user2_photon.f90
- unnused dummy variable DMASS in subr. photon is removed

util.F
- Added seterrorflag routine that sets the error flag in c part if an error
occured.

FORTRAN Include Files:
----

plot.fi,touschek.fi, twiss0.fi, twissc.fi, win32calls.fi
- Clean-up of fi files
- Momentum compaction "alfa" included into TWISS table for matching.
- Add "polarity" parameter to the twiss table
- Adding node value "kmax" (maximum K value) and "imax" (maximum Current value)

==========

PTC MODULES

Changes:

- Throw out unused variables
- Cleaned code so NAG warnings are minimized now: mainly unused variables
- Fortran Clean-Up

madx_ptc_distrib.inc
- New ptc_twiss, so A_ is tracked. This makes possible tracking of moments
(to be completed).

madx_ptc_knobs.inc
- Bug Corrected: Parametric twiss results where not scaled with energy
- Moments calculation fully imlemented; map buffering in ptc_twiss
- New ptc_twiss, so A_ is tracked. This makes possible tracking of moments
(to be completed).
- Implemented:
1. ptc_setfieldcomp that set any order field strengh
to requested value. It enables matching of higher order field components.
2. Special matching mode use_ptcknob. It implements kind of macro
that emplys parametric PTC calculations to perform matching in a faster manner.
For further details see the comments at the top of matchptcknobs.c file.
3. Minor corrections and protections against segmentation vilation.
- knobs implemented with PTC with pol_blocks; command to dump parametric
results to file or stdout; content of ptc_madx_tablepush.f90 moved to
ptc_madx_knobs.f90, the former one removed

madx_ptc_distrib.f90
- Moving from DAmap to Gmap
- Fortran Clean-Up
- Moments updated, initialization for 5D in twiss, moments not available in 5D
due to a bug and few others
- Removed charge setting to the my_ring layout to make ptc_twiss running,
redundant printouts removed
- Moments calculation fully imlemented; map buffering in ptc_twiss
- Few bugs corrected (f.g. map initialized to nd2 instead of npara when
initial twiss provided). Moments seem to work (to be tested yet)
- New ptc_twiss, so A_ is tracked. This makes possible tracking of moments
(to be completed).

madx_ptc_eplacement.f90
- Root display support for new elements
- serious memory leak removed
- Added sextupoles, octupoles and not powered elements
- Corrected RBEND drawing
- Set of kinds added to drawing in root
- In root macro generation: added protection against inclusion of headers in
interpreted mode; automatical switch to white background; redundant debug
printout removed.
- Added header lines to ROOT macro that display layout geometry. This makes
 possible to compile a macro, what is a must in the case of lengthy machines.
- Removed redundant debug printout
- 2 interface routines added needeed by rplot
- Bugs with rotations corrected
- element placement works since now

madx_ptc_intstate.f90
- New ptc_twiss, so A_ is tracked. This makes possible tracking of moments
(to be completed).
- add (lp) to logical function; indenting
- Enforce 6d implemented
- PTC_Enforce6D implemented
- defaultlevel default has been 0 now 1 as planned originally

madx_ptc_knobs.f90
- Bug Corrected: Parametric twiss results where not scaled with energy
- Exact name matching implemented, now passing name with :x
- Knobs for Initial parameters
- New ptc_twiss, so A_ is tracked. This makes possible tracking of moments

(to be completed).
- PTC first changes stay October 2006 thinlens, cutting
- Updated to the new nomenclature (beta12->beta22,...)
- Bug corrected in the treatment of 4D and 5D cases; cosmetics;
- Universal taylor nullified at the initialization level
- Implemented:
1. ptc_setfieldcomp that set any order field strengh
to requested value. It enables matching of higher order field components.
2. Special matching mode use_ptcknob. It implements kind of macro
that emplys parametric PTC calculations to perform matching in a faster manner.
For further details see the comments at the top of matchptcknobs.c file.
3. Minor corrections and protections against segmentation vilation.
- Do not print to file trailing blanks in the buffer
- PTC knobs (pol_blocks) almost completely interfaced to MAD-X.
User sets a knob with ptc_knob command.
Twiss parameters and user specified (with ptc_select) map components are
buffered in memory after every element in form of taylor series.
They can be dumped to text file in two formats with ptc_printparametric command.
They can be also visualized and further studied with rviewer from rplot plugin.
Further, user can set numeric values of knobs with ptc_setknobvalue what
updates all numeric values of the parameters in the tables.
This way knobs can be used in matching.
-Severe bug in knobs corrected: attempt to delete not properly assocoated
taylors in case no knobs are set by the user
- knobs implemented with PTC with pol_blocks; command to dump parametric
results to file or stdout; content of ptc_madx_tablepush.f90 moved to
ptc_madx_knobs.f90, the former one removed
- Debug info printed only at appropriate debuglevel
- Sceleton for knobs and arbitrary element placement implemented.
Lattice visualization via ROOT macro. Printing detailed lattice geometry in PTC.
Several small bug corrections and some code cosmetics.

madx_ptc_module.f90
- RF cavity treated as TW cavity now
-
1) madx_ptc_module without the ptc_normal stuff
2) fixing the "savemaps" bug
- Fix to restricted print out format
-
1) Se_status: preliminary fix of uninitialized variable RADIATION_NEW
2) madx_ptc_module: Fix of "ptc_normal" by fixing the string comparison
3) madx_ptc_module & madx_ptc_twiss: write & read traditional DA map format
-
PTC version including spin
- madX-3_03_48: "Final" fix of the TILT saga! Tilt is calculated exclusively
in twiss.F following the strategy:
0) These changes concern quad, sext, oct, elec separator;
   but NOT dipole or multipole
1) TILT input is the external tilt (+k ==> +ks for tilt < 0)
2) k & ks represent an internal tilt
3) at each element the total tilt & sqrt(k**2 + ks**2) is calculated
   including field errors, i.e. the correct way which might cause
   differences with MAD8
4) PTC has been adjusted appropriate
5) Possible effects on "survey" and "emit" will be tested
6) Many Thanks for HG for his help!
- Redundant debug info available only in high level debug mode
- VORNAME assigned, the same as name of node in MADX but with capital letters
- Too long line split into 3 lines
- useless print statements
- Fixing the crash for sbend + exact + multipoles larger than 10. This set-up
requires to solve Maxwell's equation up to SECTOR_NMUL_MAX. The default is
set to 10 to avoid excessive computing time. This is now safeguarded in
madxp. To this end the parameters SECTOR_NMUL and SECTOR_NMUL_MAX are
transfered from "ptc_create_layout" to "ptc_create_universe" such that
these global parameters can be set early enough. Internally in PTC the
parameter "lda_used" is incremented where needed from 1500 to 3000 and set
back. Moreover Etienne has done the following modifications to make this
possible:
"The modification I made in the new PTC I sent you are as follows:
You first select SECTOR_NMUL and SECTOR_NMUL_MAX. For all
multipole <= SECTOR_NMUL then maxwell's is solved to order SECTOR_NMUL_MAX.
For multipole above SECTOR_NMUL , they are treated a la Sixtrack.
So for example, it you have errors to order 20, you may bother with maxwells
only to order nmul=4 and nmul_max=10 as far as Maxwell's is concerned.
Multipole higher will be sixtrack multipoles.
- Updated match with knobs
- Fix dum1 dum2 definition
- Missing declarations of dum1 and dum2 added
- Debugging
a) madxp: Wrong sign of TILT when calculated from k & KS
b) madx_ptc_module: Total rewrite of TILT stuff!
- Corrected state for 56D
- In 4D, before setting in internal state only_4D we remove delta, otherwise
delta stays
- replacing "asin" by "-atan2" thereby fixing the sign for quad, sext, oct
- set my_ring%charge=1 !preliminary fix
- Removed charge setting to the my_ring layout to make ptc_twiss running,
redundant printouts removed
- Moments calculation fully imlemented; map buffering in ptc_twiss
- Initialize mass(pma) and charge of MY_RING before set_madx
- New ptc_twiss, so A_ is tracked. This makes possible tracking of moments
(to be completed).
- Initial orbit NOT closed orbit for initial betax
- In case of instabilility in normal form, the code sets the global error
flag and returns to the main command loop instead of fatal
- fulfilled formalistic request for a change of the definition of numenclature
of the ptc_twiss variables: beta, alfa and gama
- Making unstable behavior in NormalForm a fatal error
- All PTC track commands and NormalForm executions are checked for unstable
behavior
- First fill user tables and at the end TWISS table
- Implemented:
1. ptc_setfieldcomp that set any order field strengh
to requested value. It enables matching of higher order field components.
2. Special matching mode use_ptcknob. It implements kind of macro
that emplys parametric PTC calculations to perform matching in a faster manner.
For further details see the comments at the top of matchptcknobs.c file.
3. Minor corrections and protections against segmentation vilation.
- PTC knobs (pol_blocks) almost completely interfaced to MAD-X.
User sets a knob with ptc_knob command.
Twiss parameters and user specified (with ptc_select) map components are
buffered in memory after every element in form of taylor series.
They can be dumped to text file in two formats with ptc_printparametric command.
They can be also visualized and further studied with rviewer from rplot plugin.
Further, user can set numeric values of knobs with ptc_setknobvalue what
updates all numeric values of the parameters in the tables.
This way knobs can be used in matching.
- Severe bug in knobs corrected: attempt to delete not properly assocoated
taylors in case no knobs are set by the user
- knobs implemented with PTC with pol_blocks; command to dump parametric
results to file or stdout; content of ptc_madx_tablepush.f90 moved to
ptc_madx_knobs.f90, the former one removed
- Corrected twiss with parameters

- Sceleton for knobs and arbitrary element placement implemented.
Lattice visualization via ROOT macro. Printing detailed lattice geometry in PTC.
Several small bug corrections and some code cosmetics.
- Introducing TRUERBEND and WEDGRBEND in PTC. To this end 2 flags have been
introduced in the NAD-X dictionary madxdict.h:
1) ptcrbend: if true it uses a PTC type RBEND
2) truerbend: if true it uses TRUERBEND; if false it uses WEDGRBEND
- logical lp -> 4 and vice versa so NAG does not cry
- problem causing compiler warning removed
- open and close of unit 21 only for "getdebug() > 2"
- Error flag implemented that signals that error code occured
- Changing the conflicting "ksl" for the integrated solenoid strength to
"ksi". This name is reserved for the vector of the integrated skew
multipoles "ksl={};". Thick solenoid can now have normal "knl" and skew
"ksl" multipole errors in PTC, ignored in madx proper. Thin solenoids
are presently not considered in PTC.
- Check of initial conditions provided by the user on imput.
- PTC_Enforce6D implemented. If 6D TWISS calculation is performed with
 initial conditions (beta0 block) then non-zero betz is required
- Fix writing 5/5 components of closed_orbit to twiss table
- Bug corrected: writing maps should not be only in debug mode

madx_ptc_normal.f90
- Pulled out the stuff for the "ptc_normal" module since
"madx_ptc_module" is already very large.

madx_ptc_script.f90
- Adding gino command following madx_ptc_script_module

madx_ptc_setcavs.f90
- Cleaned code so NAG warnings are minimized now: mainly unused variables
- RF cavity treated as TW cavity now, bug correction
- temporary fix of non-existing "cav21" member
- RF cavity treated as TW cavity now
- Error flag implemented that signals that error code occured

madx_ptc_tablepush.f90
- knobs implemented with PTC with pol_blocks; command to dump parametric
results to file or stdout; content of ptc_madx_tablepush.f90 moved to
ptc_madx_knobs.f90,
the former one removed
- Sceleton for knobs and arbitrary element placement implemented.
Lattice visualization via ROOT macro. Printing detailed lattice geometry in PTC.
Several small bug corrections and some code cosmetics.
- Proper handling of 6D: 5th column and row are swapped with 6th ones.
- Bug corrected. Added support for 6D case - 5th column and row is swapped with
the 6th one then.
- debug level 9 removed completely

madx_ptc_track_run.f90
- unnused dummy variable in subr. photon is removed
- Unused variables detected with Makefile_nag are removed
- the re-initialization of NaN-flags
- ICASE=56 created a fatal bug, fixed by setting ICASE=5.
- ICASE other than 4, 5, (56->5), 6 throws a fatal error
- For safety a "implicit none" statement plugged into every subroutine and also
every function, independent if strictly needed or not!
- PTC_Track tables: The last element is END (not START)
- NaN-tracks by TRACK of PTC are blocked and tracking is terminated
- Serious crash in ptc_track for unstable particles due to unitialized PTC
aperture check variable.
- Converted from windows to unix encoding
- Bug in madx_ptc_track_run.f90 for ICASE=5 is fixed (Numb. of Eigens => 4)
- Debug output only at debuglevel=4

madx_ptc_trackcavs.f90
- Removed bug in node tracking making impossible tracking for closed layouts
- First step for node layout tracking in ptc_trackline
- New ptc_twiss, so A_ is tracked. This makes possible tracking of moments
(to be completed).
- Track Global Coordinates corrected -> position is given with respect to
fibre not magnet
- Turn number added in rplot
- Added switch to ptc_trackline command so user can choose if tracks
shall be written to a ROOT ntuple. The feature is only accessible if rplot
plugin is installed and madx is compiled with plugin support.
- Sceleton for knobs and arbitrary element placement implemented.
Lattice visualization via ROOT macro. Printing detailed lattice geometry in PTC.
Several small bug corrections and some code cosmetics.
- Change "logical(dp)" to the correct "logical(lp)"
- Error flag implemented that signals that error code occured; Option for
track position given in global coordinate system added.
- debug level 9 removed completely

madx_ptc_twiss.f90,v
- mad-X_3_66 bugs found by EK
- Problems:
1) pt not displayed in table.
2) deltap not in ptc_twiss header unless a twiss command was done before
   ptc_twiss!
Solution:
1) Set up y properly
2) Put "deltap" into header of the ptc_twiss table permanently
- Fix routine readrematrix by exchanging x(5) and x(6)
- Adding deltap to ptc_twiss in case of initial conditions.
-
1) madx_ptc_module without the ptc_normal stuff
2) fixing the "savemaps" bug
- Don't display orbit in debug 0 mode
-
1) Se_status: preliminary fix of uninitialized variable RADIATION_NEW
2) madx_ptc_module: Fix of "ptc_normal" by fixing the string comparison
3) madx_ptc_module & madx_ptc_twiss: write & read traditional DA map format
- Type twiss becomes "public" to overcome problem with g95. Probably okay
in g95 but may be overly picky.
- Bug corrected (p0c was written to twiss table instead of energy
- Bug Corrected: Parametric twiss results where not scaled with energy
- Few protections agains seg faults added. Redundant debug info available
only in high level debug mode
- Knobs for Initial parameters
- Moments updated, initialization for 5D in twiss, moments not available
in 5D due to a bug and few others
- Removed charge setting to the my_ring layout to make ptc_twiss running,
redundant printouts removed
- Moments calculation fully imlemented; map buffering in ptc_twiss
- Few bugs corrected (f.g. map initialized to nd2 instead of npara when
initial twiss provided). Moments seem to work (to be tested yet)
- New ptc_twiss, so A_ is tracked. This makes possible tracking of moments
(to be completed).

wrap.f90
- needed changes to accept new "madx_ptc_normal_module"
- First step for node layout tracking in ptc_trackline
- Clean-up

- Added feature that allows to set values of several knobs
and only at the end recalculate values in tables.
Normally all tables are recalculated after setting a new value.
However, it slows dows parametric matching.
New command ptc_refreshtable
- Updated match with knobs
- In addmoment, t and delta swapped so the MADX input corresponds to the MADX
nomenclature.
- Moments calculation fully imlemented; map buffering in ptc_twiss
- New ptc_twiss, so A_ is tracked. This makes possible tracking of moments
(to be completed).
- Etienne's Gino stuff
- Adding gino command following madx_ptc_script_module
- Implemented:
1. ptc_setfieldcomp that set any order field strengh
to requested value. It enables matching of higher order field components.
2. Special matching mode use_ptcknob. It implements kind of macro
that emplys parametric PTC calculations to perform matching in a faster manner.
For further details see the comments at the top of matchptcknobs.c file.
3. Minor corrections and protections against segmentation violation.
- PTC knobs (pol_blocks) almost completely interfaced to MAD-X.
User sets a knob with ptc_knob command.
Twiss parameters and user specified (with ptc_select) map components are
buffered in memory after every element in form of taylor series.
They can be dumped to text file in two formats with ptc_printparametric command.
They can be also visualized and further studied with rviewer from rplot plugin.
Further, user can set numeric values of knobs with ptc_setknobvalue what
updates all numeric values of the parameters in the tables.
This way knobs can be used in matching.
- knobs implemented with PTC with pol_blocks; command to dump parametric results
to file or stdout; content of ptc_madx_tablepush.f90 moved to ptc_madx_knobs.f90,
the former one removed
- Sceleton for knobs and arbitrary element placement implemented. Lattice
visualization via ROOT macro. Printing detailed lattice geometry in PTC. Several
small bug corrections and some code cosmetics.
- PTC_Enforce6D implemented

==========

PTC proper

Files:

Sa_extend_poly.f90, Sb_sagan_pol_arbitrary.f90, Sc_euclidean.f90, Sd_frame.f90,
Se_status.f90, Sf_def_all_kinds.f90, Sh_def_kind.f90, Si_def_element.f90,
Si_def_element.f90, Sj_elements.f90, Sk_link_list.f90, Sl_family.f90,
Sm_tracking.f90, Sma_multiparticle.f90, Sn_mad_like.f90, So_fitting.f90,
Sp_keywords.f90, Spb_fake_gino_sub.f90, Sq_orbit_ptc.f90, Sqa_beam_beam_ptc.f90,
Sqb_accel_ptc.f90, Sr_spin.f90, St_pointers.f90,

Changes:

- PTC May 2007
- VORNAME assigned, the same as name of node in MADX but with capital letters
- Fixing the crash for sbend + exact + multipoles larger than 10. This set-up
requires to solve Maxwell's equation up to SECTOR_NMUL_MAX. The default is
set to 10 to avoid excessive computing time. This is now safeguarded in
madxp. To this end the parameters SECTOR_NMUL and SECTOR_NMUL_MAX are
transfered from "ptc_create_layout" to "ptc_create_universe" such that
these global parameters can be set early enough. Internally in PTC the
parameter "lda_used" is incremented where needed from 1500 to 3000 and set
back. Moreover Etienne has done the following modifications to make this
possible:
"The modification I made in the new PTC I sent you are as follows:
You first select SECTOR_NMUL and SECTOR_NMUL_MAX. For all
multipole <= SECTOR_NMUL then maxwell's is solved to order SECTOR_NMUL_MAX.
For multipole above SECTOR_NMUL , they are treated a la Sixtrack.
So for example, it you have errors to order 20, you may bother with maxwells
only to order nmul=4 and nmul_max=10 as far as Maxwell's is concerned.
Multipole higher will be sixtrack multipoles.
- New PTC 2007
- PTC with crash security!
- First BB
- PTC version including spin
- O-tone Etienne:
Therefore I included this Zip file which contains the newest PTC. There are
a few minor bugs related to patches in the present CVS version of PTC. This
could affect the CHANGEREF command of Frank. It is fixed in this new PTC. In
addition I included some routines in pointers.f90 and the script file for the
example.
- Etienne's clean-up
- Change faulty print out of Totalpath
1) Se_status: preliminary fix of uninitialized variable RADIATION_NEW
2) madx_ptc_module: Fix of "ptc_normal" by fixing the string comparison
3) madx_ptc_module & madx_ptc_twiss: write & read traditional DA map format
- PTC first changes stay October 2006 thinlens, cutting
- Introducing TRUERBEND and WEDGRBEND in PTC. To this end 2 flags have been
introduced in the NAD-X dictionary madxdict.h:
1) ptcrbend: if true it uses a PTC type RBEND
2) truerbend: if true it uses TRUERBEND; if false it uses WEDGRBEND
- Etienne O-tone:
I fixed a bug in the exit patches part of the backward survey. This bug was
noticed while doing the Daphne backward ring. Patching was done correctly
but the survey command was moving the layout. This reflects a bug in either
patching or survey: dangerous.
- The definition is:
integer, pointer:: CAVITY_TOTALPATH ! REAL PILL B0X =1 , FAKE =0 default
Accidentally it was set CAVITY_TOTALPATH=0
- Big bug in GETMAT7R and GETMAT7d DH wrong: "basically model 2 method 4 is
messed up"
- New PTC Etienne end of visit a) thin lense b) BB

Spb_fake_gino_sub.f90
- New file needed for "Gino Version"

pointers.f90
- remove bad temper comment
- Etienne's Gino stuff
- Adding gino command following madx_ptc_script_module
- logical lp -> 4 and vice versa so NAG does not cry

Spc_pointers.f90, St_pointers.f90,
- Replaced by St_pointers.f90
- New 2007 PTC: this file replaces pointers.f90

Sq_orbit_ptc.f90
- New routines needed by PTC including spin

Sqa_beam_beam_ptc.f90, Sqb_accel_ptc.f90, Sr_spin.f90
- New routines needed by PTC including spin

==========

```
FPP
---

Files:
------
a_def_all_kind.inc, a_def_element_fibre_layout.inc, a_def_frame_patch_chart.inc,
a_def_sagan.inc, a_def_worm.inc, a_scratch_size.f90, b_da_arrays_all.f90,
c_dabnew.f90, d_lielib.f90, h_definition.f90, i_tpsa.f90, j_tpsalie.f90,
k_tpsalie_analysis.f90, l_complex_taylor.f90, m_real_polymorph.f90,
n_complex_polymorph.f90, o_tree_element.f90

Changes:
--------

- Throw out unused variables
- Fix single/double precision definition that crashed NAG f95
- logical=>logical(lp) needed for NAG f95
- Bug corrected - only first 10 elements of an array was zeroed instead of whole
- Check of initial conditions provided by the user on imput.
- pointers initialized to null in universal_taylor

e_define_newda.f90, f_newda.f90, g_newLielib.f90
- Experimental NEWDA no longer for this PTC version

==========================================================================

Makefiles
---------

Files:
------
Makefile, Makefile.bat, Makefile.prof, Makefile_develop, Makefile_g95,
Makefile_gdb, Makefile_gfortran, Makefile_nag, Makeonline

Changes:
--------
- moved aperture code from madxn.c to new file aperture.c
corrections:
- more apex in halo polygon
- corrected the construction of rectellipse in the general case
- secured potentially dangerous division by zero
- Cleaned code so NAG warnings are minimized now: mainly unused variables
- PTC May 2007
- "madx_ptc_normal_module"
- PTC version including spin
- Changes needed to compile routine dlamc1 without optimization in extra file
matchlib2.F. Otherwise madx gets stuck in matching procedures. In all Linux
Makefiles matchlib2.F is compiled when using g77. For the Fortran90 compilers
lf95, g95, f95(NAG) and gfortran an optimized routine is used as provided by
Andy Vaught, the "g95" maintainer. For Windows the special compile flag:
-lfe "-D_G95" was needed (special undocumented Fujitsu compile flag of
the Lahey lf95) to compile this special Fortran90 version of dlamc1.
- Adjustments to produce 32bit executables on AMD64 (not complete yet)
- New PTC 2007: mod to Makefile due to filename change
- plugin support off
- New ptc_twiss, so A_ is tracked. This makes possible tracking of moments
(to be completed).
- New gino PTC version
- Take out "-fno-second-underscore" from the gcc flags. Add "LIBX" for FC5
as a comment.
- change in the print out of match summary when USE_MACRO
- Implemented:
1. ptc_setfieldcomp that set any order field strengh
to requested value. It enables matching of higher order field components.
2. Special matching mode use_ptcknob. It implements kind of macro
that emplys parametric PTC calculations to perform matching in a faster manner.
For further details see the comments at the top of matchptcknobs.c file.
3. Minor corrections and protections against segmentation vilation.
- linker option added to export main program symbols so the function can be
used from plugins
- plugin support switched off by default
- Optional plugin support added, that requires dynamic linking. Switched-Off
by default.
- corrected error
- knobs implemented with PTC with pol_blocks; command to dump parametric
results to file or stdout; content of ptc_madx_tablepush.f90 moved to
ptc_madx_knobs.f90, the former one removed
- Sceleton for knobs and arbitrary element placement implemented. Lattice
visualization via ROOT macro. Printing detailed lattice geometry in PTC.
Several small bug corrections and some code cosmetics.
- Missing dependencies added
- PTC with crash security!
First BB

Makefile.bat
- JMJ: Added line to compile fatchlib2.f and modified lines to link it into
madx.exe and madxp.exe.

Makefile_gfortran
- Makefile using gfortran - not working yet for PTC

Makeonline
New files for MAD-X On-Line Modeling Version

==========================================================================
```

MAD-X production version 3.03: 04.05.2006

```
1) Documentation of standard PTC modules
2) New PTC module "ptc_track_line", i.e. lines with acceleration
3) Thin lens tracking in agreement with Ripken theory and PTC
4) Non-linear matching via encapsulated ptc_normal commands
```

All changes for each file:

```
Makefile
-replacing madxdev by madxp
-put pointers to the end on request of Etienne
- Adapting Makefiles for non-linear matching and PTC upgrade
- PTC upgrade: Proper Thin Lens Lattice
================================================================================

Makefile.bat
-replacing madxdev by madxp
-put pointers to the end on request of Etienne
-Take out old Sb_1 and Sb_1 obj files
-some misplaced commands
- Adapting Makefiles for non-linear matching and PTC upgrade
- PTC upgrade: Proper Thin Lens Lattice
================================================================================

madX/Makefile.prof
-replacing madxdev by madxp
-put pointers to the end on request of Etienne
-Makefile for profiling courtesy PS
================================================================================

Makefile_develop
-replacing madxdev by madxp
-put pointers to the end on request of Etienne
- Adapting Makefiles for non-linear matching and PTC upgrade
- PTC upgrade: Proper Thin Lens Lattice
================================================================================

Makefile_g95
-replacing madxdev by madxp
-put pointers to the end on request of Etienne
- Adapting Makefiles for non-linear matching and PTC upgrade
- PTC upgrade: Proper Thin Lens Lattice
-Merged with version MAD-X 3.02.29
================================================================================

Makefile_nag
-Makefile for the NAG compiler -- for the moment not operational
================================================================================

Sa_extend_poly.f90
-PTC with dvds implemented in the travelling wave cavity. The voltage is
given by: V=V0-dvds*z
-Updated to           energybefore = nfen%energy*1000.
================================================================================

Sb_1_pol_template.f90
-No longer needed after PTC upgrade 25.04.2005
================================================================================

Sb_2_pol_template.f90
-No longer needed after PTC upgrade 25.04.2005
================================================================================

Sd_frame.f90
-Updated to           energybefore = nfen%energy*1000.
================================================================================

Se_status.f90
-Drop the printing of "NO=10" in curvbend
- Adapting Makefiles for non-linear matching and PTC upgrade
- PTC upgrade: Proper Thin Lens Lattice
================================================================================

Sf_def_all_kinds.f90
- Adapting Makefiles for non-linear matching and PTC upgrade
- PTC upgrade: Proper Thin Lens Lattice
1) Remove residual left-over definition of double precision numbers. Should
all be in a_scratch_size.f90.
2) The logical needs to be defined as "logical(lp)". Several instances found.
-updated latest head developements
================================================================================

Sg_1_fitted.f90
-No longer needed after PTC upgrade 25.04.2005
1) Remove residual left-over definition of double precision numbers. Should
all be in a_scratch_size.f90.
2) The logical needs to be defined as "logical(lp)". Several instances found.
-updated latest head developements
-Updated to           energybefore = nfen%energy*1000.
================================================================================

Sg_1_template_my_kind.f90
-No longer needed after PTC upgrade 25.04.2005
================================================================================

Sg_2_template_my_kind.f90
-No longer needed after PTC upgrade 25.04.2005
================================================================================

Sg_sagan_wiggler.f90
- Adapting Makefiles for non-linear matching and PTC upgrade
- PTC upgrade: Proper Thin Lens Lattice
1) Remove residual left-over definition of double precision numbers. Should
all be in a_scratch_size.f90.
2) The logical needs to be defined as "logical(lp)". Several instances found.
-updated latest head developements
-Updated to madX-3_02_16; bug corrected in madx_ptc_setcavs.f90
================================================================================

Sh_def_kind.f90
- Adapting Makefiles for non-linear matching and PTC upgrade
- PTC upgrade: Proper Thin Lens Lattice
-PTC with dvds implemented in the travelling wave cavity. The voltage is
given by: V=V0-dvds*z
-Updated to madX-3_02_16; bug corrected in madx_ptc_setcavs.f90
================================================================================

Si_def_element.f90
- Adapting Makefiles for non-linear matching and PTC upgrade
- PTC upgrade: Proper Thin Lens Lattice
-PTC with dvds implemented in the travelling wave cavity. The voltage is
given by: V=V0-dvds*z
-Updated to madX-3_02_16; bug corrected in madx_ptc_setcavs.f90
================================================================================

Sk_link_list.f90
- Adapting Makefiles for non-linear matching and PTC upgrade
- PTC upgrade: Proper Thin Lens Lattice
1) Remove residual left-over definition of double precision numbers. Should
all be in a_scratch_size.f90.
2) The logical needs to be defined as "logical(lp)". Several instances found.
-updated latest head developements
================================================================================

Sl_family.f90
- Adapting Makefiles for non-linear matching and PTC upgrade
- PTC upgrade: Proper Thin Lens Lattice
1) Remove residual left-over definition of double precision numbers. Should
all be in a_scratch_size.f90.
2) The logical needs to be defined as "logical(lp)". Several instances found.
-updated latest head developements
-Updated to madX-3_02_16; bug corrected in madx_ptc_setcavs.f90
================================================================================

Sm_tracking.f90
- Adapting Makefiles for non-linear matching and PTC upgrade
- PTC upgrade: Proper Thin Lens Lattice
1) Remove residual left-over definition of double precision numbers. Should
all be in a_scratch_size.f90.
2) The logical needs to be defined as "logical(lp)". Several instances found.
-updated latest head developements
```

-Updated to madX-3_02_16; bug corrected in madx_ptc_setcavs.f90
==========

Sma_multiparticle.f90
-Needed for PTC upgrade 25.04.2005
==========

Sn_mad_like.f90
- Adapting Makefiles for non-linear matching and PTC upgrade
- PTC upgrade: Proper Thin Lens Lattice
1) Remove residual left-over definition of double precision numbers. Should
all be in a_scratch_size.f90.
2) The logical needs to be defined as "logical(lp)". Several instances found.
-PTC with dvds implemented in the travelling wave cavity. The voltage is
given by: V=V0-dvds*z
-updated latest head developements
-Updated to madX-3_02_16; bug corrected in madx_ptc_setcavs.f90
==========

So_fitting.f90
-Fixing initialization problems
- Adapting Makefiles for non-linear matching and PTC upgrade
- PTC upgrade: Proper Thin Lens Lattice
1) Remove residual left-over definition of double precision numbers. Should
all be in a_scratch_size.f90.
2) The logical needs to be defined as "logical(lp)". Several instances found.
-Wrong definition of JMIN and EPSNOW!!!
-PTC with dvds implemented in the travelling wave cavity. The voltage is
given by: V=V0-dvds*z
-updated latest head developements
-Updated to madX-3_02_16; bug corrected in madx_ptc_setcavs.f90
==========

Sp_keywords.f90
- Adapting Makefiles for non-linear matching and PTC upgrade
- PTC upgrade: Proper Thin Lens Lattice
==========

a_def_all_kind.inc
- Adapting Makefiles for non-linear matching and PTC upgrade
- PTC upgrade: Proper Thin Lens Lattice
-PTC with dvds implemented in the travelling wave cavity. The voltage is
given by: V=V0-dvds*z
-Updated to madX-3_02_16; bug corrected in madx_ptc_setcavs.f90
==========

a_def_element_fibre_layout.inc
- Adapting Makefiles for non-linear matching and PTC upgrade
- PTC upgrade: Proper Thin Lens Lattice
-PTC with dvds implemented in the travelling wave cavity. The voltage is
given by: V=V0-dvds*z
-Updated to madX-3_02_16; bug corrected in madx_ptc_setcavs.f90
==========

a_def_user1.inc
-No longer needed after PTC upgrade 25.04.2005
==========

a_def_user2.inc
-No longer needed after PTC upgrade 25.04.2005
==========

a_scratch_size.f90
-logical must be without (lp) when going into INQUIRE function
- Adapting Makefiles for non-linear matching and PTC upgrade
- PTC upgrade: Proper Thin Lens Lattice
-All constants collected here. Carefully checked and reordered.
-PTC with dvds implemented in the travelling wave cavity. The voltage is
given by: V=V0-dvds*z
-updated latest head developements
-Updated to madX-3_02_16; bug corrected in madx_ptc_setcavs.f90
==========

b_da_arrays_all.f90
1) Remove residual left-over definition of double precision numbers. Should
all be in a_scratch_size.f90.
2) The logical needs to be defined as "logical(lp)". Several instances found.
-updated latest head developements
-Updated to madX-3_02_16; bug corrected in madx_ptc_setcavs.f90
==========

d_lielib.f90
-Updated to madX-3_02_16; bug corrected in madx_ptc_setcavs.f90
==========

f_newda.f90
1) Remove residual left-over definition of double precision numbers. Should
all be in a_scratch_size.f90.
2) The logical needs to be defined as "logical(lp)". Several instances found.
-updated latest head developements
==========

h_definition.f90
-Updated to madX-3_02_16; bug corrected in madx_ptc_setcavs.f90
==========

j_tpsalie.f90
-Fixing initialization problems
==========

k_tpsalie_analysis.f90
-PTC with dvds implemented in the travelling wave cavity. The voltage is
given by: V=V0-dvds*z
-Updated to madX-3_02_16; bug corrected in madx_ptc_setcavs.f90
==========

madx_ptc_intstate.f90
-bug correction: always using eternal states instead of current ones.
-Clean-up
-New matching with macros that enables fitting of non-linear parameters with PTC
1) Remove residual left-over definition of double precision numbers. Should
all be in a_scratch_size.f90.
2) The logical needs to be defined as "logical(lp)". Several instances found.
-updated latest head developements
==========

madx_ptc_module.f90
-The restrictions "IF(1.ne.0)" are removed. =>
-bug correction: always using eternal states instead of current ones.
-Drop useless "make_states" call
-in my_state: if requested dimensionality 6 and there are cavities, enforce dalta and only_4d to false.
-Fixing division by 1=zero in the multipole block
-if icase=6 then only_4d=false
-Reduntant debug printouts present only in debug mode
-Fine tuning debug print-out
-Fixing "eigen" print-out, the screw-up was due to 5D versus 6D
-Clean-up
-New matching with macros that enables fitting of non-linear parameters with PTC
-output of eigen have proper row/column swap from (pt,t) to (-t,pt)
-Fix the i2 variable bug of eigen in equaltwiss found Piotr - thanks
-bugs in subr. SUMM_MULTIPOLES..:
INTENT(INOUT) for key & normal_0123 initialized.
-eigenvector calculation in ptc_twiss
-6D "eign" in ptc_normal
1) Adding eigenvectors to ptc_normal
2) Suppress debug printing to unit 18/19
-remove bug (division by zero) due to dipole errors
-Finishing multipoles in thick elements
-Multipoles and Errors of any order are added to thick elements (for MADX-PTC only)
1) State "time" is default and can be set in create_layout.
2) C routines that write or read from TFS tables only operate with double
precision numbers. This will ensure a proper operation when PTC is
calculating in four-fold precision.
-Merged with version MAD-X 3.02.29
-Updated to head, bug corrected in equaltwiss
-compilation problem corrected
-updated latest head developements
-Reduntant debug printout removed
-Updated to madX-3_02_16; bug corrected in madx_ptc_setcavs.f90
==========

madx_ptc_script.f90
-put pointers to the end on request of Etienne
- Adapting Makefiles for non-linear matching and PTC upgrade
- PTC upgrade: Proper Thin Lens Lattice
==========

madx_ptc_setcavs.f90
-Reduntant debug printouts present only in debug mode
-Clean-up
-New matching with macros that enables fitting of non-linear parameters with PTC
1) Remove residual left-over definition of double precision numbers. Should
all be in a_scratch_size.f90.
2) The logical needs to be defined as "logical(lp)". Several instances found.
-updated latest head developements
-Updated to madX-3_02_16; bug corrected in madx_ptc_setcavs.f90
==========

madx_ptc_tablepush.f90
-Clean-up
-New matching with macros that enables fitting of non-linear parameters with PTC
1) Remove residual left-over definition of double precision numbers. Should
all be in a_scratch_size.f90.
2) The logical needs to be defined as "logical(lp)". Several instances found.
-Updated to head, bug corrected in equaltwiss
updated latest head developements
==========

madx_ptc_track_run.f90
-Fixing the closed orbit at the observation points for ELEMENT_BY_ELEMENT
-calculation of CO is removed when ELEMENT_BY_ELEMENT is forced to ON
at Closed_ORBIT=OFF.
-text of print-out is corrected
1) Fixing "element_by_element=false"
2) Add CT variable print-out in 5D
-remove bug (division by zero) due to dipole errors
-The sign for the second coord. system {-pathelength,deltap} is corrected
-Finishing multipoles in thick elements
-use "real(kind(1d0) :: dble_num_C" variable as a double precision buffer
number for an input parameter at all CALLs of the C-routine "double_to_table"
-clean-up estetics
1) State "time" is default and can be set in create_layout.
2) deltap is transfered to pt when "time" is on.
-Merged with version MAD-X 3.02.29
-Merged with version MAD-X 3.02.29
-Updated to head, bug corrected in equaltwiss
-updated latest head developements
-Updated to madX-3_02_16; bug corrected in madx_ptc_setcavs.f90
==========

madx_ptc_trackcavs.f90
-Clean-up
-New matching with macros that enables fitting of non-linear parameters with PTC
1) Remove residual left-over definition of double precision numbers. Should
all be in a_scratch_size.f90.
2) The logical needs to be defined as "logical(lp)". Several instances found.
-fill tables with single precision values
-updated latest head developements
-Updated to madX-3_02_16; bug corrected in madx_ptc_setcavs.f90
==========

madxd.h
-New matching with macros that enables fitting of non-linear parameters with PTC
-version 3.02.25 => PTC examples (_track, _normal, _twiss) are checked on
abploc with "strict" version of madxdev
-The eigenvalue keyword is consistent set to the 2 characters "eign"
-eigenvector calculation in ptc_twiss
1) Adding "eign" to ptc_normal table "normal_results"
2) Grow table "normal_results" if needed
-Merged with version MAD-X 3.02.29
-Updated to head, bug corrected in equaltwiss
-Updated to madX-3_02_16; bug corrected in madx_ptc_setcavs.f90
-weight paramter in constraint command to be used with use macro
==========

madxdict.h
-option no_fatal_stop to not let a fatal error kill madx
-Clean-up
-New matching with macros that enables fitting of non-linear parameters with PTC
-Cleaned version of thintrack. No delta_p dependence internally any more.
Only radiation part of code still contains delta.
Full 6D equations (Ripken) used.
Some further improvements.
Closed orbit still computed by twiss.
1) Adding "eign" to ptc_normal table "normal_results"
2) Grow table "normal_results" if needed
-Multipoles and Errors of any order are added to thick elements (for MADX-PTC only)
1) Remove residual left-over definition of double precision numbers. Should
all be in a_scratch_size.f90.
2) The logical needs to be defined as "logical(lp)". Several instances found.
-Merged with version MAD-X 3.02.29
-updated latest head developements
-weight paramter in constraint command to be used with use macro
==========

madxl.h
-The eigenvalue keyword is consistent set to the 2 characters "eign"
-eigenvector calculation in ptc_twiss
1) Adding "eign" to ptc_normal table "normal_results"
2) Grow table "normal_results" if needed
-updated latest head developements
==========

madxn.c
-Reverting to old warning routine
-Clean-up
-New matching with macros that enables fitting of non-linear parameters with PTC
-The eigenvalue keyword is consistent set to the 2 characters "eign"
-eigenvector calculation in ptc_twiss
-6D "eign" in ptc_normal
1) Adding "eign" to ptc_normal table "normal_results"
2) Grow table "normal_results" if needed
-Merged with version MAD-X 3.02.29
-updated latest head developements
-weight paramter in constraint command to be used with use macro
==========

madxp.c
-option no_fatal_stop to not let a fatal error kill madx
-Reverting to old warning routine
-fmt can not be register variable
-Clean-up
-New matching with macros that enables fitting of non-linear parameters with PTC
-Merged with version MAD-X 3.02.29
-weight paramter in constraint command to be used with use macro
==========

madxu.c
-Clean-up
-New matching with macros that enables fitting of non-linear parameters with PTC
-weight paramter in constraint command to be used with use macro
==========

matchc.c
-Clean-up
-New matching with macros that enables fitting of non-linear parameters with PTC
-output fix
-weight paramter in constraint command to be used with use macro
==========

matchc2.c
-New match library needed for non-linear matching -- readded due to dead
revision
-clean-up
-fix compiler complain
-output fix
-weight paramter in constraint command to be used with use macro
==========

matchjc.F
-Clean-up

-Jacobian fix. Avoid twiss or macro to be called before a check on the variables limits
-New matching with macros that enables fitting of non-linear parameters with PTC
-weight paramter in constraint command to be used with use macro
==

matchlib.F
-New match library needed for non-linear matching -- readded due to dead
revision
==

o_tree_element.f90
-Updated to madX-3_02_16; bug corrected in madx_ptc_setcavs.f90
==

pointers.f90
put pointers to the end on request of Etienne
-Fix write statement that was changed by automatic clean-up, IE replacing
"pause" statements.
-Needed for PTC upgrade 25.04.2005
==

track.fi
-Cleaned version of thintrack. No delta_p dependence internally any more.
Only radiation part of code still contains delta.
Full 6D equations (Ripken) used.
Some further improvements.
Closed orbit still computed by twiss.
-Merged with version MAD-X 3.02.29
==

trrun.F
-Change the definition of the kicker. The acting on px/py now instead of
x'/y' AK/FS
-Cleaned version of thintrack. No delta_p dependence internally any more.
Only radiation part of code still contains delta.
Full 6D equations (Ripken) used.
Some further improvements.
Closed orbit still computed by twiss.
-Merged with version MAD-X 3.02.29
==

twiss.F
-Take out debug printing of eigen
-remove bug (division by zero) due to dipole errors
==

user2_photon.f90
1) Remove residual left-over definition of double precision numbers. Should
all be in a_scratch_size.f90.
2) The logical needs to be defined as "logical(lp)". Several instances found.
-updated latest head developements
==

wrap.f90
-clean-up
1) Remove residual left-over definition of double precision numbers. Should
all be in a_scratch_size.f90.
2) The logical needs to be defined as "logical(lp)". Several instances found.
-updated latest head developements
==

MAD-X version 3.02.14: 12.04.2006

1) PTC modules have been cleaned up and are all documented by now
2) New "PTC_TRACK_LINE" for tracking lines including acceleration
written and maintained by Piotr Skowronski
3) Numerous bug fixes and clean-ups

All changes for each file:

Makefile

-Reverting to the previous version
-Temporal work around implemented: there is some problem with internal state
settings; \ in setcaenforcing preparing to merge with the recent HEAD
developements hoping that the problem was alread solved over there.
-new match mode
==

Makefile_develop

-new match mode
==

Makefile_gdb

-file Makefile_gdb was initially added on branch newmatch.
-Temporal work around implemented: there is some problem with internal state
settings; \ in setcaenforcing preparing to merge with the recent HEAD
developements hoping that the problem was alread solved over there.
-Dependances corrected so it can be made with -j N option
-new match mode
==

Sh_def_kind.f90,So_fitting.f90,a_scratch_size.f90,i_tpsa.f90,j_tpsalie.f90,

-Replace "double precision" by "real(dp)
-Merged newmatch-060411 with recent HEAD developement
==

madx_ptc_intstate.f90

-global debuglevel integer added (0 completely silent, 1 normal printout,
2 most important debug information, 3 everything
==

madx_ptc_module.f90

-
1) Fix priority order between ICASE and DELTAP and cavities
2) Convert DELTAP to PT
3) Proper conversion between variables for 5D
4) Clean-Up
5) Replace "double precision" by generic "real(dp)

MAD-X version 3.02.05: 22.03.2006

0) Stable Production Version
1) Tracking Linacs including acceleration using PTC
2) General PTC upgrade
3) Strict Compile Flags revealed a couple of subtle Fortran bugs like
the use of initialized variables and out-of bound usage of arrays
4) Examples including Documentation brought up to date

All changes for each file:

Makefile

-New functionality for PTC: track linac, ptc_twiss with acceleration,
ptc_select, ptc_script, ptc_dumpmaps
-Take out Sg_0_fitted.f90 since no longer needed for PTC upgrade.
==========

Makefile.bat

-Fixes needed to get it to run on Windows
-Missing file added
-New functionality for PTC: track linac, ptc_twiss with acceleration,
ptc_select, ptc_script, ptc_dumpmaps
-Take out Sg_0_fitted.f90 since no longer needed for PTC upgrade.
-Updated to include matchjx.
-Default location for checked-out files changed, comment added.
==========

Makefile_develop

-More interesting flags: --chk a,e,s,u,x --chkglobal --info
-Adapting to Piotr's new PTC additions
-Switching to lf95 with very tough compile flags
-New functionality for PTC: track linac, ptc_twiss with acceleration,
ptc_select, ptc_script, ptc_dumpmaps
-Take out Sg_0_fitted.f90 since no longer needed for PTC upgrade.
==========

Makefile_g95

-New functionality for PTC: track linac, ptc_twiss with acceleration,
ptc_select, ptc_script, ptc_dumpmaps
-Take out Sg_0_fitted.f90 since no longer needed for PTC upgrade.
==========

madxp.c

-Indenting
-Fixing of: madxp.c:396: warning: ISO C90 forbids mixed declarations and code
-Redundant debug printout removed
==========

madxn.c

-Indenting
-Compiler warnings removed
-Fix a memory leak in "pro_ptc_twiss"
==========

madxreq.c, madxu.c

-Indenting
==========

makethin.c

-Warning on inconsistent child/parent slicing removed as this has become an
allowed feature + clean up of old, now rather obsolete comments.
==========

madxd.h

-Compiler warnings removed
==========

madx_ptc_setcavs.f90

-the variable "givendene" was uninitialized
==========

madxdict.h

-In ptc_create_layout by default it is closed now
-fix defaults for matching with chrom
==========

c6t.c

1) In case drifts are combined at the end of the machine, the "end_marker"
in fc.34 (input file for sodd) came with erroneous values for position,
beta-functions and phase advances.

2) For various reasons the number of elements with field errors and/or
alignment errors may vary between SixTrack and MAD-X. Obviously, the
physics is identical!
==========

dynap.F

Various bug fixes, write out of distance in phase space into file
lyapunov.data   F2&F5
==========

gxx11c.c

Defining templates mycalloc and myfree
==========

ibadb.F

In Routine twsint variable "alam" was used before being initialized, again
no effect on the results.
==========

trrun.F

-Various bug fixes, write out of distance in phase space into file
lyapunov.data   F2&F5
-Fix bug in aperture check of "rectellipse":
The parameters of rectangle/ellipse were swapped, thus wrong.
==========

twiss.F

-fixing the restsum problem of overwriten arrays and lack of initialization
-logical cpiky & dorad were uninitialize
==========

poisson.F

-Fortran clean-up: implicit none etc.
==========

madx_main.f90

Clean-up: missing public & implicit none statements
==========

Sg_0_fitted.f90

-No longer needed after PTC upgrade
==========

Sh_def_kind.f90

changed some stuff in kind7 and kind6(sixtrack)
==========

So_fitting.f90

-unintialized. Line 979 added:
   mx=zero
==========

Sa_extend_poly.f90, Sh_1_pol_template.f90, Sh_2_pol_template.f90,
Sh_sagan_pol_arbitrary.f90, Sh_sagan_pol_arbitrary.f90, Sc_euclidean.f90,
Sd_frame.f90, Se_status.f90, Sf_def_all_kinds.f90, Sg_1_fitted.f90,
Sg_1_template_my_kind.f90, Sg_2_template_my_kind.f90, Sg_sagan_wiggler.f90,
Sh_def_kind.f90, Si_def_element.f90, Si_elements.f90, Sk_link_list.f90,
Sl_family.f90, Sm_tracking.f90,Sn_mad_like.f90, So_fitting.f90,
Sp_keywords.f90, s_def_all_kind.inc, s_def_element_fibre_layout.inc,
a_scratch_size.f90, b_da_arrays_all.f90, c_dabnew.f90,e_define_newda.f90,
h_definition.f90, l_tpsa.f90, l_tpsalie.f90, k_tpsalie_analysis.f90,
l_complex_taylor.f90, m_real_polymorph.f90, madx_ptc_module.f90,
n_complex_polymorph.f90, o_tree_element.f90, run_madx.f90

0) Features kindfitted, ZGOUBI are taken out.
1) Plug in "public" in each module
2) Partial unitialized array in "FIND_ORBIT_LAYOUT_node" in "So_fitting.f90".
3) A couple of missing "implicit none" statements as described before.
4) check_iteration, check_interpolate_x, check_interpolate_y where
un-associate pointers after PTC upgrade.
==========

Sg_1_fitted.f90, Sg_1_template_my_kind.f90, Sg_2_template_my_kind.f90,
Sg_sagan_wiggler.f90, Sh_def_kind.f90, Sp_keywords.f90, b_da_arrays_all.f90,

- Missing "implicit none" and "public" statement
- File "b_da_arrays_all.f90": ndamaxi initialized to zero
- File "Sh_def_kind.f90": unused variable "copyMULTIP" & "find_fn"
  removed
==========

a_def_element_fibre_layout.inc

-Passive field for kind21 (twcavity) added containing a lag
==========

madx_ptc_intstate.f90, madx_ptc_script.f90, madx_ptc_setcavs.f90,
madx_ptc_tablepush.f90, madx_ptc_track_run.f90, madx_ptc_trackcavs.f90,
madx_ptc_module.f90, wrap.f90, ptc_dummy.F, Sn_mad_like.f90, madxp.c, madxn.c,
madxn.c, rplot.c, madxd.h, madxdict.h, madxl.h, rplot.h

-Fortran code processed by frs's indentation script
-Clean-up: missing public & implicit none statements
-New functionality for PTC: track linac, ptc_twiss with acceleration,
ptc_select, ptc_script, ptc_dumpmaps
==========

madx_ptc_module.f90

-Fortran code processed by frs's indentation script
-Normalition error found in the calculation of higher order chromaticity (RT)
==========

madx_ptc_setcavs.f90
locate_all_cav substituted with new locate_all_twcav that counts only
twcavities
==========

MAD-X version 3.02.01: 08.02.2006

1) Many bug fixes
2) "Jacobian" matching
4) PTC upgrade
3) Worldwide CVS read access from:
MAD-X Source

All changes for each file:

Makefile

-Modified to allow more flexible PTC modeling using mads_main.f90 and
run_mads.f90
-Adding new PTC file bg_l_fitted.f90
-Missing dependence on madxreg.c

Makefile.bat

-Updated to include matchjc.
Default location for checked-out files changed, comment added.

-Modified to allow more flexible PTC modeling using mads_main.f90 and
run_mads.f90

-Adding new PTC file bg_l_fitted.f90

Makefile_develop

-New match routine "jacobian".
In the matching routine there is the new command "jacobian" similar to lmdif.
It allows to use more variable than constraints, finding the least square
solution for the parameters to be varied.

-Modified to allow more flexible PTC modeling using mads_main.f90 and
run_mads.f90

-Bug fix of dependence of bg_l_fitted.f90

-Adding new PTC file bg_l_fitted.f90

1) Add missing dependence on madxreg.c
2) Replace obsolete g95 compiler flag "-g90" by "-g"

---

makefile_g95

-The link option 'static' does not yet work for 'madadev' using g95.

-Modified to allow more flexible PTC modeling using mads_main.f90 and
run_mads.f90

-Makefile using the g95 compiler

---

bc_euclidean.f90, bq_frame.f90, bs_status.f90, bf_def_all_kinds.f90,
bq_l_fitted.f90, bq_l_fitted.f90, bq_l_template_sq_kind.f90,
bq_l_template_sq_kind.f90, bq_sagan_wiggler.f90, bh_def_kind.f90,
bl_def_element.f90, bl_elements.f90, bk_link_list.f90, bl_family.f90
bm_tracking.f90, bc_fitting.f90, bp_keywords.f90, a_def_all_kind.inc,
a_def_element_fiber_layout.inc, a_scratch_size.f90, b_definition.f90,
l_tpsa.f90, k_tpsalie_analysis.f90, l_complex_taylor.f90,
m_real_polynorph.f90, n_complex_polynorph.f90, o_tree_element.f90

-PTC Upgrade:
1) Improved Knobs
2) Acceleration
3) Add rectellipse aperture
4) Kill memory leaks

-More on PTC Upgrade:

In Etienne's words:

The type pancake will be made of equidistant slices in cylindrical
coordinates amenable to rk2 or rk4 integration (nonsymplectic).
I will tentatively represent Bx,By,Bz by a polynomial in s=y. This
polynomial will be extracted using Ben's wavelets.... This is my plan.

I will most likely destroy fitted magnets and this experimental multip.
Math will be taken over by pancake....

-Fixing ngaus

-adding KINDMU

-Mytrue & Myfalse in call to ENGE_COS

-ENGE_COS has to have LOGICAL(LP) as the third variable

-add Runge Kutta, Zgoubi stuff

-Lethal bug in patch of tracking

-'_global, OTHER PROGRAM, TRACK_K

-ZGOUBI_MULTIP, patch, CHANGEREF, other_program

2) Add rectellipse aperture
4) Kill memory leaks

-Fix normalisation of arc length in PTC. MAD-X always gives arc length
internally:
1) rbarc=true   input straight length ==> internally arc length
2) rbarc=false arc length always

-Destructors and Constructors for my_ID_taylor

---

c_dabnew.f90

-Serious DA bug: do ib1 =2,isvo-1 in routine atree

PTC Upgrade:
1) Improved Knobs
2) Acceleration
3) Add rectellipse aperture
4) Kill memory leaks

---

a_def_frame_patch_chart.inc, a_def_worm.ic,

New PTC include file

---

c6t.c

-For SixTrack runs the length of the machine has to be known to lmu.
This value is transfered to SixTrack via fo.3.mad.

-Sequence length is written out to lmu meter

-More work on the f34 fake crash

-Under certain conditions writing to unit 34 (twiss parameters for SIXTR)
failed leading to a fake MAD-X crash when running "sixtrack" or "madx".

-Remove tabs introduce by earlier indenting. Indenting by 2 characters &
brackets below operator for better readability. F5 has recipe to do it
semi-automatic.

---

c6t.h, gxalloc.c, gxalfpsc.c, madx.h, madx1.h, madxreg.c, madxreg.h, madxu.c,
maketbln.c, matchc.c, mfc.c, timel.c

-Remove tabs introduce by earlier indenting. Indenting by 2 characters &
brackets below operator for better readability. F5 has recipe to do it
semi-automatic.

---

gxalloc.c

-Replacing calloc & free by the wrappers mycalloc & myfree

---

madx.h, madxd.h, maddclct.h, madxn.c, madxp.c, madxu.c, maketbln.c

1) Warning if sequedit of lines and no action
2) New Attribute "save" on "save" command
Example:

t2:line=(bl6,qtl301,mqn,qtl301,bl7,qtl302,mqn,qtl302,bl8,ison);
SLTL3 : LINE=(delay, t2);
use, period=slt3;

save,sequence=slt13,file=t1,havej // new parameter "save": only sequ. saved
call,file=t1; // sequence is read in and is now a "real" sequence
// if the two preceding lines are suppressed, sequdit will print a warning
// and else do nothing
use, period=slt13;
twiss, save, beta=bxa, alfa=alfxa, bet=dya, alf=alfya;
plot, vaxis=beta, bet, haxis=s, colour:=30;
SEQEDIT, SEQUENCE=slt13;
  remove,element=s.bhe0330;
  remove,element=s.bhe0330;
ENDEDIT;

use, period=slt13;
twiss, save, beta=bxa, alfa=alfxa, bet=dya, alf=alfya;

---

madx_main.f90

-2 of 3 routines to replace madxn.f: routines added in run_mads can be called
here

---

madx_ptc_module.f90

-Normalisition error found in the calculation of higher order chromaticity (NT)

-Proper print-out of closed orbit in PTC

-Skew quadrupole, sextupole and octupole components ks1, ks2, ks3 have
been added respectively. Field error components are added as well.
Results have been confirmed with using k1, k2, k3 and the the proper tilt

-Finish patch; Survey output

1) Mass Grote non-dynamic buffer fix
2) Adding Generating Function to ptc_normal (more work needed!)

-Initialization Bug courtesy FS

-trivial formating like indenting trailing blanks etc

---

madx_ptc_track_run.f90

-Fix t pt which have reversed order in PTC

-trivial formating like indenting trailing blanks etc

---

madxc.c

-Hans Grote non-dynamic buffer fix

-Name added for nameslst

---

madxd.h

-In the matching routine there is the new command "jacobian" similar to lmdif.
It allows to use more variable than constraints, finding the least square
solution for the parameters to be varied.

-Issuing the number of warnings during a MAD-X run

-Hans Grote non-dynamic buffer fix

-fixing the crash after "save" command courtesy MS

-name in nameslst MS

1) Add fill_twiss_header_ptc routine and template to produce short but
proper ptc_twiss table headers.
2) Remove tabs introduce by earlier indenting. Indenting by 2 characters &
brackets below operator for better readability. FS has recipe to do it
semi-automatic.
3) madam.c: remove unused variables
4) madam.h: new version number

-Missing "for" loop over all sequences in memory in add_to_el_list.c
Adverse effect when trying to read in 2 previously "save"d sequences
(courtesy MS)

---

madalloct.h

-New match routine "jacobian".
In the matching routine there is the new command "jacobian" similar to lmdif.
It allows to use more variable than constraints, finding the least square
solution for the parameters to be varied.

-Hans Grote non-dynamic buffer fix

-solenoid and makethin selection consistency

-The deltap variable was wrongly defined in the match section, this lead to
a crash when matching was attempted with deltap. Crash fixed but
functionality still has to be tested (kindly pointed at WW)

---

madal.h

1) Hans Grote non-dynamic buffer fix
2) Adapt for adding Generating function to madx_ptc_module.f90

-EXP upgrade in particular adding the "angle" attribute HM

-Memory Leak Detection courtesy KM

-Wrong Touschek table definition in madal.h

---

madam.c

1) New match routine "jacobian". HGM
In the matching routine there is the new command "jacobian" similar to lmdif.
It allows to use more variable than constraints, finding the least square
solution for the parameters to be varied.
2) Fixing the Windows end-of-line problems of TFS tables. (courtesy MS)

-The normalization of mu to 2*pi has been done in "twiss" and "embedded twiss"
in the C part, as a result the value of a BETAG block had an inconsistent
normalization (found my Frank Tecker).

-Write out sectormap file with variable format.

1) Hans Grote non-dynamic buffer fix
2) Adding Generating Function to ptc_normal (more work needed!)

-Names added for name_list MS
-allow "/" as the first letter of a file DH
-reset curr_cmd_pointer=1

-drop unused variable

-Replacing malloc by the wrapper mymalloc

1) Add fill_twiss_header_ptc routine and template to produce short but
proper ptc_twiss table headers.
2) Remove tabs introduce by earlier indenting. Indenting by 2 characters &
brackets below operator for better readability. FS has recipe to do it
semi-automatic.
3) madam.c: remove unused variables
4) madam.h: new version number

---

madmap.c

-Issuing the number of warnings during a MAD-X run

-Fatal_error if row name is not found in TFS table.

1) Modifications to allow 'S' format in routine get_val_num (courtesy MS)
2) The table_access with 2 variables table(s,y,z) has usually implied:
"accesses value named "z" for element "y" of table "s".
To specify a certain row number 'n' in table access one can use: table(s,z,n).

-Hans Grote non-dynamic buffer fix

-fixing the crash after "save" command courtesy MS

- Names for namelist MS
- Fix input error la "a-2d" MS

1) Fix save bug that produced wrong (straight) length of rbend in case
shared or un-flattened sequences are used. Bug fix in make_element
(courtesy MS)
2) Remove tabs introduce by earlier indenting. Indenting by 2 characters &
brackets below operator for better readability. FS has recipe to do it
semi-automatic.
3) Remove unused variables.

-Missing "for" loop over all sequences in memory in add_to_el_list.c
Adverse effect when trying to read in 2 previously "save"d sequences
(courtesy MS)

---

madmeq.c

-Replacing calloc/malloc/free by the wrappers mycalloc/mymalloc/myfree

---

madmu.c

-Modifications to allow 'S' format (courtesy MS)

-Avoid potentially uninitialised variable

-Hans Grote non-dynamic buffer fix

-fixing the crash after "save" command courtesy MS

-Memory Leak Detection courtesy KM

-fixing a=2d input error MS
-names in namelist MS

-Missing "for" loop over all sequences in memory in add_to_el_list.c
Adverse effect when trying to read in 2 previously "save"d sequences
(courtesy MS)

---

makethin.c

-indentation

1) Hans Grote non-dynamic buffer fix
2) Issue message if thick bend has fringe fields

-improved treatement of selection conflicts

-selection consistency and solenoid

---

match.f, matchc.c, matchjo.f, matchas.f

-New match routine "jacobian".
In the matching routine there is the new command "jacobian" similar to lmdif.
It allows to use more variable than constraints, finding the least square
solution for the parameters to be varied.

---

matcho.c

-name in namelist MS

-Replacing malloc by the wrapper mymalloc

---

plot.fi

-Increased buffer size for long axis annotation

---

run_mada.f90

-1 of 2 codes to replace madam.f90: one can add more routines at the end

---

saf.c

-Remove unused variables

-Hans Grote non-dynamic buffer fix

-EXP upgrade in particular adding the "angle" attribute HM

-name in namelist

---

trrun.F

-solenoid, Koschick + hbu

---

twiss.F

The normalization of mu to 2*pi has been done in "twiss" and "embedded twiss"
in the C part, as a result the value of a BETAG block had an inconsistent
normalization (found my Frank Tecker).

-Add thin solenoid

-Fix rmatrix calculation - inverse similarity trasformation

MAD-X version 3.00.01: 07.09.2005

Overview:

1) New major release featuring PTC as an integral part of MAD-X
PTC modules are:
a) ptc_twiss: Higher style Twiss parameters now with initial optics conditions
b) ptc_track: the thick lens tracking module using PTC with most
features of the thin lens tracking
c) ptc_normal: Nonlinear parameters, Hamiltonian terms etc.

2) Essential upgrade of makethin

3) CAD upgrade

4) Adding PS file in the plotting

5) Removing limit on length of macros

6) Clean-up

All changes for each file:

```
==============================================================================
Makefile

-drop not needed epause.o & usleep.o

-Add missing dependence of madxp.c on madxreg.h

1) Mods needed due to added files in V3.00
2) Missing dependencies for touschek
3) maxcontine=100

a) Raising optimization to O4 in:  Makefile
b) Full checking in:               Makefile_develop
==============================================================================
Makefile.bat

1) Mods needed due to added files in V3.00
2) Missing dependencies for touschek
3) maxcontine=100
==============================================================================
Makefile_develop

-drop not needed epause.o & usleep.o

1) Mods needed due to added files in V3.00
2) Missing dependencies for touschek
3) maxcontine=100

a) Raising optimization to O4 in:  Makefile
b) Full checking in:               Makefile_develop

-FOPT=-Bstatic does not work on the MAC and has been taken out.

-The "-c" option was missing for MAC compilation.

-Fix bad link static flag "-static" by the proper "-Bstatic" one
==============================================================================
a_scratch_size.f90 - Sp_keywords.f90

PTC Version -February 2005 (courtesy Etienne Forest)
==============================================================================
Sb_sagan_pol_arbitrary.f90, Sg_sagan_wiggler.f90, a_def_sagan.inc

-new code entity for PTC Version -February 2005 (courtesy Etienne Forest)
==============================================================================
a_def_arbitrary.inc

-no longer needed in PTC Version -February 2005 (courtesy Etienne Forest)
==============================================================================
c6t.c,v

-simple clean-up

-frs + hbu : General cleaning. Tabs to blanks. Remove blanks at eol.
Convert any C++ comments to ansi c comments.
Fix possible uninitialized variables in madxn.c
==============================================================================
dynap.F

-Modified dynap to track several pairs of particles;
write tunes, smear and lyapunov exponent of each
pair to tables dynap and dynaptune;
improve estimate of Lyapunov exponent by considering
3 angles in normalized phase space instead of 6-D distance
in original phase space.
==============================================================================
epause.c, usleep.c

-No longer needed
==============================================================================
gxx11.F, gxx11ps.F

-new track plot routine, plots are appended to existing ps file

-Flush in a portable way for ps plot

-During interactive use the ps plot file is consistent and can be viewed
after each plot command. (Not for tracking plot which uses a different system)
==============================================================================
gxx11psc.c

-frs + hbu : General cleaning. Tabs to blanks. Remove blanks at eol.
Convert any C++ comments to ansi c comments.
Fix possible uninitialized variables in madxn.c
==============================================================================
ibsdb.F

-Code logic unchanged but: Fortran90 compatibility changes

-Remove one obsolete write command.

-IBS horizontal growth rate corrected; effect of vertical dispersion added to al three IBS growth rates.
==============================================================================
madx.h

-simple clean-up

-frs + hbu : General cleaning. Tabs to blanks. Remove blanks at eol.
Convert any C++ comments to ansi c comments.
Fix possible uninitialized variables in madxn.c
==============================================================================
madx_ptc_module.f90

-Mods needed for the ptc_twiss upgrade and the new ptc_track module
==============================================================================
madx_ptc_track_run.f90

-New ptc_track thick lens tracking module
==============================================================================
madxc.c

-Revert "clean-up"

-frs + hbu : General cleaning. Tabs to blanks. Remove blanks at eol.
Convert any C++ comments to ansi c comments.
Fix possible uninitialized variables in madxn.c

-Fixing the limit of the length of 10000  characters for macros (courtesy HG)

-Clean up to avoid compiler warnings
==============================================================================
madxd.h

-version number 3.00.01

-Changes to the Core Code needed for PTC upgrade V3.00

-frs + hbu : General cleaning. Tabs to blanks. Remove blanks at eol.
Convert any C++ comments to ansi c comments.
Fix possible uninitialized variables in madxn.c

-new track plot routine, plots are appended to existing ps file

-Version number V2.13.12

-Fixing the limit of the length of 10000  characters for macros (courtesy HG)

-new version number

-Fix of install bug (courtesy HG)

-Clean up to avoid compiler warnings
==============================================================================
madxdict.h

-Fix of the "start" command bug in ptc_track. For the time being
"ptc_start" has to be used. To be followed up...

-Changes to the Core Code needed for PTC upgrade V3.00

-frs + hbu : General cleaning. Tabs to blanks. Remove blanks at eol.
Convert any C++ comments to ansi c comments.
Fix possible uninitialized variables in madxn.c

-introduce Werner's improved readtable
==============================================================================
madxe.c

-Revert "clean-up"

-frs + hbu : General cleaning. Tabs to blanks. Remove blanks at eol.
Convert any C++ comments to ansi c comments.
Fix possible uninitialized variables in madxn.c

-Clean up to avoid compiler warnings
==============================================================================
madxl.h

-Changes to the Core Code needed for PTC upgrade V3.00

-frs + hbu : General cleaning. Tabs to blanks. Remove blanks at eol.
Convert any C++ comments to ansi c comments.
Fix possible uninitialized variables in madxn.c
==============================================================================
madxn.c

-Changes to the Core Code needed for PTC upgrade V3.00

-frs + hbu : General cleaning. Tabs to blanks. Remove blanks at eol.
Convert any C++ comments to ansi c comments.
Fix possible uninitialized variables in madxn.c

-remove set_selected_elements()   now done in makethin.c

-new track plot routine, plots are appended to existing ps file

-Fixing the limit of the length of 10000  characters for macros (courtesy HG)

-Doubled the number of tracked particles in track_dynap so as to include
the Lyapunov partners.

-Tracking plot fix: now different postscript output filenames can be used.
Note: the plotting routine plots only at the end of mad executing a gnuplot
command file. The data used for plotting are last data stored in the table file.

-introduce Werner's improved readtable, further clean-up
==============================================================================
madxp.c

-Changes to the Core Code needed for PTC upgrade V3.00

-frs + hbu : General cleaning. Tabs to blanks. Remove blanks at eol.
Convert any C++ comments to ansi c comments.
Fix possible uninitialized variables in madxn.c

-new track plot routine, plots are appended to existing ps file

-Fixing the limit of the length of 10000  characters for macros (courtesy HG)
==============================================================================
madxu.c

-simple clean-up

-frs + hbu : General cleaning. Tabs to blanks. Remove blanks at eol.
Convert any C++ comments to ansi c comments.
Fix possible uninitialized variables in madxn.c

-Fix of install bug (courtesy HG)
==============================================================================
makethin.c

-compound_expr moved from madxn.c to makethin.c as only used here
and increased precision   sprintf(tmp, "%e"  ->   sprintf(tmp, "%.14g"

-additional check in set_selected_elements() that current_sequ is not NULL

-frs + hbu : General cleaning. Tabs to blanks. Remove blanks at eol.
Convert any C++ comments to ansi c comments.
Fix possible uninitialized variables in madxn.c

-improved selection

-code for select,flag=makethin rewrittes, with class, range and pattern
==============================================================================
matchc.c

-simple clean-up

-frs + hbu : General cleaning. Tabs to blanks. Remove blanks at eol.
Convert any C++ comments to ansi c comments.
Fix possible uninitialized variables in madxn.c
==============================================================================
photoni.inc

-New routine needed for Frank's quadrupole radiation in ptc_track
==============================================================================
plot.F

-The momemtum offset label is not displayed when deltap is the horizzontal axis
==============================================================================
plot.fi

-simple clean-up
==============================================================================
plot_b.fi

-simple clean-up
==============================================================================
user2_photon.f90, poisson.F

-New routine needed for Frank's quadrupole radiation in ptc_track
==============================================================================
resindex.fi

-simple clean-up
==============================================================================

-simple clean-up

-frs + hbu : General cleaning. Tabs to blanks. Remove blanks at eol.
Convert any C++ comments to ansi c comments.
Fix possible uninitialized variables in madxn.c

-Fixing the limit of the length of 10000  characters for macros (courtesy HG)
==============================================================================
timel.c

-simple clean-up

-frs + hbu : General cleaning. Tabs to blanks. Remove blanks at eol.
Convert any C++ comments to ansi c comments.
Fix possible uninitialized variables in madxn.c
==============================================================================
touschek.F

-simple clean-up
==============================================================================
trrun.F

-simple clean-up

-Fix bug introduced to upgrade dynap

-Made TRRUN consistent with new version of DYNAP: removed switch=3, which was never active, and added Lyapunov calculation to switch=2.
==============================================================================
wrap.f90

-Mods needed for the ptc_twiss upgrade and the new ptc_track module
==============================================================================
```

MAD-X version 2.13.09: 09.03.2005

Overview:

```
1) New Touscheck module (CM & FZ)

2) Hamiltonian terms from the ptc_normal module put into TFS table
(Td'A)

3) Orbit correction: Changes to orbit correction: setcorr and input of
external file (WH)

4) Error module (WH):
- Changes to correction factor in EFCOMP
as requested by Massimo
- Additional routines to read errors from
ESAVE: seterror command

5) Aperture Module (Ivar Iwaarum): Various changes

6) Improved "readtable" command (WH)

7) New "dipedge" element in trrun & twiss module: this allows to track
(thin lens) through all systems that can be described with MAD-X
except higher order non-symplectic terms. (AV & FS)

8) Deselect command for twiss tables (HG)

9) Include tilt of various elements into the survey module (FT)

10) Complete clean-up of all Fortan77 & C routines to produce code
compliant with Fortran and C standards, suppress all warnings from
compilers with strict checking, unless acceptable (E.g. unused
variables due to include files)

10) Various bug fixes. Special thanks to Hans Grote for correcting
some tricky ones!
```

All changes for each file:

```
============================================================================
Makefile

- add resindex dependence + touschek typo
============================================================================
Makefile.bat

- naming bug of resindex.F
- Reintroduce location of proper .h files for CL compiler
- drop Include add resindex
============================================================================
Makefile_develop

- Fix bad link static flag "-static" by the proper "-Bstatic" one
- fully for testing now
- This development Makefile makes the most stringent tests both Fortran & C
- add resindex subroutine
- Produce static executables
============================================================================
madX/c6t.c

- avoid possible undefined "tag_element" & automatic indent
============================================================================
dynap.F

- Fortran clean-up with mymod, f2c, lf95, f95
============================================================================
emit.F

- In routine emdamp bv0 was uninitialized
- Fortran clean-up with mymod, f2c, lf95, f95
============================================================================
gxx11.F

- Fortran clean-up with mymod, f2c, lf95, f95
============================================================================
gxx11c.c

- solid & dash_list changed from "static unsigned char" to "static char"
courtesy HG
- get rid of C++ style
============================================================================
gxx11ps.F

- Fortran clean-up with mymod, f2c, lf95, f95
============================================================================
ibsdb.F

- minor cleaning
- Fortran clean-up with mymod, f2c, lf95, f95
============================================================================
madx.h

- Removed pedantic errors.
- Added possibility to give an elements true profile, and its offset
w.r.t. a point in space.
- Aperture command no longer prints aper1.out as default.
- Deselect command for tables (courtesy H.G.)
- Repair:
1) multiple negative Lines
2) bug in seq_replace
- Add:
deselect command                        (Courtesy HG)
- Changes for error, orbit and aperture module
- revert Ivar's last changes
- Aperture
madx.h: Added struct aper_node.
madxd.h: Added aperture module function declarations, the aperture
table and char aptwfile as global variables.
madxdict.h: Added aperture command and aper_tol argument for all elements.
madxl.h: Added MAXARRAY definition, racetrack as an apertype, an
aperture table definition and added n1 to the Twiss table definition.
madxn.c: Added all aperture functions, and made the exec_plot able to
print aperture tables.
madxp.c: Added call to pro_aperture.
============================================================================
madx_ptc_module.f90

- change to mode definition: qx ==> q1 etc
- add Hamiltonian terms to ptc_normal          Td'A
============================================================================
madxc.c

- Clean up to avoid compiler warnings
- Changes to orbit correction: setcorr and input of
external file.
============================================================================
madxd.h

- Clean up to avoid compiler warnings
- Removed pedantic errors.
- Added possibility to give an elements true profile, and its offset
w.r.t. a point in space.
- Aperture command no longer prints aper1.out as default.
- Deselect command for tables (courtesy H.G.)
- Fortran clean-up with mymod, f2c, lf95, f95
- fix res_index format fix
- Repair:
1) multiple negative Lines
2) bug in seq_replace
- Add:
deselect command                        (Courtesy HG)
- change to mode definition: qx ==> q1 etc
- add Hamiltonian terms to ptc_normal          Td'A
- IWAARUM 07.01.2005:
aperture module now supports long file names for Twiss table output.
- IWAARUM 06.01.2005:
Altered function header of aper_write_table().
- Changes for error, orbit and aperture modules
- revert Ivar's last changes
- Aperture
madx.h: Added struct aper_node.
madxd.h: Added aperture module function declarations, the aperture
table and char aptwfile as global variables.
madxdict.h: Added aperture command and aper_tol argument for all elements.
madxl.h: Added MAXARRAY definition, racetrack as an apertype, an
aperture table definition and added n1 to the Twiss table definition.
madxn.c: Added all aperture functions, and made the exec_plot able to
print aperture tables.
madxp.c: Added call to pro_aperture.
============================================================================
madxdict.h

- introduce Werner's improved readtable
- Removed bug due to uppercase/lowercase in offsetelem file.
- Added the possibility to assign self-made apertures from external
coordinate files.
- Removed pedantic errors.
- Added possibility to give an elements true profile, and its offset
w.r.t. a point in space.
- Aperture command no longer prints aper1.out as default.
- Deselect command for tables (courtesy H.G.)
- Adding a "dipedge" element so as to have all elements available in the
thin-lens tracking except second order (non-symplectic) terms.  FS & AV
- Fortran clean-up with mymod, f2c, lf95, f95
- Changes needed for the touschek module
- Repair:
1) multiple negative Lines
2) bug in seq_replace
- Add:
deselect command                        (Courtesy HG)
- change to mode definition: qx ==> q1 etc
- add Hamiltonian terms to ptc_normal          Td'A
- Changes to error, orbit and aperture modules
- New options for errors to allow time memory
effects, extern option in orbit correction
- revert Ivar's last changes
- Aperture
madx.h: Added struct aper_node.
madxd.h: Added aperture module function declarations, the aperture
table and char aptwfile as global variables.
madxdict.h: Added aperture command and aper_tol argument for all elements.
madxl.h: Added MAXARRAY definition, racetrack as an apertype, an
aperture table definition and added n1 to the Twiss table definition.
madxn.c: Added all aperture functions, and made the exec_plot able to
print aperture tables.
madxp.c: Added call to pro_aperture.
============================================================================
madxe.

- Clean up to avoid compiler warnings
- Changes to correction factor in EFCOMP
as requested by Massimo
- Additional routines to read errors from
ESAVE: seterror command
============================================================================
madxl.h

- Removed pedantic errors.
- Added possibility to give an elements true profile, and its offset
w.r.t. a point in space.
- Aperture command no longer prints aper1.out as default.
- Changes needed for the touschek module
- change to mode definition: qx ==> q1 etc
- add Hamiltonian terms to ptc_normal          Td'A
- IWAARUM 06.01.2005:
Cleaned up aperture table.
Added n1 in SI units to the aperture table.
- Changes for orbit and aperture modules.
- Aperture table and additional apertypes
- n1 into Twiss table
- revert Ivar's last changes
- Aperture
madx.h: Added struct aper_node.
madxd.h: Added aperture module function declarations, the aperture
table and char aptwfile as global variables.
madxdict.h: Added aperture command and aper_tol argument for all elements.
madxl.h: Added MAXARRAY definition, racetrack as an apertype, an
aperture table definition and added n1 to the Twiss table definition.
madxn.c: Added all aperture functions, and made the exec_plot able to
print aperture tables.
madxp.c: Added call to pro_aperture.
============================================================================
madxn.c

- introduce Werner's improved readtable, further clean-up
- letting point "at" of an freshly installed element to the position
Courtesy HG
- Fix the "no haxis" bug introduced in the clean-up (found by JBJ)
- Removed bug due to uppercase/lowercase in offsetelem file.
- Added the possibility to assign self-made apertures from external
coordinate files.
- Suppress unused variables, safeguard potentially uninitialized variables,
sanity checks in embedded_twiss, exec_plot & sodd
- Removed pedantic errors.
- Added possibility to describe an elements true profile, and its
offset w.r.t. a point in space.
- Deselect command for tables (courtesy H.G.)
- Fix of pseudo-crash when an element had been re-installed (with class
attribute) in a seqedit - now simply ignored            (courtesy HG)
- Changes needed for the touschek module
- Repair:
1) multiple negative Lines
2) bug in seq_replace
- Add:
deselect command                        (Courtesy HG)
- change to mode definition: qx ==> q1 etc
- add Hamiltonian terms to ptc_normal          Td'A
- IWAARUM 06.01.2005:
No longer uses isnan().
No longer uses strcasecmp() and strncasecmp().
Corrected printing of n1 to Twiss table.
Changed ap# to aper_# in aperture table.
Removed printing of "node_minimum" line in aperture table.
- Added aperture module functions, plot function
can plot aperture table
- revert Ivar's last changes
- Aperture
madx.h: Added struct aper_node.
madxd.h: Added aperture module function declarations, the aperture
table and char aptwfile as global variables.
madxdict.h: Added aperture command and aper_tol argument for all elements.
madxl.h: Added MAXARRAY definition, racetrack as an apertype, an
aperture table definition and added n1 to the Twiss table definition.
madxn.c: Added all aperture functions, and made the exec_plot able to
print aperture tables.
madxp.c: Added call to pro_aperture.
============================================================================
madxp.c

- Suppress unused variables, safeguard potentially uninitialized variables,
sanity checks in embedded_twiss, exec_plot & sodd
- Deselect command for tables (courtesy H.G.)
- Repair:
1) multiple negative Lines
2) bug in seq_replace
- Add:
deselect command                        (Courtesy HG)
- Call to aperture module
- revert Ivar's last changes
- Aperture
madx.h: Added struct aper_node.
madxd.h: Added aperture module function declarations, the aperture
table and char aptwfile as global variables.
madxdict.h: Added aperture command and aper_tol argument for all elements.
madxl.h: Added MAXARRAY definition, racetrack as an apertype, an
aperture table definition and added n1 to the Twiss table definition.
madxn.c: Added all aperture functions, and made the exec_plot able to
print aperture tables.
madxp.c: Added call to pro_aperture.
============================================================================
madxu.c

- Fix of pseudo-crash when an element had been re-installed (with class
attribute) in a seqedit - now simply ignored            (courtesy HG)
============================================================================
makethin.c

- implement element instrument
============================================================================
match.

- Fortran clean-up with mymod, f2c, lf95, f95
============================================================================
matchc.c

- Fortran clean-up with mymod, f2c, lf95, f95
============================================================================
matchsa.F

- Fortran clean-up with mymod, f2c, lf95, f95
============================================================================
orbf.F

- Formal changes to satisfy f2c and Fortran90 compatibility
============================================================================
plot.F

- Fortran clean-up with mymod, f2c, lf95, f95
- plot_style & plot_symbol have to be defined as integer arrays
- bug corrected in plot.F (check of style+symbol in wrong place).
Only the first variable was plotted.
============================================================================
plot_math.f

- add missing newline
============================================================================
resindex.F

- Fortran clean-up with mymod, f2c, lf95, f95
- Finding resonance indices
============================================================================
resindex.fi

- Fortran clean-up with mymod, f2c, lf95, f95
- Finding resonance indices
============================================================================
sodd.F

- Fortran clean-up with mymod, f2c, lf95, f95
- Fix of mix-up of data and define statements
- adjust sodd results to PTC NormalForm by multiplying with multiples of 1000
============================================================================
survey.F

- suelem in util.F returns tilt for all elements (except BEND + MULT
with angle=0)
```

```
- GLOBALTILT takes into account tilt for all elements now
- repeated angle initialization taken out
==========================================================================
sxf.c

- automatic indent
==========================================================================
touschek.F

- Fortran clean-up with mymod, f2c, lf95, f95
- Well tested touschek module
==========================================================================
touschek.f

- General input for touschek.F
==========================================================================
trrun.F

- Adding a "dipedge" element so as to have all elements available in the
thin-lens tracking except second order (non-symplectic) terms.  FS & AV
- Fortran clean-up with mymod, f2c, lf95, f95
- Introduction of dipedge. AV
==========================================================================
twiss.F

- Adding a "dipedge" element so as to have all elements available in the
thin-lens tracking except second order (non-symplectic) terms.  FS & AV
- Fortran clean-up with mymod, f2c, lf95, f95
- Activating synch_1, synch_2,synch_3,synch_5, first use in touschek
- Centre option corrected when closed orbit present
==========================================================================
util.F

- suelem in util.F returns tilt for all elements (except BEND + MULT
with angle=0)
- GLOBALTILT takes into account tilt for all elements now
- Fortran clean-up with mymod, f2c, lf95, f95
==========================================================================
```

MAD-X version 2.13: Update-I WH 09.12.2004

```
error module:
1. allows to add time memory effects to
   field errors, see documentation on
   EFCOMP
2. set errors directly from a file previously
   produced by an ESAVE command

orbit module:
1. set correctors directly from a file
   produced by a CORRECT command
2. allow correction of orbits read from
   and external file.
3. READMYTABLE as new generic table access
   function.

Aperture Module:
***************

madxn.c:
16 function definitions for the aperture module.
A change in
exec_plot allows the PLOT-command to take the argument table=aperture to
print an aperture table.
Inserted a line in pro_twiss which saves the output filename for the twiss
table.

madxp.c:
A call to pro_aperture.

madxl.h:
Added racetrack as a possible apertype.
Defined the aperture table.
Added n1 to the Twiss table.

madxd.h:
16 aperture function declarations.
Added char aptwfile as global variable.

madx.h:
New aperture struct declaration.

madxdict.h:
Aperture command definition.
Added aper_tol as a possible argument to all elements.
```

MAD-X version 2.13: FS 23.11.2004

```
Summary:
The first step has been done for MAD-X matching with nonlinear terms
from PTC. By the end of the year matching should be operation in the
latest MAD-X version 2.13.

Changes with respect to the previous version listed per modified routine:
==========================================================================
Makefile

-Extra link flag to work under SLC3
==========================================================================
Makefile.bat

-Fix of SET INCLUDE path for stdio.h
==========================================================================
emit.F

-Fix the notorious tilt issues since this was still in the intermediate
state using k0/k0s instead of tilt.
==========================================================================
madx_ptc_module.f90

-nonlinear ptc acquisition -- Td'A
-fixed illegal extension "<>" by ".ne."
-use current index in structure table instead of external indexing
-length instead of dimension in character definition for "string_from_table"

-Allow any rest mass
==========================================================================
madxd.h

-nonlinear ptc acquisition -- Td'A
-use current index in structure table instead of external indexing

-add cf77flush template
==========================================================================
madxdict.h

-nonlinear ptc acquisition -- Td'A
==========================================================================
madxl.h,v

nonlinear ptc acquisition -- Td'A
==========================================================================
madxn.c,v

-fix "read_table" bug (courtesy HG)

-nonlinear ptc acquisition Td'A
-use current index in structure table instead of external indexing

-Two essential select bugs (courtesy HG)
==========================================================================
madxp.c,v

-fix "read_table" bug (courtesy HG)

-nonlinear ptc acquisition Td'A
-use current index in structure table instead of external indexing

-Fix of faulty "resbeam" command  found by EK and fixed courtesy HG
==========================================================================
madxu.c

-C routine cf77flush for flushing unit 6 in C & F subroutines
==========================================================================
plot.F

-fixed 4th argument of comm_para

-attribute 'trackfile' for plot.

-correct treatment of tracking plots for particles for which no tracking is
available.

-check of the zero values of the style & symbols attributes.
==========================================================================
timest.f90

-First timing routine needed for PTC
==========================================================================
timex.f90

-Second timing routine needed for PTC
==========================================================================
trrun.F

-correction of bug in thin multipole focusing. AV
==========================================================================
```

MAD-X version 2.12: FS 29.09.2004

[illegible]

MAD-X version 2.11: FS 02.06.2004

[illegible]

MAD-X version 2.10: FS 27.03.2004

MAD-X version 2.00: FS 24.11.2003

MAD-X version 1.12: FS 04.07.2003

MAD-X version 1.11: FS 26.04.2003

MAD-X version 1.10: HG & FS 20.01.2003

MAD-X version 1.09: FS 09.12.2002

MAD-X version 1.08: FS 18.11.2002

MAD-X version 1.06: FS 16.10.2002

MAD-X version 1.05: HG 25.9.2002

New sample job to create footprints for LHC: HG 18.9.2002

EUROPEAN ORGANIZATION FOR NUCLEAR RESEARCH 

RPLOT

Introduction

RPLOT is a MAD-X plug-in that privides additional functionality using ROOT . It contains several tools

RVIEWER

plotting tool that handles the results in paramremtric form

What makes it different from the standard PLOT module of MAD-X is that it is also able to deal with the parmateric results. RPLOT proviedes graphical user interface that allows to choose which functions shall be drawn, set its ranges and adjust all the details of the plot formatting. Of course, the result is immendiately visible on the screen, in contrary to the standard plot tool that is able to work solely in the batch mode. The user can choose several formats to save his plot, including postscript, gif, pdf, root macro and many others.

RVIEWER is able to draw the lattice functions

1. along the layout
2. at given position in function of one or two knobs

It provides a convienient way to set the knob values. As the value is set, the plotted functions are immediately drawn for the new value.

In order to run RVIEWER simpy issue "rviewer;" command

RTRACKSTORE

enables storage of the tracking data in ROOT NTuple/Tree format

Ntuple and its modern extension called Tree are formats designed for storing particle tracking data. It is proven to provide the fastest data writing and reading thanks to column wise I/O operations. It is commonly used for data storage by HEP experiments. Additionally, ROOT provides automatical ZIP data compression that is transparent for the user algorithms. Morover, ROOT provides wide set of very comfortable tools for advanced analysis and plotting of the data stored in Trees.

Addtionally, we plan to extend RVIEWER functionality that would provide intuitive graphical user interface to most commonly used features in particle tracking in accelerators. Thanks to that, the user is not forced to learn how to use the ROOT package.

Currently the feature is enabled only for tracking using the ptc_trackline command, however, it will be extended to other tracking modes.

Download

The newest version is available here

Installation

Prerequisite: ROOT must be installed beforehand compilation and whenever the user wants to use the plug-in. See explanations on ROOT webpage.

To install RPLOT

1. Unpack the archive, it will create directory rplot

```
tar xvzf rplot-X.XX.tgz
```

2. Change to rplot directory

```
cd rplot
```

3. Type

```
make install
```

Examples

SYNOPSIS

```
RVIEWER;
```

PROGRAMMERS MANUAL

To be continued...

References

1
The Graphical Kernel System (GKS). ISO, Geneva, July 1985. International Standard ISO 7942.

2
B. Autin and Y. Marti. *Closed Orbit Correction of Alternating Gradient Machines using a small Number of Magnets*. CERN/ISR-MA/73-17, CERN, 1973.

3
D.P. Barber, K. Heinemann, H. Mais and G. Ripken, *A Fokker--Planck Treatment of Stochastic Particle Motion within the Framework of a Fully Coupled 6-dimensional Formalism for Electron-Positron Storage Rings including Classical Spin Motion in Linear Approximation*, DESY report 91-146, 1991.

4
R. Bartolini, A. Bazzani, M. Giovannozzi, W. Scandale and E. Todesco, *Tune evaluation in simulations and experiments*, CERN SL/95-84 (AP) (1995).

5
J. D. Bjorken and S. K. Mtingwa. *Particle Accelerators* **13**, pg. 115.

6
E. M. Bollt and J. D. Meiss, *Targeting chaotic orbits to the Moon through recurrence*, Phys. Lett. A 204,373 (1995).

7
P. Bramham and H. Henke. private communication and LEP Note LEP-70/107, CERN.

8
Karl L. Brown. *A First-and Second-Order Matrix Theory for the Design of Beam Transport Systems and Charged Particle Spectrometers*. SLAC 75, Revision 3, SLAC, 1972.

9
Karl L. Brown, D. C. Carey, Ch. Iselin, and F. Rothacker. *TRANSPORT - A Computer Program for Designing Charged Particle Beam Transport Systems*. CERN 73-16, revised as CERN 80-4, CERN, 1980.

10
A. Chao. *Evaluation of beam distribution parameters in an electron storage ring*. Journal of Applied Physics, 50:595-598, 1979.

11
A. W. Chao and M. J. Lee. *SPEAR II Touschek lifetime*. SPEAR-181, SLAC, October 1974.

12
M. Conte and M. Martini. *Particle Accelerators* **17**, 1 (1985).

13
E. D. Courant and H. S. Snyder. *Theory of the alternating gradient synchrotron*. Annals of Physics, 3:1-48, 1958.

14
Ph. Defert, Ph. Hofmann, and R. Keyser. *The Table File System, the C Interfaces*. LAW Note 9, CERN, 1989.

15
M. Donald and D. Schofield. *A User's Guide to the* `HARMON` *Program*. LEP Note 420, CERN, 1982.

16
A. Dragt. *Lectures on Nonlinear Orbit Dynamics*, 1981 Summer School on High Energy Particle Accelerators, Fermi National Accelerator Laboratory, July 1981. American Institute of Physics, 1982.

17
D. A. Edwards and L. C. Teng. *Parametrisation of linear coupled motion in periodic systems*. IEEE Trans. on Nucl. Sc., 20:885, 1973.

18
M. Giovannozzi, *Analysis of the stability domain of planar symplectic maps using invariant manifolds*, CERN/PS 96-05 (PA) (1996).

19
H. Grote. *GXPLOT User's Guide and Reference Manual*. LEP TH Note 57, CERN, 1988.

20
LEP Design Group. *Design Study of a 22 to 130 GeV electron-positron Colliding Beam Machine (LEP)*. CERN/ISR-LEP/79-33, CERN, 1979.

M. Hanney, J. M. Jowett, and E. Keil. *BEAMPARAM - A program for computing beam dynamics and performance of electron-positron storage rings*. CERN/LEP-TH/88-2, CERN, 1988.

22
R. H. Helm, M. J. Lee, P. L. Morton, and M. Sands. *Evaluation of synchrotron radiation integrals*. IEEE Trans. Nucl. Sc., NS-20, 1973.

23
F. James. *MINUIT, A package of programs to minimise a function of n variables, compute the covariance matrix, and find the true errors*. program library code D507, CERN, 1978.

24
E. Keil. *Synchrotron radiation from a large electron-positron storage ring*. CERN/ISR-LTD/76-23, CERN, 1976.

25
D. E. Knuth. *The Art of Computer Programming*. Volume 2, Addison-Wesley, second edition, 1981. Semi-numerical Algorithms.

26
J. Laskar, C. Froeschle and A. Celletti, *The measure of chaos by the numerical analysis of the fundamental frequencies. Application to the standard mapping*, Physica D 56, 253 (1992).

27
H. Mais and G. Ripken, *Theory of Coupled Synchro-Betatron Oscillations*. DESY internal Report, DESY M-82-05, 1982.

28
M. Meddahi, *Chromaticity correction for the 108/60 degree lattice*, CERN SL/Note 96-19 (AP) (1996).

29
J. Milutinovic and S. Ruggiero. *Comparison of Accelerator Codes for a RHIC Lattice*. AD/AP/TN-9, BNL, 1988.

30
B. W. Montague. *Linear Optics for Improved Chromaticity Correction*. LEP Note 165, CERN, 1979.

31
Gerhard Ripken, *Untersuchungen zur Strahlführung und Stabilität der Teilchenbewegung in Beschleunigern und Storage-Ringen unter strenger Berücksichtigung einer Kopplung der Betatronschwingungen*. DESY internal Report R1-70/4, 1970.

32
F. Ruggiero, *Dynamic Aperture for LEP 2 with various optics and tunes*, Proc. Sixth Workshop on LEP Performance, Chamonix, 1996, ed. J. Poole (CERN SL/96-05 (DI),1996), pp. 132--136.

33
L. C. Teng. *Concerning n-Dimensional Coupled Motion*. FN 229, FNAL, 1971.

34
U. Völkel. *Particle loss by Touschek effect in a storage ring*. DESY 67-5, DESY, 1967.

35
R. P. Walker. *Calculation of the Touschek lifetime in electron storage rings*. 1987. Also SERC Daresbury Laboratory preprint, DL/SCI/P542A.

36
P. B. Wilson. *Proc. 8th Int. Conf. on High-Energy Accelerators*. Stanford, 1974.

37
A. Wrulich and H. Meyer. *Life time due to the beam-beam bremsstrahlung effect*. PET-75-2, DESY, 1975.

38
H. Grote, J. Holt, N. Malitsky, F. Pilat, R. Talman, C.G. Trahern. *SXF (Standard eXchange Format): definition, syntax, examples*. RHIC/AP/155, August, 1998.

39
F. Schmidt. *SixTrack, User's Reference Manual*. CERN SL/94-56 (AP).

40
M. Hayes and F. Schmidt. *Run Environment for SixTrack*. Physics Note 53 (unpublished) & LHC Project Note 300.

41
F. Schmidt. *SODD: A computer code to calculate detuning and distortion function terms in first and second order*. CERN SL/Note 99-009 (AP).

42
R.Talman and L.Schachinger. *TEAPOT. A Thin Element Accelerator Program for Optics and Tracking*. SSC-52.

43
J.D. Bjorken and S.K. Mtingwa, *Intrabeam Scattering*, FERMILAB-Pub-82/47-THY, July 1982.

---

frs, May 01, 2003