The MAD-X Program
(Methodical Accelerator Design)
Version 5.08.01
User's Reference Manual

Laurent Deniau (editor)
Hans Grote
Ghislain Roy
Frank Schmidt

Abstract

`MAD-X` is a general-purpose tool for charged-particle optics design and studies in alternating-gradient accelerators and beam lines. It can handle medium size to very large accelerators and solves various problems on such machines.

`MAD-X` is the successor of `MAD-8` and was specifically adapted to the needs of the design of the LHC. The `PTC` library of E. Forest is also embedded in `MAD-X` as an addition to better support small and low energy accelerators. A large part of the present document is based on the `MAD-8` documentation originally written and published by F.C. Iselin.

This documentation is updated regularly as corrections, improvements and additions are made to the program. It is also available online on the `MAD-X` website.

Comments and corrections from readers are most welcome. They may be sent to the email address: `mad@cern.ch`

Copyright Notice

Contents

List of Tables

List of Figures

Part I

Control

Chapter 1. Conventions

1.1 Reference System

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 1.1).
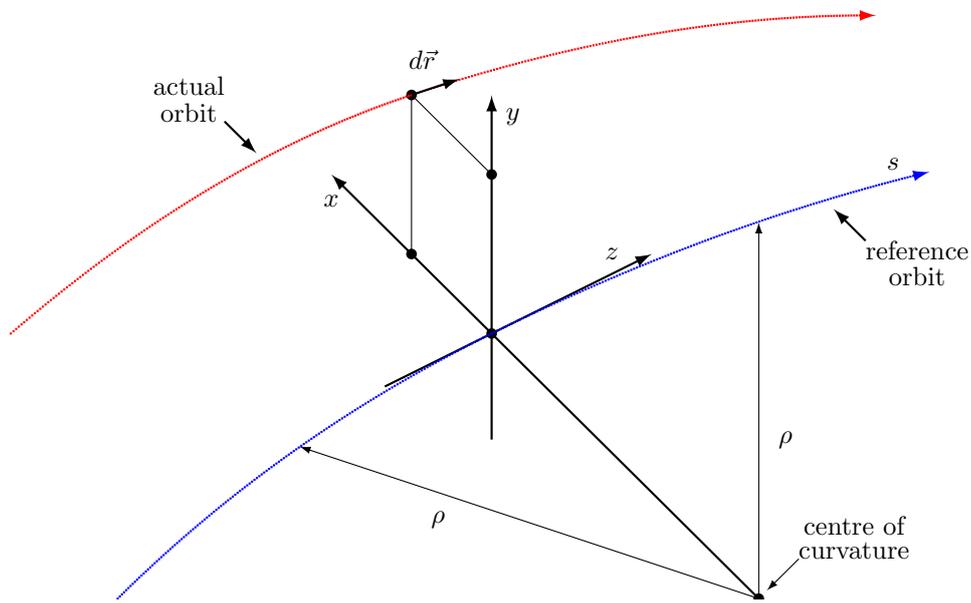
Figure 1.1: Local Reference System

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).

1.2 Closed Orbit

Due to various errors like misalignment errors, field errors, fringe fields etc., the closed orbit does not coincide with the reference orbit. The closed orbit 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-X` 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.

1.3 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 1.2).

The positions between beam elements are indexed with $i = 0, \ldots n$. 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 $(\theta_i, \phi_i, \psi_i)$
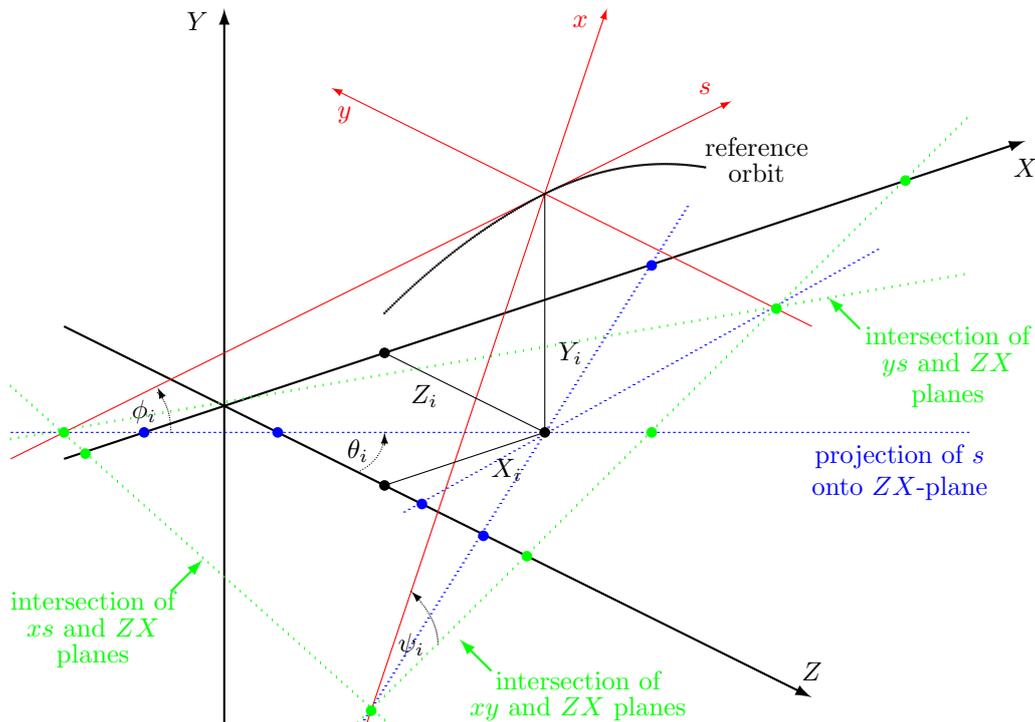
Figure 1.2: Global Reference System showing the global Cartesian system (X, Y, Z) in black and the local reference system (x, y, s) in red after translation (X_i, Y_i, Z_i) and rotation $(\theta_i, \phi_i, \psi_i)$. The projections of the local reference system axes onto the horizontal ZX plane of the Cartesian system are figured with blue dashed lines. The intersections of planes ys, xy and xs of the local reference system with the horizontal ZX plane of the Cartesian system are figured in green dashed lines.

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	θ is the 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 ϕ is the elevation angle, i.e. the angle between the reference orbit and its pro-
                       jection onto the (Z, X)-plane. A positive angle PHI corresponds to increasing
                       Y.
                       If only horizontal bends are present, the reference orbit remains in the (Z, X)-plane and PHI is always zero.

PSI ψ is the roll angle about the local s-axis, i.e. the angle between the line defined
                       by the intersection of the (x, y)-plane and (Z, X)-plane on one hand, and the
                       local x-axis on the other hand. A positive angle PSI forms a right-hand screw
                       with the s-axis.

The angles (θ, ϕ, ψ) 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 initial values $(X_0, Y_0, Z_0, \theta_0, \phi_0, \psi_0)$ are
therefore all zero unless the user specifies 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:

$$V = \begin{pmatrix} X \\ Y \\ Z \end{pmatrix}, \qquad W = \Theta \quad \Phi \quad \Psi \tag{1.1}$$

where

$$\Theta = \begin{pmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{pmatrix}, \quad \Phi = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & \sin\phi \\ 0 & -\sin\phi & \cos\phi \end{pmatrix}, \quad \Psi = \begin{pmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{1.2}$$

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 2π) with the revolution
frequency. If ψ is not periodic (modulo 2π), coupling effects are introduced. When advancing
through a beam element, MAD-X computes V_i and W_i by the recurrence relations

$$V_i = W_{i-1}R_i + V_{i-1}, \qquad W_i = W_{i-1}S_i \tag{1.3}$$

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 below for different physical element types.

1.4   Local Reference Systems

1.4.1   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 spaces, Quadrupoles, Sextupoles, Octupoles,
Solenoids, Crab cavities, RF cavities, Electrostatic separators, Closed orbit correctors and
Beam position monitors.

The corresponding R, S are

$$R = \begin{pmatrix} 0 \\ 0 \\ L \end{pmatrix}, \quad S = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}. \tag{1.4}$$

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.
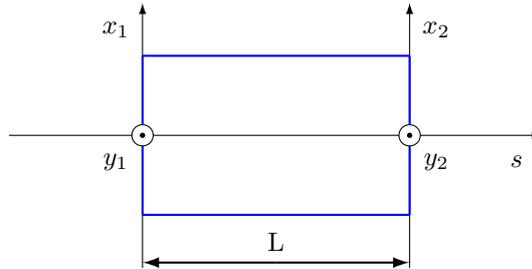
Figure 1.3: Reference System for Straight Beam Elements

1.4.2 Reference System for Bending Magnets

Bending magnets have a curved reference orbit. For both rectangular and sector bending magnets, the R and S are expressed as function the bend angle α:

$$R = \begin{pmatrix} \rho\,(\cos\alpha - 1) \\ 0 \\ \rho\sin\alpha \end{pmatrix}, \quad S = \begin{pmatrix} \cos\alpha & 0 & -\sin\alpha \\ 0 & 1 & 0 \\ \sin\alpha & 0 & \cos\alpha \end{pmatrix} \tag{1.5}$$

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 ρ, 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 ψ, R and S are transformed by

$$\overline{R} = TR, \qquad \overline{S} = TST^{-1} \tag{1.6}$$

where T is the orthogonal rotation matrix

$$T = \begin{pmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{1.7}$$

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

1.5 Sign Conventions for Magnetic Fields

The `MAD-X` program uses the following Taylor expansion for the field on the mid-plane $y = 0$, described in [1] (Note the factorial in the denominator):

$$B_y(x, 0) = \sum_{n=0}^{\infty} \frac{B_n\,x^n}{n!} \tag{1.8}$$

The field coefficients have the following meaning:
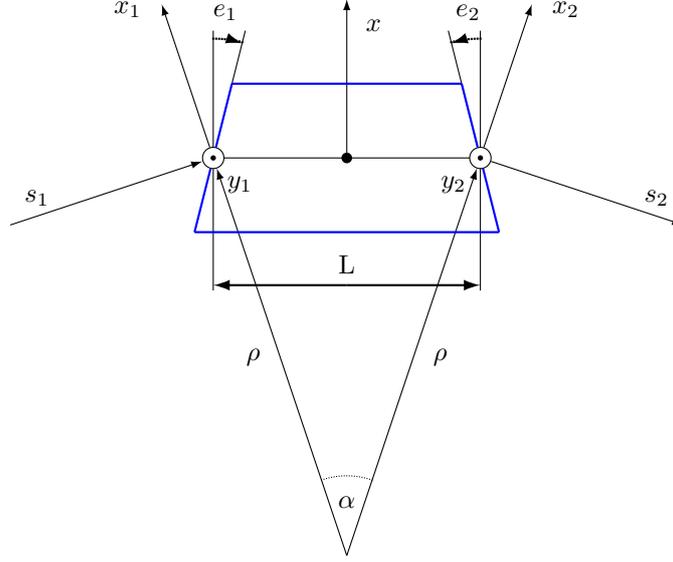
Figure 1.4: Reference System for a Rectangular Bending Magnet; the signs of pole-face rotations are positive as shown.

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 $B_1 = (\partial B_y/\partial x)$; a positive value corresponds to horizontal focussing of a positively charged particle.

B_2 Sextupole coefficient $B_2 = (\partial^2 B_y/\partial x^2)$.

B_3 Octupole coefficient $B_3 = (\partial^3 B_y/\partial x^3)$.

...   etc.

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

$$
\begin{aligned}
A_s = & + B_0 \left(x - \frac{hx^2}{2(1+hx)} \right) \\
& + B_1 \left(\frac{1}{2}(x^2 - y^2) - \frac{h}{6}x^3 + \frac{h^2}{24}(4x^4 - y^4) + \cdots \right) \\
& + B_2 \left(\frac{1}{6}(x^3 - 3xy^2) - \frac{h}{24}(x^4 - y^4) + \cdots \right) \\
& + B_3 \left(\frac{1}{24}(x^4 - 6x^2y^2 + y^4) \cdots \right) \\
& + \cdots
\end{aligned}
\tag{1.9}
$$
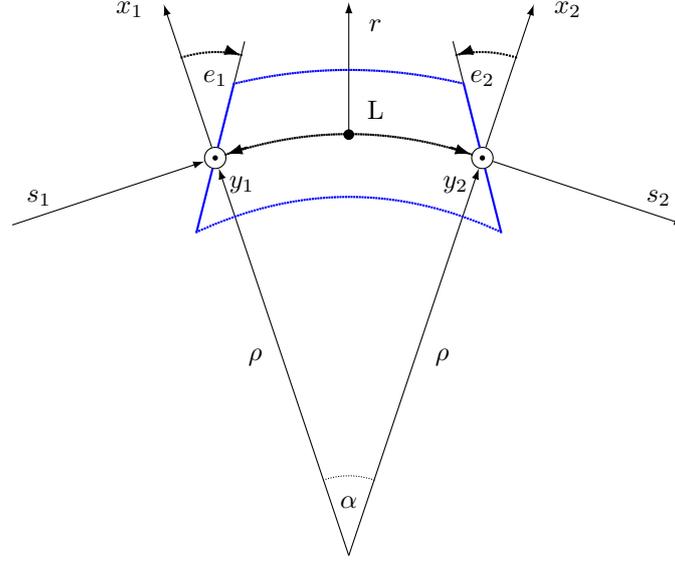
Figure 1.5: Reference System for a Sector Bending Magnet; the signs of pole-face rotations are positive as shown.

Taking $\vec{B} = \nabla \times \vec{A}$ in curvilinear coordinates, the field components can be computed as

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

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

$$(1.10)$$

It can be easily verified that both $\nabla \times \vec{B}$ and $\nabla . \vec{B}$ are zero to the order of the B_3 term.

Introducing the magnetic rigidity $B\rho = p_s/q$ as the momentum of the particle divided by its charge, the multipole coefficients are computed as

$$K_n = qB_n/p_s = B_n/B\rho \tag{1.11}$$

1.6   Generalisation to normal and skew components

The previous section assumed an expansion at the mid-plane ($y = 0$), symmetry around the mid-plane and considered only the vertical component of the field.

An extension using complex notation for the position ($x + iy$) and the field is given as

$$B_y + iB_x = \sum_{n=0}^{\infty}(b_n + ia_n)\frac{(x + iy)^n}{n^{n-1}} \tag{1.12}$$

By introducing the normal and skew multipole coefficients KN and KS at order n as

$$KN_n = q\, b_n/p_s = b_n/B\rho \tag{1.13}$$

and

$$KS_n = q\, a_n/p_s = a_n/B\rho \tag{1.14}$$

the kicks received in each plane can be expressed as the summation over all components

$$\Delta P_x - i\Delta P_y = \sum_{n=0}^{\infty} -(KN_n + iKS_n)\frac{(x + iy)^n}{n!} \tag{1.15}$$

Remark: need to add references to the (a_n, b_n) field conventions in the magnet world.

1.7   Variables

For each variable listed in this section, the physical units are given between square brackets, where [1] denotes a dimensionless variable.

1.7.1   Canonical Variables Describing Orbits

`MAD-X` 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 $= p_y/p_0$, [1].

T   Velocity of light times the negative time difference with respect to the reference particle: T $= -c\Delta t$, [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 $= \Delta E/p_s c$ where $p_s = p_0(1+\text{DELTAP})$, [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 between the reference momentum and the design momentum, divided by the design 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 ($\gamma \gg 1$, $v = c$, $pc = E$), the variables T and PT used here agree with the longitudinal variables used in [2]. 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.

1.7.2   Normalised Variables and other Derived Quantities

XN   The normalised horizontal displacement, [sqrt(m)]
                $x_n = Re(E_1^T \, S \, Z)$

PXN   The normalised horizontal transverse momentum, [sqrt(m)]
                $p_{xn} = Im(E_1^T \, S \, Z)$

WX   The horizontal Courant-Snyder invariant, [m]
                $\text{WX} = x_n^2 + p_{xn}^2$

PHIX   The horizontal phase, [1]
                $\phi_x = -\arctan(p_{xn}/x_n)/2\pi$

YN   The normalised vertical displacement, [sqrt(m)]
                $y_n = Re(E_2^T \, S \, Z)$

PYN   The normalised vertical transverse momentum, [sqrt(m)]
                $p_{yn} = Im(E_2^T \, S \, Z)$

WY   The vertical Courant-Snyder invariant, [m]
                $\text{WY} = y_n^2 + p_{yn}^2$

PHIY   The vertical phase, [1]
                $\phi_y = -\arctan(p_{yn}/y_n)/2\pi$

TN   The normalised longitudinal displacement, [sqrt(m)]
                $t_n = Re(E_3^T \, S \, Z)$

PTN   The normalised longitudinal transverse momentum, [sqrt(m)]
                $p_{tn} = Im(E_3^T \, S \, Z)$

WT   The longitudinal invariant, [m]
                $\text{WT} = t_n^2 + p_{tn}^2$

PHIT   The longitudinal phase, [1]
                $\phi_t = -\arctan(p_{tn}/t_n)/2\pi$

In the above formulas the vectors E_i are the three complex eigenvectors, Z is the phase space vector, and the matrix S is the "symplectic unit matrix":

$$
Z = \begin{pmatrix} x \\ p_x \\ y \\ p_y \\ t \\ p_t \end{pmatrix}, \quad
S = \begin{pmatrix}
0 & 1 & 0 & 0 & 0 & 0 \\
-1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & -1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & -1 & 0
\end{pmatrix}
\tag{1.16}
$$

1.7.3   Linear Lattice Functions (Optical Functions)

Several MAD-X commands refer to linear lattice functions or optical functions.

Because MAD-X uses the canonical momenta (p_x, p_y) instead of the slopes (x', y'), the definitions of the linear lattice functions differ slightly from those in Courant and Snyder[3].

Notice that in MAD-X, PT substitutes DELTAP as longitudinal variable. Dispersive and chromatic functions are hence derivatives with respect to PT. And since PT≈BETA*DELTAP (see derivation in chapter 39), where BETA is the relativistic Lorentz factor, those functions given by MAD-X must be multiplied by BETA a number of time equal to the order of the derivative to find the functions given in the literature.

The linear lattice functions are known to MAD-X under the following names:

BETX   Amplitude function β_x, [m].

ALFX   Correlation function $\alpha_x = -\frac{1}{2}(\partial \beta_x / \partial s)$, [1]

MUX   Phase function $\mu_x = \int ds/\beta_x$, [2π]

DX   Dispersion of x: $D_x = (\partial x/\partial p_t)$, [m]

DPX   Dispersion of p_x: $D_{px} = (\partial p_x/\partial p_t)/p_s$, [1]

BETY   Amplitude function β_y, [m]

ALFY   Correlation function $\alpha_y = -\frac{1}{2}(\partial \beta_y / \partial s)$, [1]

MUY   Phase function $\mu_y = \int ds/\beta_y$, [2π]

DY   Dispersion of y: $D_y = (\partial y/\partial p_t)$, [m]

DPY   Dispersion of p_y: $D_{py} = (\partial p_y/\partial p_t)/p_s$, [1]

R11, R12, R21, R22 : Coupling Matrix

1.7.4   Chromatic Functions

Several MAD-X commands refer to the chromatic functions.

Because MAD-X uses the canonical momenta (p_x, p_y) instead of the slopes (x', y'), the definitions of the chromatic functions differ slightly from those in [4].

Notice also that in MAD-X, PT substitutes DELTAP as longitudinal variable. Dispersive and chromatic functions are hence derivatives with respect to PT. Since PT≈BETA*DELTAP (see derivation in chapter 39), where BETA is the relativistic Lorentz factor, those functions

given by `MAD-X` must be multiplied by BETA a number of times equal to the order of the derivative to find the functions given in the literature.

The chromatic functions ([5] and references therein) are known to `MAD-X` under the following names:

WX   Chromatic amplitude function $W_x = \sqrt{a_x^2 + b_x^2}$, [1], where

$$b_x = \frac{1}{\beta_x}\frac{\partial\beta_x}{\partial p_t}, \qquad a_x = \frac{\partial\alpha_x}{\partial p_t} - \frac{\alpha_x}{\beta_x}\frac{\partial\beta_x}{\partial p_t}$$

PHIX   Chromatic phase function $\Phi_x = \arctan(a_x/b_x)$, [$2\pi$]

DMUX   Chromatic derivative of phase function: $DMUX = (\partial\mu_x/\partial p_t)$, [$2\pi$]

DDX   Chromatic derivative of dispersion D_x : $DDX = \frac{1}{2}(\partial^2 x/\partial p_t^2)$, [m]

DDPX   Chromatic derivative of dispersion D_{px} : $DDPX = \frac{1}{2}(\partial^2 p_x/\partial p_t^2)/p_s$, [1]

WY   Chromatic amplitude function $W_y = \sqrt{a_y^2 + b_y^2}$, [1], where

$$b_y = \frac{1}{\beta_y}\frac{\partial\beta_y}{\partial p_t}, \qquad a_y = \frac{\partial\alpha_y}{\partial p_t} - \frac{\alpha_y}{\beta_y}\frac{\partial\beta_y}{\partial p_t}$$

PHIY   Chromatic phase function $\Phi_y = \arctan(a_y/b_y)$, [$2\pi$]

DMUY   Chromatic derivative of phase function: $DMUY = (\partial\mu_y/\partial p_t)$, [$2\pi$]

DDY   Chromatic derivative of dispersion D_y : $DDY = \frac{1}{2}(\partial^2 y/\partial p_t^2)$, [m]

DDPY   Chromatic derivative of dispersion D_{py} : $DDPY = \frac{1}{2}(\partial^2 p_y/\partial p_t^2)/p_s$, [1]

1.7.5 Variables in the SUMM Table

After a successful TWISS command a summary table, with name SUMM, is created which contains the following variables:

LENGTH   The length of the machine, [m].

ORBIT5   The (ORBIT5 $= -T = c\Delta t$, [m]) component of the closed orbit.

ALFA   The momentum compaction factor α_c, [1].

GAMMATR The transition energy γ_{tr}, [1].

Q1   The horizontal tune Q_1 [1].

DQ1   The horizontal chromaticity $dq_1 = \partial Q_1/\partial p_t$, [1]

BETXMAX The largest horizontal β_x, [m].

DXMAX   The maximum of the absolute horizontal dispersion D_x, [m].

DXRMS   The r.m.s. of the horizontal dispersion D_x, [m].

XCOMAX The maximum of the absolute 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 $dq_2 = \partial Q_2/\partial p_t$, [1]
`BETYMAX`	The largest vertical β_y, [m].
`DYMAX`	The maximum of the absolute vertical dispersion D_y, [m].
`DYRMS`	The r.m.s. of the vertical dispersion D_y, [m].
`YCOMAX`	The maximum of the absolute vertical closed orbit deviation [m].
`YCORMS`	The r.m.s. of the vertical closed orbit deviation [m].
`DELTAP`	**Momentum difference, divided by the reference momentum [1]. DELTAP** $= \Delta p/p_0$
`SYNCH_1`	First synchrotron radiation integral
`SYNCH_2`	Second synchrotron radiation integral
`SYNCH_3`	Third synchrotron radiation integral
`SYNCH_4`	Fourth synchrotron radiation integral
`SYNCH_5`	Fifth synchrotron radiation integral
`SYNCH_6`	Sixth synchrotron radiation integral
`SYNCH_8`	Eighth synchrotron radiation integral

Notice that in `MAD-X`, PT substitutes DELTAP as longitudinal variable. Dispersive and chromatic functions are hence derivatives with respect to PT. And since PT≈BETA*DELTAP (see derivation in chapter 39) ,where BETA is the relativistic Lorentz factor, those functions given by `MAD-X` must be multiplied by BETA a number of time equal to the order of the derivative to find the functions given in the literature.

The first five radiation integrals, SYNCH_1 through to SYNCH_5, are defined and computed using the expressions in [6]. It should be noted that these integrals assume that the beam traverses through the centre of each element and does not take into account effects from non-zero closed orbits, including orbits resulting from chromatic effects. These integrals are therefore useful for estimating various radiation properties of an ideal machine but the EMIT module should be used for a more general case.

The integrals six and eight, SYNCH_6 and SYNCH_8, as defined in [7] have been implemented to capture phenomena not considered by the first five integrals. Integral six is required for computing the radiation in the quadrupoles and integral eight can be used to understand how the partition function varies for off momentum beams as detailed in [7]. Again, these estimates should be used with caution and the EMIT module should be used for precise computations.

1.7.6 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_y 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, $T = -c\Delta t$, [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, 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.

1.8   Physical Units

`MAD-X` uses units derived from the "Système International" (SI). These units are summarised in the Units table.

Table 1.1: Physical Units used by `MAD-X`

Quantity	Unit
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	2π
Particle energy	GeV
Particle mass	GeV/c^2
Particle momentum	GeV/c
Beam current	A (ampères)
Particle charge	e (elementary charges)
Impedance	MΩ (Megohms)
Emittance	$\pi * 10^{-3}$ m.rad
RF power	MW (megawatts)
Higher order mode loss factor	V/pc

Chapter 2. Command Format

2.1 Input Statements

Input for `MAD-X` follows in broad lines the `MAD-9` format, *i.e.* free format with commas (,) as separators, although blanks may be used as separators outside {...} enclosures.

Blank input lines do not affect program execution.

The input is not case sensitive except for strings enclosed in double quotes (" ").

The input file consists of a sequence of statements. A statement may occupy any number of input lines. Several statements may be placed on the same line. A statement must be terminated by a semicolon (;).

Several statements may be grouped into a block enclosed inside a {...} enclosure. In this case the terminating semmicolon can be omitted.

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

or

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

are both valid.

2.2 Comments

When an exclamation mark (`!`) or double forward slash (`//`) is found in the input line, the remaining characters until the end of the line are treated as a comment and are skipped.

A comment spreading over multiple lines starts with a string `/*` and ends with a string `*/`.

2.3 Commands

The general format for a command is

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

where the { } are not part of the command and the items enclosed in { } can be omitted or repeated any number of times.

A command contains three parts:

label   A label is required for a definition statement. A label gives a name to the stored command.

keyword   A keyword identifies the action desired.

attributes   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, and

`attribute-value` provides its value.

If a value is to be assigned to an attribute, the `attribute-name` is mandatory.

Whenever an attribute is not explicitly given a value, the default `attribute-value` specified in the command dictionary is assumed.

2.4 Keywords

A keyword begins with a letter and consists of letters and digits.

The `MAD-X` keywords are protected; attempting to use a protected keyword as a label results in a fatal error.

The keyword `VERSION` has been introduced since version 5.02.07. It contains the version number of the `MAD-X` release quoted as a decimal. For example:

```
X:> value VERSION;
version = 50207 ;
```

This allows testing of the version number of the running `MAD-X` process. Note that any version prior to version 5.02.07 reports the value `version = 0;`

2.5 Attribute Types

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 this command in memory. This allows repeated execution of the same command by simply entering `EXEC label;`. This construct may be nested.

2.6   Variable Declarations

In the following, "=" means that the variable on the left receives the current value of the expression on the right, but does not depend on it any further, whereas ":=" makes the variable on the left depend on the expression on the right, *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;
```

2.7   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.

2.8   Command Attributes

- A name or string attribute refers to an object, or a string.

- A logical attribute selects or deselects an option.

- An integer attribute defines a value stored as integer data.

- A real attribute defines a value stored as double precision data.

- 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.

2.9   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 lowercase 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 named my.file, in the last two it will try to read the file named MY.FILE.

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";
```

2.10 Logical Attributes

Many commands in `MAD-X` 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;
```

2.11   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.

2.12   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;
```

2.13   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-X` 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.

2.13.1   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,
`SINC(x)`	cardinal sine function,
`ABS(x)`	absolute value,
`ERF(x)`	Gauss error,
`ERFC(x)`	complementary error,
`FLOOR(x)`	floor, largest previous integer,
`CEIL(x)`	ceiling, smallest next integer,
`ROUND(x)`	round, closest integer;
`FRAC(x)`	fractional part of number;

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 or COPTION commands. Note that the command TRACK with QUANTUM attribute set to true may set or reset the seed too.

The command `OPTION` has two attributes to control the new random generator of MAD-X. The attribute `RAND` can take the value `default` (for backward compatibility) or `best` for the new generator. The latter supports 10 multiple independent generators numbered from 1 to 10, and can be selected with the attribute `RANDID=id` in combination to `RAND`. If `RANDID` is not specified and `RAND=best`, then the last `id` set is used (init: 1). The user can switch between the 10 independent generators at will to use different streams for the next generated random numbers. When a new seed is set through to command `EOPTION`, `COPTION` or `TRACK`, it will only affect the currently selected stream. The special initial seed for the `best` generator can be restored by using the value zero instead of the commands default 123456789.

The following example displays the first 10 random numbers of the 10 streams of the `best` generator, using the default initial seed.

```
option, -info;
i = 1;
while (i <= 10) {
  option, rand=best, randid=i;
  j = 1;
  while(j <= 10) {
    value, i, j, ranf();
    j = j+1;
  }
  i = i+1;
}
```

Table Access Functions

`TABLE(x,z)`   identical to TABLE(x,z,1); 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". When the element is called with its proper name, as in the example above, the value is returned at the first occurrence of the element of this name. If the value is needed at an occurrence number: NNN, then "[NNN]" has to be appended to the name, e.g. in the above example one obtains the betx of the 23th occurrence of the element "mb.12" by changing the example to: "table(twiss,mb.12[23],betx)". Mind you that the old, but little known, form: "table(twiss,mb.12->23,betx)" continues to work. Lastly, if NNN exceeds the maximum occurrence number the return value is arbitrarily small.

`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.

`TABINDEX(x,z,key)` identical to TABINDEX(x,z,1,key).

`TABINDEX(x,z,N_row,key)` returns the index of the first occurrence of a string starting by

> "key" in table column "z" (must be a column of strings) starting at the "N_row"
> number of rows of table "x"; example: tabindex(survey,name,102,bpm) returns
> the row number of the first element found starting from row 102 with a "name"
> beginning by "bpm" in the table "survey".

TABSTRING(x,z,N_row) accesses value of the named table column "z" (must be a column of
strings) at the "N_row" number of rows of table "x"; example: tabstring(sur-
vey,name,10) returns the name of the 10th element of the survey table "survey".
Row count starts at 1. The function can be used in any context where a string
appears; in case there is no match, it returns "_void_".

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

```
gettab(tblname, colname, rowidx) : macro = {
  gettab.val = table(tblname, colname, rowidx) ;
}
getidx(tblname, colname, rowidx, keyname) : macro = {
  getidx.val = tabindex(tblname, colname, rowidx, keyname) ;
}
row = 10 ;
exec, gettab(twiss,betx,$row) ;
exec, getidx(twiss,name,$row,qf) ;
value, gettab.val, getidx.val ;
```

Beware:

- Unnamed Drifts are not included in the table name database, so as to speed up the
  search for "real" elements. Therefore, those unnamed 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.

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:

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

Use macro in loop:

```
i = 0;
incval = 0;
while (i < 100) {
```

```
    value, i;
    exec, mymacro(twiss,k1l,$i);
    incval = incval + myval;
    value, i, myval, incval;
    i = i + 1;
}
```

2.13.2   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 INTE-
GER 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-X` recognizes 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;
```

A number of predefined symbolic constants exist in `MAD-X` and can be used in expressions.
The values of physical constants are regularly updated to reflect the values published by the
Particle Data Group [8].

The `NMASS` constant in `MAD-X` is now really the neutron mass and the atomic mass unit is
`UMASS`.

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;
```

Table 2.1: Predefined Symbolic Constants in `MAD-X`

MAD-X name	symbol	value	unit
PI	π	4 * atan(1)	1
TWOPI	2π	2 * PI	1
DEGRAD	$180/\pi$	180 / PI	deg/rad
RADDEG	$\pi/180$	PI / 180	rad/deg
E	e	exp(1)	1
EMASS	m_e	$0.51099895000 \times 10^{-3}$	GeV
PMASS	m_p	0.93827208816	GeV
NMASS	m_n	0.93956542052	GeV
UMASS	u	0.93149410242	GeV
MUMASS	m_μ	0.1056583715	GeV
CLIGHT	c	2.99792458×10^8	m/s
QELECT	e	$1.602176634 \times 10^{-19}$	A.s
HBAR	$\hbar$	$6.582119569 \times 10^{-25}$	MeV.s
ERAD	r_e	$2.8179403262 \times 10^{-15}$	m
PRAD	$r_e(m_e/m_p)$	ERAD*EMASS/PMASS	m

`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 deprecated `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, which results in abnormnal termination of `MAD-X`.

Of course, the following definitions are equivalent:

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

When such a parameter is used in an expression, `MAD-X` 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, while the length of the drift `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`.

2.13.3   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.

2.14   Logical Expressions

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

```
name = expression                      !  equality constraint
name < expression                      !  upper limit
name > expression                      !  lower limit
name < expression, name > expression !  both upper and lower limit
                                       !  for the same name
```

2.15 Variable Names

A variable name can have one of the formats:

```
parameter-name
element-name->attribute-name
command-name->attribute-name
beam%sequence-name->attribute-name
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`.

The fourth format is specific to beams belonging to a particular sequence (for details see sequences and beams).

The fifth format allows extraction of variables from existing tables, as specified in table access.

2.16 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).

`"\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.

2.17   Relations between Variable Parameters

A relation is established between variables by one of two statements

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

The first form evaluates the expression on the right immediately and assigns its value to the parameter. It is an immediate assignment.

The second form assigns the value by evaluating the expression on the right every time the parameter is actually used. It is a deferred assignment.

This mechanism holds as well for element parameters that can be defined with either immediate or deferred assignments.

Attention! 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 at the time of declaration, and will not change when ak1 later changes.

Parameters that have not yet been defined at time of evaluation have a zero value.

Example:

```
gev = 100;
BEAM, ENERGY=gev;
```

The parameter on the left may appear on the right as well in the computer science form of assignments:

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

increases the value of x by 1.

Successive definitions are allowed in the first form of relations or immediate assignments:

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

But circular definitions in the second form of relations, or deferred assignments, are forbidden:

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

results in an error.

Chapter 3. Program Flow Statements

3.1 IF...ELSEIF...ELSE

```
IF (logical_expression) { statements; }

ELSEIF (logical_expression) { statements; }

ELSE { statements; }
```

where "logical_expression" is one of

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

and `oper` is 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` constructs are only possible (in any number) behind an `IF`, or another `ELSEIF`; the branch is executed if "logical_expression" is true, and if none of the preceding `IF` or `ELSEIF` logical conditions was true.

`ELSE` construct is only possible once behind an `IF`, or an `ELSEIF`; the branch is executed if "logical_expression" is true, and if none of the preceding `IF` or `ELSEIF` logical conditions was true.

Warning:
Because `IF ...  ELSEIF ...  ELSE` constructs are a `MAD-X` special feature and not part of a full language, `MAD-X` does not deal gracefully with other special constructs such as `MACRO` or `LINE` when they are placed inside `IF ...  ELSEIF ...  ELSE` statements: this can lead to silent and/or catastrophic errors and is due to the fact that `MACRO` and `LINE` constructs contain, either explicitly or implicitly, a closing curly bracket that unbalances the `IF ... ELSEIF ...  ELSE` statements.

However it is possible to nest `IF ...  ELSEIF ...  ELSE` constructs to at least six levels deep.

3.2 WHILE

```
WHILE (logical_condition) { statements; }
```

executes the statements in curly brackets while the logical_expression is true.

Warning:
Because `WHILE` constructs are a `MAD-X` special feature and not part of a full language, `MAD-X` does not deal gracefully with other special constructs such as `MACRO` or `LINE` when they are placed inside `WHILE` statement blocks: this can lead to silent and/or catastrophic errors and is due to the fact that `MACRO` and `LINE` constructs contain, either explicitly or implicitly, a closing curly bracket that unbalances the `WHILE` statements.

However it is possible to nest `WHILE` statements to at least six levels deep.

Example giving the value of the first ten factorials:

```
n = 1; m = 1;
while (n <= 10) {
    m = m * n; value, m;
    n = n + 1;
};
```

3.3 MACRO

The MACRO construct allows the execution of a group of statements via a single command. Optionally the MACRO construct takes arguments.

```
label:  MACRO = { statements; };

label(arg1, ...,argn):  MACRO = { statements; };
```

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

```
EXEC, label;
```

that executes the statements defined between curly brackets exactly once. The `EXEC` command can then 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:

```
simple(xx,yy):  MACRO = { xx = yy*yy + xx; VALUE, xx;};
a = 3; b = 5;
EXEC, simple(a,b);
```

yields

```
a = 28 ;
```

Passing arguments

In the following example we use the fact that 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);
};
```

Because `MACRO` statements are a `MAD-X` construct and not part of a full language, `MAD-X` allows only one level of inclusion of another `IF ...  ELSEIF ...  ELSE`, `WHILE` or `MACRO` statements.

Macros cannot be called with number arguments but always with string arguments. In case numerical values should be passed to a `MACRO` in an `EXEC` statement, one can conveniently use variables names:

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

instead of

```
EXEC, thismacro($99, $129); !  fails...
```

Chapter 4. General Control Statements

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

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, *etc.*

4.1 EXIT, QUIT, STOP

Any of these three commands ends the execution of `MAD-X`:

```
EXIT;
```

```
QUIT;
```

```
STOP;
```

4.2 HELP

The `HELP` command prints all parameters, and their defaults values, for the statement given; this includes basic element types.

```
HELP, statement_name;
```

4.3 SHOW

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

```
SHOW, command;
```

4.4 VALUE

The `VALUE` command evaluates the current value of all listed expressions, constants or variables, and prints the result in the form of `MAD-X` statements on the assigned output file.

```
VALUE, expression{, expression} ;
```

Example:

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

results in

```
a = 299792.458 ;
pmass = 0.938272046 ;
exp(sqrt(2)) = 4.113250379 ;
```

4.5   OPTION

The `OPTION` commands sets the logical value of a number of flags that control the behavior
of `MAD-X`.

```
OPTION, flag=logical;
```

Because all attributes of `OPTION` are logical flags, the following two statements are identical:

```
OPTION, flag = true;
OPTION, flag;
```

And the following two statements are also identical:

```
OPTION, flag = false;
OPTION, -flag;
```

Several flags can be set in a single `OPTION` command, e.g.

```
OPTION, ECHO, WARN=true, -INFO, VERBOSE=false;
```

The available flags, their default values and their effect on `MAD-X` when they are set to `TRUE`
are listed in table 4.1. Note that to obtain the proper physics in `TRACK` with `BBORBIT` set to
false, one must enable the search for the closed orbit (i.e. not use `ONEPASS`).

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.

4.6   EXEC

Each statement may be preceded by a label, when parsed and executed the statement is then
also stored and can be executed again with

```
EXEC, label;
```

This mechanism can be invoked any number of times, and the executed statement may be
calling another `EXEC`, etc.

```
tw:  TWISS, FILE, SAVE; !  first execution of TWISS
...
EXEC, tw; !  second execution of the same TWISS command
```

Note however, that the main usage of this `MAD-X` construct is the execution of a `MACRO`.

Table 4.1: Flags available to `OPTION` command

FLAG	default	effect if TRUE
ECHO	true	echoes the input on the standard output file
WARN	true	issues warning statements
INFO	true	issues information statements
DEBUG	false	issues debugging information
ECHOMACRO	false	issues macro expansion printout for debugging
VERBOSE	false	issues additional printout in makethin
TRACE	false	prints the system time after each command
VERIFY	false	issues a warning if an undefined variable is used
TELL	false	prints the current value of all options
RESET	false	resets all options to their defaults
MISSING_FILE_FATAL	false	missing a file triggers a fatal fault. This flag ONLY impacts the INTER$\land$
NO_FATAL_STOP	false	Prevents madx from stopping in case of a fatal error **Use at your own risk !**
KEEP_EXP_MOVE	true	keeps the expression of the location after a move command
RBARC	true	converts the RBEND straight length into the arc length
THIN_FOC	true	enables the $1/\rho^2$ focusing of thin dipoles
BBORBIT	false	the closed orbit is modified by beam-beam kicks
SYMPL	true	all element matrices are symplectified in Twiss
TWISS_PRINT	true	controls whether the twiss command produces output
THREADER	false	enables the threader for closed orbit finding in Twiss (see Twiss module)

4.7 SET

The `SET` command is used in two forms:

```
SET, FORMAT=string {, string} ;
SET, SEQUENCE=string;
```

The first form of the `SET` command defines the formats for the output precision that `MAD-X` uses with the `SAVE`, `SHOW`, `VALUE` and `TABLE` commands. The formats can be given in any order and stay valid until replaced.

The formats follow the C convention and must be included in double quotes. The allowed formats are
*n*d for integers with a field-width $= n$,
*n.m*f or *n.m*g or *n.m*e for floats with field-width $= n$ and precision $= m$,
*n*s for strings with a field-width $= n$.
The default is "right adjusted", a "-" changes it to "left adjusted".

Example:

```
SET, FORMAT="12d", "-18.5e", "25s";
```

The default formats are `"10d", "18.10g"` and `"-18s"`.

Example:

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

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

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

sets all formats to automatic adjustment according to C conventions.

The second form of the `SET` command allows to select the current sequence without the `USE` command, which would bring back to a bare lattice without errors. The command only works if the chosen sequence has been previously activated with a `USE` command, otherwise a warning is issued and `MAD-X` continues 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`.

4.8 SYSTEM

```
SYSTEM, "string";
```

transfers the quoted `string` to the operating system for execution. The quotes are stripped and no check of the return status is performed by `MAD-X`.

Example:

```
SYSTEM, "ln -s /afs/cern.ch/user/j/joe/input shortname";
```

makes a local link to an AFS directory with the name `shortname` on a `UNIX` system.

Attention: Although this command is very convenient, it is clearly not portable across systems and it should probably be avoided if one intends to share `MAD-X` scripts with collaborators working on other platforms.

4.9 TITLE

```
TITLE, "string";
```

defines a `string` that is inserted as title in various table outputs and plot results.

4.10 USE

`MAD-X` operates on beamlines that must be loaded and expanded in memory before other commands can be invoked. The `USE` command allows this loading and expansion.

```
USE, SEQUENCE=sequence_name, PERIOD=sequence_name,
     RANGE=range,
     SURVEY=logical;
```

The attributes to the `USE` command are:

SEQUENCE   name of the sequence to be loaded and expanded.

PERIOD   name of the sequence to be loaded and expanded.
              `PERIOD` is an alias to `SEQUENCE` that was kept for backwards compatibility with
              `MAD-8` and only one of them should be specified in a `USE` statement.

RANGE   specifies a range. restriction so that only the specified part of the named
              sequence is loaded and expanded.

SURVEY   option to plug the survey data into the sequence elements nodes on the first
              pass (see `SURVEY`).

Note that reloading a sequence with the `USE` command reloads a bare sequence and that any
`ERROR` or orbit correction previously assigned or associated to the sequence are discarded. A
mechanism to select a sequence without this side effect of the `USE` command is provided with
the `SET, SEQUENCE=...` command.

4.11 SELECT

Some `MAD-X` commands can perform specific operations on selected elements or ranges of
elements and can produce specific output for selected elements or ranges of elements.

The selection is made through the `SELECT` command and applies to subsequent commands.

```
SELECT, FLAG=string, RANGE=string, CLASS=string, PATTERN=string,
        SEQUENCE=string, FULL=logical, CLEAR=logical,
        COLUMN=string{,string}, SLICE=integer, THICK=logical,
        STEP=real, AT={real, ...};
```

The attributes to the `SELECT` command are:

FLAG   determines the applicability of the `SELECT` statement and the attribute value
              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` command that converts
                            the sequence into one with thin elements.

              SECTORMAP   selection of elements for the `SECTORMAP` output file from the
                            `TWISS` module.

              SAVE   selection of elements for the `SAVE` command.

              INTERPOLATE   selection of interpolation points for the `TWISS` command.

              tablename   is a table name such as `twiss`, `track` etc., and the rows and
                            columns to be written are selected.

RANGE   the range of elements to be selected as defined in section 12.1 on range selection.

CLASS   the class of elements to be selected as defined in section 12.2 on class selection.

PATTERN   the regular expression pattern for the element names to be selected as defined
              in section 2.16 on selection via regular expressions.

SEQUENCE   the name of a sequence to which the selection is applied.

FULL   a logical falg to select ALL positions in the sequence for the named flag.
              For the flag TWISS, this attribute includes all standard columns for a TWISS
              table, and therefore the following two statements are equivalent:

                    SELECT, FLAG=twiss, COLUMN= name, s, betx, ..., var1;
                    SELECT, FLAG=twiss, FULL, COLUMN= var1;

              FULL=true is the default for the MAKETHIN flag and for tables: *e.g.* SELECT,
              FLAG=makethin; is equivalent to SELECT, FLAG=makethin, FULL;

CLEAR   deselects ALL positions in the sequence for the flag "name".

COLUMN   is only valid for tables and takes as attribute value a list of columns to be
              written into the TFS file. The special "_name" argument refers to the actual
              name of the element.

SLICE   is the number of slices into which the selected elements have to be cut and is
              only used by MAKETHIN and FLAG=INTERPOLATE. (Default = 1).

THICK   is a logical flag to indicate whether the selected elements are treated as thick
              elements by the MAKETHIN command.
              This only applies up to now to QUADRUPOLEs and BENDs for which thick maps
              have been explicitly derived.

STEP   output intermediate values every STEP meters in the TWISS command.

AT   manual specification of interpolation points for the TWISS command. Specified
              as a fraction of the node length, i.e. a value of 0.5 slices at the centre of the
              element.

Composition of SELECT statements:
The selection criteria provided on a single SELECT statement are logically ANDed, *i.e.* selected
elements have to fulfill **all** provided criteria in the single SELECT statement.

The selection criteria on different SELECT statements are logically ORed, *i.e.* selected elements
have to fulfill **any** of the selection criteria provided by the different SELECT statements.

All selections for a given flag remain valid until a SELECT statement with the CLEAR argument
is specified for the same flag.

Note that because of these composition rules, it is considered good practice to start by clearing
the selection for a given flag before making a new selection, eg:

    SELECT, FLAG=twiss, CLEAR;
    SELECT, FLAG=twiss, CLASS=MQ;
    SELECT, FLAG=twiss, RANGE=MQ[5]/MQ[7];
    ...

Examples:

```
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], as well as all elements (in the whole sequence) with name starting with "mqw", for treatment by the ERROR module.

```
SELECT, FLAG=SAVE, CLASS=variable, PATTERN="abc.*";
SAVE, FILE=mysave;
```

saves all variables containing "abc" in their name, but does not save elements with names containing "abc" since the class "variable" does not exist.

```
sig1 := sqrt(beam->ex*table(twiss,betx));
SELECT, FLAG=twiss, COLUMN= _name, s, betx, ..., sig1; !  or equivalently
SELECT, FLAG=twiss, FULL, COLUMN= sig1; !  default columns + new
```

writes the current value of "sig1" into the TWISS table each time a new line is added; Note that the values from the same (current) line can be are accessed by the variable "sig1". The PLOT command also accepts the new variable in the table.

4.12 Add2expr

The ADD2EXPR command is used to add a variable to an already defined deferred expression.

```
ADD2EXPR, var=string, expr=string
```

An example of how it is used is given below.

```
a := b + c; add2expr, var=a, expr=-d; show, a; a := b + c - d;
```

Chapter 5. File Handling Statements

Note that the filenames given as attribute values in File Handling statements must be explicit names and should not contain wildcard characters since the filename strings are not passed to the underlying Operating System for evaluation.

5.1   ASSIGN

```
ASSIGN, ECHO="filename", TRUNCATE;
```

where `filename` is either the name of an output file, or the string `terminal`. This allows switching the echo stream to a file or back to the terminal, but only for the commands VALUE, SHOW, and PRINT. Allows easy composition of future `MAD-X` input files.

`TRUNCATE` specifies whether the file is truncated when opened (ignored for terminal).

5.2   CALL

```
CALL, FILE="filename";
```

where `filename` is the name of an input file. The named file is then read until a `RETURN` statement is encountered, or until the End_Of_File; The file being "called" may in turn contain any number of `CALL` statements itself, and so on to any depth.

5.3   CHDIR

```
CHDIR, DIR="path/to/folder";
```

changes the current working directory to the specified folder. Both relative and absolute pathes are possible.

5.4   RETURN

```
RETURN;
```

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

5.5   PRINT

```
PRINT, TEXT="string";
```

prints the quoted text `string` to the current output file (see ASSIGN above). The text can be edited with the help of a macro statement.

5.6 PRINTF

```
PRINTF, TEXT="string", VALUE= expr,expr;
```

prints the numerical values specified in the `VALUE` field to the current output file, with formatting according to the format string provided in the `TEXT` field.

The string format can take numeric C or `MAD-X` format specifiers for double real values. Integer and string formats are not supported but can be approximated with the `%g` format in the case of integers, and via `MACRO` statements, which perform string substitution, themselves containing a `PRINT` statement.

The maximum number of values that can be printed in one statement is limited to 100.

If the number of format specifiers given in the string is higher than the number of values in the value field, undefined values are printed where they are not explicitly provided.

If the number of format specifiers given in the string is lower than the number of values in the value field, the values that do not correspond to a format specifier are ignored.

Example:

```
a = 1.2; b = 3.4/0.3; c := 0.8*a/b;
PRINTF,TEXT="String with floats a=%f, b=%.3g, text and MAD float c=%F;",
       VALUE = a,b,c;
PRINTF,TEXT="More specifiers than values:  %f, %.3g, %F",  VALUE = a,b;
PRINTF,TEXT="More values than specifiers:  %f, %.3g",      VALUE = a,b,c;
```

produces the following output:

```
String with floats a=1.200000, b=11.3, text and MAD float c=  0.08470588235;
More specifiers than values: 1.200000, 11.3,   6.953222976e-310
More values than specifiers: 1.200000, 11.3
```

Note that `PRINTF`, like `PRINT`, produces output that cannot be read back by `MAD-X`. For output that can be read back by `MAD-X`, use the command `VALUE` or TFS tables.

Note also that a percent sign (%) can be printed using the format `text="%%"`.

5.7 RENAMEFILE

```
RENAMEFILE, FILE="filename", TO="new_filename";
```

renames the file `filename` to `new_filename` on disk.
This command is more portable than an equivalent `SYSTEM` call:

```
SYSTEM("mv filename new_filename"); !  Unix specific
```

5.8 COPYFILE

```
COPYFILE, FILE="filename", TO="new_filename", APPEND=logical;
```

copies the file `filename` to the file `new_filename` on disk.

The attribute `APPEND=true` causes `COPYFILE` to append the content of `filename` at the end of the file `new_filename`.
The default value `APPEND=false` causes the replacement of the content of `new_filename` with the content of `filename`.

`COPYFILE, APPEND=true...` is more portable than an equivalent SYSTEM call:

```
SYSTEM("copy /y filename new_filename"); !  Windows specific
```

5.9   REMOVEFILE

```
REMOVEFILE, FILE="filename";
```

removes the file `filename` from disk.
It is more portable than an equivalent SYSTEM call:

```
SYSTEM("rm filename"); !  Unix specific
```

Chapter 6. Table Handling Statements

6.1 CREATE

```
CREATE, TABLE=tabname, COLUMN= var{, var} {, _name} ;
```

creates a table with the specified variables as columns. The table created is initially empty and can be subsequently filled, and eventually written to file in TFS format.

The special variable name attribute _name (name preceded by underscore) adds the element name to the table at the specified column.

6.2 DELETE

```
DELETE, SEQUENCE=seqname, TABLE=tabname;
```

deletes a sequence with name `seqname` or a table with name `tabname` from memory. The sequence deletion is done without influence on other sequences that may have elements that werein common with the deleted sequence.

6.3 READTABLE

```
READTABLE, FILE="filename", TABLE=tabname;
```

reads the TFS file `filename` containing a MAD-X table loads the table into memory with the name `tabname`. If the table name is not specified, use the name specified in the information section of the TFS file. The table can then be manipulated as any other table, *i.e.* its values can be accessed, its data can be plotted or changed, and it can be written out again.

6.4 READMYTABLE

```
READMYTABLE, FILE="filename", TABLE=tabname;
```

deprecated alias for `READTABLE`.

6.5 WRITE

```
WRITE, TABLE=tabname, FILE="filename";
```

writes the table "tabname" onto the file "filename"; only the rows and columns of a preceding `SELECT, FLAG=table,...;` are written. If no `SELECT` has been issued for this table, only the header is written to file. If the FILE argument is omitted, the table is written to standard output.

6.6   SETVARS

The `SETVARS` command sets the variables with values extracted from the row of a table.

```
SETVARS, TABLE=tabname, ROW=integer;
```

The attributes of `SETVARS` are:

TABLE   the name of the table. (Default: none)

ROW   the row number containing the values. (Default: -1)

Negative `ROW` values are allowed and count the row numbers from the last row, allowing access to the table in reverse order of rows: `ROW = -1` accesses the last row of the table, `ROW = -2` accesses the penultimate (one before last) row, etc. . .

Trying to access the table forward beyond the last row, i.e. `ROW` strictly greater than `nrow` the number of rows in the table, or trying to access the table backwards before the first row, i.e. `ROW` strictly lower than `-nrow`, or trying to access the illegal `ROW=0`, all result in a "row out of bound" message and no variable values are returned and set.

6.7   SETVARS_LIN

The `SETVARS_LIN` command sets the variables with values calculated by linear interpolation, or extrapolation, between two rows of a table.

```
SETVARS_LIN, TABLE=tabname,
             ROW1=integer, ROW2=integer, PARAM=string;
```

The attributes of `SETVARS_LIN` are:

TABLE   the name of the table. (Default: none)

ROW1   a first row number with values for interpolation. (Default: 0)

ROW2   a second row number with values for interpolation. (Default: 0)

PARAM   a string containing the linear interpolation factor or the name of a variable or
               expression containing the interpolation factor. If the resulting value of `PARAM`
               is outside the $[0, 1]$ interval, the result is a linear extrapolation.
               (Default: "interp", itself defaulting to a value of 0.0 when evaluated)

`SETVARS_LIN` sets the variables with values calculated through the following formula that `MAD-X` constructs internally as a deferred expression which is immediately evaluated:

```
value := value(row1)*(1-param) + value(row2)*param;
```

Both the expression and the value of the expression are available to the user through respectively the commands `SHOW` and `VALUE`.

When the values are represented as strings, *e.g.* the name or keyword of elements, the resulting value is the string in `ROW1`.

Negative `ROW`*i* values are allowed and count the row numbers from the last row, allowing access to the table in reverse order of rows: `ROW`*i* `= -1` accesses the last row of the table, `ROW`*i* `= -2` accesses the penultimate (one before last) row, etc. . .

Trying to access the table forward beyond the last row, i.e. `ROW`*i* strictly greater than `nrow` the number of rows in the table, or trying to access the table backwards before the first row, i.e. `ROW`*i* strictly lower than `-nrow`, or trying to access the illegal `ROW`*i*`=0`, all result in a "row out of bound" message and the expression is not constructed or evaluated.

Example:

```
!  extracts the position of the centre of each element from a standard
!  TWISS table giving positions at end of elements:
len = table(twiss,tablelength);
interpolate = 0.5;
i = 2;
WHILE (i < len) {
      SETVARS_LIN, TABLE=twiss, ROW1=i-1, ROW2=i, PARAM=interpolate;
      !  now variables are interpolated at the center of the elements.
      !  in particular S holds the position of the center of the element.
      SHOW, s; VALUE, s;
      ...
      i = i + 1; };
```

6.8 SETVARS_KNOB

The `SETVARS_KNOB` command creates or manipulates the expression for the variables by adding a term `"+ val * knob"` where val is extracted from the row of a table and knob from the command.

> `SETVARS_KNOB, TABLE=tabname, ROW=integer, KNOB=knobname;`

The attributes of `SETVARS` are:

TABLE   the name of the table. (Default: none)

ROW   the row number containing the values. (Default: -1)

KNOB   the name of the knob. (Default: none)

NOAPPEND   when true the term is not appended but replace the exisint value or expression.

Negative `ROW` values are allowed and count the row numbers from the last row, allowing access to the table in reverse order of rows: `ROW = -1` accesses the last row of the table, `ROW = -2` accesses the penultimate (one before last) row, etc. . .

6.9 SETVARS_CONST

The `SETVARS_CONST` sets the value for the variables to constant.

> `SETVARS_CONST, TABLE=tabname, CONST=value;`

The attributes of `SETVARS` are:

TABLE   the name of the table. (Default: none)

CONST   the value used to set the variables. (Default: 0)

6.10   FILL

The `FILL` command fills a row of a table with the current values of all declared column variables of the table.

> FILL, TABLE=tabname, ROW=integer;

The `FILL` command takes two arguments:

TABLE   is the name of the table to be filled. The table must have been created before-
               hand. The table can then be written to file in `TFS` format.

ROW   is the row number to be filled with the current values of all column variables.
               `ROW=0`, or `ROW=nrow + 1`, where `nrow` is the current number of rows in the
               table, causes `FILL` to add a row at the end of the table and fill it with the
               current values of all column variables.
               (Default: 0)

Negative `ROW` values are allowed and count the row numbers from the last row, allowing access to the table in reverse order of rows: `ROW = -1` accesses the last row of the table, `ROW = -2` accesses the penultimate (one before last) row, etc. . .

Trying to access the table forward beyond the last row, i.e. `ROW` strictly greater than `nrow + 1`, where `nrow` is the number of rows in the table, or trying to access the table backwards before the first row, i.e. `ROW` strictly lower than `-nrow`, both result in a "row out of bound" message and no values are filled in the table.

Reminder: One can get access to the current number of rows in a table using the variable

    TABLE(tablenanme, TABLELENGTH)

6.11   FILL_KNOB

The `FILL_KNOB` command fills a row of a table with the variation (multiplied by scaling factor) of all declared column variables of the table when the knob value change by 1.

> FILL_KNOB, TABLE=tabname, ROW=integer, KNOB=knobname, SCALE=scaling;

The `FILL` command takes two arguments:

TABLE   is the name of the table to be filled. The table must have been created before-
               hand.

ROW   is the row number to be filled with the current values of all column variables.
               `ROW=0`, or `ROW=nrow + 1`, where `nrow` is the current number of rows in the
               table, causes `FILL` to add a row at the end of the table and fill it with the

current values of all column variables.
(Default: 0)

KNOB   the name of the variable that is varied to calculate the values to fill a table.

SCALE   scaling factor applied to all the variation calculated by varying the knob.

6.12 SHRINK

The `SHRINK` command removes a number of rows at the end of a table.

```
SHRINK, TABLE=tabname, ROW=integer;
```

The `SHRINK` command takes two arguments:

TABLE   is the name of the table from which rows should be removed. The table must
            have been previously created and filled or read from file with `READTABLE`.

ROW   is the number of the last row to be kept in the table. All rows beyond the
            given row number are removed.
            Negative values are allowed and count the row numbers from the last row,
            allowing access to the table in reverse order of rows: `ROW = -1` removes the
            last row of the table, `ROW = -2` removes the last two rows of the table, etc...
            (Default: -1)

Trying to access the table forward beyond the last row, i.e. `ROW` strictly greater than `nrow`,
where `nrow` is the number of rows in the table, or trying to access the table backwards before
the first row, *i.e.* `ROW` strictly lower than `-nrow`, both result in a "row out of bound" message
and no values are filled in the table.

Chapter 7. Beam Handling Statements

Many commands in `MAD-X` require the prior 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.

7.1 BEAM

The quantities are entered by a `BEAM` command:

```
BEAM, PARTICLE=string, MASS=real, CHARGE=real,
      ENERGY=real, PC=real, GAMMA=real, BETA=real, BRHO=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=string;
```

The attributes of the `BEAM` command are:

PARTICLE   The name of particles in the beam. Default=POSITRON
            `MAD-X` 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_μ, CHARGE=1)

  NEGMUON   The particles are negative muons (MASS=m_μ, CHARGE=-1)

  ION   The particles are simple generic ions (MASS=u, CHARGE=1)

MASS   the restmass of the particles in the beam in GeV.
            (Default=$m_e \approx 0.511 \ 10^{-3}$ GeV).
            Note that a zero mass particle is not allowed in `MAD-X`.

CHARGE   the electrical charge of the particles in the beam in units of q_p, the proton charge. (Default=1)
            Note that a zero charge particle is not allowed in `MAD-X`.

The order of precedence for arguments is: `particle->(mass+charge)`
If the particle name given is recognized in the list above, the restmass and charge are set directly by `MAD-X`, and the `MASS` and `CHARGE` arguments provided in the `BEAM` command are simply ignored. For other particles, and in particular for ions, any combination of name, mass and charge can be entered independently.

ENERGY	Total energy per particle in GeV.
	If given, it must be greater than the particle restmass. (Default=1 GeV)
PC	Particle momentum times the speed of light, in GeV.
	If given, it must be greater than zero.
GAMMA	Relativistic factor, *i.e.* ratio between total energy and rest energy of the particles: `GAMMA = ENERGY/MASS` = $E/m_0 c^2$.
	`GAMMA` must be greater than one.
BETA	Ratio between the speed of the particle and the speed of light: `BETA`= v/c.
	`BETA` must be strictly less than one.
BRHO	Magnetic rigidity of the particles in T.m.
	`BRHO`= $P/abs(q)$ = `PC / ( abs(CHARGE) * c * 1.e-9)`.

The order of precedence for arguments is: `energy->pc->gamma->beta->brho`

Note that if the restmass is changed after the energy has been set, ie in separate `BEAM` commands, the energy is left unchanged and the momentum `PC` and relativistic factor `GAMMA` are recalculated.

EX	The horizontal emittance ϵ_x (default: 1 m).
EY	The vertical emittance ϵ_y (default: 1 m).
ET	The longitudinal emittance ϵ_t (default: 1 m).
EXN	The normalised horizontal emittance [m]: $\epsilon_{xn} = \sqrt{\gamma^2 - 1}\ \epsilon_x = \beta\gamma\ \epsilon_x$
EYN	The normalised vertical emittance [m]: $\epsilon_{yn} = \sqrt{\gamma^2 - 1}\ \epsilon_y = \beta\gamma\ \epsilon_y$
SIGT	The bunch length $c\ \sigma_t$ in [m].
SIGE	The *relative* energy spread σ_E/E in [1].

The order of precedence for arguments is: `ex->exn, ey->eyn`.

The order of precedens for the logitudinal emittance is in the following order `ET->(SIGT and SIGE), (SIGT and SIGE)->ET, (SIGT)->(ET and SIGE, (SIGE)->(ET and SIGT`. This means that if you define a `ET` this values will be used to override any value that you might have set for `SIGT and SIGE`.

Note that up to version 5.02.04 the definition of normalised emittance used in `MAD-X` was referring to the so-called 2-sigma geometric emittance: $\epsilon_n = 4\sqrt{\gamma^2 - 1}\ \epsilon = 4\beta\gamma\ \epsilon$ This definition was different from the definition usually found in literature and used for example in the APERTURE module.
The standard one sigma definition is now used across all `MAD-X` modules.

Certain commands compute the synchrotron tune Q_s taking into account the settings of RF cavities. If Q_s is non-zero, the relative energy spread and the bunch length are calculated

with

$$\sigma_E/p_0 c = \sqrt{\epsilon_t \; \frac{2\pi Q_s}{\eta \; C}} \tag{7.1}$$

$$c \; \sigma_t = \sqrt{\epsilon_t \; \frac{\eta \; C}{2\pi Q_s}} \tag{7.2}$$

where C is the machine circumference, and

$$\eta = 1/\gamma^2 - 1/\gamma_t^2 \tag{7.3}$$

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 dipole 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 the section below on bv flag.

SEQUENCE   attaches the defined beam to the named sequence; if the name is omitted, the BEAM command refers to the default beam which is always present. Sequences without attached beam use this default beam. When updating a beam with a corresponding sequence name, tye sequence name must always be mentioned.

Order and Precedence:
Internally the BEAM command processes the parameters in the following order and with the following precedence (left to right):

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

Warning: BEAM updates, i.e. it replaces attributes explicitly mentioned, may calculate other attributes according to the precedence rules given, but does NOT return attributes not specified to default values! In order to reset to reset BEAM attributes to their default values, use the RESBEAM command.

Additional variables:
Some MAD-X modules may also compute and store data into a beam data block. These attributes may NOT be set directly through the BEAM command. The corresponding variables are:

CIRC   total length or circumference of the machine [m].

FREQ0	revolution frequency [MHz].
DTBYDS	phase slip factor.
DELTAP	momentum deviation.
ALFA	momentum compaction factor.
U0	radiation loss per turn [GeV].
QS	synchrotron tune [1].
ARAD	classical particle radius [m].
PDAMP	damping partition numbers; Default is 1,1,2.
N1MIN	minimum available aperture, set by the APERTURE module.

7.2 RESBEAM

The `RESBEAM` command resets the default values of the beam belonging to the specified sequence, or of the default beam if no sequence is given.

```
RESBEAM, SEQUENCE=string;
```

The only argument to `RESBEAM` is a string for the sequence name. If the sequence name is omitted, the default beam is reset.

Table 7.1: Default Beam Data

Attribute	Value	Unit
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	

7.3 Referring to BEAM data attributes

Expressions may refer to data in the beam data block using the notation

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

or

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

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

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

but also for storing values in the beam bank, e.g.

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

Note however that this does NOT trigger an update of dependent variables and you are strongly advised against setting `BEAM` parameters with this method.

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

```
SHOW, BEAM;
```

or to obtain the data for a beam linked to a named sequence:

```
SHOW, BEAM%sequence-name;
```

Example:

```
! select electrons, set energy and emittances
BEAM, PARTICLE=ELECTRON, ENERGY=50, EX=1.E-6, EY=1.E-8, SIGE=1.E-3;
...
! turn on synchrotron radiation
BEAM, RADIATE;
...
! reset all values to their defaults,
! ie positrons, energy = 1GeV, default emittances, no radiation...
RESBEAM;
...
! select new emittances
BEAM, EX=2.E-5, EY=3.E-7, SIGE=4.E-3;
```

7.4   BV FLAG

When reversing the direction $(\vec{V})$ of a particle in a magnetic field $(\vec{B})$ while keeping its charge constant, the resulting force $\vec{V} \times \vec{B}$ changes sign. This is equivalent to flipping the field, but not the direction. Depending on the symmetry of the magnetic field upon 180° rotation around the vertical axis, this can result in a sign change of the field's gradient and hence the sign of its field strength. E.g. Normal Quadrupoles, Skew Sextupoles, Normal Octupoles etc. change sign, while Skew Quadrupoles, Normal Sextupoles, Skew Octupoles etc. are invariant.

Please note: In the current implementation, this sign change is visible in the TWISS tables, yet not in the ERROR tables!

Usage of the BV-Flag in the LHC:
For practical reasons the properties of all elements of the default LHC sequence file are defined in the `MAD-X` input as if they apply to a clockwise proton beam. This allows a single definition

for elements traversed by both beams. For particles travelling in this direction ("LHC Beam 1"), the BV-flag needs to be positive. Effects on a beam with identical particle charge but running in the opposite direction ("LHC Beam 2") must be reversed by a negative BV-flag, which is then taken into account in calculations.

The `BEAM` commands look therefore as follows:

```
BEAM, SEQUENCE=lhcb1, PARTICLE=proton, PC=450, BV = +1;

BEAM, SEQUENCE=lhcb2, PARTICLE=proton, PC=450, BV = -1;
```

For "LHC Beam 4", which mirrors "LHC Beam 2" and can be found in a separate sequence file, the lattice itself is defined in counter-clockwise direction and the BV-flag is therefore positive:

```
BEAM, SEQUENCE=lhcb2, PARTICLE=proton, PC=450, BV = +1;
```

Please note: With these definitions and the difference between TWISS and ERROR tables mentioned above, the following relations need to be taken into consideration for the LHC implementation.

Always:

- LHCB2 and LHCB4 have opposite (D)X-Orbit in TWISS and ERROR.

For upon mirroring on the vertical axis anti-symmetric common-aperture magnets (K0SL, K1L, K2SL, K3L, K4SL, K5L ...):

- LHCB4 circuits are implemented with "-" in the sequence, so that the value of the circuit itself can be identical for all beams.

- LHCB2 and LHCB4 have the **same** sign KNL/KNSL-Values in TWISS, but the **opposite** in ERROR.

- LHCB1 and LHCB2 have the **opposite** sign KNL/KNSL-Values in TWISS, but the **same** in ERROR.

For the other common magnets all beams have the same signs.

Chapter 8. Sequence Editor

With the help of a few commands, an existing sequence may be modified in many ways: in the case of a circular machine, the starting point of the sequence 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. This is particularly useful for the SAVE command to properly save *shared sequences* and to write them out in MAD-8 format.

8.1   SEQEDIT

MAD-X provides an environment for the edition of sequences that is invoked with the command:

```
SEQEDIT, SEQUENCE=string;
```

The only attribute is the name of the sequence to be edited.

The editing is performed on the sequence as provided by the user and before it is expanded with the USE command. At the end of sequence edition, the resulting sequence must be expanded through the USE command as necessary.

8.2   FLATTEN

Sequences can be built from elements but also sub-sequences resulting in a nested structure (see chapter 14 on sequence definition). The positioning of elements within a sequence can also be specified with values or expressions, and by reference to other elements.

MAD-X provides a command to resolve these dependencies and transform a complex sequence into a simple list of elements with positioning values referring to the start of the sequence, discarding the user-specified expressions for the positioning.

This command takes no argument:

```
FLATTEN;
```

If the sequence being edited contains sub-sequences, FLATTEN recursively includes all sub-sequences until the sequence is only composed of a simple list of elements.

FLATTEN also resolves the positioning of each element within the sequence to a single value with reference to the start of the sequence and updates the AT attribute of that element while also discarding the user-specified expression if present.

The FLATTEN command is recommended at the beginning of sequence edition as well as at the very end as in

```
SEQEDIT, SEQUENCE=name;
        FLATTEN;
        ...commands to edit the named sequence...
        FLATTEN;
ENDEDIT;
```

8.3 CYCLE

```
CYCLE, START=string;
```

This makes the sequence start at the location given as attribute value of the `START` attribute. The element named by the `START` attribute must be a marker.

In case there is a shared sequence in the used sequence, the command `FLATTEN` should be used before the command `CYCLE`.

Example:

```
FLATTEN;
CYCLE, START=place;
```

Note that the `FLATTEN` command inserts another marker before the start location, with a name composed of the name of the sequence being edited, followed by the start location name and the string "_P_".

8.4 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;
```

8.5 INSTALL

```
INSTALL, ELEMENT=string, CLASS=string,
         AT=real, FROM={string|SELECTED}, LAST={true|false};
```

where the parameters have the following meaning:

ELEMENT   name of the element to be inserted (mandatory)

CLASS   class of the new element to be inserted.

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 `SELECT`; in the
              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=..;

LAST   if there are one or more elements at the same location the new element will be
              placed after the last element unless LAST is FALSE.

Note: No element definition can occur inside a `SEQEDIT ...  ENDEDIT` block.

8.6 MOVE

> MOVE, ELEMENT={string|SELECTED}, BY=real, TO=real, FROM=string, LAST={true|false};

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,
              the `BY` attribute must be given.

BY   distance by which the element(s) is/are to be moved; no `TO` or `FROM` attributes
              should be given in this case.

TO   position to which the element has to be moved; if no `FROM` attribute is given,
              the position is relative to the start of the sequence; otherwise, it is relative to
              the location given in the `FROM` argument

FROM   place in the sequence with respect to which the element is to be positioned.

LAST   if there are one or more elements at the same location the new element will be
              placed after the last element unless LAST is FALSE.

8.7 REMOVE

> REMOVE, ELEMENT={string|SELECTED};

ELEMENT   name of existing element(s) to be removed.
              The special case `ELEMENT = SELECTED` removes all elements previously selected
              by `SELECT` commands

Note: `MAD-X` expects to find some special markers in a beam line and it is therefore dangerous
to remove all markers from a sequence! In particular the `START=...` marker and markers
added by a `CYCLE` command must never be removed from a sequence.

8.8 REPLACE

> REPLACE, ELEMENT={string|SELECTED}, BY=string;

The parameters are defined as:

ELEMENT   names the elements to be replaced.
                The special case `ELEMENT = SELECTED` replaces all elements previously selected
                by `SELECT` commands

BY   names the elements replacing the elements selected for replacement.

8.9 EXTRACT

A new sequence can be extracted as a subset of an existing sequence. The extracted sequence
is given a new name and can be `USE`d as any user defined sequence.

```
EXTRACT, SEQUENCE=string, FROM=string, TO=string,
        NEWNAME=string;
```

The parameters are defined as:

SEQUENCE   the name of the existing sequence from which the new sequence is extracted.

FROM   the name of an element in the sequence that becomes the first element of the
                extracted sequence.

TO   the name of an element in the sequence that becomes the last element of the
                extracted sequence.

NEWNAME   the name of the extracted sequence.

The extracted sequence is declared as `SHARE`d and can therefore be combined *e.g.* into the
cycled original sequence.

Note: the element given by the `FROM` attribute must be located, in the existing sequence,
before the element given by the `TO` attribute, or `MAD-X` fails with a fatal error. In the case of
circular sequences, this can be ensured by performing a `CYCLE` of the original sequence with
`START` pointing to the same element given in the `FROM` attribute of the `EXTRACT` command.

8.10 ENDEDIT

The sequence editing environment is closed with the command

```
ENDEDIT;
```

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

8.11 SAVE

The `SAVE` command saves a sequence to a specified file together with all relevant information.

```
SAVE, SEQUENCE=string,string, FILE=filename,
      BEAM=logical, BARE=logical, MAD8=logical,
      NOEXPR=logical, NEWNAME=string;
```

The parameters are defined as:

SEQUENCE   lists the sequences to be saved, separated by commas. This attribute is optional
             and when omitted, all known sequences are saved.
             However, because of internal inconsistencies that can result in spurious entries
             in the output file, the user is strongly advised to always provide explicitly the
             names of sequences to be saved.

FILE   the filename of the output file. (Default: "save")

BEAM   an optional flag to specify that all beams linked to the specified sequences are
             saved at the top of the output file.

BARE   an optional flag to save only 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 in order to further edit it with SEQEDIT.

MAD8   an optional flag to request that the sequences should be saved using MAD-8
             input format.

NOEXPR   an optional flag to save values of expressions instead of the expressions them-
             selves: the expressions in commands and variables are expanded and evaluated
             before saving. This option must be used with care because the exported values
             were not deeply checked and the code that writes variables and commands is
             widely spread in the internal structure.
             This option does not apply only for the saving of sequences in MAD-8 format.

NEWNAME   provides a name for the saved sequence, overriding the original name. (see
             EXTRACT above)

CSAVE   an option that saves the strength of the corrector magnets used for an orbit
             correction.

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.

The precision of the output of the SAVE command depends on the defined output precision
for MAD-X, which can be adjusted with the SET, FORMAT... command.

Note that with BARE=false the saved sequence may contain redundant efinitions of elements,
i.e. the same element is defined in the declaration of elements in the form label: type...
and in the sequence itself in the form label: type, at=.... This is flagged by MAD-X as
implicit element redefinition and is ignored but a warning is issued.

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; // only sequence is 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;
```

8.12 DUMPSEQU

```
DUMPSEQU, SEQUENCE=string, LEVEL=integer;
```

This command is actually more of a debug statement, but it may come handy at certain occasions. The argument of the `SEQUENCE` attribute is the name of an already expanded (i.e. `USE`d) sequence. The amount of detail in the output is controlled by the `LEVEL` argument:

$= 0$: 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

Chapter 9. Save state

It is sometimes convientine to save the state of a MAD-X script in order to reload it at a later occation or in order to distribute it to another users. The `SAVE_STATE` command provide the user with the possiblity to save a sequence, the beam information, the errors, and the macros in a single command. A MAD-X file is also created in order to load the produced files. The files are saved in a separate folder and in hexadecimal format in order not to lose any information in the saving.

```
SAVE_STATE, SEQUENCE=string,string, FILE=filenames,
            BEAM=logical, FOLDER=foldername, csave=logical ;
```

The parameters are defined as:

SEQUENCE   lists the sequences to be saved, separated by commas. This attribute is optional and when omitted, all known sequences are saved.
            However, because of internal inconsistencies that can result in spurious entries in the output file, the user is strongly advised to always provide explicitly the names of sequences to be saved.

FILE   the starting of the filenames the output files will have. (Default: ”save”)

BEAM   an optional flag to specify that all beams linked to the specified sequences are saved at the top of the output file. (Default=TRUE)

FOLDER   the name of the folder where the files are saved (Default:”save_state”)

CSAVE   an option that saves the strength of the corrector magnets used for an orbit correction (Default=TRUE).

Part II

Elements, Beamlines and Sequences

Chapter 10. Definition of Elements

10.1   Element Input Format

All physical elements are defined by statements of the form

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

where

label   is a name to be given to the element.

keyword   is an element type keyword.

attribute   normally – with exception for multipoles – takes one of the two forms:

> attribute-name = attribute-value
> attribute-name := attribute-value

> **attribute-name** selects the attribute, as defined for the element type keyword.

> **attribute-value** provides a value to the **attribute_name**. 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.

Example:

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

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 component and a skew decapole component.

10.2   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 = 2, K1 = 0.02;    ! Replace whole definition of QF

QF, K1 = 0.03;                       ! Replace value of K1 for QF
```

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 only affect already defined objects which belong to its element class in case the:

```
option, update_from_parent
```

is used. In case this option is used all elements which belong to that class will be updated. An example is given in element class.

10.3   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 these 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 from their class all attributes that are not explicitly specified at time of declaration. 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-X` 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.

The default option is that if you change the parent of an element you will not change the children of that class. Let's see in the following example how it works:

```
mb:  sbend,l:=lenseq, angle =0.001;
mb2:  mb;
```

This will create an element `mb` that then is used as the parent for `mb2`. If we now do want to give mb an aperture we can do the following:

```
apertype=ellipse, aperture:= 1,4 ;
```

This, however, does not modify `mb2`. In case we want `mb2` to also inherent the change from `mb` we need to use the flag

```
option, update_from_parent
```

. This should be used with care and is only changing the direct parent.

Chapter 11. Element Types

11.1   Marker

```
label:  MARKER;
```

The simplest element which can occur in a beam line is a `MARKER`. It has zero length and 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;
```

11.2   Drift Space

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

A drift space has one real attribute:

L   The drift length (Default: 0 m)

Example:

```
DR1:  DRIFT, L=1.5;
DR2:  DRIFT, L=DR1->L;
```

The length of DR2 is always equal to the length of DR1.

The straight reference system for a drift space is a Cartesian coordinate system.

11.3   Bending Magnet

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

SBEND   is a sector bending magnet.
             The planes of the pole faces intersect at the centre of curvature of the curved sbend reference system.

RBEND   is a specific case of SBEND.
             The pole faces are parallel. The reference system is the curved rbend reference system.

Bendig magnets are defined by the statements:

```
label:   SBEND, L=real, ANGLE=real, TILT=real,
                K0=real, K1=real, K2=real, K1S=real,
                E1=real, E2=real, FINT=real, FINTX=real,
                HGAP=real, H1=real, H2=real,
                KTAP=real, THICK=logical;



label:   RBEND, L=real, ANGLE=real, TILT=real,
                K0=real, K1=real, K2=real, K1S=real,
                E1=real, E2=real, FINT=real, FINTX=real,
                HGAP=real, H1=real, H2=real,
                KTAP=real, THICK=logical,
                ADD_ANGLE='array';
```

Bending magnets have the following attributes:

L   The length of the magnet (default: 0 m).
                   For sector bends the declared length is the arc length of the reference orbit.
                   For rectangular bends the declared length is normally the length of a straight
                   line joining the entry and exit points, as in the Figure.
                   Internally MAD-X only uses the arc length of the reference orbit for both bend
                   types.
                   **In order to define RBEND's with a declared length equal to the arc
                   length of the reference orbit, the option RBARC must be previously
                   set to FALSE in MAD-X with Option, RBARC=false;**

ANGLE   The bend angle (default: 0 rad).
                   A positive bend angle represents a bend to the right, i.e. towards negative x
                   values. Please notice that ANGLE is used to define bend strength, K0. To define
                   the roll angle use TILT.

ADD_ANGLE   An array of (maximum 5) bending angles for multiple passes. See ADD_PASS
                   option of the SEQUENCE command. This is only allowed for RBEND elements
                   and is ignored for SBEND elements.

TILT   The roll angle about the longitudinal axis (default: 0 rad, i.e. a horizontal
                   bend).
                   A positive angle represents a clockwise rotation.
                   An attribute TILT=pi/2 turns a horizontal into a vertical bend, and a positive
                   ANGLE then denotes a downwards deflection.

K1   The quadrupole coefficient (Default: 0 m^{-2})
                   $K_1 = (1/B\rho)(\partial B_y/\partial x)$.
                   A positive quadrupole strength implies horizontal focussing of particles, irre-
                   spective of their charge. K1 is taken into account in thick bend transfer map.
                   Only K0 and K1 are considered for think transfer map, not any other strengths
                   (like K2, K1S).

E1   The rotation angle for the entrance pole face (Default: 0 rad).
```

E2       The rotation angle for the exit pole face (Default: 0 rad).
The pole face rotation angles are referred to the magnet model for rectangular bend and sector bend respectively. If E1 and E2 are of the same sign as the bending angle, they reduce the external length (i.e. opposite to the centre of curvature) of the magnet. E1 and E2 must be specified as half of the bending angle of the magnet to get a SBEND with parallel faces, i.e. turning it into a RBEND. E1 and E2 must be specified as *minus* half of the bending angle of the magnet to get a RBEND with sector faces, i.e. turning it into a SBEND.

FINT       The fringe field integral at entrance and exit of the bend. (Default: 0).

FINTX       If defined and positive, the fringe field integral at the exit of the element, overriding FINT for the exit. (Default: =FINT)
This allows to set different fringe field integrals at entrance (FINT) and exit (FINTX) of the element.

HGAP       The half gap of the magnet (default: 0 m).

K2       The sextupole coefficient. (Default: $0 \text{ m}^{-3}$)
$K_2 = (1/B\rho)(\partial^2 B_y/\partial x^2)$. Please note that K2 is not taken into account for bend transfer map.

K1S       The skew quadrupole coefficient $K_s = 1/(2B\rho)(\partial B_x/\partial x - \partial B_y/\partial y)$
where (x,y) is a coordinate system rotated by $-45°$ around s with respect to the normal one. The default is $0 \text{ m}^{-2}$. A positive skew quadrupole strength implies *defocussing*, irrespective of the charge of the particles, in the (x,s) plane rotated by $45°$ around *s* (particles in this plane have x = y > 0). Please note that K1S is not taken into account for bend transfer map.

H1       The curvature of the entrance pole face. (Default: $0 \text{ m}^{-1}$).

H2       The curvature of the exit pole face. (Default: $0 \text{ m}^{-1}$)
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.

K0       Please take note that $K_0$ and $K_0 S$ are left in the data base but are no longer used for the MAP of the bends, instead ANGLE and TILT are used exclusively. However, K0 is assignment of **relative** field errors to a bending magnet because K0 is used for the normalization instead of the ANGLE. (see EFCOMP).
With $K_0 = (1/B\rho)B_y$, one gets K0 = ANGLE / arclength.

KTAP       The relative change of the dipole strength (See K0 above) to account for energy change of the reference particle due to RF cavities or synchrotron radiation. The actual strength of the dipole is calculated as K0 * L = ANGLE * (1 + KTAP). See the TAPER command. (Default: 0)

THICK       If this logical flag is set to true the bending magnet is kept as a thick-element, instead of being converted into thin-lenses after MAKETHIN. Note that if used in tracking without a MAKETHIN command it will remain THICK even if this attribute is false.
(Default: false)

KILL_ENT_FRINGE If this logical flag is set to true the fringe fields on the entrance of the element are not taken into account (not calculated).
(Default: false)

KILL_EXI_FRINGE If this logical flag is set to true the fringe fields on the exit of the element are not taken into account (not calculated).
(Default: false)

**Note:** Additional attributes can be given to bending magnets; They are useful for PTC and are defined in 32.9.

**Fringe Fields:**

The quantities FINT and HGAP specify the finite extent of the fringe fields as defined in SLAC-75 [1]:

$$\text{FINT} = \int_{-\infty}^{\infty} \frac{B_y(s)(B_0 - B_y(s))}{g \cdot B_0^2}\, \mathrm{d}s, \qquad g = 2 \cdot \text{HGAP}. \tag{11.1}$$

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, which is a reasonable average of the above values.

Note also that the possibility to specify both FINT and FINTX allows one to set different values at entrance and exit of a bend element.
This can be particularly useful to set the fringe field integral to zero on one side only, *e.g.* when slicing a dipole.

**Examples:**

```
BR: RBEND, L=5., ANGLE=+0.001;                ! Deflection to the right
BD: RBEND, L=5., ANGLE=+0.001, TILT= pi/2; ! Deflection down
BL: RBEND, L=5., ANGLE=+0.001, TILT= pi;   ! Deflection to the left
BU: RBEND, L=5., ANGLE=+0.001, TILT=-pi/2; ! Deflection up
```

11.4   Dipole edge

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 make the tracking non-symplectic. The transformation of the machine elements into thin lenses leaves dipole edge (DIPEDGE) elements untouched and splits correctly the SBEND's.

It does not make sense to use a `DIPEDGE` alone. It can be specified at the entrance and the exit of a `SBEND`. A dipole edge element is defined by the command:

```
label:  DIPEDGE, H=real, E1=real, FINT=real,
                 HGAP=real, TILT=real;
```

A `DIPEDGE` has zero length and five attributes.

H   Is angle/length or 1/rho (default: $0\,\mathrm{m}^{-1}$ - 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 that specifying `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.

11.5 Quadrupole

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

A `QUADRUPOLE` has six attributes:

L   The quadrupole length (default: 0 m).

K1   The normal quadrupole coefficient: $K_1 = 1/(B\rho)(\partial B_y/\partial x)$.
             The default is $0\,\mathrm{m}^{-2}$. A positive normal quadrupole strength implies horizontal focussing, irrespective of the charge of the particles.

K1S   The skew quadrupole coefficient $K_{1s} = 1/(2B\rho)(\partial B_x/\partial x - \partial B_y/\partial y)$
             where (x,y) is now a coordinate system rotated by $-45°$ around s with respect to the normal one. The default is $0\,\mathrm{m}^{-2}$. A positive skew quadrupole strength implies *defocussing*, irrespective of the charge of the particles, in the (x,s) plane rotated by $45°$ 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 `MAD-8` one has to specify the desired TILT angle, otherwise it is taken as 0 rad. This was needed to avoid the confusion in `MAD-8` about the actual meaning of the `TILT` attribute for various elements.

KTAP   The relative change of the quadrupoles strengths (both `K1` and `K1S`) to account
                for energy change of the reference particle due to RF cavities or synchrotron
                radiation. The actual strength of the quadrupole is calculated as `K1act = K1`
                `* (1 + KTAP)`. See the TAPER command. (Default: 0)

THICK   If this logical flag is set to true the bending magnet is kept as a thick-element,
                instead of being converted into thin-lenses after MAKETHIN. Note that if
                used in tracking without a MAKETHIN command it will remain THICK even
                if this attribute is false.
                (Default: false)

**Note also that K_1 or 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 a tilt alignment error in the machine. In fact, a positive K_1 with a TILT
= 0 is equivalent to a positive K_{1s} with TILT = $+\pi/4$**

Example:

        QF: QUADRUPOLE, L=1.5, K1=0.001, THICK=true;

Note: Quadrupoles cannot have any of the attributes of other magnetic elements, for example, bend (`K0`) or sextupole (`K2` opr `K2S`). If these have to be considered in the element, one
should use the `MULTIPOLE` element.

Note: Additional attributes can be given to quadrupoles; They are useful for `PTC` and are
defined in 32.9.

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

11.6   Sextupole

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

A `SEXTUPOLE` has five real attributes:

L   The sextupole length (default: 0 m).

K2   The normal sextupole coefficient $K_2 = \frac{1}{B\rho}(\partial^2 B_y/\partial x^2)$.
                (default: 0 m^{-3}).

K2S   The skew sextupole coefficient $K_{2S} = \frac{1}{B\rho}(\partial^2 B_x/\partial x^2)$
                where (x,y) is now a coordinate system rotated by $-30°$ around s with respect
                to the normal one. (default: 0 m^{-3}). A positive skew sextupole strength
                implies *defocussing* (!) irrespective of the charge of the particles, in the (x,s)
                plane rotated by 30° 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 `MAD-8` one has to specify the desired TILT
                angle, otherwise it is taken as 0 rad. This was needed to avoid the**

> **confusion in MAD-8 about the actual meaning of the TILT attribute for various elements.**

KTAP The relative change of the sextupole strengths (both K2 and K2S) to account for energy change of the reference particle due to RF cavities or synchrotron radiation. The actual strength of the sextupole is calculated as K2act = K2 * (1 + KTAP). See the TAPER command.27. (Default: 0)

Note also that K_2 or 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;

Note: Sextupoles cannot have any of the attributes of other magnetic elements, for example, bend (K0) or quadrupole (K1 or K1S). If these have to be considered in the element, one should use the MULTIPOLE element.

Note: Additional attributes can be given to sextupoles; They are useful for PTC and are defined in 32.9.

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

11.7 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 $K_3 = \frac{1}{B\rho}(\partial^3 B_y/\partial x^3)$
(default: 0 m^{-4}).

K3S The skew octupole coefficient $K_{3S} = \frac{1}{2B\rho}(\partial^3 B_x/\partial x^3 - \partial^3 B_y/\partial y^3)$
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* (!) irrespective of the charge of the 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 MAD-8 one has to specify the desired TILT angle, otherwise it is taken as 0 rad. This was needed to avoid the confusion in MAD-8 about the actual meaning of the TILT attribute for various elements.**

Note also that K_3 or 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;
```

Note: Additional attributes can be given to octupoles; They are useful for PTC and are defined in 32.9.

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.

11.8   General Thin Multipole

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

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

The deflection imparted by a magnetic multipole on a particle is expressed as *integrated multipole magnetic strength.* The n-th order integrated normal- and skew- magnetic strengths, respectively $K_{N,n}L$ and $K_{S,n}L$, are defined as:

$$K_{N,n}L = \frac{L}{B\rho}\frac{\partial^n B_y}{\partial x^n}; \quad \text{and} \quad K_{S,n}L = \frac{L}{B\rho}\frac{\partial^n B_x}{\partial x^n}, \tag{11.2}$$

as mentioned in section 1.6. They are both expressed in $[\mathrm{m}^{-n}]$.

LRAD   A fictitious length, originally only used to compute synchrotron radiation effects.
           A non-zero LRAD in conjunction with OPTION, THIN_FOC=true takes into account 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. The roll angle affects all components.

KNL   An array of the integrated normal multipole coefficients, starting from order zero and up to the maximum order (currently 20). The parameters are positional in the array, therefore zeros must be filled in for components that do not exist.
           The coefficient of rank i in the array corresponds to the integrated strength $K_iL = B_i.L/(B\rho)$ where the strength is given by equation 1.11. Hence the first argument of the array, the argument for a normal multipole of order zero, $K_0L = B_0.L/(B\rho)$ is equal to the normal horizontal rotation angle of the thin dipole.

KSL   An array of the integrated skew multipole coefficients, starting from order zero and up to the maximum order (currently 20). The parameters are positional in the array, therefore zeros must be filled in for components that do not exist.

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

Contrary to `MAD-8` the desired `TILT` angle must be explicitly specified, and defaults otherwise to 0 rad. The roll angle specified with `TILT` is global to all multipolar components. Hence the `KNL` and `KSL` components can be considered as the normal or skew multipole components of the magnet as measured on the bench, while the `TILT` attribute can be considered as an alignment error as measured in the machine.

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, total deflection `angle` defined by:

$$\begin{aligned} \mathtt{angle} &= \sqrt{\mathtt{KNL}(0)^2 + \mathtt{KSL}(0)^2} \\ \mathtt{tilt} &= \mathtt{TILT} - \arctan(\mathtt{KSL}(0)/\mathtt{KNL}(0)) \end{aligned}$$

(11.3)

Note that the `TILT` attribute of the `MULTIPOLE` is added to the intrinsic tilt calculated from `KNL` and `KSL`.

Examples:
A thin-lens sextupole:

```
ms:  MULTIPOLE, KNL={0, 0, k2l};
```

A thin-lens skew octupole:

```
mso:  MULTIPOLE, KSL={0, 0, 0, k3sl};
```

A thin-lens multipole with a normal octupole component and a skew decapole component:

```
mod:  MULTIPOLE, KNL={0,0,0,myoct*lrad}, KSL={0,0,0,0,-1.e-5};
```

A thin-lens dipole bending to the right and down for a total angle of 2 milliradians and a tilt of $\pi/4$ can be equivalently defined as:

```
hvbend:  MULTIPOLE, KNL={1.414e-3}, KSL={1.414e-3};
hvbend:  MULTIPOLE, KNL={2.e-3}, TILT= pi/4;
hvbend:  MULTIPOLE, KSL={2.e-3}, TILT=-pi/4;
```

11.9   Solenoid

Solenoids can be defined in two forms, a thick and a thin version:

```
label:  SOLENOID, L=real, KS=real;            !  thick version
label:  SOLENOID, L=0,    KS=real, KSI=real;  !  thin version
```

A `SOLENOID` has three real attributes:

L   The length of the solenoid (default: 0 m)

KS   The solenoid strength $K_s = B_0/B\rho$ (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_sL (default: 0 rad). This additional attribute is needed only when using the thin solenoid, where $L = 0$.

Example:

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

Note: Additional attributes can be given to solenoids. They are useful for PTC and are defined in 32.9. In particular multipole coefficients KNL and KSL can also be specified for solenoids. They have no effect in MAD-X proper but are used in PTC for solenoid with multipoles.

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

11.10   Nonlinear Lens with Elliptic Potential

```
label:  NLLENS, KNLL=real, CNLL=real;
```

The NLLENS element represents a thin nonlinear lens with the potential of 'Elliptic' type as specified in [9]. The lens is used to create fully integrable 2D nonlinear accelerator lattice with very large nonlinear tune spread/shift. The NLLENS element is recognized by the thin tracking module. The quadrupole term of the potential is included in the transport map and, consequently, affects the calculation of tunes and Twiss functions.

KNLL   The integrated strength of lens (m). The strength is parametrized so that the quadrupole term of the multipole expansion is k1=2*KNLL/CNLL^2.

CNLL   The dimensional parameter of lens (m). The singularities of the potential are located at X=-CNLL,+CNLL and Y=0.

The scalar potential function of the element is given by

$$U(x,y) = \frac{k}{c} \frac{\xi\sqrt{\xi^2 - 1}\,\mathrm{acosh}\xi + \eta\sqrt{1-\eta^2}(\mathrm{acos}\eta - \pi/2)}{\xi^2 - \eta^2} \tag{11.4}$$

where $k = $ KNLL, $c = $ CNLL and

$$\xi = \frac{\sqrt{(x+c)^2 + y^2} + \sqrt{(x-c)^2 + y^2}}{2c}, \quad \eta = \frac{\sqrt{(x+c)^2 + y^2} - \sqrt{(x-c)^2 + y^2}}{2c}, \tag{11.5}$$

Figure below shows the contour plot of the scalar potential:

The multipole expansion of the scalar potential is

$$U(x,y) = k \cdot Re\left\{ \left(\frac{x+iy}{c}\right)^2 + \frac{2}{3}\left(\frac{x+iy}{c}\right)^4 + \frac{8}{15}\left(\frac{x+iy}{c}\right)^6 + \frac{16}{35}\left(\frac{x+iy}{c}\right)^8 + \cdots \right\}$$
$$\tag{11.6}$$

Figure 11.1: Contour plot of the scalar potential

Note that this expansion is only valid inside the $r=c$ circle on the x,y plane.

In order to create integrable optics, one needs to shape the potential along z axis according to the beta-function. Below is an example nonlinear section representing the necessary nonlinear field with 20 thin lenses:

```
mu0 = 0.3;  ! phase advance over straight section
l0 = 2.0;   ! length of the straight section
nn = 20;    ! number of nonlinear elements
tn = 0.45;  ! strength of nonlinear lens
cn = 0.01;  ! dimentional parameter of nonlinear lens
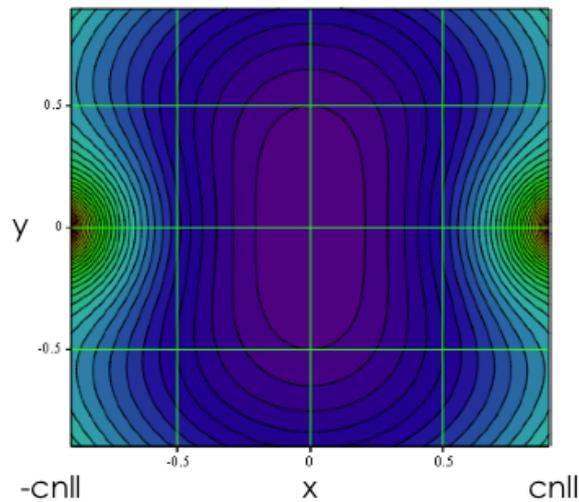
musect = mu0 + 0.5;
f0 = l0/4.0*(1.0+1.0/tan(pi*mu0)^2);
betae = l0/sqrt(1.0-(1.0-l0/2.0/f0)^2);
alfae = l0/2.0/f0/sqrt(1.0-(1.0-l0/2.0/f0)^2);
betas = l0*(1-l0/4.0/f0)/sqrt(1.0-(1.0-l0/2.0/f0)^2);
value, f0, betae, alfae, betas;

ncreate(ii,kk,cc): macro = {n.ii: nllens, knll=kk, cnll=cc;};

i=0;
while(i <  nn)
  {
    i = i+1;
    sn = l0/nn*(i-0.5);
    bn = l0*(1-sn*(l0-sn)/l0/f0)/sqrt(1.0-(1.0-l0/2.0/f0)^2);
    knn = l0*tn*cn^2/nn/bn;
    cnn = cn*sqrt(bn);
```

```
    exec, ncreate($i,knn,cnn);
    value, i, bn, cnn, knn;
};
```

11.11   Closed Orbit Corrector

Three types of magnetic 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 momentum change $\delta PX = \delta p_x/p_0$ or $\delta PY = \delta p_y/p_0$ for respectively horizontal or vertical correctors. (default: 0).

HKICK   The horizontal momentum change $\delta PX = \delta p_x/p_0$ for a corrector acting in both planes (default: 0).

VKICK   The vertical momentum change $\delta PY = \delta p_y/p_0$ for a corrector acting in both planes (default: 0).

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.

The deviation angle θ of the particle trajectory is related to the momentum change through $\sin\theta = \delta P = \delta p/p_0$.

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.

Please note that there is a new feature introduced by Stefan Sorge from GSI. Here his decription:

The elements `KICKER, HKICKER`, and `VKICKER` can also be used as magnetic exciters providing sinusoidal momentum kicks. The usage in this case is:

```
xykick:  KICKER, SINKICK=integer, SINPEAK=real, SINTUNE=real, SINPHASE=real;
xkick :  HKICKER,SINKICK=integer, SINPEAK=real, SINTUNE=real, SINPHASE=real;
ykick :  VKICKER,SINKICK=integer, SINPEAK=real, SINTUNE=real, SINPHASE=real;
```

where a sinusoidal momentum kick `dpz` as a function of the revolution number `n` given by
`dpz(n)=SINPEAK * sin(2*PI*SINTUNE*n + SINPHASE), pz=px,py`
is provided.

The `KICKER` element generates synchronous kicks in both horizontal and vertical planes. `HKICKER` generates only a horizontal kick, and `VKICKER` generates only a vertical kick.

The variables are

`SINKICK`   must be set to 1 to switch on the sinusoidal signal, default: 0.

`SINPEAK`   amplitude of the bending angle (rad); default: 0 rad.

`SINTUNE`   frequency of the signal times the revolution frequency. Hence, the phase per revolution is 2*PI*SINTUNE; default: 0.

`SINPHASE`   initial phase; default: 0 rad.

The momentum kick of a kicker has only a single frequency. An element having a finite bandwidth can approximately created by defining thin kickers with all amplitudes `SINPEAK`, frequencies `SINTUNE`, and initial phases `SINPHASE` desired and putting them at the same position s in the accelerator.

11.12 Transverse Kicker

The type `TKICKER` should be used to create horizontal, vertical or combined transverse magnetic kickers physically equivalent to elements of type `KICKER`, but **not used** by the COR-RECT command of the closed orbit correction module.

Examples of elements that may use the type `TKICKER`:

- Fast kickers for injection, dump and tune

- Magnetic septa towards beam dump

- Dampers of transverse beam oscillations

- Undulator and Wiggler magnets

For further information on element type `TKICKER` and its attributes, look at the documentation of the orbit corrector type `KICKER`.

11.13   RF Cavity

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

An `RFCAVITY` has eight real attributes and one integer attribute:

L   The length of the cavity (DEFAULT: 0 m)

VOLT   The peak electrical 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 $[2\pi]$ (DEFAULT: 0).

FREQ   The frequency [MHz] (no DEFAULT).

                **Note that if the RF frequency is not given, it is computed from the
                harmonic number and the revolution frequency f_0. For accelerating
                structures this makes no sense, and the input of the frequency is
                mandatory.**

HARMON   The harmonic number h (no DEFAULT). This attribute is only used if the
                frequency is not given.

Caveats:

- Please take note, that the following `MAD-8` attributes: `BETRF, PG, SHUNT` and `TFILL`
  are currently not implemented in `MAD-X`.

- Important Note: The `TWISS` command 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. Please refer to `PTC_SETSWITCH` command, in particular to `TIME` and `TOTALPATH` switches, concerning RF behaviour of cavities in PTC.

- The charge is not taken into account by the RF-cavity.

- The RF-cavity has to be thin for TRACKING in MAD-X. This is handled by `MAKETHIN` but note that it is always sliced into a single slice.

The `RFCAVITY` can also have attributes that only become active in `PTC`:

N_BESSEL   (DEFAULT: 0):
                Transverse focussing effects are typically ignored in the cavity in `MAD-X` or
                even `PTC`. This effect is being calculated to order `n_bessel`, with `n_bessel=0`
                disregarding this effect and with a correct treatment when `n_bessel` goes to
                infinity.

NO_CAVITY_TOTALPATH (Default: false):
>> flag to select whether the transit time factor in the cavity is to be considered (NO_CAVITY_TOTALPATH = false) or if the particle is kept on constant RF phase when passing through cavity. (NO_CAVITY_TOTALPATH = true).

A cavity requires the particle ENERGY and the particle 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.

11.14   Travelling Wave Cavity

```
label:  TWCAVITY, L=real, VOLT=real, FREQ=real, LAG=real,
                  PSI=real, DELTA_LAG=real ;
```

The travelling wave cavities model the accelerating structures of linacs where wavefront moves together with the accelerated particles. This element is implemented only by PTC and is treated as drift by native MADX commands. The phase velocity is fixed to speed of light and can not be varied.

L   The length of the cavity (DEFAULT: 0 m)

VOLT   The peak electrical RF voltage (DEFAULT: 0 MV).

FREQ   The frequency [MHz] (no DEFAULT).

LAG   The phase lag $[2\pi]$ (DEFAULT: 0). Zero lag means the highest acceleration, i.e. the reference particle (starting at x=0, px=0, y=0, py=0,t=0, pt=0) is on the crest of the wave. Please note that it is different from the RFCAVITY nomenclature.

PSI

DELTA_LAG

11.15   Thin Radio-Frequency Multipole

```
label:  RFMULTIPOLE, L=real, VOLT=real, LAG=real,
                     FREQ=real, HARMON=integer,
                     LRAD=real, TILT=real,
                     KNL={real, ...}, KSL={real, ...},
                     PNL={real, ...}, PSL={real, ...};
```

A `RFMULTIPOLE` is a element which exhibits the properties of an RF cavity as well as those of a multipole, of arbitrary order, oscillating at the same RF frequency.

The accelerating kick is `delta(E) = VOLT * sin(2π * (LAG - HARMON * f0 t))`, and the multipole moments oscillate with the same RF frequency.

An `RFMULTIPOLE` requires the particle `ENERGY` and the particle `CHARGE` to be set with a `BEAM` command before any calculation is performed.

Contrary to the regular multipole, where the dipole component has an effect on the reference orbit, an RF-Multipole that includes a dipole component does not bend the reference orbit.

In PTC there must be at least one cavity with non-zero voltage, otherwise all RF elements, including RF multipoles, are disabled. Concerning RF behaviour of cavities in PTC please refer to `PTC_SETSWITCH` command, in particular to `TIME` and `TOTALPATH` switches.

VOLT	The peak voltage of the RF accelerating mode (DEFAULT: 0 MV).
LAG	The phase lag $[2\pi]$ (DEFAULT: 0)
FREQ	The frequency [MHz] (no DEFAULT). Note that if the RF frequency is not given, it is computed from the harmonic number and the revolution frequency f0 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.
LRAD	A fictitious length, which was originally just used to compute synchrotron radiation effects.
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 `MAD-8` one has to specify the desired `TILT` angle, otherwise it is taken as `0` rad. We believe that the `MAD-8` 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	An array of the integrated normal rfmultipole coefficients, starting from order zero and up to the maximum order (currently 20). The parameters are positional in the array, therefore leading zeros must be filled in for components that do not exist.
KSL	An array of the integrated skew rfmultipole coefficients, starting from order zero and up to the maximum order (currently 20). The parameters are positional in the array, therefore leading zeros must be filled in for components that do not exist.
PNL	The phase for each normal rfmultipole coefficients from order zero to the maximum; the parameters are positional, therefore leading zeros must be filled in

for components that do not exist.

PSL   The phase for each skew rfmultipole coefficients from order zero to the maximum; the parameters are positional, therefore leading zeros must be filled in for components that do not exist.

Examples

- A skew RF octupole

      MS: RFMULTIPOLE, KSL={0, 0, 0, k3sl};

  Notice that both KNL and KSL for the same multipole mode can be specified simultaneously.

- An RFMULTIPOLE used as a CRABCAVITY:

      rfdipole: RFMULTIPOLE, L=Lcrab, KNL={ Vcrab / PC0 },
                              PNL={ 0.25 + LAGcrab };

  is equivalent to

      crab: CRABCAVITY, L=Lcrab, VOLT=Vcrab, LAG=LAGcrab;

11.16 Crab Cavity

```
label:  CRABCAVITY, L=real, TILT=real, VOLT=real, LAG=real,
                    FREQ=real, HARMON=integer,
                    RV1=integer, RV2=integer,
                    RV3=integer, RV4=integer,
                    RPH1=integer, RPH2=integer, LAGF=real ;
```

A CRABCAVITY has six attributes describing its steady state and seven attributes to describe dynamic behaviour:

L   The length of the cavity (default: 0 m)

TILT   The tilt of the cavity in radian (default: 0 m). If the tilt is $\pi/2$, then the export to sixtrack will automatically convert it to a vertical cravity (default is horizontal).

VOLT   The peak RF voltage (default: 0 MV).

LAG   The initial phase lag $[2\pi]$ (default: 0).

FREQ   The RF frequency [MHz] (no default).

               Note that if the RF frequency is not given, it is computed from the harmonic number and the revolution frequency f_0. For deflecting structures this makes no sense, and the frequency is mandatory.

HARMON   The harmonic number h (no default).
               Only if the frequency is not given.

The other attributes describe the time evolution of a CRABCAVITY behaviour:

RV1   Number of initial turns with zero voltage (default: 0).

RV2   Number of turns to ramp voltage from zero to nominal value (default: 0).

RV3   Number of turns with nominal voltage (default: 0).

RV4   Number of turns to ramp voltage from nominal value to zero (default: 0).

LAGF   Value of the final crab RF phase lag $[2\pi]$ (default: 0).

RPH1   Number of initial turns with nominal phase (default: 0).

RPH2   Number of turns to ramp phase $[2\pi]$ from nominal to specified value
            (default: 0).

Caveats:

- Please take note, that the following `MAD-8` attributes: `BETRF, PG, SHUNT` and `TFILL` are currently not implemented in `MAD-X`!

- Note that crab cavities are only implemented for tracking purposes.
  `TWISS` ignores any effect of the crab cavity.

- Important Note: The `TWISS` command 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.

Before any calculation is performed with a `CRABCAVITY`, the particle `ENERGY` and the particle `CHARGE` must be set with the `BEAM` command.

The effect of a `CRABCAVITY` on particle coordinates during tracking is

$\delta\mathtt{p_x} = \mathtt{VOLT} * \sin(\mathtt{PHI} - \mathtt{OMEGA} * \mathtt{t})$
$\delta\mathtt{E} = -\mathtt{VOLT} * \mathtt{OMEGA} * \mathtt{x} * \cos(\mathtt{PHI} - \mathtt{OMEGA} * \mathtt{t})$

where $\mathtt{PHI} = 2\pi * (\mathtt{LAG} - \mathtt{HARMON} * \mathtt{f_0 t})$,
and $\mathtt{OMEGA} = 2\pi * \mathtt{FREQ}/\mathtt{c}$

Example:

```
BEAM, PARTICLE=PROTON, ENERGY=7000.0;
CAVITY: CRABCAVITY, L=10.0, VOLT=5.0, LAG=0.0, FREQ=400,
        RV1=0, RV2=50, RV3=1000, RV4=50,
        RPH1=100, RPH2=500, LAGF=0.125;
```

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

11.17   AC Dipole

Alternating Current (AC) dipole is implemented with:

HACDIPOLE   an AC dipole for the horizontal plane,

VACDIPOLE   an AC dipole for the vertical plane,

```
label:  HACDIPOLE, L=real, VOLT=real, FREQ=real, LAG=real,
                RAMP1=integer, RAMP2=integer,
                RAMP3=integer, RAMP4=integer;
label:  VACDIPOLE, L=real, VOLT=real, FREQ=real, LAG=real,
                RAMP1=integer, RAMP2=integer,
                RAMP3=integer, RAMP4=integer;
```

In fact this element is rather oscilating (time varying) orbit corrector (kicker) rather than dipole because it does not change the reference frame. The oscillation is assumed to be slow compared with bunch length and the kick is constant along the bunch. These devices are used for beam excitations in circular machines for optics measurements. They are linearly ramped up to the full amplitude (and ramped down) to avoid emittance growth.

The attributes have the following meaning:

L   The length of the device (default: 0 m).

VOLT   The amplitude of the kick (default: 0).

FREQ   Tune of the oscillation in 2π units.
                **The name of the parameter is misleading and will be changed in the
                future for TUNE.**
                (default: 0).

RAMP1   Starting turn of amplitude ramp-up.
                (default: 0).

RAMP2   Last turn of amplitude ramp-up.
                (default: 0).

RAMP3   Starting turn of amplitude ramp-down.
                (default: 0).

RAMP4   Last turn of amplitude ramp-down.
                (default: 0).

AC dipole is not implemented in TWISS.

In TRACK it is implemented as a zero-length element changing transverse momentum by $(0.3 \cdot VOLT/p0c) \cdot sin(2\pi \cdot FREQ \cdot turn)$, where $turn$ is turn number. If ramp settings are left with default values (all zero) AC dipole has no effect in TRACK.

PTC requires that MODULATION flag is set to true via PTC_SETSWITCH command to enable AC dipole, otherwise it is treated as a drift.

For computation of the maps in PTC_TWISS and PTC_NORMAL the phasespace is than extended by extra dimensions corresponding to the frequency clocks defined with AC dipoles. Maximum of three distinct frequencies are allowed. There can be more then three AC dipoles but than some of them need to have the same frequencies. PTC_TWISS, when invoked with NORMAL=true, calculates dependence of optical paramters on amplitudes of the clocks. Results of PTC_TRACK and PTC_TRACKLINE may be slightly different from TRACK because of some implementation

details, in particular for not fully relativistic beams. PTC internaly uses frequency so time of flight effects are taken into account and kick amplitude is inversely proportional to relativistic beta function. Please note that at the moment `LAG` parameter is ignored in translation to PTC. If ramp settings are left with default values (all zero) PTC tracking will assume maximum amplitude.

11.18   Electrostatic Separator

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

An `ELSEPARATOR` element 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 in-
                    creases 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.

EX_L   The integrated vertical electric field strength (default: 0 MV). This parameter
                    is only used in THINTRACK and ignored by TWISS.

EY_L   The integrated horizontal electric field strength (default: 0 MV). This param-
                    eter is only used in THINTRACK and ignored by TWISS.

An electrostatic separator requires the particle `ENERGY` and the particle `CHARGE` to be set by a `BEAM` command before any calculation is performed. Note that no thin elseparator currently exists in TWISS. 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.

11.19   Beam Position Monitor

A beam monitor has no specific effect on the beam and behaves like a drift space. In addition it serves to record the beam position for closed orbit correction.

Three 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.

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

A beam position monitor has one real attribute:

L   The length of the monitor (default: 0 m).

Examples:

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

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

11.20   Instrument and Placeholder

An instrument or a placeholder has no specific effect on the beam and behaves like a drift space. An instrument is different from beam a position monitor and is not used for closed orbit correction.

Two different types of instruments are recognised:

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.

PLACEHOLDER   A place holder for any type of element. Internally it is equivalent to an INSTRUMENT: optically it behaves as a drift space, it returns *no beam observation*. It represent a class of elements which is completely independent from drifts and monitors.

```
label:  INSTRUMENT,  L=real;
label:  PLACEHOLDER, L=real;
```

An instrument or placeholder has one real attribute:

L   The length of the instrument (default: 0 m).

The straight reference system for an instrument is a Cartesian coordinate system.

11.21   Collimator

A COLLIMATOR has no specific effect on beam optics and behaves like a drift space.

```
label:  COLLIMATOR, L=real,
                    APERTYPE=string, APERTURE={values},
                    APER_OFFSET={values}, APER_TOL={values};
```

A COLLIMATOR has one specific real attribute:

L   The collimator length (default: 0 m).

Additionally, like any other element, except `DRIFT` space, a `COLLIMATOR` can have specific aperture related attributes as defined in the related section Defining aperture in `MAD-X`

During tracking in `MAD-X`, particle loss is checked at the entrance of the element by comparing the particle coordinates and the defined aperture, provided that the `APERTURE` flag is `true` in the `TRACK` command, and that the `APERTYPE` attribute value of the element is one of the predefined types. An aperture model defined in an external file (`APERTYPE=filename`) is not used to check particle loss during tracking.

Example:

```
COLLIM: COLLIMATOR, L=0.5, APERTYPE=ellipse, APERTURE=0.01,0.005;
```

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

NOTE: A collimator can be displaced transversally in order to model an asymmetric collimator by means of the `APER_OFFSET` attributes; During tracking particle losses are then reported with coordinates with respect to the **displaced** reference system, not with respect to the surrounding beam line.

Other collimator elements have been inherited from `MAD-8` and still exist in `MAD-X` for backward compatibility. `ECOLLIMATOR` (elliptic aperture collimator) and `RCOLLIMATOR` (rectangular aperture collimator) behave both as drift spaces in `MAD-X` They are declared with

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

Either type has several real attributes:

L   The collimator length (default: 0 m).

XSIZE   The horizontal half-aperture (default: unlimited).
                OBSOLETE : parsed and stored but not used.

YSIZE   The vertical half-aperture (default: unlimited).
                OBSOLETE : parsed and stored but not used.

Users are STRONGLY advised to replaced all instances of `RCOLLIMATOR` and `ECOLLIMATOR` in input files with appropriate `COLLIMATOR` elements. The `RCOLLIMATOR` and `ECOLLIMATOR` elements are only kept for the time being for backward compatibility and will be removed in the near future.

Note also that the `XSIZE` and `YSIZE` parameters can be declared but are simply ignored both in the `APERTURE` command an in tracking.

11.22   Beam-beam Interaction

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

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

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).

CHARGE   The charge of particles in the opposite beam, in elementary charges.
            (default: +1)
            In order to describe interactions between beams containing the same particles
            and having a charge different from +1, one should set the CHARGE explicitly in
            the BEAM command as well as in the BEAMBEAM element.

NPART   The number of particles. In case this attribute is not defined then MAD-X
            will use the NPART from the beam command.

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).

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.

WIDTH   The extent of the edge region, relative to the SIGX, SIGY parameters. The ab-
            solute value is given by WIDTH*SIGX and WIDTH*SIGY in horizontal and vertical
            directions respectively.

BBSHAPE   A parameter to select the radial density shape of the opposite beam
            (default: 1)

            • BBSHAPE=1: Gaussian shape (default). SIGX, SIGY are the standard de-
              viations in horizontal and vertical directions. WIDTH is ignored.

            • BBSHAPE=2: trapezoidal shape (see fig.11.2). SIGX, SIGY are the half
              widths of density profile, *i.e.* distance from the centre to half edge region
              with linear decrease of density in horizontal and vertical directions.
              Only circular opposite beam is possible, *i.e.* in the calculations SIGX' =
              SIGY' = (SIGX+SIGY)/2 is used, if SIGX and SIGY have different values.
              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=5mm, the edge region has a full width of 0.05 mm.
              Condition: WIDTH <2.0.

            • BBSHAPE=3: hollow-parabolic shape (see fig.11.3). SIGX, SIGY are the
              distances from the centre to the maximum of the parabolic density profile

Figure 11.2: Trapezoidal shape of radial density for beam-beam lens.

> in horizontal and vertical directions.
>
> Only circular opposite beam is possible, *i.e.* in the calculations `SIGX'` = `SIGY'` = `(SIGX+SIGY)/2` is used, if `SIGX` and `SIGY` have different values. `WIDTH` denotes the full width at half maximum of the parabolic density profile in units of `SIGX`, or `SIGX'` and `SIGY'`, respectively, if `SIGX` and `SIGY` are not equal. Condition: `WIDTH` <`SQRT(2.0)`

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.

BBDIR   A parameter to select the direction of motion of the opposite beam relative to the beam being studied. It determines the sign of the Lorentz force between both beams (default: -1):

- `BBDIR=-1`: Beams move in opposite directions as in a collider. The Lorentz force enhances the beam-beam interaction.

- `BBDIR=0`: Opposite beam does not move. The Lorentz force is neglected

- `BBDIR=1`: Beams move in the same direction as in an electron cooler. The Lorentz force reduces the beam-beam interaction.

Figure 11.3: Hollow parabolic shape of radial density for beam-beam lens.

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 of the particles in the beam considered. 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*clight)`, 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. Note that the factor 1/2 is inserted only for `BBDIR=-1` to calculate correct results.

A beam-beam element requires the particle `ENERGY` and the particle `CHARGE`, as well as the number of particles per bunch (`NPART`) to be set by a `BEAM` command before any calculation is performed.

Example of a four-dimensional beam-beam element definition in Collider regime:

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

11.23   Wire

```
label:  WIRE, CURRENT={real, ...}, L=real,
               L_INT={real, ...}, L_PHY={real, ...},
               XMA={real, ...}, ..., YMA={real, ...};
```

The `WIRE` element allows defining one or several WIRE elements at once.

L   The length of the element in the sequence.

CURRENT   The current for the wires in Amperes .

L_PHY   This should correspond to the actual physical length of the wires.

L_INT   The length where we stop integrating the effect of the field. It needs to be
                larger or the same as `L_PHY`.

XMA   This specifies the horizontal offset of the wire compared to the reference orbit.

YMA   This specifies the vertical offset of the wire compared to the reference orbit.

The WIRE as implemented in MAD-X is imlemented as a thin kick. However, it is still
possible to define a length of the wire but internally the element is placed in the middle and
drift spaces are placed on each side of the `WIRE`. Note also that the collimator element has
the same attiributes as the `WIRE` and by defining it using the same parameters you can also
get the collimator with a wire inside.

11.24   Arbitrary Matrix Element

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

The `MATRIX` element allows 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 terms) of the element.

TMikl   Defines the second-order terms ($6 \times 6 \times 6$ terms) of the element.

Data values that are not explicitly entered are taken from the identity transformation for the
`RMik` matrix elements, and taken as zero for the `KICKi` kick factors and the `TMikl` second order
terms. In the thin-lens tracking module a non-zero length for an arbitrary matrix is accepted,

however no non-zero second order terms are allowed to avoid non symplectic tracking runs. In the latter case the tracking run is aborted.

11.25 Rotation around the vertical axis

The element `YROTATION` rotates the straight reference system about the vertical (y) axis.

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

`YROTATION` has no effect on the beam, but it causes the beam to be referred to the new coordinate system

$$x_2 = x_1 \cos\theta - s_1 \sin\theta$$
$$s_2 = x_1 \sin\theta + s_1 \cos\theta \tag{11.7}$$

It has one real attribute:

ANGLE   The rotation angle θ (default: 0 rad).

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

Important note:
The rotation angle θ must be *small*, *i.e.* it must be at most comparable to the transverse angles of the orbit.

Example:

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

11.26 Rotation around the horizontal axis

The element `XROTATION` rotates the straight reference system about the vertical (x) axis.

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

`XROTATION` has no effect on the beam, but it causes the beam to be referred to the new coordinate system

$$y_2 = y_1 \cos\theta - s_1 \sin\theta$$
$$s_2 = y_1 \sin\theta + s_1 \cos\theta \tag{11.8}$$

It has one real attribute:

ANGLE   The rotation angle θ (default: 0 rad).

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

Important note:
The rotation angle θ must be *small*, *i.e.* it must be at most comparable to the transverse angles of the orbit.

Example:

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

11.27   Rotation around the longitudinal axis

The element `SROTATION` rotates the straight reference system about the longitudinal (s) axis.

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

`SROTATION` has no effect on the beam, but it causes the beam to be referred to the new coordinate system

$$
\begin{aligned}
x_2 &= x_1 \cos\psi - y_1 \sin\psi \\
y_2 &= x_1 \sin\psi + y_1 \cos\psi
\end{aligned}
\tag{11.9}
$$

It has one real attribute:

`ANGLE`   The rotation angle ψ (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 `VBEND` definition 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;
```

11.28   Coordinate translation

The element `TRANSLATION` changes the reference system by applying a translation of the reference system.

```
    label:   TRANSLATION, DX=real, DY=real, DS=real;
```

`TRANSLATION` moves the coordinate system. `DX` and `DY` will translate the position and `DS` will introduce a virtual drift.

11.29   Change of reference system

The element `CHANGEREF` changes the reference system by applying both translations and rotations.

```
label:  CHANGEREF, PATCH_ANG=real, real, real,
                   PATCH_TRANS=real, real, real;
```

CHANGEREF has no effect on the beam, but it causes the beam to be referred to the new coordinate system where both translations and rotations are applied.

11.30   SixTrack Marker

```
label:  SIXMARKER, ELTYPE=real, ATTR={real, ...},
                   F3STRING=string, F3VECTOR={real, ...} ;
```

The SIXMARKER element allows the definition of a marker that is converted to SixTrack. The element has no effect in MAD-X.

ELTYPE   is the element type number in SixTrack, converted to the first column in fc.2.

ATTR   This defines the remaining columns of the fc.2 file.

F3STRING   Defines the String that is written to the fc.3 file, {newline} creates a new line in the output and {0}...{10} are being replaced with the values in F3VECTOR.

F3VECTOR   The values defined will be converted to the location defined in the F3STRING.

Example:

```
sm1:SIXMARKER, ELTYPE=2, ATTR={0.1,2,3,4,5,6,7},
,F3STRING="NEW_ELEMENT {newline} {0}, {1} {newline} NEXT",
F3VECTOR={0.2, 0.5};
```

Will produce in the output file fc.3:

```
NEW_ELEMENT
2.00000000e-01, 5.00000000e-01
NEXT
```

Chapter 12. Range and Class Selection Format

12.1 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 that are between `position1` and `position2` are selected.
NOTE: `position1` must appear before `position2` in the sequence.

The `position` is composed of a single element name if that name is unique in the sequence, or an element class name followed by the occurrence count `n` in the sequence within square brackets to specify the nth occurence of an element of that class in the sequence:

```
mq.ir5.l6..1  !  unique element in the sequence
mb[17]        !  17th occurence of an element of class mb
```

There are two special predefined psoitions in `MAD-X`:

`#S`   The start of the beam line or sequence expanded by USE,

`#E`   The end of the beam line or sequence expanded by USE.

If a range is selected in a `USE` statement:

```
USE, PERIOD=lhcb1, RANGE=ir1/ir5;
```

then the `#S` and `#E` indices refer to the start and end of the range expanded by the `USE` statement.

Examples for ranges:

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

12.2 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; !  makes the element MQ
Q1:  MQ;        !  makes the element Q1 from class MQ
Q2:  MQ;        !  makes the element Q2 from class MQ
Q1..A: Q1;      !  makes an element from class Q1
```

```
    Q2..B: Q2;      !  makes an element from class Q2
```

Subsequent selection with criterion `CLASS="MQ"` will actually select `Q1, Q2, Q1..A`, and `Q2..B` in this example.

Chapter 13. 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.

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 (see 13.2)

Warning:
`MAD-X` has been developed using sequences. Machine description using beamlines was kept in `MAD-X` to a minimal extent to keep backward compatibility but also because it is often easier to start from beamlines when designing an accelerator *ab initio*. But since `MAD-X` works only with sequences internally, there may exist some inconveniences when only beamlines are defined by the user. It is therefore recommended to convert beamlines into sequences (e.g. by means of the `SAVE` command) as soon as possible lines in the design phase and to use only sequences for a finalised machine.

Attempting to do sequence edition on a sequence that has been expanded in memory from a beam line specified by the user is strongly discouraged.

13.1   Simple Beam Lines

The simplest beam line consists of single elements:

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

Example:

```
exmp1:  LINE=(a,b,c,d,a,d);
USE,    PERIOD=exmp1;
```

The `USE` command causes `MAD-X` to load and expand the specified beamline. This in particular loads the beamline as a `SEQUENCE` representation in memory. All subsequent calculations are done on this sequence in memory.

13.2   Nested Beam 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 or nested beam lines which can 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:

```
top:  LINE=(a,b,S,b,a,S,a,b);
S:    LINE=(c,d,e);
USE,  PERIOD=top;
```

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
   ```

13.3   Reflection and Repetition

An unsigned repetition count and an asterisk (multiplication sign) indicate repetition of a beam line member. A repetition count can be applied to sub-lines as well as elements.

A minus prefix causes reflection, i.e. all elements in the sub-line are taken in reverse order. Sub-lines of reflected lines are also reflected. Physical elements are not reflected head-to-tail, hence a negative repetition count for a single element is treated as positive.

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);
top: LINE= (2*S, 2*(e,f), -S, -(a,b), -2*c);
USE, PERIOD=top;
```

produces the following expansion steps:

1. Replace sub-line S:

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

2. Replace sub-line R:

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

3. Omit parentheses:

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

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

Note also that `-2*c` at the end of the `top` line is equivalent to `2*c` since single elements are not reflected, or `2*(c)` which would first promote `c` as a sub-line of a single element, or `-2*(c)` since reverting a sub-line of a single element gives the same single element.

13.4   Replaceable Arguments

A beam line definition may contain a formal argument list, consisting of labels separated by commas and enclosed in parentheses.

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

where `member` can be one of the `arg` arguments.

A beam line defined with arguments can be expanded for different values of its arguments. The arguments can be beam lines or element names. The number of actual arguments given at time of use must agree with the number of formal arguments defined at declaration time. All occurrences of a formal argument on the right-hand side of the line definition are replaced by the corresponding actual argument.

Example:

```
fodo(Q1,Q2):  LINE=(Q1, d, Q2, d);
top:          LINE = ( fodo(qf12, qd23), fodo(qd34,qf45) );
USE, PERIOD=top;
```

expands to

```
qf12, d, qd23, d, qd34, d, qf45, d
```

Example showing that arguments can also be beam lines:

```
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
```

13.5   Limits of Construction of Beam Lines

Since LINEs are in fact deprecated there are some limits to how they can be constructed. Below is a running MAD-X example which shows a collection of valid (marked OK) and invalid cases (marked WRONG).

```
!-----------------------------------------------------------------------
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;
!-----------------------------------------------------------------------
```

Chapter 14. Sequences

`MAD-X` accepts two forms of an accelerator definition: sequences and beam lines (See 13). However, the sequence definition is the only one used internally by `MAD-X`.

A sequence is declared with the following statements:

```
name:  SEQUENCE, L=real, REFER=string, REFPOS=string,
                 ADD_PASS=integer, NEXT_SEQU=string;
       ...
       label: class,  AT=real, FROM=string {, add.  attributes};
       ...
              class,  AT=real, FROM=string {, add.  attributes};
       ...
       seqname,        AT=real, FROM=string;
       ...
ENDSEQUENCE;
```

The first statement declares a sequence, giving it a name in the form of a label, and providing some key parameters:

L
: (Default: 0)
the total length of the sequence in meters.

REFER
: = keyword in {entry, centre, exit} (Default: centre)
specifies which part of the element is taken as the reference point at which the position along the beamline is given.

REFPOS
: (Default: none)
specifies the name of an element in this sequence to be used as the reference point for the insertion of this sequence in another sequence.

ADD_PASS
: (Default: 0)
specifies a number of additional passes (max. 5) through the structure; in case of an RBEND the angle will be overwritten in survey using the i-th component ($1 \leq i \leq$ add_pass ≤ 5) of its `array_of_angles` attribute (see RBEND).

NEXT_SEQU
: (Default: none)
specifies the name of a sequence to be concatenated to the end of this sequence.

The `ENDSEQUENCE` statement terminates the declaration of a sequence.

Inside the `SEQUENCE ...ENDSEQUENCE` bracketing keywords, several types of statements may appear:

- An element declaration with label:

```
label:  class, AT=real, FROM=string {, attributes} ;
```

an element declaration as usual, with additional `AT` attribute giving the relative element position, and an optional `FROM` attribute;

CAUTION: an existing definition for an element with the same name (label) 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.

- An element declaration from class:

```
class, AT=real, FROM=string {, attributes} ;
```

a class name causing an instance of that class to be created with specified attributes, with additional `AT` attribute giving the relative element position, and an optional `FROM` attribute;
For uses inside ranges, instances of the same class can then be accessed with an occurrence count.

- A sub-sequence name:

```
seqname, AT=real, FROM=string ;
```

a sequence name with additional `AT` attribute giving the relative element position, and an optional `FROM` attribute;
Depending upon the `REFER` attribute of the current (containing) sequence, the `entry,` `centre`, or `exit` of the inserted sequence is placed at the position specified.
HOWEVER, if the inserted sequence has a `REFPOS` attribute containing the name of an element in the inserted sequence, the inserted sequence is placed such that the element pointed to by `REFPOS` is at the location specified in the current sequence.

The additional arguments that can be given in the declaration of sequence elements or subsequences are:

AT   mandatory argument giving the location at which the reference point (`entry,` `centre` or `exit`) of the element is to be placed in the sequence.
            The value is absolute with the zero reference at the start of the sequence, unless a `FROM` argument is specified.

FROM   optional argument giving the name of an element of the same sequence.
            The location given for the current element or sequence in the `AT` argument is then taken as **relative** to the position of the center of the element given by the `FROM` argument.
            Only simple elements should be used in `FROM`; specifying a sequence as the `FROM` reference can lead to erroneous sequence expansion at best.

When the sequence is expanded in a USE command, `MAD-X` generates the drift spaces between elements according to the following rules:

1. When the distance between the exit of the previous element and the entrance of the next element is positive and greater than a threshold of $1\mu m$ an explicit drit is generated with its own name, unless an already existing drift space with the same length (within $10^{-12}m$ tolerance) can be re-used.

2. When the absolute value of the distance between the exit of the previous element and the entrance of the next element is less than a given tolerance of $1\mu m$, no drift space is

created and the elements are considered as contiguous.
Note that in very specific cases this can cause very small errors to accumulate and the actual length of the sequence can vary slightly from the declared length. (see second example below)

3. When the distance between the exit of the previous element and the entrance of the next element is negative and less than $-1\mu m$, the elements are considered to be overlapping and MAD-X terminates with a "`negative drift length`" fatal error.

When the sequence is expanded in a USE command, MAD-X also checks that the calculated sequence length is positive and that the lengths of all elements in the sequence are positive or zero. A negative sequence length or negative element length stops the MAD-X run with a fatal error.

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 vary element lengths during matching.

Example:

```
! 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;
```

Example of very small drift space being ignored during sequence expansion:

```
QTEST: QUADRUPOLE, L=1.000001;

TEST: SEQUENCE, REFER=centre, L=2.;
```

```
   QTEST, AT=1.5;
ENDSEQUENCE;

USE, SEQUENCE=TEST;
SURVEY, FILE='test';
```

The above sequence will expand to a total length of $2.0000005m$, half a micron longer than the claimed length of $2m$, but will not fail.

Part III

Input and Output

Chapter 15. TFS File Format

TFS[10] is a an acronym for the "Table File System". TFS files have been used in the LEP control system. The MAD-X 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.

15.1 Descriptor Lines

MAD-X writes the following descriptors in 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.

15.2 Column Formats

The column formats used are listed below:

Table 15.1: 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\"")

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 largest of the column name and the width of data values of the column.

15.3 Twiss TFS file header

MAD-X gives access to parameters from TWISS and other tables using the table access commands.

Chapter 16. Conversion to *SixTrack*

SixTrack[11][12] is a separate beam optics code that is often used for long term tracking of particles, *e.g.* for dynamic aperture studies, because of its speed and controllability.

However the input files are notoriously difficult to produce by hand. This command may be used to generate *SixTrack* input files from a sequence loaded in `MAD-X`.

```
SIXTRACK, CAVALL=logical,
          MULT_AUTO_OFF=logical, MAX_MULT_ORD=integer,
          SPLIT=logical, APERTURE=logical, RADIUS=real, MARKALL=logical;
```

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, default = FALSE) If TRUE, this module does not process zero value multipoles. Moreover, multipoles are prepared by `SIXTRACK` (output file fc.3) to be treated up to the order as specified with `MAX_MULT_ORD`.

MAX_MULT_ORD   (optional parameter, default = 11) Process up to this order for `MULT_AUTO_OFF = TRUE`.

SPLIT   (optional flag) OBSOLETE. This splits all the elements in two. This is for backward compatibilty only. The user should now use the `MAKETHIN` command instead.

APERTURE   (optional flag) flag to convert the apertures from `MAD-X` to *SixTrack* so that *SixTrack* can track with apertures defined. The aperture data is found in file `fc.3.aper`.

RADIUS   (optional, default value is 1 m). This sets the reference radius for the magnets. This argument is optional but should normally be set.

MARKALL   (optional flag, default false) flag to convert all markers through the conversion. If false, only markers with names beginning with "ip" or "mt_" are kept.

LONG_NAMES   (optional flag, default true) flag to increase the possible length of names (from 17 to 48) and increase the number of digits in the output in the conversion.

SIX_VERSION   (optional, default value=0) Some new formats are only supported in recent versions of SixTrack. In order to output in these formats a version number higher or equal to the number when it was implemented needs to be set. If the needed SixTrack version is 5.02.03 then the version should be set to 50203.

Important Notes:

- 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 before calling the SIXTRACK command.

- The BV flag is presently ignored by SIXTRACK.

- *SixTrack* and the MAD-X command SIXTRACK 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 SIXTRACK command always produces at least one output file:

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

It may also produce any or all of the following files, depending on the sequence and command attributes:

- fc.3 - multipole mask(s).

- fc.3.aux - various beam parameters.

- fc.3.aper - aperture element data (units are mm and degrees).

- fc.8 - misalignments and tilts.

- fc.16 - field errors and/or combined multipole kicks.

- fc.34 - various optics parameters at various locations
  This file is not needed by *SixTrack* but may be used as input to the program *SODD*[13].)

For a full description of these files see the *SixTrack* website[12], the *SixTrack* user manual[11]; and for information on running *S*ixTrack see the *SixTrack* run environment description[14].

Chapter 17. SXF file format

An SXF[15] lattice description is an ASCII listing that contains one named, "flat", ordered list of elements, delimited as {...}, with one entry for each element. The list resembles a MAD-X "sequence" describing the entire machine. The syntax is supposed to be adapted for ease of reading by human beings and for ease of parsing by LEX and YACC.

17.1 SXFWRITE

The command

```
SXFWRITE, FILE=filename;
```

writes the current 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.

17.2 SXFREAD

The command

```
SXFREAD, FILE=filename;
```

reads the file "filename" in SXF format, stores the sequence away and loads the sequence in memory through the USE mechanism in order to keep the existing errors.

It is therefore possible to write a lattice complete with errors to a named file and reload it later in a different MAD-X job:

```
!  define sequence MYSEQU
USE, mysequ;

!  add alignment errors and field errors

SXFWRITE, FILE = file;

STOP;
```

and later:

```
SXFREAD, FILE = file;
!  sequence mysequ is now reloaded and active, complete with errors.

TWISS;
...
```

Chapter 18. Plotting data

Values contained in `MAD-X` tables can be plotted directly in `MAD-X` in the form column versus column, with up to four differently scaled vertical axes and up to 10 different variables in total for all vertical axes.

If the horizontal axis is the position "s" of the elements in a sequence, a symbolic representation of the beamline can also be displayed at the top of the plot.

In certain conditions true interpolation of optical functions and parameters inside the element is available through internal slicing of the element and a call to the `TWISS` module for each slice.

The basic plot attributes, such as line thickness, annotation size, and PostScript format and interpolation can be set with the `SETPLOT` command and reset with the `RESPLOT` command.

Note also that for various reasons a sequence must be defined before the `PLOT` command can be invoked.

18.1   PLOT

```
PLOT, HAXIS=  string, HMIN=real, HMAX=real,
      VAXIS=  string1, ..., stringn,
      VAXIS1= string1, ..., stringn,
      VAXIS2= string1, ..., stringn,
      VAXIS3= string1, ..., stringn,
      VAXIS4= string1, ..., stringn,
      VMIN=reals, VMAX=reals,
      TITLE=string,
      BARS=integer, STYLE=integer,
      COLOUR=integer, SYMBOL=integer,
      INTERPOLATE=logical, ZERO_SUPPR=logical,
      NOVERSION=logical, NOLINE=logical, NOTITLE=logical,
      MARKER_PLOT=logical, RANGE_PLOT=logical, RANGE=range,
      TABLE=tabname, PARTICLE=particle1,particle2,...,particlen,
      MULTIPLE=logical, FILE=file_name_start, TRACKFILE=basename,
      PTC=logical, PTC_TABLE=tabname;
```

where the parameters have the following meaning:

HAXIS   name of the horizontal variable

HMIN, HMAX lower and upper edge for horizontal axis. Both values must be provided.

VAXIS   one or several variables from the table to be plotted against a single vertical axis. If more than 10 variables are specified, the plot is not produced.

VAXIS*i*   one or several variables from the table to be plotted against vertical axis number *i*. There is a maximum of 4 vertical axes.

If the number of variables given for a single vaxis*i* would push the total number of variables beyond the maximum of 10, the variables given for this vaxis*i*, as well as those for subsequent vaxes, are ignored but the plot is produced for the variables accumulated so far.

Important: VAXIS and VAXIS1..4 are exclusive in their application! if VAXIS is given, VAXIS1..4 will be simply ignored.

VMIN, VMAX lower and upper edge(s) for vertical axis or axes, up to four numbers separated by commas.

Note that both vmin and vmax must be given for an axis to be effective.

TITLE plot title string; if absent, the last overall title is used; if no such overall title as well, the sequence name is used.

BARS 0 (default) or 1 - with bars=1, all data points are connected to the horizontal axis with vertical bars.

STYLE 1 (default), 2, 3, or 4: line style for connecting the successive data points, respectively solid, dashed, dotted, and dot-dashed; the special value style=100 uses the four styles in turn for successive curves in the same plot. With style=0 successive data points are not connected.

COLOUR 1 (default), 2, 3, 4, or 5: colour for the symbols and lines, respectively black, red, green, blue, and magenta; The special value colour=100 uses the five colours in turn for successive symbols and lines.

SYMBOL 0 (default), 1, 2, 3, 4, or 5: The symbols to be plotted at data points, respectively none, dot (".."), plus ("+"), star ("*"), circle ("o"), and cross ("x"). The symbol size may have to be adapted through the SETPLOT command (see below).

Note that if symbol and style are both zero, style is forced to its default value (style=1) otherwise the plot would be invisible.

INTERPOLATE logical, default=false. The data points are normally connected by straight lines; if **INTERPOLATE** is specified, the following on-momentum Twiss parameters are interpolated with calls to the Twiss module inside each element: beta, sqrt(beta), alfa, phase advance, orbit, angle, dispersion and its first derivative, for both planes.

For all other variables splines are used to smooth the curves.

Note that setting this option is ineffective if the **INTERPOLATE** option of the SETPLOT command has been set to true; in this case all plots will be interpolated.

Note also that because the **INTERPOLATE** option causes **TWISS** to be called internally with a range, any range that might have been defined in a previous USE command is lost. In this case the USE statement must be reiterated with the range, which in turn could cause assigned errors to be lost.

Caution the attribute **INTERPOLATE** actually replaces expressions stored into element attributes by their value. It is plan to correct the problem, but it is not yet fixed.

ZERO_SUPPR To be documented (default = false)

NOVERSION   logical, default=false. If `NOVERSION=true`, the information concerning the version of `MAD-X` and the date of the run are suppressed from the title. This option frees additional space available for the user specified title of the plot.

NOLINE   logical, default=false. If `s` is the horizontal variable, then a symbolic representation of the beamline is plotted above the plot, except for tables that have been read back into `MAD-X`. In case the horizontal scale is too large and the density of beamline elements is too high, this may result in hardly legible representation and a thick black block in the worst case. The `NOLINE=true` option suppresses this symbolic representation of the beamline.

NOTITLE   logical, default=false. If true, suppresses the title line, including the information on the version and date.

MARKER_PLOT   logical, default=false. If true, plotting is done also at the location of marker elements. This is only useful for the plotting of non-continuous functions like the "N1" from the aperture module. Beware that the PS file might become very large if this flag is invoked.

RANGE_PLOT   logical, default=false. Allows the specification of a plotting range for the user defined horizontal axis.

RANGE   horizontal plot range given by elements.

TABLE   name of the table from which data is plotted (default: twiss).
If the first part of the name of the table is "track", the data to be plotted are taken from the tracking file(s) generated by a previous TRACK command for each requested particle as defined by the *particle* attribute. If the required file has not been generated by a preceding TRACK command, no plot is done for that particle.
The plot is generated through the GNUPLOT program and is available in the format specified by the SETPLOT command.
The preceding TRACK command should contain the attribute *DUMP* and may contain the attribute *ONETABLE*. The tracking plots are appended to the file file_name.ps where file_name can be specified via the attribute *file-name=file_name*. Note that the plots are appended to this file and the file is not overwritten.
The PLOT command uses the following tracking output files depending on the name of the table.
With the attribute *table=trackone*, the data file is assumed to have been generated with the *ONETABLE=true* attribute of the TRACK command, and the file name has the following format: *basisone* where the basis for the file name is defined by the attribute *trackfile=basis* (default=track).
With the attribute *table=trackxxx* where xxx is any string other than "one", the data files are assumed to have been generated with the *ONETABLE=false* attribute of the TRACK command, and the file names have the following format: *basis.obs0001.p00i* where the basis for the file name is defined by the attribute *trackfile=basis* (default=track), the observation point fixed is to 1 and the particle number *i* is given by the attribute *particle=i*.

PARTICLE   one or several numbers associated to the tracked particles for which the spec-
                  ified 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.

FILE   basename 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.
                  Note: in the case of several `PLOT` commands in a single `MAD-X` job, the first
                  `FILE` argument determines the basename and other `FILE` arguments in subse-
                  quent `PLOT` commands are ignored.

TRACKFILE   basename of the files containing tracking data for each particle (default: track)

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 pre-
                  viously 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.
                  This option is recommended when plotting data obtained from `PTC_TWISS` since
                  there is no mechanism for `PLOT` to physically interpolate the optical functions
                  beyond using splines with no mechanism for constraints with derivatives. In
                  most cases the `INTERPOLATE` option with data obtained with `PTC_TWISS` pro-
                  duces unphysical data representation.

PTC_TABLE   name of the ptc twiss table from which data is plotted (default: ptc_twiss)

18.2   SETPLOT

```
SETPLOT, POST= integer, FONT= integer, LWIDTH= real,
         XSIZE= real, YSIZE= real,
         ASCALE= real, LSCALE= real, SSCALE= real, RSCALE= real,
         INTERPOLATE= logical;
```

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.

INTERPOLATE (default=false) The data points are normally connected by straight lines; if
             INTERPOLATE is specified, the following on-momentum Twiss parameters are in-
             terpolated with calls to the Twiss module inside each element: beta, sqrt(beta),
             alfa, phase advance, orbit, angle, dispersion and its first derivative, for both
             planes.
             For all other variables splines are used to smooth the curves.
             Note that if INTERPOLATE=true, all subsequent PLOT commands use interpo-
             lation, irrespective of the setting of their INTERPOLATE attribute;
             If INTERPOLATE=false (default), all subsequent PLOT commands respect the
             setting of their INTERPOLATE attribute.

18.3 RESPLOT

```
RESPLOT;
```

resets all defaults for the SETPLOT command.

18.4 First example for plots of tracking data

The following MAD-X code sample defines the tracking of four particles with the generation of
a single file with name *basisone* holding the tracking data for all four particles.

```
// track particles
track, file=basis, dump, onetable;
 start, x= 2e-3, px=0, y= 2e-3, py=0;
 start, x= 4e-3, px=0, y= 4e-3, py=0;
 start, x= 6e-3, px=0, y= 6e-3, py=0;
 start, x= 8e-3, px=0, y= 8e-3, py=0;
 run,turns=1024;
endtrack;
```

The following sample code defines the plotting of the x-px and y-py phase space coordinates
for all four particles. It takes into account the fact that all coordinates are in a single file
with *table=trackone* and defines the filename where tracking data is to be found (*basisone*)
with *trackfile=basis*.

```
// plot trajectories
setplot, post=1;
title, "FODO phase-space test";
plot, file=plot, table=trackone, trackfile=basis, noversion, multiple,
      haxis=x, vaxis=px, particle=1,2,3,4;
plot, file=plot, table=trackone, trackfile=basis, noversion, multiple,
      haxis=y, vaxis=py, particle=1,2,3,4;
```

With each plot command a temporary file *gnu_plot.gp* containing GNUPLOT instructions is generated. The file generated by the first plot command reads:

```
set terminal postscript color
set pointsize 0.48
set output 'tmpplot.ps'
set title "FODO phase-space test"
set xlabel 'x'
set ylabel 'px'
plot 'basisone' using 3:($1==1 ? $4 : NaN) title 'particle 1' with points pointtype 1 , \
     'basisone' using 3:($1==2 ? $4 : NaN) title 'particle 2' with points pointtype 2 , \
     'basisone' using 3:($1==3 ? $4 : NaN) title 'particle 3' with points pointtype 3 , \
     'basisone' using 3:($1==4 ? $4 : NaN) title 'particle 4' with points pointtype 4
```

MAD-X then calls GNUPLOT as a subprocess to execute this file, which generates the file *tmpplot.ps*. The file *tmpplot.ps* is then **appended** to the file *plot.ps* determined by the attribute *file=plot*. The files *gnu_plot.gp* and *tmpplot.ps* are then discarded.

The same process is repeated for the second plot command, resulting in a growing file *plot.ps*.

18.5   Second example for plots of tracking data

The following MAD-X code sample defines the tracking of four particles with the generation of individual files with name *basis.obs0001.p000i* with *i=1..4* holding the tracking data for each of the four particles.

```
// track particles
track, file=basis, dump;
 start, x= 2e-3, px=0, y= 2e-3, py=0;
 start, x= 4e-3, px=0, y= 4e-3, py=0;
 start, x= 6e-3, px=0, y= 6e-3, py=0;
 start, x= 8e-3, px=0, y= 8e-3, py=0;
 run,turns=1024;
endtrack;
```

The following sample code defines the plotting of the x-px and y-py phase space coordinates for all four particles with the data of all four particles on a single plot. It takes into account the fact that coordinates for all four particles are in separate files with *table=trackfodo* and defines the filename where tracking data is to be found (*basis.obs0001.p000i*) with *trackfile=basis*.

```
// plot trajectories
setplot, post=1;
title, "FODO phase-space test";
plot, file=plot, table=trackfodo, trackfile=basis, noversion, multiple,
      haxis=x, vaxis=px, particle=1,2,3,4;
plot, file=plot, table=trackfodo, trackfile=basis, noversion, multiple,
      haxis=y, vaxis=py, particle=1,2,3,4;
```

With each plot command a temporary file gnu_plot.gp containing GNUPLOT instruction is generated. The file generated by the first plot command reads:

```
set terminal postscript color
set pointsize 0.48
set output 'tmpplot.ps'
set title "FODO phase-space test"
set xlabel 'x'
set ylabel 'px'
plot 'basis.obs0001.p0001' using 3:4 title 'particle 1' with points pointtype 1 , \
     'basis.obs0001.p0002' using 3:4 title 'particle 2' with points pointtype 2 , \
     'basis.obs0001.p0003' using 3:4 title 'particle 3' with points pointtype 3 , \
     'basis.obs0001.p0004' using 5:4 title 'particle 4' with points pointtype 4
```

`MAD-X` then calls GNUPLOT as a subprocess to execute this file, which generates the file *tmpplot.ps*. The file *tmpplot.ps* is then **appended** to the file *plot.ps* determined by the attribute *file=plot*. The files *gnu_plot.gp* and *tmpplot.ps* are then discarded.

The same process is repeated for the second plot command, resulting in a growing file *plot.ps*.

18.6   MAD-X PLUGINS

`MAD-X` provides a plug-in mechanism for functionality extensions. Plug-in technique does not require linking at build time. The job is done at the time plug-in is loaded by the dynamic linker. In order to use any plug-in, the plugin support must be compiled in `MAD-X`. At the top of every `MAD-X` makefile there is variable PLUGIN_SUPPORT that must be set to "YES". Then, the appropriate C interface is compiled in and `MAD-X` is linked dynamically. Plug-ins must be accessible to the dynamic linker. The default location where plug-ins are searched is $HOME/.madx/plugins. Otherwise, the directory containing the library must be either listed in

1. the configuration file of the dynamic linker (on SLC3 it is /etc/ld.so.conf)

2. `LD_LIBRARY_PATH` environment variable

Existing plug-ins

1. `RPLOT`

Example

PROGRAMMERS MANUAL

The interface is not yet fully defined. The documentation apears at the moment it happens.

18.7 RPLOT

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 provides 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.

Installation

Prerequisite: ROOT must be installed before 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...

Part IV

MAD-X Modules

Chapter 19. SURVEY

The `SURVEY` command computes the geometrical layout, *i.e.* 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.

```
SURVEY, SEQUENCE=string, FILE= string, PERM_ALIGN_SURVEY= logical
        X0=real, Y0=real, Z0=real,
        THETA0=real, PHI0=real, PSI0=real;
```

The `SURVEY` command has the following attributes:

`SEQUENCE`   the name of sequence to be surveyed [Note: this attribute is actually ignored in favor of previous `USE`]. By default the last sequence expanded with the `USE` command will be surveyed.

`FILE`   the name of external file to which the results are written.
(Default: survey)

`PERM_ALIGN_SURVEY` If TRUE the output includes the permanent misalignments and the start of each element ends with a .ENT and the exit of each element with .EXI.
(Default: FALSE)

`X0, Y0, Z0`   the initial horizontal, vertical and longitudinal coordinates in meters.
(Default: 0.0, 0.0, 0.0)

`THETA0, PHI0, PSI0` the initial horizontal and vertical angles and transverse tilt in radians.
(Default: 0.0, 0.0, 0.0)

The computation results are written on the internal table named "survey". Results can also be written on an external file. Each line contains the global coordinates of an element taken at the end of the element.

The computation takes into account the length of each element, as well as the rotation angles defined for `SBEND`, `RBEND`, thin `MULTIPOLE` and thin `RFMULTIPOLE` elements exclusively. Rotation angles introduced via the `KNL`, `KSL` mechanism for other elements are ignored by `SURVEY`, other `MAD-X` commands, as well as `PTC` commands.

Example: average LHC ring with CERN coordinates.

```
REAL CONST R0 = 1.0; !  to obtain the average ring
USE, SEQUENCE=lhcb1;
SURVEY, X0=-2202.21027, Y0=2359.00656, Z0=2710.63882,
        THETA0=-4.315508007, PHI0=0.0124279564, PSI0=-0.0065309236,
        FILE=survey.lhcb1;
```

WARNING :
In the case a machine geometry is constructed with thick lenses, the circumference changes when the structure is converted into thin lenses via the `MAKETHIN` command. This is an

unavoidable feature of `MAKETHIN`. ONLY the structure with thick lenses must be used for practical purposes.

Chapter 20. Twiss Module

The `TWISS` command calculates the linear lattice functions [3], and optionally the chromatic functions. The coupled functions are calculated in the sense of Edwards and Teng[16]. For the uncoupled cases they reduce to the C and S functions.

```
TWISS, SEQUENCE=seqname, LINE=linename, RANGE=range,
       DELTAP=real{,real}||initial:final:step,
       CHROM=logical, RMATRIX=logical, TAPERING=logical,
       CENTRE=logical, TOLERANCE=real,
       FILE=filename, TABLE=tabname, NOTABLE=logical,
       SECTORMAP=logical, SECTORTABLE=tabname,
       SECTORFILE=filename, SECTORPURE=logical,
       EIGENVECTOR=tabname, EIGENFILE=filename,
       KEEPORBIT=name, USEORBIT=name,
       COUPLE=logical, EXACT=logical,
       RIPKEN=logical, TAPERING=logical
   ;
```

The attributes of the `TWISS` command are:

SEQUENCE   the name of a valid sequence for which the calculation of optical functions should be performed.
            `SEQUENCE` and `LINE` are mutually exclusive.
            (Default: sequence or beam line defined in the latest `USE` command)

LINE   the name of a valid beamline for which the calculation of optical functions should be performed.
            `SEQUENCE` and `LINE` are mutually exclusive.
            (Default: sequence or beam line defined in the latest `USE` command)

RANGE   (Default: `#S/#E`)
            The `TWISS` calculation is restricted to the specified range. See `RANGE`.

DELTAP   =`real{,real}` or `initial:final:step` (Default: 0.0)
            The relative energy error `DELTAP` may be entered in one of the two forms above. 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.
            For example, `DELTAP=0.001` defines a single value, `DELTAP=0.001,0.005` defines two values and `DELTAP=0.001:0.007:0.002` defines four values.

CHROM   a logical flag to trigger computation of the chromatic functions as well as the radiation synchrotron integrals.
            Please note that the calculation is done without taking coupling into account. In case of coupled lattice the warning isissued that the calculation of chromatic functions can be inaccurate!
            Please note that this option also changes the way that the chromaticities are calculated: The chromaticities are normally calculated from the analysis of

the first and second order matrices. With `CHROM`*, the chromaticities are re-calculated by explicitly calculating the tunes for the case of the specified momentum deviation* `DELTAP` *and for the case of a momentum deviation equal to* `DELTAP+1.e-6`*. The tune differences divided by 1.e-6 yield the chromaticities.*

CENTRE a logical flag to enforce the calculation of the linear lattice functions at the center of the element instead of the end of the element. The values in the tables and in the output files are affected by this flag.
(Default: false)
Since the lattice functions are calculated inside the element the closed orbit coordinates in the output also include the misalignment of the element.

TOLERANCE the maximum closed orbit error, for all six orbit components, that can be tolerated during the closed orbit search. The value given in the `TWISS` command is only valid for the current calculation; the `COGUESS` command allows to change the default value for all subsequent closed orbit search calculations.
(Default: 1.e-6)

FILE causes `MAD-X` to write a TFS Twiss table to the file specified. (Default: "twiss")
The columns of the table can be selected using the `SELECT` command with the `FLAG=twiss` attribute.

TABLE the name of the table where linear lattice functions as well as chromatic functions are stored. (Default: "twiss")
Note: If the `TABLE` option is given the selection of column names via the `SELECT` command is ignored. Hence if both `TABLE` and `FILE` options are given, the table written to file is the full twiss table, containing all elements as rows and all known Twiss parameters as columns.

NOTABLE logical flag to prevent the creation of the internal twiss table. Consequently, no output file is created either.
(Default: false)

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 a logical flag to initiate the calculation of a sector map. Default the Rij contains feed-down from higher order maps. In order to turn it off use the flag SECTORPURE.

SECTORACC a logical flag to save composition of maps instead of individual maps of a sector map.

SECTORPURE a logical flag to save the transfer map Rij without effects from higher order map (Tijk). This option should be used to get it in the correct format for TRAIN.

SECTORTABLE the name of the table containing the `SECTORMAP` values. The elements (lines) and parameters (columns) of the table can be tailored using the `SELECT` com-

mand as specified in SECTORMAP
(Default: sectortable)

SECTORFILE   the name of the file to which the SECTORMAP is written. In order to specify the
             output use the SECTORTABLE as specified in SECTORMAP
             (Default: "sectormap")

EIGENVECTOR a logical flag to output the eigenvectors in the beginning of the sequence.
             Multiplying with this matrix brings normalized coordinates to physical coor-
             dinates.

EIGENFILE   the name of the file to which the EIGENVECTOR is written.

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 MAD-8 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 [16]. For the uncoupled cases they reduce to
             the C and S functions.

EXACT   If this is used the dirft is expanded around the actual closed orbit instead of
             the ideal orbit. (Default: false)

             Twiss calculation is 4D only! : The TWISS command calculates an approxi-
             mate 6D closed orbit when the accelerator structure includes an active cavity.
             However, the calculation of the Twiss parameters are 4D only. This may result
             in apparently non-closure of the beta values in the plane with non-zero disper-
             sion. The full 6D Twiss parameters can be calculated with the PTC_TWISS
             command.

RIPKEN   This flag calculates the Ripken-Mais Twiss parameters (beta11, beta12,
             beta21, beta22, alfa11, alfa12, alfa21, alfa22, gamma11, gamma12,
             gamma21 and gamma22) using the parameters betx, bety, alfx, alfy,
             gamax, gamay, R11, R12, R21 and R22 as input.

TAPERING   Adjust the strengths of the quadrupoles and sextupoles in order to compensate
             for the offset in energy. This flag triggers a call to the TAPER command with
             default parameters and no output file.

The tables are suitable for PLOT.

After a successful TWISS run MAD-X creates a table of summary parameters named "SUMM"
which includes tunes, chromaticities, etc. versus the selected values of DELTAP. Please note
that with the CHROM attribute the calculation is done without taking coupling into account.
In case of coupled lattice the warning isissued that the calculation of chromatic functions can
be inaccurate!

Notice also that in `MAD-X`, `DELTAP` is converted to `PT`, which is used as longitudinal variable. Dispersive and chromatic functions are hence derivatives with respect to `PT`. (see summ table).

These summary parameters can later be accessed via the table access functions using the "SUMM" table.

The direction of the beam, as set via BV-flag, is considered in the sign of the magnetic field strengths in the table.

20.1 Twiss Parameters for a Period

The simplest form of the `TWISS` command is

```
TWISS;
```

which calculates the periodic solution for the last beamline or sequence declared in a `USE` statement, and with zero `DELTAP`. Chromatic functions are not calculated. Standard tables ("TWISS" and "SUMM") are created in memory but no file is written to disk.

The slightly more elaborate version

```
TWISS, DELTAP=real{,real}, CHROM, TABLE=tabname;
```

computes the periodic solution, including chromatic functions, for the last beam line or sequence declared in a `USE` statement, for all values of DELTAP entered (or for `DELTAP=0`, if none is entered). The tables "tabname" and "SUMM" are created in memory and no file is written to disk.

Example:

```
USE, period=OCT;
TWISS, DELTAP=0.001, CHROM;
```

computes the periodic solution for the linear lattice and chromatic functions for the beam line OCT and for DELTAP=0.001.

20.2 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{,real}, LINE=beamline,
       MUX=real, MUY=real,
       TABLE=tabname;
```

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=beamline: 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 beamline CELL: Output is also written on the file TWISS:

If any of the beam lines was defined with formal arguments, actual arguments must be pro-
vided:

```
CELL(SF,SD): LINE=(...);
INSERT(X):   LINE=(...);
USE, period=INSERT;
TWISS, LINE=CELL(SF1,SD1);
```

20.3   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, DDY=real, DDPX=real, DDPY=real,
       R11=real, R12=real, R21=real, R22=real, !coupling matrix
       TABLE=tabname,
       TOLERANCE=real,
       DELTAP=real: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 (see below). The lattice parameters are taken from this block, but are also over-
written by lattice parameters explicitly decalred on the command line. As entered, the initial
conditions cannot depend on `DELTAP`, and can thus be correct only for one such value.

20.4   SAVEBETA

Initial lattice parameters can be saved and transfered for later commands, in particular for
TWISS or the match module, with the `SAVEBETA` command sequence.

```
SAVEBETA, LABEL=string, PLACE=string, SEQUENCE=sequencename;
```

marks a location given by attribute `PLACE` in an expanded sequence `sequence_name`; at the next `TWISS` command execution, a `BETA0` block is saved at that location with the label given by the attribute `LABEL`. This is done only once; in order to get a new `BETA0` block at the same location in a subsequent `TWISS` command, the `SAVEBETA` command must be repeated. The content 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;
```

saves and then shows the `BETA0` block parameters at the end (!) of the fifth element of type `mb` in the sequence.

Parameters in tables can also be accessed using the table access functions.

```
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 saves 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 tracks lattice parameters through `INSERT`, using all end conditions (including orbit) in `CELL` at the start of `INSERT`.

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).

20.5 BETA0: Set Initial Lattice Parameters

Initial lattice parameters can be set 'manually' for later commands, in particular for `TWISS` or the `MATCH` module, by using the `BETA0` command attached to a label.

```
label:  BETA0, BETX=real, ALFX=real, MUX=real,
               BETY=real, ALFY=real, MUY=real,
               {etc for linear and chromatic lattice functions};
```

A `BETA0` block can be used as a whole with all values declared, as a block with overriden values explicitly, or by extracting single values as shown in the three examples below:

Example of `BETA0` block used as a whole in `TWISS`:

```
initial:  BETA0, BETX=10., ALFX=0.0, MUX=0.0,
                 BETY=10., ALFY=0.0, MUY=0.0,
                 DX=1., DPX=0.0;
TWISS, BETA0=initial;
```

Example of `BETA0` block used as a whole but with overriden values in the `TWISS` command:

```
initial:  BETA0, BETX=10., ALFX=0.0, MUX=0.0,
                 BETY=10., ALFY=0.0, MUY=0.0,
                 DX=1., DPX=0.0;
TWISS, BETA0=initial, ALFX=-0.1, ALFY=0.1;
```

Example of using `BETA0` block by extracting single values in the `TWISS` command:

```
initial:  BETA0, BETX=10., ALFX=0.0, MUX=0.0,
                 BETY=10., ALFY=0.0, MUY=0.0,
                 DX=1., DPX=0.0;
TWISS, BETX=initial->BETX, BETY=initial->BETY;
```

20.6   Sectormap output

The flag `SECTORMAP` of the `TWISS` command 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` (6×6 values) and of the second-order map `T` ($6 \times 6 \times 6$ values) If only certain elemenents and values are desired a SELECT command can be used. Note that it should be used on the TABLE as shown in the following example:

```
TWISS,,SECTORMAP,SECTORTABLE=my_sect_table; SELECT, FLAG=my_sect_table, COLUMN=name, p
```

The sectormap file contains for each selected element, one element per line, the set of chosen K, R, and 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
...	...	...
R16	...	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. Note that effects of the higher order are included in the Rij. In order to disable this use the flag SECTORPURE. This flag should be activated when used to interface TRAIN.

20.7   Beam Threader

In a machine with magnetic and alignment errors it can happen that the beam does not circulate and the closed orbit cannot be established and measured. This can also happen in `MAD-X` and the closed orbit finder does not converge.

The `THREADER` simulates beam steering through such a machine with repeated measurement of trajectory over a certain number of monitors and correction of the trajectory with upstream correctors.

When enabled, threading is executed whenever a trajectory or closed orbit search is carried out by the TWISS module.

The threader is controlled as an option. The following `MAD-X` command enables the threader:

    OPTION, THREADER;

and the threader can be disabled with

    OPTION, -THREADER;

During the trajectory search in TWISS, or the first turn of the orbit search for a closed machine, the threader calculates at each monitor the difference between the present trajectory reading and a reference value. If the difference exceeds a threshold (see below), the threader searches backwards for the first corrector that will efficiently cancel the difference and calculates the corresponding kick. The trajectory is then recalculated starting again from that corrector and progressing forward. The calculated kicks are added to already existing kicks. If Twiss is searching for a closed orbit which involves tracking this trajectory over many turns, the threader is only active during the first turn.

The reference value for the trajectory difference is zero by default but can also be obtained from a previous orbit calculation if the current `TWISS` command has the `USEORBIT` flag enabled and a previous `TWISS` command had the `KEEPORBIT` flag enabled. This allows for example to thread the beam into a machine with orbit bumps present.

The threshold values for triggering the threader correction can be set with the command

    THREADER, VECTOR={xmax, ymax, att};

where

xmax, ymax   threshold orbit excursions beyond which the threader is applied.
              Defaults: 0.005, 0.005

att   attenuation factor for the kicks applied by the threader
              Default: 1.0

The attenuation factor defines the fraction of the calculated kick that is actually applied by the threader. An attenuation factor of 0.5 will apply 50% of the calculated kicks.

20.8 Closed Orbit Guess

In order to help the initial finding of the closed orbit by the `TWISS` module, it is possible to specify an initial guess for the coordinates of the fixed point at the start of the lattice.

```
COGUESS, X=real, PX=real, Y=real, PY=real, T=real, PT=real,
         TOLERANCE=real,
         CLEAR=logical;
```

The `COGUESS` command has the following attributes:

X, PX, Y, PY, T, PT each parameter specified defines a first guess for all future closed orbit searches in case they are different from zero.

TOLERANCE sets the required convergence precision in the closed orbit search. (Default: 1.e-6)

CLEAR a flag to reset the tolerance to its default value and to cancel the effect of a previous `COGUESS` in defining a first guess for subsequent closed orbit searches. (Default: false)

Example

```
COGUESS, X=1.e-3;
...
TWISS;
...
COGUESS, Y=-2.e-3;
...
TWISS;
...
COGUESS, CLEAR;
...
TWISS;
...
```

20.9 Coupling

In order to calculate Twiss parameters (β, α and ϕ) MAD-X uses Edwards-Teng approach.

For coupled motion the matrix transformation can be written in a form $M = VUV^{-1}$, where M is a symplectic 1-turn map, U is a symplectic decoupled block-diagonal matrix, V is a symplectic "rotation" matrix. Initial coupling parameters are calclulated based on decoupled U matrix.

Given that M_1 is the one-turn matrix at point 1, M_2 is the one-turn matrix at point 2 , and we know M_{12}, the transfer matrix between points 1 and 2, we can express the change of the one-turn map from point 1 to point 2 by $M_2 = M_{12} M_1 M_{12}^{-1}$. Then $U_2 = V_2^{-1} M_2 V_2 = V_2^{-1} M_{12} M_1 M_{12}^{-1} V_2 = (V_2^{-1} M_{12} V_1) U_1 (V_1^{-1} M_{12}^{-1} V_2)$.

So, $U_2 = W_{12} U_1 W_{12}^{-1}$, where $W_{12} \equiv V_2^{-1} M_{12} V_1$ Knowing V_1 and M_{12} (transfer matrix of the element), we can calculate $V_2 W_{12} = M_{12} V_1$, where we use V_2 for computing the coupling parameters and W_{12} to propagate to the end of the element. One can find V_2 such that W_{12} is block-diagonal or off block-diagonal.

For weakly coupled lattice MAD-X always uses **block-diagonal** solution, i.e. large tune (mode1) is associated with x-plane, and smaller tune (mode2) is associated with y-plane by convention. For a highly coupled lattice, it may happen at some places in the lattice that mode1 is associated with y-plane, and mode2 with x-plane. Switching the association of mode1 and mode2 with x and y-planes as one propagates the twiss parameters through the lattice is called "modes flip" (use **off-block diagonal** solution for W_{12}). In case the modes flip is triggered, there is an additional stability check, either the calculation of twiss parameters can be done with modes flip or not.

20.10   Intermediate values

The `SELECT, FLAG=interpolate` command allows to select locations for output of intermediate values for the `TWISS` command. If interpolation points have been selected for an element, the `TWISS` command will add the values at these locations to the TFS table rather than at the end of the element.

```
SELECT, FLAG=interpolate, SEQUENCE=string, RANGE=string, CLASS=string,
        CLEAR,
        SLICE=integer,
        STEP=real,
        AT={real, ...};
```

Note: This feature may behave unexpectecly with `CENTRE`.

Selected elements can optionally be filtered based on sequence, range and class. Every selection specifies exactly one of the following actions which overrides previous selections for the matching elements:

CLEAR   remove interpolation positions.

SLICE   slice using a fixed number of slices.

STEP   slice at every `STEP` meters.

AT   slice at given locations (as fraction of the node length).

Chapter 21. Matching Module

Before a match operation at least one sequence must be selected by means of a `USE` command. Matching is 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-X accepts the matching commands listed below. For a mathematical description of the minimisation procedures see [17].

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`: Initiate Matching Mode

  – `ENDMATCH`: Leave Matching Mode

- Define Variable Parameter

  – `VARY`: Set Parameter to Vary

- Define 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`

21.1   MATCH . . . ENDMATCH

The `MATCH` command is used for matching either a periodic cell or an insertion to another part of the machine.

Both matching modes are initiated by the `MATCH` command using one of several forms outlined below.

The `ENDMATCH` command terminates the matching section and deletes all tables related to the matching run.

```
ENDMATCH;
```

21.2   Cell Matching

The matching of a periodic cell is initiated by a `MATCH` command of the form:

```
MATCH, SEQUENCE='name1', 'name2', ..., 'name-n',
       SLOW=logical;
```

In this 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', ...,'name-n';
```

21.3   Insertion Matching

The matching of an insertion to another part of the machine is initiated by a `MATCH` command taking one of the following forms:

```
MATCH, SEQUENCE= 'name1', 'name2', ..., 'name-n',
       BETA0= 'beta01', 'beta02', ..., 'beta0n',
       SLOW=logical;
```

or

```
MATCH, SEQUENCE='seqname',
       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,
       DELTAP=real, SLOW=logical;
```

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='seqname',
           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;
```

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.

21.4   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 functions 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.

21.5   SLOW attribute

The `SLOW` logical flag enforces the old and slow matching procedure which allows to use the special columns `mvar1, ..., mvar4`, if they are added to the twiss table.

Recently a number of parameter, like `RE56`, have been added to the list of parameters that can be matched.

Nevertheless, some parameters might only be available when using the `SLOW` attribute.

21.6   Useful TWISS attributes

Some of the attributes of the `TWISS` command can be used in the `MATCH` command and are transmitted to the `TWISS` command when it is internally invoked during the matching process.

The main `TWISS` attributes that can be used also in the `MATCH` command 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.

CHROM   This logical flag sets the matching process to transmit the `CHROM` attribute to the `TWISS` command when it is invoked, enforcing the calculation of chromatic functions and synchrotron radiation integrals, and the alternative calculation of chromaticities as documented in `TWISS`.

    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 repective keywords `WX, PHIX, WY, PHIY,...` for the chromatic functions.

TAPERING   Adjust the strengths of the quadrupoles and sextupoles in order to compensate
              for the offset in energy. This flag triggers a call to the `TAPER` command with
              default parameters and no output file.

Examples:

```
MATCH, RMATRIX, SEQUENCE='name', BETA0='beta-block-name';
CONSTRAINT, SEQUENCE=insert, RANGE=#e, RE11=-2.808, RE22=2.748;
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.

21.7 VARY

A parameter to be varied is specified by the command

```
VARY, NAME=variable, STEP=real, LOWER=real, UPPER=real
      SLOPE=integer, OPT=real;
```

It has six 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-X` 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`) or 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 than 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.

21.8   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=name1/name2`

- the optics functions to be constrained.

The optic functions can be constrained in four different ways:

1. lower limit: `BETX > value`

2. upper limit: `BETX < value`

3. lower and upper limits: `BETX < value1, BETX > value2`

4. target value: `BETX=value`

In case one element is affected by 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 100 meter at element 'name1':

```
CONSTRAINT, SEQUENCE=seqname, RANGE='marker1'/'marker2', BETX<200.0;
CONSTRAINT, SEQUENCE=seqname, RANGE='name1'/'marker2', BETX=100.0;
```

When the two constraint statements are interchanged, and supposing that name1 is an element in the range marker1/marker2, 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=seqname, 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=seqname, 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.

21.9 User Defined Matching Constraints

In addition to the nominal `TWISS` variables, the user can define a limited set of 'user-defined' variables in the constraint statement. This allows, for example, the matching of the normalized 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 expressions. 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 details in the TWISS module and SELECT statement):

```
SELECT, FLAG=twiss, CLEAR;
SELECT, FLAG=twiss, COLUMN=keyword, name, s, betx, dx, mvar1;
TWISS, SEQUENCE=seqname, FILE=twissfile;
```

The variable can now be referenced like any other `TWISS` variable in the constraint command:

```
CONSTRAINT, SEQUENCE=seqname, RANGE=range, MVAR1=targetvalue;
```

21.10 GLOBAL

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=seqname,
        Q1=real, DQ1=real,
        Q2=real, DQ2=real;
```

All attributes are optional and have the following meaning:

SEQUENCE   the name of the sequence on which to operate the matching.

Q1, Q2   enable a correction of tunes in presence of magnetic imperfections or misalignments

DQ1, DQ2   enable a correction of chromaticities in presence of magnetic imperfections or misalignments

21.11   WEIGHT, GWEIGHT

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 default values listed in the table below, and may be reset at any time to different values. Values set in this way remain valid until changed again.

The WEIGHT command changes the weights for subsequent constraints:

```
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;
```

The weights are entered with the same name as the linear lattice functions and orbit coordinate to which the weight applies. Frequently the matching weights serve to select a restricted set of functions to be matched.

Matching weights for global matching constraints can be set by the GWEIGHT command, having attributes identical to those of GLOBAL.

```
GWEIGHT, Q1=real, DQ1=real,
         Q2=real, DQ2=real;
```

Default values for matching weights are given in the table below.

21.12   Matching Methods

MAD-X currently supports four different matching algorithms each associated to a command with its own attributes.

Table 21.1: Default Matching Weights

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	10.0	PT	100.0		
DX	10.0	DPX	100.0		
DY	10.0	DPY	100.0		
WX	1.0	PHIX	1.0	DMUX	1.0
WY	1.0	PHIY	1.0	DMUY	1.0
DDX	1.0	DDPX	1.0		
DDY	1.0	DDPY	1.0		
MVARi	10.0				
Q1	10.0	DQ1	1.0		
Q2	10.0	DQ2	1.0		

21.12.1 LMDIF: Fast Gradient Minimisation

The `LMDIF` command minimises the sum of squares of the constraint functions using their numerical derivatives. It is the fastest minimisation method available in `MAD-X`.

```
LMDIF, CALLS=integer, TOLERANCE=real;
```

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: 1E-6)

21.12.2 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=integer;
```

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: 1E-6)

`STRATEGY`   specifies the strategy to be used. (Default: 1)
Details are given in [17].

21.12.3 SIMPLEX: Simplex Minimisation

The `SIMPLEX` command minimises the penalty function by the simplex method. Details are given in [17].

```
SIMPLEX, CALLS=integer, TOLERANCE=real;
```

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: 1E-6)

21.12.4   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 folowing attributes:

CALLS   The maximum number of calls to the penalty function. (Default: 30)

TOLERANCE   The desired tolerance for the minimum. (Default: 1E-6)

REPEAT   The number of calls of the JACOBIAN routine. (Default: 1)

BISEC   Selects the maximum number of iteration used to determine the step length which reduces the penalty function during the main iteration. A large number (i.e. 6) reduces the probability to diverge from the solution, but increases the probability of being trapped in a local minimum.

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, minimum and 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.

21.13 USE_MACRO

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, weight=2, 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 left hand side (lhs) and the right hand side (rhs) of 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.

21.13.1 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.

21.13.2 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

21.13.3   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 a set of constraints should be defined, with the following syntax for the `CONSTRAINT` command:

    CONSTRAINT, expr = "lhs = rhs";
    CONSTRAINT, expr = "lhs < rhs";
    CONSTRAINT, expr = "lhs > rhs";

where "lhs" and "rhs" are well defined `MAD-X` expressions. Another set of macro and constraints can be defined afterwards.

21.13.4   Examples

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;
```

21.14 Matching Examples

All matching examples and the related files for executing the `MAD-X` sample jobs can be found in the examples directory (https://github.com/MethodicalAcceleratorDesign/madx-examples) of the `MAD-X` website.

- 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

- Matching 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

- Matching s position using VLENGTH flag.
  match the positions of elements and the total sequence length for a simple sample sequence. s position matching

- Matching 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

Chapter 22. EMIT: Equilibrium emittances

`EMIT` calculates the equilibrium emittances:

> `EMIT, DELTAP=real, TOL=real;`

The attributes for the `EMIT` command are:

`DELTAP`   the average energy error.
`EMIT` adjusts the RF frequencies in order to obtain this average energy error: the revolution frequency f_0 is determined for a fictitious particle with constant momentum error `DELTAP` $= \delta_s = \delta(E)/p_s c$ travelling along the design orbit. The RF frequencies are then set to $f_{RF} = h f_0$.

`TOL`   The tolerance attribute is for the distinction between static and dynamic cases: 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.
(Default: 1.000001)

Output:
Traditionally the output from the `EMIT` module has only been shown in the standard output. However, since 5.08.00 there output is also stored in the table : emit and emitsumm. They can be written to file using the normal table commands. **Example:**

```
Emit, DELTAP=0.0 ;
write, table=emit, file="emittab.tfs" ;
write, table=emitsumm, file="emitsums.tfs" ;
```

If the machine contains at least one RF cavity, and if synchrotron radiation is enabled with `BEAM, RADIATE=true;`, the `EMIT` command computes the equilibrium emittances and other electron beam parameters using the method in [18].

In this calculation the effects of quadrupoles, sextupoles and octupoles along the closed orbit are also considered. Thin multipoles are used only if they have a fictitious length `LRAD` different from zero.

If the machine does not contain any RF cavity, if synchrotron radiation is turned off (`BEAM, RADIATE=false;`, or if the longitudinal motion is not stable, `EMIT` only computes the parameters that are not related to radiation and does not update the `BEAM` values.

If synchrotron radiation is enabled (`BEAM, RADIATE=true;`, and the `DELTAP` attribute is zero, and the longitudinal motion is stable, `EMIT` calculates and updates the following values for the `BEAM` attached to the current sequence: both geometric and normalized transverse emittances, longitudinal emittance and beam sizes (σ_E and σ_t), damping partition numbers, energy loss per turn and synchrotron tune.

Example:

```
RFC: RFCAVITY, HARMON..., VOLT=...;
...
```

```
BEAM, ENERGY = 100.0, RADIATE=true;
EMIT, DELTAP = 0.01;
```

Remark:
This module assumes nearly constant lattice functions inside elements. This assumption works for many machines, like LEP but it fails when the lattice functions largely vary inside single elements. In the later case it is advised to slice the elements.

Chapter 23. Physical Aperture

Physical apertures can be defined and associated to most elements in `MAD-X`.

The `APERTURE` command calculates the beam stay clear values (n1 values).

During tracking the particle excursion can be checked against the available aperture, and the particle lost if it falls outside the defined aperture.

23.1 Aperture definition

The aperture for a particular element or class of elements can be set in `MAD-X` at the time of definition or instantiation of the element or class.

Note that instantiation can also happen at the time of declaration of an element in a sequence.

The aperture can be specified for any element or class of elements, with the exception of drift spaces.

The definition of the aperture takes the following form and parameters:

```
..., APERTYPE=string, APERTURE={values},
     APER_OFFSET={values}, APER_TOL={values},...
```

where the four specific attributes are attributes of an element declaration or instantiation.

The minimum aperture definition uses the following two attributes:

APERTYPE   defines the aperture type from a set of preselected types, or from a file if a filename is provided as argument. The preselected types are:

     CIRCLE

     RECTANGLE

     ELLIPSE

     RECTCIRCLE   a superposition of a `CIRCLE` and a `RECTANGLE`

     LHCSCREEN   an alias for `RECTCIRCLE`

     RECTELLIPSE   a superposition or intersection of an `ELLIPSE` and a `RECTANGLE`

     RACETRACK   the union of an `ELLIPSE` and a `RECTANGLE` forming a `RACETRACK` shape with four quarters of ellipse, one per quadrant, connected by straight lines

     OCTAGON   a simply connex octagon

     If a filename is provided as argument, the file must contain a list of x and y coordinates, one pair per line, outlining the complete aperture shape, i.e. no symmetry is assumed. In order to speed up the tracking it is also possible to provide an rectangle that is checked before the arbitrary polygon is checked.

If the rectangel is bigger than the polygon the arbitrary shape is not checked. When a filename is provided as argument, the `APERTURE` attribute is ignored.

APERTURE   is an array of values, the number and meaning of which depends on the `APERTYPE`.

Table 23.1: Defintion of aperture types

APERTYPE	**# of values**	**meaning of values**
`CIRCLE`	1	radius of circle
`RECTANGLE`	2	half width and half height of rectangle
`ELLIPSE`	2	horizontal and vertical semi-axes of ellipse
`RECTCIRCLE` or `LHCSCREEN`	3	half width and half height of rectangle, radius of circle
`RECTELLIPSE`	4	half width and half height of rectangle, horizontal and vertical semi-axes of ellipse
`RACETRACK`	4	full half width and full half height along main axes, and horizontal and vertical semi-axes of ellipse for the rounding of the corners
`OCTAGON`	4	half width and half height along main axes, two angles sustaining the cut corner in the first quadrant, given in radians and in order of increasing values.
`"FileName"`	2	In addition to the points in the file which are defining geometry it is also possible add two parameters to APERTURE attribute. These values are used for a check identical to the RECTANGLE check.

Examples

The following statements are equivalent in setting an elliptic aperture using the variables `APH` and `APV`:

```
..., APERTYPE=ellipse, APERTURE={APH, APV};
..., APERTYPE=rectellipse, APERTURE={APH, APV, APH, APV};
..., APERTYPE=racetrack, APERTURE={APH, APV, APH, APV};
```

Similarly the following statements are equivalent in setting a rectangular aperture using the variables `APX` and `APY`, and `APR:=SQRT(APX**2+APY**2)`:

```
..., APERTYPE=rectangle, APERTURE={APX,APY};
..., APERTYPE=rectellipse, APERTURE={APX, APY, APR, APR};
..., APERTYPE=racetrack, APERTURE={APX, APY, 0.0, 0.0};
```

When the aperture is described in a file, the only parameter to be provided is a filename given as value to the `APERTYPE` attribute:

```
MB: SBEND, L=l.MB, APERTYPE="MB-aper.txt", APERTURE={APH, APV} ;
```

where `"MB-aper.txt"` contains the list of x-y coordinates defining the aperture shape:

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

and `APX` and `APY` are the settings for the rectangle that is checked first. The rectangle parameters are optional and are only used to speed up the aperture checks in tracking.

There is also another way to define an arbitrary polygon aperture using the attributes `APER_VX` and `APER_VY`. They can also be defined in combination standard aperture defintions. The standard aperture definitions are then checked first and in case it is outside the standard definition of aperture the arbitrary is checked. An example of how to use it can be found: here. **Notes**

- There are inconsistencies in the parameter definition for the different aperture types. This is historical and is kept for backwards compatibility. Pay some attention to the parameters you introduce!

- It is considered good practice to always set the first two `APERTURE` parameters for predefined shapes to be the maximum horizontal and vertical extents. This proves useful when trying to plot horizontal and vertical beam stay-clear. For example, it is better to use

    ```
    ..., APERTYPE=rectellipse, APERTURE=APX, APY, APX, APY;
    ```

    rather than

    ```
    ..., APERTYPE=rectellipse, APERTURE=100, 100, APX, APY;
    ```

- When `MAKETHIN` is called all thin slices inherit the aperture of their respective original thick lens version.

- When the `SIXTRACK` command is called (see the SixTrack converter module) the apertures are ignored by default. To convert the apertures as well the `APERTURE` flag has to be set in `C6T`.

- Aperture parameters are like all parameters and are inherited by derived elements. Like other parameters they can also be overridden by the derived elements if necessary.

- The `APERTYPE` and the `APERTURE`s themselves can be conveniently added to the `TWISS` (chapter 20) table by using the `SELECT` command (section 4.11). For example 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 put the aperture information together with the specified `TWISS` parameters into the `TWISS` table.

- The minimum beam stay-clear value (n1 value) computed along the beamline is printed to standard out and is also available as an output variable `N1MIN` of the `BEAM` command:

      Value, beam->n1min;
      nmin = beam->n1min;

23.2   Aperture tolerance definition

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 misalignment of the element in the accelerator, which also causes a decrease of aperture. The tolerance is given in the transverse plane like in picture 23.1 below.

Figure 23.1: Definition of aperture tolerances

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

    APER_TOL = {r, g, s};

Example:

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

23.3   Aperture offset definition

An aperture offset can be assigned to each element in a `MAD-X` sequence as a vector:

    APER_OFFSET = {real, real};

where the two real values are respectively the horizontal and vertical offsets of the aperture inside the element.

The offsets are only used in the `tracking` module of `MAD-X` and are ignored by the `APERTURE` command and by `PTC_TRACK`.

23.4 APERTURE

The APERTURE module was developed specifically for the LHC. Default parameter values are LHC values.

The `APERTURE` module computes the `n1` values for a piece of machine. Each element is sliced into thick sub-elements at given intervals, and the available aperture is computed at the end of each slice. The `APERTURE` module gets the geometric emittances from the values given or calculated in the BEAM command. The computation is based on the last Twiss table computed by the TWISS command. It is important to properly define a `BEAM`, and run TWISS and `APERTURE` commands on the same period or sequence.

The `APERTURE` example below also shows how to plot the resulting `n1` values.

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

```
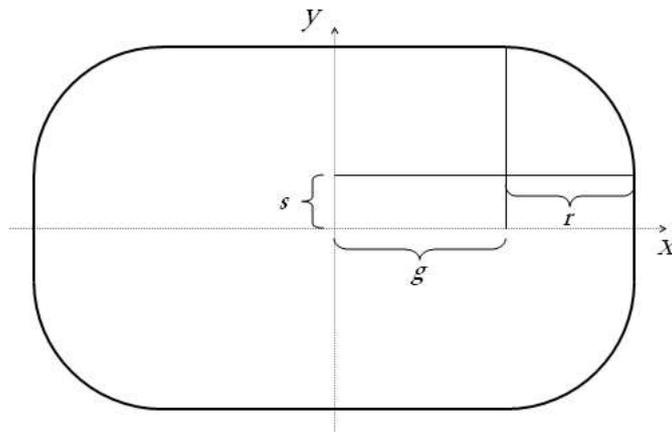APERTURE, RANGE=range,
         DQF=real, DPARX=real, DPARY=real,
         BETAQFX=real, BBEAT=real, DP=real,
         COR=real, NCO=integer,
         HALO={real,real,real,real}, HALOFILE=filename,
         INTERVAL=real, SPEC=real, NOTSIMPLE=logical,
         TRUEPROFILE=filename, OFFSETELEM=filename,
         FILE=filename;
```

where the parameters have the following meaning:

RANGE   Range given by elements. Default = `#s/#e`

DQF   Peak linear dispersion $(\partial x/\partial p_t)$[m]. Default = 2.086

DPARX   Fractional horizontal parasitic dispersion. Default = 0.273

DPARY   Fractional vertical parasitic dispersion. Default = 0.273

BETAQFX   Beta x in standard qf [m]. Default = 170.25

BBEAT   Beam size increase. This can be used to include beta-beat but note that it is a increase of the beamsize. Default = 1.1

DP   Bucket edge at the current beam energy. Note that this should be given in p_t and not deltap. For high energy machines such as the LHC this makes no difference. Default = 0.0015

COR   Maximum radial closed orbit uncertainty [m]. Default = 0.004

NCO   Number of azimuth per quadrant for halo 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}

HALOFILE   Input file with halo polygon coordinates. Will suppress an eventual halo parameter. Default = none

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 beam-screens in the range are considered not to be a
                "simply connex". Since all predefined MAD-X aperture types 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.
                **Note that the reference point should be within the range of elements
                given for the offsets to be taken into account.**

FILE   Output file with aperture table. Default = none

23.5   Not simply connex beam pipe profiles

The algorithm for finding the largest possible halo is the following:
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 in figure 23.2 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. (Geo-
metrical 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 23.3 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 sorts of polygons, the logical attribute NOTSIMPLE must
be specified. 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 henceforth not miss a dent in the beam pipe. See figure 23.4. The use of the
notsimple option significantly increases computation time.

Figure 23.2: Determination of maximum halo size

23.6 Dispersion

The aperture module uses both the parasitic dispersion which is based on inputs from the aperture module as well as the design dispersion which is obtained from `TWISS`. The spurious dispersion is given by:

$$D_{spur}(s) = D_m(s)(\sigma_{increase} - 1) + \frac{\sigma_{increase}(\sqrt{\beta_m(s)}f_{dqf}D_{para}}{\sqrt{\beta_{arc}}}), \qquad (23.1)$$

where $D_m(s)$, $\beta_m(s)$ is the dispersion and β-function from TWISS at the given location, `DQF`$=f_{dqf}$, `DPARX`$=D_{para}$, `BETAQFX` $= \beta_{arc}$, `BBEAT` $=\sigma_{increase}$.

When the aperture is calculated the two extrem points are checked for the model dispersion, i.e. `+DP` and `-DP`. The spurious dispersion is then added to this point in the way that it reduces the aperture the most for the given angle. This is the same method that is used for mechanical tolerances as well. The way the dispersion is added is shown in figure 23.5. In case `NOTSIMPLE` is chosen 10 points between `+DP` and `-DP` is evaluated. This is only needed in case of concave apertures and slows down the calculations significantly.

23.7 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 determined by the number of Si.
Layout of file:

```
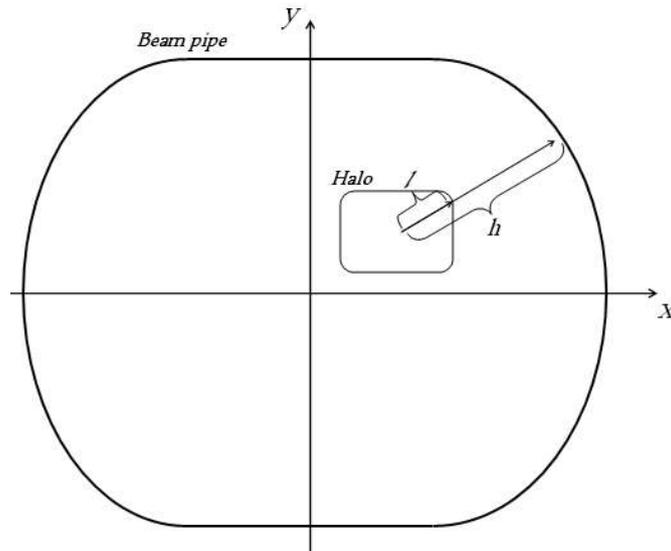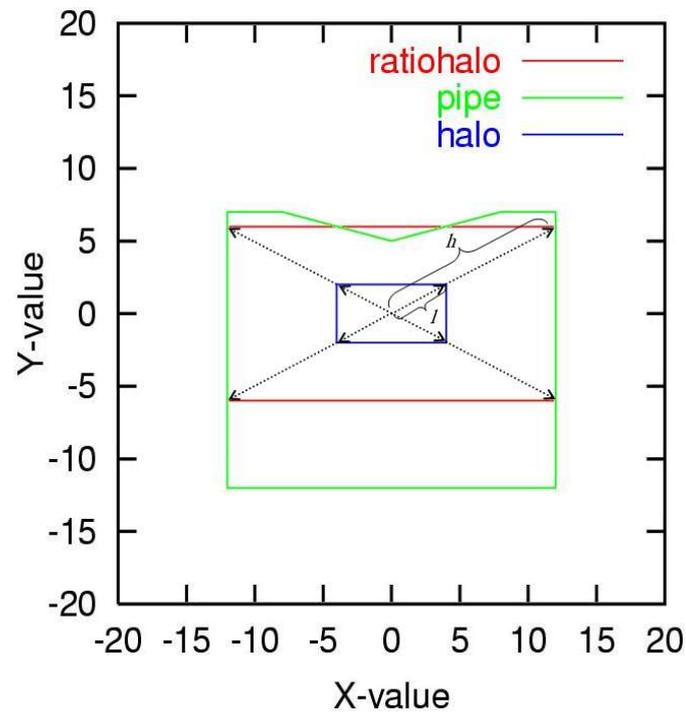magnet.name1
```

Figure 23.3: Not connex beam pipe profile: problem

```
        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:

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


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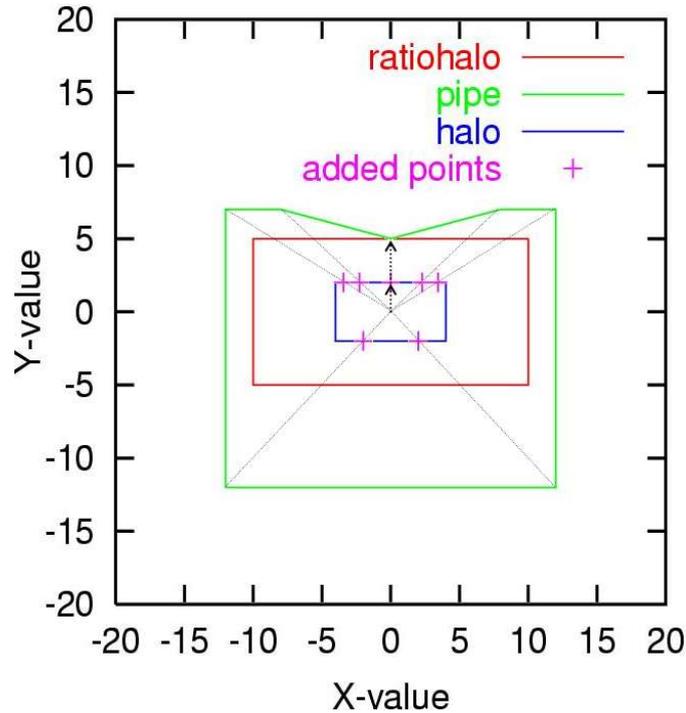1.033   0.00034         0.3e-9
2.066   0               0.00e-2
3.1     4.232e-4        0.00000003
```

Figure 23.4: Not connex beam pipe profile: solution

23.8 Offsetelem file syntax

This file contains parameters describing how certain elements are actually located in space with respect to a given reference element in the machine.

The basis for this file is a pair of coordinate systems, {s,x} and {s,y} with the origin at the reference point. The units for s, x and y are meters.

The actual location of the magnetic axis of a given element can be described, in each plane, as a parabola defined with three coefficients:

```
X_act(s) = DDX_OFF * s**2 + DX_OFF * s + X_OFF
Y_act(s) = DDY_OFF * s**2 + DY_OFF * s + Y_OFF
```

The reference position for the element, X_ref(s) and Y_ref(s), is calculated by `MAD-X` via an internal call to the SURVEY module, taking the reference element as the origin.

The resulting offset, in each plane, which is taken into account in the aperture calculation, is the difference between reference position and actual position:

```
X_offset(s) = X_ref(s) - X_act(s)
Y_offset(s) = Y_ref(s) - Y_act(s)
```

The offsets are only calculated for elements for which actual positions have been given through the `OFFSETELEM` file mechanism.

The file must be given in `TFS` format according to the following template, with mandatory header and any number of data lines, one per element.

Figure 23.5: The closed orbit is the black spot and the blue spots are displaced due to the design dispersion. The green rectangles show the spurious dispersion being added. Note that the points in the left corner of the figure has the same parameters but with p_t being negative.

```
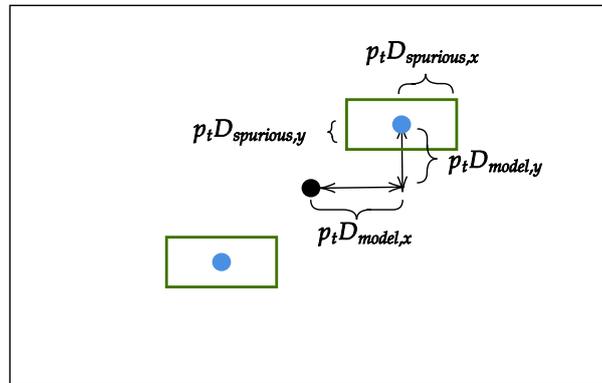@ NAME            %06s "OFFSET"
@ TYPE            %06s "OFFSET"
@ REFERENCE       %10s "reference-element-name"
* NAME      S_IP       X_OFF      DX_OFF     DDX_OFF    Y_OFF     DY_OFF    DDY_OFF
$ %s        %le        %le        %le        %le        %le       %le       %le
"elementname"  real    real       real       real       real      real      real
```

Note that the column S_IP is actually not used, and the values are ignored, but the column and values are parsed nevertheless and must be present. The absence of this column will trigger an error.

Example:

```
@ NAME            %06s "OFFSET"
@ TYPE            %06s "OFFSET"
@ REFERENCE       %10s "mq.12r1.b1"
* NAME      S_IP       X_OFF      DX_OFF     DDX_OFF    Y_OFF     DY_OFF    DDY_OFF
$ %s        %le        %le        %le        %le        %le       %le       %le
"mq.12r1.b1"  0.0      -3.0       -2.56545   0.0        0.0       -2.344366 0.0
"mcbxa.3r2"   0.0      -0.84      32.443355  0.3323     0.0       32.554363 0.2522
```

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

Note that prior to MAD-X version 4, the layout of the file was different and not formatted as TFS file:

```
reference-element-name


elementname
DDX_OFF DX_OFF X_OFF
DDY_OFF DY_OFF Y_OFF
```

Example:

```
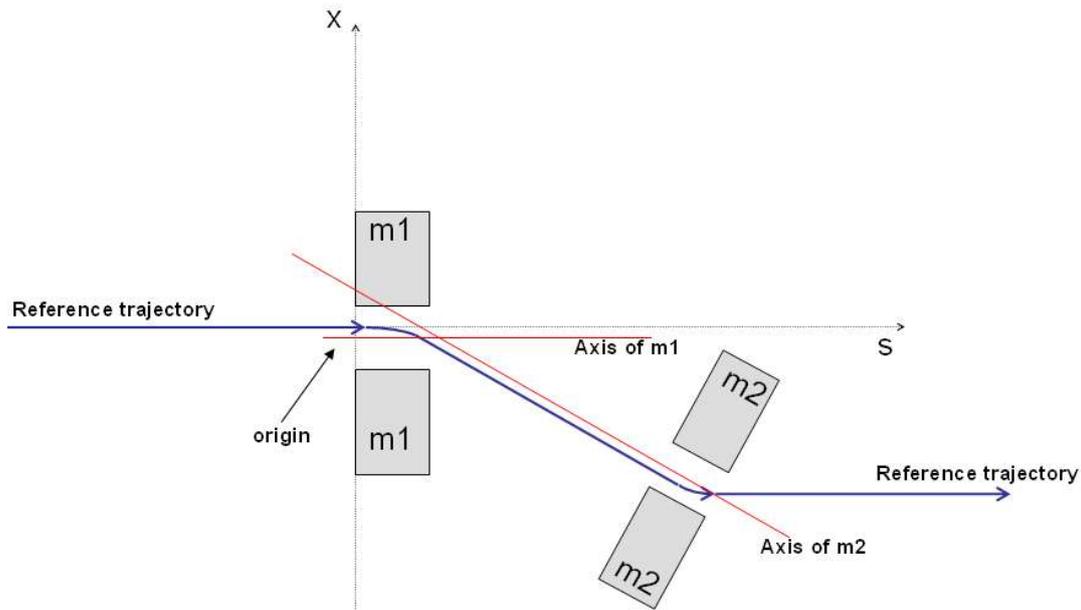!  Reference point
```

Figure 23.6: Illustration of effect of `OFFSETELEM`

```
mq.12r1.b1

!  List of elements and their displacement w.r.t.  reference point.
mcbxa.3l2
0 -2.5654500 -3
0 -2.3443666 0

!  The next element uses 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
```

23.9 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.

Note that specifying the `apertype` column automatically selects also the `aper_1, aper_2,`
`aper_3` and `aper_4` columns. The statement

```
SELECT, FLAG=aperture, COLUMN=apertype;
```

is equivalent to

```
SELECT, FLAG=aperture, COLUMN=apertype, aper_1, aper_2, aper_3, aper_4;
```

aper_# means for rectellipse:
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

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. The following plot shows the n1, beta functions, and
the hardware symbolized by ON_ELEM.

Figure 23.7: Example of plot showing aperture limits and n1

Chapter 24. Slicing a sequence into thin lenses

This module takes a sequence with thick elements and creates another one with similar functionality but composed of thin (zero length) element slices or simplified thick slices as required by `MAD-X` tracking or conversion to `SIXTRACK` input format.

DISCLAIMER: Several `MAD-X` commands such as `ELIGN`, `EFCOMP` won't directly work on sequences produced by `MAKETHIN`. In order to use such commands on a thin sequence, it is advisable to save the sequence on a file, and then re-load it. Also note that after `MAKETHIN` you will have to do a `USE` command.

24.1 MAKETHIN

The slicing is performed by the command:

```
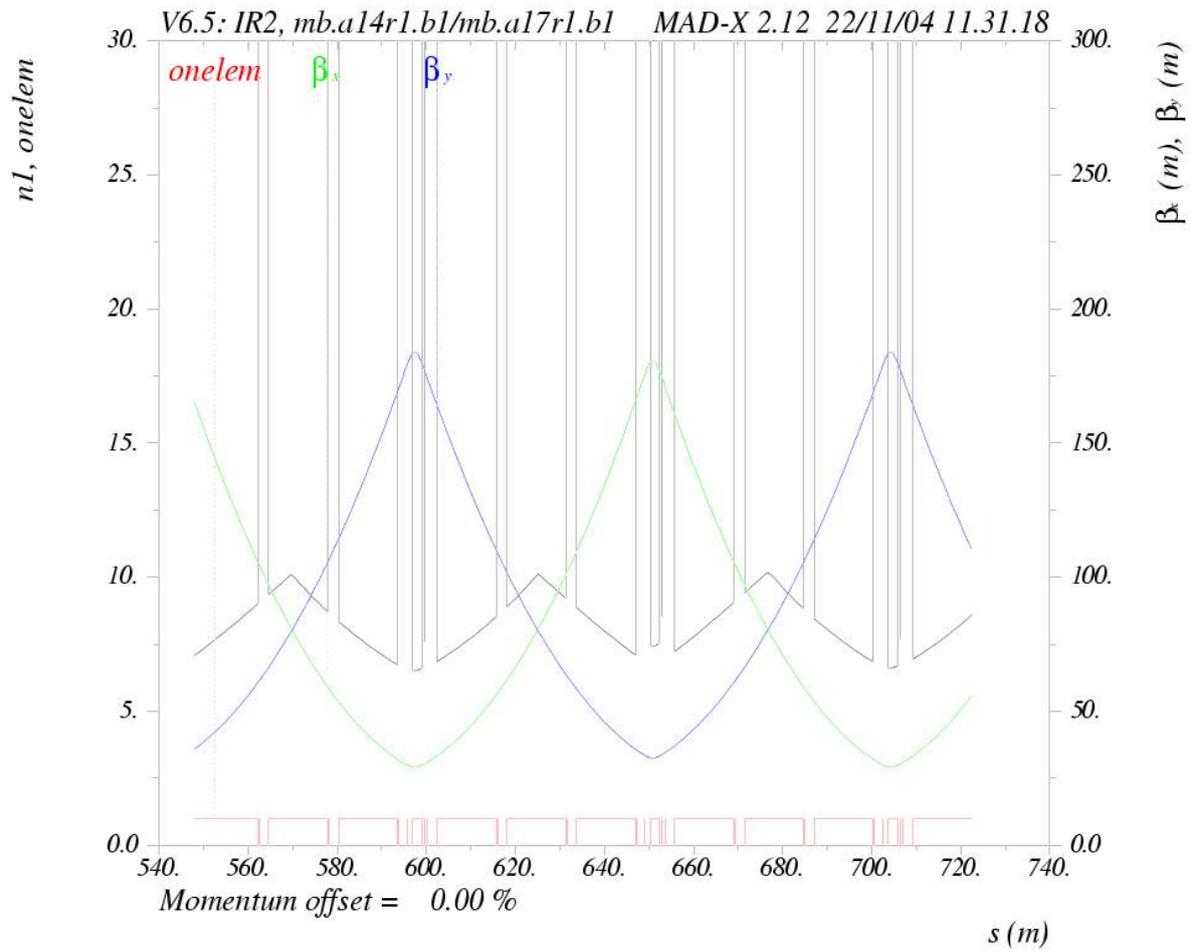MAKETHIN, SEQUENCE=seqname, STYLE=string, MAKEDIPEDGE=logical;
```

The parameters are defined as:

SEQUENCE
: seqname is the name of the sequence to be processed to thin slices. The sequence must be active, i.e. it should have been previously loaded with a USE command. The sequence must use the default positioning of elements (`REFER=centre`). To use the sliced sequence you have to again use the command USE with the sequence name.

STYLE
: the slicing style to be used for all active elements (excluding corrector magnets). The corrector magnets and the inactive elements are instead sliced so there is a slice in the beginning and in end of the location of the thick element. Available slicing styles are:

TEAPOT
: is the default slicing algorithm for all elements except collimators and is described in [19]. `TEAPOT` is a clear improvement over the `SIMPLE` algorithm.

SIMPLE
: produces slices of equal strength and length at equidistant positions.

COLLIM
: is the default slicing algorithm 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.

HYBRID
: is the previous default algorithm that was active up to version 5.02.05 when the `STYLE` attribute was not given; the standard slicing for all elements, except collimators, is `TEAPOT` if the number of slices is less or equal to four, and `SIMPLE` otherwise.

MAKEDIPEDGE
: is a logical flag to control the generation of `DIPEDGE` elements at the start and/or end of bending magnets, to conserve edge focusing from pole face angles `E1, E2` or extra fields described by `FINT, FINTX`, in the process of slicing

bending magnets to thin multipole slices. Selection with `THICK=true` will translate a complex thick `RBEND` or `SBEND`, including edge effects, to a simple thick `SBEND` with edge focusing transferred to extra `DIPEDGE` elements. (Default: true)

Example:

```
! keep translated rbend as thick sbend
SELECT, FLAG=makethin, CLASS=rbend, THICK=true;
```

24.2 Controlling the number of slices

The number of slices can be set individually for elements or groups of elements using the `SELECT` command

```
SELECT, FLAG=makethin,
        RANGE=range, CLASS=class, PATTERN=pattern[,FULL][,CLEAR],
        THICK=logical, SLICE=integer;
```

where the argument to the attribute `SLICE` stands for the number of slices for the selected elements. The default is one slice and `THICK=false` for all elements, i.e. conversion of all thick elements to a single thin slice positioned at the centre of the original thick element.

Note that `THICK=true` only applies to dipole or quadrupole magnetic elements and is ignored otherwise.

`MAKETHIN` allows for thick quadrupole slicing with insertion of thin `MULTIPOLE` elements between thick slices. Positioning is done with markers between slices, here however with thick slice quadrupole piece filling the whole length.

Examples:

```
! slice quadrupoles in three thick slices, insert 2 markers per quadrupole
SELECT, FLAG=makethin, CLASS=quadrupole, THICK=true, SLICE=3;

! thick slicing for quadrupoles named mqxa, insert one marker in the middle
SELECT, FLAG=makethin, PATTERN=mqxa\., THICK=true, SLICE=2;
```

Slicing can be turned off for certain elements or classes by specifying a number of slices < 1. Examples:

```
! turn off slicing for sextupoles
SELECT, FLAG=makethin, CLASS=sextupole, SLICE=0;

! keep elements unchanged with names starting by mbxw
SELECT, FLAG=makethin, PATTERN=mbxw\., SLICE=0;
```

This option allows to introduce slicing step by step and monitor the resulting changes in optics parameters.

Keep in mind however that subsequent tracking generally requires full slicing, with possible exception of dipole and quadrupole magnetic elements.

24.3   Choice of options for dipoles

There are several options that affect the slicing of a sequence. Depending whether dipole magnets (`RBEND or SBEND`) are kept as thick elements (See `SELECT, FLAG=makethin`) and whether the `MAKEDIPEDGE` option of `MAKETHIN` is used to generate `DIPEDGE` elements around these bending magnets, the resulting sequence is more adapted to optics calculation or tracking. The following table gives an indication of the best choice of options for dipole elements

Table 24.1: Best choice of options in `MAKETHIN`

	`THICK=false`	`THICK=true`
`MAKEDIPEDGE=false`	backward compatibility	thick optics calculations
`MAKEDIPEDGE=true`	thin tracking	thick tracking

The combination of `THICK=false` (dipoles converted to thin lenses) and `MAKEDIPEDGE=false` (no `DIPEDGE` thin elements inserted around the dipoles) results in a lattice with Twiss parameters that can differ significantly from those of the original sequence. This is most obvious with the tune or phase advance parameters that are generally differing by a significant amount (see first hint below). This combination is however useful for backward compatibility with previous versions of `MAKETHIN` before `DIPEDGE` elements were implemented.

The combination of `THICK=true` (dipoles are kept as thick elements and not converted to thin lenses) and `MAKEDIPEDGE=false` (`DIPEDGE` thin elements are not inserted around the thick dipoles to replace the edge effects that stay with the bends themselves) results in a lattice with `TWISS` parameters very comparable to those of the original sequence. The dipoles are still thick elements, eventually sliced, with proper edge effects up to second order, although non-symplectic. This is the best combination for optics studies without particle tracking, e.g. with `TWISS`.

The combination of `THICK=false` for dipoles and `MAKEDIPEDGE=true` transforms the original dipoles into one or several thin slices without any edge effect, surrounded by a pair of `DIPEDGE` elements at the original location of dipole entrance and exit. Note however that these `DIPEDGE` elements only contain first order effects for the purpose of tracking.

The combination of `THICK=true` for dipoles and `MAKEDIPEDGE=true` conserves the dipoles as thick element dipole bodies only while their associated edge effects are transferred to `DIPEDGE` elements that are taken into account at first order only for symplectic tracking.

24.4   Additional information

The generated thin lens sequence has the following properties:

- The new sequence has the same name as the original. The original sequence is replaced by the new one in memory. If the original sequence is needed for further processing in `MAD-X`, it should be reloaded.

- The algorithm also processes any sub-sequence inserted in the main sequence. These sub-sequences are also given the same names as the original ones.

- Any element transformed into a single thin lens element has the same name as the original.

- If an element is sliced into more than one **thin** slices, the individual slices have the same basename as the original element plus a suffix `..1`, `..2`, etc. and a marker with the name of the original element is placed at the location of the center of the original element.

- If an element is sliced into more than one **thick** slices, the individual slices have the same basename as the original element plus a suffix `_EN` the first one, a suffix `_EX` the last one, and a suffix `_BO` the ones in between.

Hints

1. Compare the main optics parameters like tunes before and after slicing with `MAKETHIN`. Rematch tunes and chromaticity as necessary after `MAKETHIN`.

2. In tests, turn off slicing for some of the main element classes to identify the main sources of changes.

3. For sextupoles and octupoles, a single slice should always be sufficient.

4. Increase the number of slices for critical elements like mini-beta quadrupoles. Even there, more than four slices should rarely be required.

5. In case of problems or doubts, consider to `FLATTEN` the sequence before slicing.

6. See the examples for makethin.
   See also the presentations on the upgrade of the makethin module:
   LCU_makethin_2012_09_18.pdf, and
   LCU_makethin_2013_04_19.pdf.
   TEAPOT is documented in IPAC'13 MOPWO027

Chapter 25. 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 with both 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.

WARNING: if errors are to be applied to a thin sequence produced by MAKETHIN, it is advisable to save the sequence on a file and then reload it, in order for MAD-X to reinitialize its data structures in a suitable way.

WARNING: The BV-flag is not considered in the magnetic field strenghts of the ERROR tables!

25.1 EALIGN: Alignment Errors and permanent misalignments

There are two ways of introducing misalignments in MAD-X.

25.1.1 EALIGN

Alignment errors are defined by the EALIGN command. The misalignments refer to the local reference system for a perfectly aligned machine. Misalignments are defined as 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.

Beam Position Monitors can be given readout errors as well as readout scaling errors in both horizontal and vertical planes. Monitor readout and scaling errors are ignored for all elements other than monitors.

Each new EALIGN statement replaces the misalignment errors for all elements in its range, unless the logical ADD attribute of EOPTION has been specified.

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;
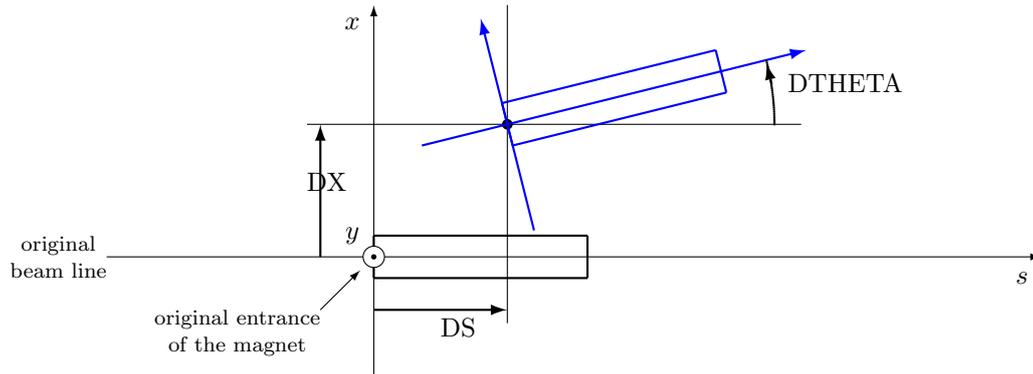```

for elements selected by the SELECT command.



Figure 25.1: Alignment errors in the $(x, s)$-plane



Figure 25.2: Alignment errors in the $(x, y)$-plane

## 25.1.2 Permanent Misalignments

The second way of introducing misalignments in MAD-X is to define them directly in the creation of a sequence. The name and effect of the orientation is the same as for the EALIGN. In case EALIGN and permanent misalignment are used togheter the values are added togheter component by component. The effect of the permanent misalignment is ignored by SURVEY by default but by adding the PERM_ALIGN_SURVEY the start and end location of each element is given. The permanent misalignemnt need to be defined in the creation of the sequence. An example is given below: **Examples:**

```
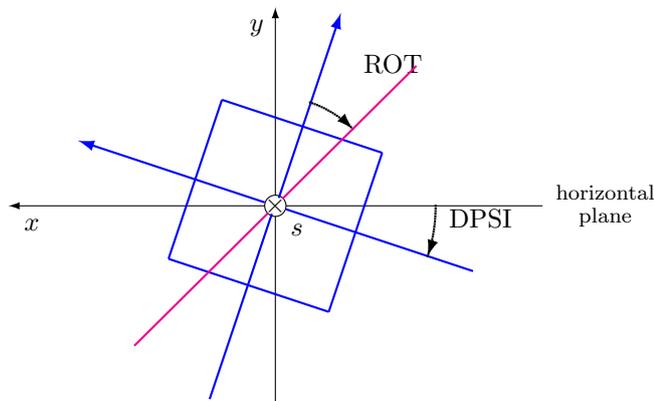a=0.06;
```

```
q1:quadrupole, l=1, k1=0.01;
myseq:  sequence, l=10;
q1, at=5, dtheta=0.1, dphi=0.2, dpsi=0.3, dx=0.04, dy=0.05, ds:=a;
endsequence;
```

It is also possible to defined the permanet misalignments using defered expresions.

Figure 25.3: Alignment errors in the (y, s)-plane

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. (Default: 0 rad). A positive angle gives a greater y-coordinate for the exit than for the entry.
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. Note that this is ignored by TWISS and TRACK and only used by the orbit correction command. 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. Note that this is ignored by TWISS and TRACK and only used by the orbit correction command. If MREY>0, the reading for y is too high (default: 0 m).

MSCALX   The relative horizontal scaling error for a monitor. This is ignored if the
             element is not a monitor. Note that this is ignored by TWISS and TRACK
             and only used by the orbit correction command.
             If MSCALX>0 the reading for x is too high (default: 0). A value of 0.5 implies
             that 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. Note that this is ignored by TWISS and TRACK and only
             used by the orbit correction command.
             If MSCALY>0 the reading for y is too high (default: 0). A value of -0.3 implies
             that the actual reading is multiplied by 0.7.

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

Figure 25.4: Readout errors in a monitor

Examples:

```
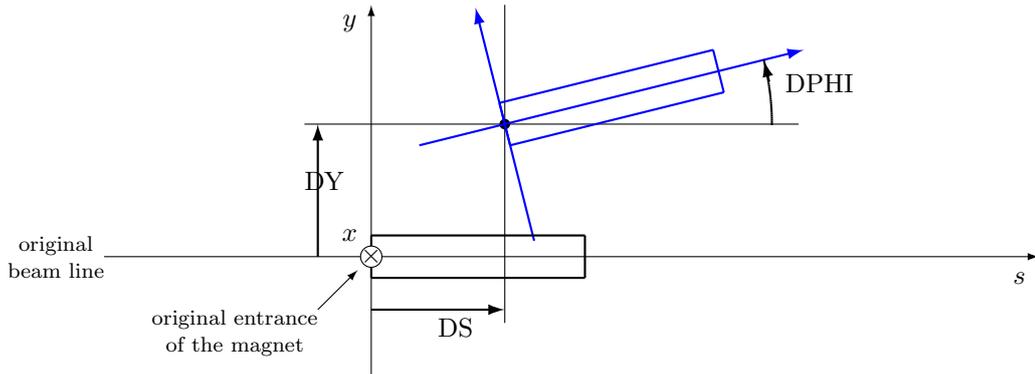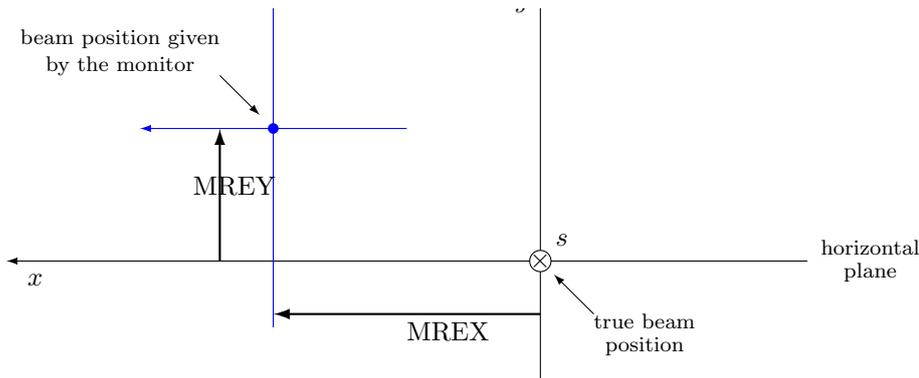SELECT, FLAG = ERROR, CLASS = MQ;
EALIGN, DX = 0.002, DY = 0.0004*RANF(), DPHI = 0.0002*GAUSS();
```

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) specifies a Gaussian distribution cut at 2.5 sigma.

25.2   EFCOMP: 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 `MAD-8` 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),...};
```

for elements selected by the `SELECT` command.

Each new `EFCOMP` statement replaces the field errors for all elements in its range(s). Previous field errors present in the range are discarded or incremented depending on the setting of `ADD` logical attribute of the `EOPTION` command. `EFCOMP` defines the field errors 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. In order to use a skew component as the reference, the reference radius should be specified as a negative number. (Default: 0)
            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 where 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 negative, 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). The absolute error is computed from DKNR(i) using the following formula

$$\Delta K_i = \text{DKNR(i)} \cdot (k_{ref}) \cdot R^{n-j} \cdot \frac{j!}{n!}$$

where n is the order of the reference strength, k_{ref}, specified in ORDER, and R is the RADIUS.

DKSR(i)   Relative error for the skewed multipole strength with (2i+2) poles. (Default: 0). The absolute error is computed from DKSR(i) using the following formula

$$\Delta K_i = \text{DKSR(i)} \cdot (k_{ref}) \cdot |R|^{n-j} \cdot \frac{j!}{n!}$$

where n is the order of the reference strength, k_{ref}, specified in ORDER, and R is the RADIUS.

Please note that even though that any order of field errors can be specified for all elements, only those which are defined for an element will be taken into account. Only K0 and K1 are be considered for BEND transfer map, and, in addition, K2 and K1S for radiation effect. Only K1 and K1S are considered for QUADRUPOLE; K2 and K2S - for SEXTUPOLE. Only THIN MULTIPOLE considers the erros of all orders.

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:

$$\Delta = \sum (c_i * k_{ref}^i) i = 0...3$$

The coefficients c_i for the polynomial 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, HCOEFFS normal and skew coefficients for the computation of the correction term. The four 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 to 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, 0, 0},
        DKSR={ 0, 4e-1, 1e-1, 2e-3, 0, 0, 0, 0, 0, 0,
               0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
```

Example to add time memory effect to relative errors:

```
SELECT, FLAG=error, PATTER="q̂.*"; [FIXME]
EFCOMP, ORDER=1, RADIUS=0.020, HYSTER=1,
        DKNR={ 0, 1e-2, 2e-4, 4e-5, 1.e-5, 0, 0, 0, 0, 0,
              0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
        DKSR={ 0, 1e-2, 2e-4, 4e-5, 1.e-5, 0, 0, 0, 0, 0,
              0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
        HCOEFFN={0.000, 0.000, 0.000, 0.000,  ! coeff.  mult.  order 0
                 0.001, 0.000, 0.000, 0.000,  ! coeff.  mult.  order 1
                 0.000, 0.000, 0.002, 0.000}; ! coeff.  mult.  order 2
```

See also: random values and deferred expressions.

25.3   EOPTION: Set Options for Error Definition

The error option command specifies different seeds for random values:

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

SEED   Seeds or starts a particular sequence of random values. A valid SEED value is
              an integer in the range [0...999999999] or an expression that evaluates to an
              integer in the same range. (default: 123456789).
              Internally the code takes as the effective seed the value ABS(seed)%1.e10 hence
              normalizing the provided seed to an appropriate value in the range irrespective
              of the provided value.
              Note that the seed set with this command is shared with the TRACK and
              COPTION commands. See also: Random Values.

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 defi-
              nitions. The default value is TRUE if it is omitted in the EOPTION command.
              The default value is false if no EOPTION command is used.
              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;
```

The random number generator for MAD-X is taken from [20].

25.4   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.

25.5   ESAVE: Writing errors to a file

```
ESAVE, FILE=string;
```

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

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.

25.6   ETABLE: Saving the errors to a table

```
ESAVE, TABLE=string;
```

This command saves a table of errors assigned to elements on a table, using a format which can be read in again to obtain the same results. This allows storing the errors and reloading them after a new USE command.

Example:

```
SELECT, FLAG=myerrortable, RANGE=range, CLASS=name, PATTERN=string;
ETABLE, TABLE=myerrortable;
```

and elements selected by the SELECT command are stored to the table.

25.7   SETERR: Reading 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 since the USE command 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 READTABLE.

The command sequence:

```
READTABLE, 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).

Please note:

1. To assign correctly the errors from the file to the elements in the sequence, all elements must have individual names, otherwise an identification is not possible. Elements in the file not identified in the active sequence are ignored.

2. Errors are assigned to ALL elements found in the file and the FLAG=ERROR is set. Therefore the number of elements selected corresponding to a command like
   SELECT, FLAG=ERROR,...;
   can be different after the execution of SETERR.

Chapter 26. Orbit Correction

This chapter describes the commands to correct a closed orbit or a trajectory. The initial orbit or trajectory to be corrected can be obtained from an internal or external TFS table.

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.

26.1 CORRECT

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 (some options are valid only for special algorithms):

```
CORRECT, SEQUENCE=seqname,
        FLAG=string, MODE=string, PLANE=string,
        COND=integer, NCORR=integer,
        SNGVAL=real, SNGCUT=real,
        MONERROR=integer, MONON=real, MONSCALE=real,
        CORRLIM=real, TWORING=logical, UNITS=real,
        CORZERO=integer, ERROR=real,
        ORBIT=table, MODEL=table, TARGET=table,
        BEAM1TAB=table, BEAM2TAB=table,
        EXTERN=logical,
        NAME_COL=string, X_COL=string, Y_COL=string,
        CLIST=filename, MLIST=filename,
        RESOUT=integer, TWISSUM=integer;
```

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   can be "ring" or "line", either a circular machine or a trajectory is corrected. (Default: ring)

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 get 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: `MICADO`)

PLANE   With `PLANE=x`, the orbit correction is performed for the horizontal plane; With `PLANE=y`, the correction is performed for the vertical plane. made. (This differs from the `MAD-8` implementation).
(Default: X for horizontal plane)

COND   When `COND=1`, a Singular Value Decomposition is performed and the response matrix is conditioned to avoid linearly dependent correctors. This can be used to avoid creation of artificial bumps during a `LSQ` or `MICADO` correction. 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 attributes `SNGVAL` and `SNGCUT`. Both parameters depend on the machine and may need adjustment. Default values are adjusted to large machines and give "reasonable" performance for smaller machines.

NCORR   Defines the number of correctors to be used by the `MICADO` algorithm. If `NCORR=0` all available correctors are used. Only used by the `MICADO` algorithm.
(Default: 0 i.e. all available correctors)

SNGVAL   Used to set the threshold for finding singular values with the `COND` attribute. (Hint: smaller number finds fewer singular values).
Use with care !
(Default: 2.0)

SNGCUT   Used to set the threshold for finding redundant correctors with the `COND` attribute. (Hint: larger number finds fewer redundant correctors).
Use with extreme care !
(Default: 50.0)

MONERROR   When `MONERROR` is 1, the `MREX` and `MREY` alignment errors on monitors assigned by EALIGN are taken into account, otherwise they are ignored.
(Default: 0)

MONSCALE   When `MONSCALE` is 1, the `MSCALX` and `MSCALY` scaling errors on monitors assigned by EALIGN are taken into account, otherwise they are ignored.
(Default: 0)

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 $(1.0 - \text{MONON})$ 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: 1.0 i.e. 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: 1.0 mrad)
TWORING	When true, the correction will be done on two rings at once. The only correction mode available in this case is MICADO. The attribute ORBIT=table is ignored and BEAM1TAB=table and BEAM2TAB=table are used instead.
	(Default: false)
UNITS	when this parameter is set the value is a multiplier [TO BE COMPLETED] Default unknown
CORZERO	an integer value to specify whether corrector settings should be all reset to zero before starting the orbit correction (CORZERO> 0) or corrector settings calculated by the orbit correction should be added to existing corrector settings (CORZERO= 0, Default).
ERROR	specifies the maximum RMS value, in meters, of the orbit to be reached by the correction algorithm, e.g. ERROR=1.e-3 for a 1 mm RMS target value.
	(Default: 1.e-5 m)

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;. 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.
EXTERN	(default: false): When false, the ORBIT and TARGET table are assumed to be computed by MAD-X with a previous TWISS command. When set to true, that option allows to use twiss tables imported from an external file (with the READTABLE command), for example to use measured BPM data. In that case, the imported twiss table is allowed to contain coordinate data only at the location of the monitors. Note that the list has to be in order and that you cannot skip any locations of the monitors you have included.
NAME_COL	The name of the column that contains the name of monitors in the tables. (Default: "name")
X_COL	The name of the column that contains the X position readings of monitors in the tables. (Default: "X")
Y_COL	The name of the column that contains the Y position readings of monitors in the tables. (Default: "Y")

Example of use of CORRECT to reproduce a measured orbit:

```
! To have a refererence optical model
twiss, table=twiss_ref;

! The bpm.tsv is a reduced Twiss file containing only lines for the BPMs
readtable, file="bpm.tsv", table="twiss_bpm";

! correct orbit using external measurements
correct, flag=ring, mode=micado, ncorr=5, cond=1 ,plane=x, extern,
        model=twiss_ref, orbit=twiss_ref, target=twiss_bpm,
        error=1.0e-21;
```

Two attributes affect the printing of tables and results:

CLIST   the name of the file where corrector settings (in units of rad) before and after
               correction are printed.

MLIST   the name of the file where monitor readings (in units of m) before and after
               correction are printed.

RESOUT   This command outputs the results for all monitors and all correctors in a com-
               puter 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 attribute value is larger than 0, CORRECT prints maximum orbit and r.m.s.
               values for both planes, taken from the Twiss summary table, in computer
               readable form. This allows to analyze orbits etc. at elements that are not
               monitors or correctors. The argument is added to the output. This attributes
               is used only to produce output: no correction is made, and all other attributes
               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;
```

Example 5: correct orbit in horizontal plane for a two-beam machine, using MICADO with no SVD conditioning, zeroing correctors before the correction.

```
CORRECT, FLAG=ring,
        TWORING, BEAM1TAB=twb1, BEAM2TAB=twb2,
        MODE=micado, COND=0, NCORR=4, ERROR=1E-6, PLANE=x,
        MLIST='mx12.tab', CLIST='cx12.tab', RESOUT=1, CORZERO=1;
```

26.2   USEMONITOR, USEKICK

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 command `USEMONITOR` activates or deactivates a selection of beam position monitor. This command affects elements of types `MONITOR, HMONITOR`, and `VMONITOR`.

The command `USEKICK` activates or deactivates a selection of orbit correctors. This command affects elements of types `KICKER, HKICKER` and `VKICKER`.

Both commands have the same attributes:

STATUS   If this flag is true (on), the selected elements are activated. Active orbit moni-
            tor readings will be considered, and active correctors can change their strengths
            in subsequent correction commands. Inactive elements will be ignored subse-
            quently.

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, STATUS = OFF, RANGE = ...;    ! deactivate selected correctors
USEMONITOR, STATUS = OFF, RANGE = ...; ! deactivate selected monitors
```

```
CORRECT, NCORR = 32;                    ! uses different set of correctors
USEKICK, STATUS = OFF, RANGE = ...;     ! deactivate different set of correctors
CORRECT, NCORR = 32;                    ! uses different set of correctors
```

26.3   CSAVE

This section is under construction, options presently only available in MADX development version.

26.4   SETCORR

This section is under construction, options presently only available in MADX development version.

26.5   COPTION

COPTION, SEED=integer, PRINT=integer;

In the orbit program monitors can be randomly disabled and the correct option command specifies different seeds for random values:

SEED   Seeds or starts a particular sequence of random values. A valid SEED value is
              an integer in the range [0...999999999], or an expression that evaluates to an
              integer in the same range (Default: 123456789).
              Internally the code takes as the effective seed the value ABS(seed)%1.e10 hence
              normalizing the provided seed to an appropriate value in the range irrespective
              of the provided value.
              Note that the seed set with this command is shared with the TRACK and
              EOPTION commands. See also: Random values.

PRINT   A flag taking integer values to control the printout.
              In general: the higher the 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;
```

Note that the random generator for MAD-X is taken from [20].

Chapter 27. TAPER: strength tapering for energy variation

`TAPER` calculates the adjustment to the strengths of elements to account for small momentum variations through RF cavities or synchrotron radiation.

```
TAPER, ITERATE=integer, STEPSIZE=real, FILE=filename, RESET=logical;
```

The attributes for the `TAPER` command are:

ITERATE the number of iterations through the element maps.
Because the energy loss in dipoles is strongly non linear, `MAD-X` can cycle `ITERATE` times through the dipole map to converge to the appropriate tapering value of the element corresponding to the energy loss through the adjusted bending magnet.
For quadrupoles, sextupoles, octupoles and multipoles, a tracking through the map is done only once for any positive non-zero value of `ITERATE`.
If `ITERATE=0` is specified, the tapering is only calculated from the momentum deviation at the entrance of the element and no account is taken of the energy loss through the current element.
A maximum value of `ITERATE=10` is enforced. (Default: 3)

STEPSIZE the tapering step size for stepwise tapering. if `STEPSIZE` is non-zero and positive, tapering values are calculated as multiples of `STEPSIZE`. (Default: 0)

FILE causes `MAD-X` to write the tapering values applied to each sequence node to the file specified. The file can be `LOAD`ed in a subsequent `MAD-X` run.
(Default: "taper.madx")

RESET causes `MAD-X` to reset tapering values of all nodes in the current sequence to zero values. All other command parameters are ignored.

The `TAPER` command is only meaningful when at least one of the following conditions is true for the `SEQUENCE` being used:

- the `SEQUENCE` contains at least one active `RFCAVITY` providing energy gain,

- the energy loss by synchrotron radiation is enabled with `BEAM, RADIATE=true;`.

In these conditions the energy of the reference particle changes along the beam line and the nominal strength of elements after the location where energy variation occurs are not ideally adapted to the actual energy of the reference particle.

The tapering is calculated as a **relative** change in strength between the nominal strength and the strength adapted to the local energy of the reference particle, relative to the nominal strength, taken as the average between the local energies at the entrance and at the exit of the element.

For a quadrupole, the focusing strength `K1act` actually used is calculated from the nominal strength `K1nom` and `KTAP` as

```
    K1act = K1nom * (1 + KTAP)
```

For a dipole with a given geometric `ANGLE` and `LENGTH`, the tapering value `KTAP` is applied through the `K0` mechanism with the actual strength of the dipole calculated as `K0 * LENGTH = ANGLE * (1 + KTAP)`.

The `KTAP` attribute can be calculated and stored, or applied from input file, for each node of type `BEND`, `QUADRUPOLE`, `SEXTUPOLE`, `OCTUPOLE` or `MULTIPOLE` in the active sequence.

If `STEPSIZE` is not specified or specfied as zero, the calculated value is stored and each node has a unique value for tapering.

If `STEPSIZE` is non-zero the calculated value is rounded to the nearest multiple of the `STEPSIZE` resulting in a quantization of the tapering values; the same tapering value can be assigned to several adjacent nodes in the sequence.

Setting `STEPSIZE` to a non-zero value is useful to explore tolerances of a beamline to tapering and determine a likely strategy for stepwise and/or piecewise tapering, *i.e.* applying the same tapering to a set of elements or part of a beamline.

`TAPER` uses the closed orbit finding algorithm of the `TWISS` package to calculate the tapering values of all elements if `STEPSIZE` is zero. If `STEPSIZE` is non-zero it could be difficult to ensure a closure condition for the tapering values and the first turn algorithm of `MAD-X` is used instead, also from the `TWISS` package. Hence for evaluation of tapering of an open line the `STEPSIZE` should be set to a non-zero value.

Every call to the `TAPER` command first resets all tapering values to zero in order to be able to adjust the `BEAM` before recalculating the orbit with tapering. This is also the case when the `TAPER` command is called internally from `TWISS` or `MATCH` command with option `TAPERING=true`.

Example:

```
    BEAM, ENERGY = 100.0, RADIATE=true;
    TAPER;
```

calculates tapering in the current sequence where each node is attributed a tapering value corresponding to the local energy of the reference particle. The `STEPSIZE` is zero and the calculation is done with closed orbit finding algorithm.

Example:

```
    BEAM, ENERGY=100.0, RADIATE=true;
    TAPER, STEPSIZE=1.e-3, FILE=''mytaper''; TWISS;
```

calculates tapering in the current sequence where each node is attributed a tapering value corresponding to the local energy of the reference particle, but rounded to the nearest multiple of the `STEPSIZE` which is one per mil. The calculation is done with the first turn algorithm only. The tapering values are written out to the file "mytaper". The following `TWISS` command will use the tapered values for calculating the closed orbit and twiss parameters.

Chapter 28. SODD: Second Order Detuning and Distortion

The `SODD` command calculates the Second Order Detuning and Distortion, as described in [21], on the beam line defined by the last `USE` command followed by a `TWISS` command.

The `SODD` command is based on the stand-alone program[13] with the same name, with analytical computation extended to the second order distortion[22].

```
SODD, DETUNE=logical, DISTORT1=logical, DISTORT2=logical,
      START_STOP = start,stop,
      MULTIPOLE_ORDER_RANGE = first,last,
      NOPRINT=logical, PRINT_ALL=logical, PRINT_AT_END=logical,
      NOSIXTRACK=logical;
```

The attributes of the `SODD` command are:

DETUNE   flag to trigger calculation of the detuning function terms in first and second order in the strength of the multipoles. (Default: false)

DISTORT1   flag to trigger the calculation of the distortion function and the Hamiltonian terms in first order in the strength of the multipoles. (Default: false)

DISTORT2   flag to trigger the calculation of the distortion function and Hamiltonian terms in second order in the strength of the multipoles. (Default: false)

START_STOP   positions (reals in meters) along the beamline defining a longitudinal interval. (Default: 0.0,0.0)

MULTIPOLE_ORDER_RANGE lowest and highest multipole order to be taken in account, given as integers. (Default: 1,2)

NOPRINT   flag to the effect that no file or internal table is created to hold the results. If true, the attributes `PRINT_ALL` or `PRINT_AT_END` have no effect. (Default: false)

PRINT_ALL   flag to generate files and internal tables with results for each mutipole. (Default: false)

PRINT_AT_END   flag to generate files and internal tables with results at the end of the longitudinal interval. (Default: false)

NOSIXTRACK   flag to signal that fc.34 shall not be generated internally by invoking the conversion routine of sixtrack. The user should provide this file before the execution of the `SODD` command. (Default: false)

Note that the first row of every file generated by `SODD` is a header containing the names of the columns. This row is absent in the internal tables.

A more detailed description can be found in [23].

28.1   DETUNE

The attribute `DETUNE` triggers the calculation of the detuning function terms in first and second order in the strength of the multipoles.

If the logical attribute `PRINT_AT_END` is set to true, the following two files, and corresponding tables, are created:

- "detune_1_end" contains five columns : multipole order, horizontal or vertical plane coded as 1 or 2, horizontal or vertical detuning, order of horizontal invariant and order of vertical invariant.

- "detune_2_end" contains five columns : first multipole order, second multipole order, horizontal detuning, order of horizontal invariant and order of vertical invariant.

If the logical attribute `PRINT_ALL` is set to true, the following two files, and corresponding tables, are created :

- "detune_1_all" contains five columns : multipole order, horizontal or vertical plane coded as 1 or 2, horizontal or vertical detuning, order of horizontal invariant and order of vertical invariant.

- "detune_2_all" contains five columns : first multipole order, second multipole order, horizontal detuning, order of horizontal invariant and order of vertical invariant.

28.2   DISTORT1

The attribute `DISTORT1` triggers the calculation of the distortion function and the Hamiltonian terms in first order in the strength of the multipoles.

If the logical attribute `PRINT_AT_END` is set to true, the following two files, and corresponding tables are created:

- "distort_1_F_end" contains eight columns : multipole order, cosine and sine part of distortion, amplitude of distortion, j, k, l, m.

- "distort_1_H_end" contains eight columns : multipole order, cosine and sine part of Hamiltonian, amplitude of Hamiltonian, j, k, l, m.

If the logical attribute `PRINT_ALL` is set to true, the following two files, and corresponding tables, are created :

- "distort_1_F_all" contains eleven columns : multipole order, appearance number in position range, number of resonance, position, cosine and sine part of distortion, amplitude of distortion, j, k, l, m.

- "distort_1_H_all" contains eleven columns : multipole order, appearance number in position range, number of resonance, position, cosine and sine part of Hamiltonian, amplitude of Hamiltonian, j, k, l, m.

28.3 DISTORT2

The attribute `DISTORT2` triggers the calculation of the distortion function and Hamiltonian terms in second order in the strength of the multipoles.

If the attribute `PRINT_AT_END` is set to true, the following two files, and corresponding tables, are created:

- "distort_2_F_end" contains nine columns : first multipole order, second multipole order, cosine and sine part of distortion, amplitude of distortion, j, k, l, m.

- "distort_2_H_end" contains nine columns : first multipole order, second multipole order, cosine and sine part of Hamiltonian, amplitude of Hamiltonian, j, k, l, m.

Chapter 29. 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 in [24] and his article on Touschek lifetime in [25].

The syntax of the TOUSCHEK command is:

```
TOUSCHEK, TOLERANCE=real, FILE=filename;
```

The arguments have the following meaning:

TOLERANCE   the tolerance for the numerical integrator DGAUSS. (Default: 1.e-7)

FILE   The name of the output file (Default: 'touschek')

TOUSCHEK should only be called after fully qualified BEAM command and 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 [26], which was generalized to the case of several harmonic rf systems. If *synch_2* is zero, 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 *cavtousch0* for that ring.)

Example:

```
BEAM, PARTICLE = PROTON, ENERGY = 450, NPART = 1.15e11,
      EX = 7.82E-9, EY = 7.82E-9, ET = 5.302e-5;
USE, PERIOD = LHCb1;
...
VRF = 400;
...
SELECT, FLAG = TWISS, CLEAR;
TWISS, CHROM, TABLE, FILE;
...
TOUSCHEK, FILE, TOLERANCE=1.e-8;
```

The first command defines the beam parameters. It is essential that the longitudinal emittance is 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 instantaneous 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.

All results can also be printed to a file using the command

```
WRITE, TABLE=touschek, FILE;
```

The `MAD-X` Touschek module was developed by Catia Milardi and Frank Zimmermann.

The `MAD-X` Touschek module was partially rewritten in November 2013 by Ghislain Roy after the discovery of a few bugs in the original code:

The first bug concerned a numerical instability in the computation of the B2 parameter as listed in Eq. 34 in [24].

The initial alogorithm implemented the calculation of square root of the difference between two expressions. It turned out that the numerical values of both expressions could sometimes be very large and nearly equal.

Because of limited precision in floating point calculations, the difference could sometimes lead to negative values and the square root returned an undefined value (NaN). The integrator then failed to compute the integral and returned a value of zero, with the printing of a faintly related message that too high accuracy was required for integrator DGAUSS. The algorithm didn't stop there and the end result was that the summation over all elements in the range was wrong and the end results were also wrong.

This bug was eliminated by evaluation the first expression in equation 34 which calculates directly the B2 factor by taking the square root of the sum of two squares, hence ensuring that an instability of the same kind cannot happen.

Another problem was that in the original algorithm the inverse Touschek lifetime was calculated by taking the average of the twiss parameters at both ends of the element as input. The resulting set of parameters was no longer consistent, resulting also in poor calculation. This has been changed by calculating the inverse Touschek lifetime at specific points, always

considering as input the Twiss parameters given by Twiss at a single location. This provides at least very accurate results for the TLI parameters.

The integration over the length of the element is now done in different ways, depending whether the preceding TWISS command calculated the Twiss parameters at the end of the element or at the centre (CENTRE option of TWISS).

In the first case (calculation at the end of the element, CENTRE=false), the inverse Touschek lifetime (TLI) is calculated at the end of each element. The weighted contribution of element i to the total inverse Touschek lifetime is then given by

```
TLIW[i] = 0.5 * (TLI[i] + TLI[i-1]) * L[i] / CIRC
```

In the second case (calculation at the center of the element, CENTRE=true), the inverse Touschek lifetime (TLI) is also calculated at the center of each element. The weighted contribution of element i to the total inverse Touschek lifetime is then given by

```
TLIW[i] = TLI[i] * L[i] / CIRC
```

Another bug that was uncovered in the original algorithm was that the vertical dispersion was wrongly taken into account and mostly ignored: DY and DPY were set uniformly to half the initial values for lack of updating in the loop over elements.

The new algorithms have been inserted in `MAD-X` as of version 5.01.04, a development release dated early december 2013.

Chapter 30. Intra-Beam Scattering

The Intra-Beam Scattering command computes the contribution to emittance growth rates due to Coulomb scattering of particles within relativistic beams. The algorithms in this module have been derived from the formalism presented in 1982 by J.D. Bjorken and S.K. Mtingwa [27], and are also using the expansion of M. Conte and M. Martini [28] developed in 1985, generalized to the case of nonzero vertical dispersion.

The present implementation of the IBS module in `MAD-X` is described in a forthcoming note [29].

The syntax of the `IBS` command is:

```
IBS, FILE=string;
```

The `IBS` command has one attribute:

FILE　　　　　outputs the resulting "ibs" table to the named file. (Default: "ibs")

The Bjorken-Mtingwa formalism takes into account the variation of the lattice parameters (beta and dispersion functions) around the machine and consequently, the knowledge of the optical functions along the machine is required: `IBS` should only be called after fully qualified `BEAM` command and a `TWISS` command.

Warning: Calling `EMIT` between the `TWISS` and `IBS` commands leads to `IBS` using wrong beam parameters, even if the `BEAM` command is reiterated.

The `IBS` module does not include a consistent treatment of linear betatron coupling.

The intra-beam scattering growth times are given by:

$$\frac{1}{\tau_i} \quad = \quad C_i \times \frac{N}{\gamma \epsilon_x \epsilon_y \epsilon_s} \qquad (i = x, y, s)$$

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. These key beam parameters must be specified through the `BEAM` command.

If the `CENTRE=true` option of `TWISS` was specified, the optical functions are calculated by `TWISS` at the center of each element and `IBS` uses these values for the element. If by default `TWISS` calculated the optical functions at the end of each element, `IBS` calculates the values at the center of each element by performing a linear interpolation between the end values for the previous element and the end values for the current element.

Input of the beam parameters:
A number of parameters have to be present in the `BEAM` command in order to run the IBS module:

PARTICLE　　　This is mandatory but `MAD-X` provides default value of `PARTICLE=proton`. For ions, this parameter specifies only the name of the ions, and the MASS

(approximated to the atomic unit number times the unified atomic mass `UMASS`) and CHARGE must be provided as well.

NPART   the number of particles (or number of ions).

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`.

An important check for the correctness of the input is the printed value of the relativistic factor γ. The latter should correspond to:

$$\gamma_{ion} \quad = \quad \gamma_{proton} \times \frac{charge}{nucleon}$$

emittances   This part of the input is used to define the normalized horizontal, vertical and longitudinal emittances. The required parameters are the `physical` transverse emittances, `EX` and `EY`, and the longitudinal emittance `ET`.

The longitudinal emittance is defined as the product of the bunch length `SIGT` times the relative energy spread `SIGE`, which are therefore required input.

If only the longitudinal emittance is defined, and `SIGT` and `SIGE` are omitted, an active RF cavity is also necessary in the lattice to infer `SIGT` and `SIGE`.

Example of BEAM input:
A beam of fully stripped Lead ions at the LHC injection energy may be defined as follows for `IBS` calculations:

```
nucleon = 208;
charge = 82;

BEAM, PARTICLE= lead, CHARGE= charge, MASS= nucleon*umass,
      ENERGY= 450*charge, NPART= 1.1E7, BUNCHED,
      EX= 7.82E-9, EY= 7.82E-9, SIGE= 4.68E-4, SIGT= 0.115;
```

Resulting Table and File:
The `IBS` command produces a table "ibs" containing the following data for each element of the machine: element name, position, optical functions (beta, alfa, dispersion and derivative) in both transverse planes, as well as the particular variables `DELS`, the length difference in meters between consecutive elements, and `TXI, TYI` and `TLI`, the IBS growth times in the two transverse and longitudinal planes.

This table can be accessed through the usual mechanisms. If the attribute `FILE="file_name"` is also given, `MAD-X` writes the table to the named file.

Features:
The average growth rates in [sec] are defined as variables called `ibs.tx`, `ibs.ty`, `ibs.tl` for the horizontal, vertical and longitudinal growth times respectively. They are directly accessible as variables after the `IBS` command, e.g.

```
IBS;
Tx = ibs.tx;
```

defines a variable Tx which is the average horizontal growth rate in seconds.

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://madx.web.cern.ch/madx/madX/examples/ibs/.

Chapter 31. Particle Tracking

31.1 Introduction to `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` keeps most of the functionality of `MAD-8`, the trajectory tracking in `MAD-X` is considerably modified compared to `MAD-8`. The reason is that in `MAD-8` 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 the form of an additional kick on either end of the element. This method requires a preliminary conversion of a sequence with thick elements into one composed of thin elements (see the `MAKETHIN` command).

The second part of this design decision is to produce a thick lens tracking module based on the `PTC` code of E. Forest 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 that it treats thick-lenses in a symplectic manner.

There is a second `PTC` tracking module called the line tracking module (`PTC_TRACKLINE`). It was developped for tracking particles in CLIC, with the specificities that it can deal with beam-lines containing traveling-wave cavities and includes actual beam acceleration.

31.2 Overview of Thin-Lens Tracking

The **thin-lens tracking module** of `MAD-X` performs element per element tracking of one or several 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.

Several commands are actually required to complete a tracking run:

```
TRACK, DELTAP=real, ONEPASS=logical, DAMP=logical;
       QUANTUM=logical, SEED=integer, UPDATE=logical,
       ONETABLE=logical, RECLOSS=logical, FILE=filename,
       APERTURE=logical,ONLY_AVERAGE=logical;
    ...
    START, X=real, PX=real, Y=real, PY=real, T=real, PT=real;
    START, FX=real, PHIX=real, FY=real, PHIY=real,
           FT=real, PHIT=real;
    ...
    OBSERVE, PLACE=string;
    ...
    RUN, TURNS=integer, MAXAPER=double_array, FFILE=integer;
    ...
    DYNAP, TURNS=real, FASTUNE=logical, LYAPUNOV=real,
           MAXAPER=real_array, ORBIT=logical;
    ...
ENDTRACK;
```

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 creates a number of internal tables and can create files on disk: `TRACKSUMM`, `TRACKLOSS`, and `TRACKONE` or `TRACK.OBS$$$$.P$$$$` (depending on the attribute `ONETABLE` of the `RUN` command) `ONLY_AVERAGE` when used with ONETABLE it will only output the average of all the particles.

These internal tables can be accessed via the [TABLE](#)-access functions.

Plotting of particle coordinates or other data in these tables is possible in `MAD-X`. Plotting can also be done with external programs by using the files created by `TRACK`.

`MAD-X` also has the capability to treat space-charge during tracking runs. There is no space-charge command per se but space charge is controlled through several options of `MAD-X` (see

OPTION) and specific attributes of the RUN command in this TRACK environment. A section specific to space charge options and particularities appears below.

31.3   TRACK

The TRACK command initiates trajectory tracking by entering the thin-lens tracking module.

```
TRACK, DELTAP=real, ONEPASS=logical, DAMP=logical;
       QUANTUM=logical, SEED=integer, UPDATE=logical,
       ONETABLE=logical, RECLOSS=logical, FILE=filename,
       APERTURE=logical;
```

The attributes of the TRACK command are:

DELTAP   relative momentum offset for reference closed orbit (switched off for ONEPASS) Defining a non-zero DELTAP results in a change of the beam momentum/energy without changing the magnetic properties in the sequence, which leads to an off-momentum closed orbit different from the on-momentum reference orbit. Particle coordinates are then given with respect to this new closed orbit, unless the option ONEPASS=true is used!
            (Default: 0.0)

ONEPASS   flag to ensure that no closed orbit search is done, which also means that no stability test is done. This is always the case for transfer lines, but this option can also be enabled for multi-turn tracking of a circular machine. ONEPASS=true does **NOT** restrict tracking to a single turn.
            With ONEPASS=true, the particle coordinates are specified with respect to the reference orbit.
            With ONEPASS=false, the closed orbit is calculated and the particle coordinates are given with respect to the closed orbit coordinates.
            This flag affects the behavior of the BBORBIT flag.
            The name of this attribute is misleading but was kept for backwards compatibility.
            (Default: false)

DAMP   flag to introduce synchrotron damping (needs RF cavity and flag RADIATE in the BEAM command).
            (Default: false)

QUANTUM   flag to introduce quantum excitation via random number generator and table look-up (SYNRAD = 1, see ref. [30]) or polynomial interpolation (SYNRAD = 2, see ref. [31]) for photon emission. The choice of the generator can be selected via the command OPTION attribute SYNRAD.
            (Default: 2)

SEED   If QUANTUM is true, seeds or starts a particular sequence of random values. A valid SEED value is an integer in the range [0...999999999], or an expression that evaluates to an integer in the same range (default: 123456789).
            Internally the code takes as the effective seed the value ABS(seed)%1.e10 hence

normalizing the provided seed to an appropriate value in the range irrespective of the provided value.

Note that the seed set with this command is shared with the COPTION and EOPTION commands. See also: Random Values.

DUMP   flag to write the particle coordinates in files, whose names are generated automatically.
(Default: false)

APERTURE a logical flag to trigger aperture check at the entrance of each element (except DRIFTs). A particle is lost from the table of tracked particles if its position lies outside the aperture of the current element at the entrance of this element.
(Default: false)

The APERTYPE and APERTURE information of each element in the sequence is used to assess the particle loss. However TRACK only takes into account the predefined aperture types listed in table 23.1

Note that if no aperture information was specified for an element, the following procedure still takes place:
$\rightarrow$ No aperture definition for element $\rightarrow$ Default apertype/aperture assigned (currently this is APERTYPE=circle, APERTURE={0})
$\rightarrow$ 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.
$\Rightarrow$ Hence even if no aperture information is specified by the user for certain elements, default values will be used!

ONETABLE flag to write all particle coordinates in a single file instead of one file per particle.
(Default: false)

RECLOSS   flag to create in memory a table named "trackloss" containing the coordinates of lost particles.
(Default: false)
Traditionally, when a particle is lost on the aperture, this information is written to stdout. To allow more flexible tracking studies, the coordinates of lost particles and additional 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

FILE   name for the track table. The default name is different depending on the value
              of the `ONETABLE` attribute.
              (Default: "track" if `ONETABLE=true`, "trackone" if `ONETABLE=false`)

UPDATE   flag to trigger parameter update per turn.
              (Default: false)
              Note that

tr$macro   needs to be defined even if only the access to the turn number `tr$turni` is
              used. Specifying `UPDATE=true` gives access to the following additions:

   tr$turni   this special variable contains the turn number; it can be used
                 in expressions like KICK := SIN(`tr$turni`) and is updated at
                 each turn during tracking.

   tr$macro   this special macro can be user-defined and is executed/updated
                 at each turn, during tracking. A macro structure is necessary
                 to provide for table access. *e.g.*
                 `tr$macro(turn):  macro={`
                 `commands that can depend on the turnnumber;`
                 `};`

   KEEPTRACK   a logical flag to keep data from previous tracking and append
                 new results to tables. (Default: false)

Remarks
IMPORTANT: If an RF cavity has a non-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.

31.4   START

After the `TRACK` command, initial trajectory coordinates must be provided for each trajectory or particle to be tracked, with one `START` command per trajectory or particle.

The coordinates can be expressed as either **canonical** or **action-angle** coordinates.

```
START, X=real, PX=real, Y=real, PY=real, T=real, PT=real;
START, FX=real, PHIX=real, FY=real, PHIY=real,
       FT=real, PHIT=real;
```

For the case of action-angle coordinates, 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 given in the `BEAM` command) **in each mode plane**. The phases are Φ_X, Φ_Y and Φ_T expressed in radian. In the uncoupled case, we have in the plane mode labelled z, and with E_z being the r.m.s. emittance in that plane:

$$Z = F_z \sqrt{E_z} \cos \Phi_z, \qquad P_z = F_z \sqrt{E_z} \sin \Phi_z \qquad (31.1)$$

The attributes of the START command are:

X, PX, Y, PY, T, PT canonical coordinates.

FX, PHIX, FY, PHIY, FT, PHIT action-angle coordinates.

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` of the TRACK is used, the coordinates are specified with respect to the reference orbit.

31.5 OBSERVE

During the tracking process, particle coordinates at specific named locations along the machine can be printed to file(s). The declaration of an observation point is with the OBSERVE command:

> OBSERVE, PLACE=string;

The single attribute of `OBSERVE` is:

PLACE   the name of the observation point.

Several `OBSERVE` commands can be given for the same tracking job, one per observation point.

If no `OBSERVE` command is given in a tracking job, but the `DUMP` option in the `TRACK` command is used, the trajectory coordinates are still recorded and one observation point is provided at the starting point of the sequence.

The output files are named automatically. The name given by the user (attribute `FILE` of the `TRACK` command) is followed by ".obsnnnn", where nnnn is the observation point number, and followed by ".pnnnn" wherer nnnn is now the particle number. Hence the default filename for the first obseration point and first particle looks like `track.obs0001.p0001`.

31.6 RUN

The actual tracking is triggered by the `RUN` command.

> RUN, TURNS=integer, MAXAPER=real_array, FFILE=integer, KEEPTRACK=logical;

The `RUN` command has three attributes:

TURNS   number of turns to be tracked.

MAXAPER   defines the maximum aperture (by aperture type) beyond which the particle
              is considered to be lost upper and, in addition, limits for the six coordinates.

(Default: {0.1, 0.01, 0.1, 0.01, 1.0, 0.1}
The limits defined by the `MAXAPER` option are only being taken into account if the `APERTURE` option of the `TRACK` command is used.

FFILE   defines the turn periodicity for printing coordinates at observation points. (Default: 1)

`FFILE=n` will print coordinates every n-th turn only.

TRACK␣HARMON is used to calculate the maximum time difference before a particle is considered lost (t_{max}). $t_{max} = \frac{C}{h_{track*\beta}}$ where h_{track}=TRACK␣HARMON and C is the total length of the machine. (Default: 1)

31.7   DYNAP

The `DYNAP` command calculates tunes, tune footprints, smear and Lyapunov exponent from tracking data. `DYNAP` can be called instead of `RUN` inside a `TRACK` command environment.

```
DYNAP, TURNS=integer, FASTUNE=logical, LYAPUNOV=real,
       MAXAPER=real_array, ORBIT=logical;
```

For each previously entered start command, `DYNAP` tracks two close-by particles over a selected number of turns (minimum 64 and maximum 1024), 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 $2 \times ($ $wxy_{max} - wxy_{min}$ $)/($ $wxy_{max} + wxy_{min}$ $)$, where the $wxy_{min,max}$ refer to the minimum 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 formula (18) in reference [32] if the number of turns is 64 or less, or formula (25) in the same reference if the number of turns is strictly larger than 64.

`DYNAP` has the following attributes:

TURNS   the number of turns to be tracked (Default: 64, minimum: 64 and maximum: 1024).

FASTUNE   a logical flag to compute the tunes. (Default: false)

MAXAPER   a vector of 6 real numbers defining the maximum aperture beyond which the particle is considered to be lost.
                 (Default: {0.1, 0.01, 0.1, 0.01, 1.0, 0.1}

LYAPUNOV   the initial distance which is added to the x coordinate of the companion particle of every particle declared with `START` commands. (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.**

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` command 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 equal to the value of the `LYAPUNOV` attribute for each `START` definition over `TURNS` revolutions, and `ENDTRACK` terminates the execution of the tracking module.

The results are stored in the `DYNAP` and `DYNAPTUNE` tables, and can be obtained by the two commands

```
VALUE, TABLE(dynap,smear);
VALUE, TABLE(dynaptune,tunx),
       TABLE(dynaptune,tuny),
       TABLE(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.

31.8   ENDTRACK

Tracking is terminated by the command `ENDTRACK` with no attributes.

```
ENDTRACK;
```

31.9   Space Charge

`MAD-X` can perform tracking using a frozen space charge model using beam-beam elements in a thin lattice. This process requires careful setting of several options and switches as well as the insertion of space-charge kicks inserted within regular elements. The Space-Charge specifics of `MAD-X` are documented in [33].

Two new additional tables mytracksumm and mytrackloss have been added with the table types and columns being defined in mad_gcst.h and mad_gcst.c. The tables themselves are created in mad_track.c and are copies of the original tracksumm and trackloss tables except that they are no longer deleted and recreated each time a RUN control command is issued. These old tables are kept to avoid potential interference with standard trrun calls. The table names needed to be changed to avoid possible confusion with the old tables.

A set of tools has been added to control various options of the mytracksumm table. It is assumed that the mytrackloss table will always be small, the maximum entries being the total number of particles in the simulation. Therefore, no further control parameters were needed for mytrackloss table.

The tools for the mytracksumm table control an optional and fixed limits on the number of lines entered, the frequency of entering a turn into the table and options to control a range

of which particle tracks are entered into the table.

The new MAD-X *options* must be refined before any RUN command.

31.9.1   Space-Charge Flags

The first set of logical flags are MAD-X *options*.

- **bb_ultra_relati (Default = FALSE)**
  For the nominal BB treatment the proper energy has been taken into account with this
  flag set to **FALSE**. For the SC treatment we need to be in control to apply the proper
  gamma and beta functions to achieve the desired SC kick. To this end the flag has to
  be set to **TRUE**.

- **bb_sxy_update (Default = FALSE)**
  This flag if **FALSE** switches off any remnant of the space charge treatment so that
  the nominal tracking operates as usual. Obviously, the flag has to be set to **TRUE** to
  activate the space charge treatment.

- **emittance_update (Default = TRUE)**
  In case the user does not want the adaptive nature of the space charge treatment it can
  be switched off by setting this flag to **FALSE**. In that case the sigma values in all 3
  planes (that depend on the emittances) are determined at the start of the run and kept
  constant. This flag becomes defunct for the implementation after 2018 since in that
  case the emittances are always updated.

- **fast_error_func (Default = FALSE)**
  An additional speed-up of some 30% can be achieved by putting this flag to **TRUE**.
  Internally to MAD-X an optimized evaluation of the complex error function has been
  implemented following the work of late G.A Erskine.

- **sc_chrom_fix (Default = TRUE)**
  This flag basically corrects for a programming bug early on that did not properly con-
  sider the fifth coordinate. It should always be kept at **TRUE** except for occasional runs
  that should be backwards compatible.

- **exit_loss_turn (Default = FALSE)**
  The problem with the **checkpnt_restart** flag (see below) is that in case of a particle
  loss a bit-by-bit precision of a continuation of the simulation run is not possible. In
  fact, it rather is the reverse, meaning that once a particle loss takes place there is no
  reasonable way of programming in MAD-X that would allow a for a correct evaluation
  of the simulations after such a particle loss. This has to do with the fact that right
  after the loss one would have to initiate a MAD-X TWISS which is not possible from
  within TRRUN. The **exit_loss_turn** flag has been introduced to minimize this problem
  by stopping the TRRUN module at the end of the turn in which the loss has occurred.
  This will cause a minor deviation only and thereafter the simulations will carry on
  correctly.

The 2018 flags to introduce a symplectic 3D SC kick and the "periodic" and "free" or self-
consistent modes are (please refer to Ref. [34] for details):

- **sc_3d_kick (Default=FALSE)**
  When this flag is set to **TRUE** the 2018 implementation becomes active that features this 3D symplectic SC kick and the following flag.

- **sc_3d_periodic (Default=FALSE)**
  This flag needs the above flag to be set to: **sc_3d_kick=TRUE**. For **sc_3d_periodic=TRUE** the "period" version is used as explained in Ref. [34] this is an adaptive mode type while for **sc_3d_kick=FALSE** the "free" or self-consistent mode is used. As attractive as the "free" mode may seem, for most practical purpose it will be of little use since significant noise is created that leads to large artificial emittance growth. To overcome this issue very large number of macro-particles would be needed resulting in prohibitive slowing of the computing speed.

- In Fig. 31.3 one finds the reference to two required optics parameter files "my_sector.tfs" which holds the Twiss sectortable at each element and the file "rmatrix.dat" which is the one-turn transfer matrix at the end of the machine. These two files maybe produced beforehand.

  Find below the MAD-X syntax to create these files:

  ```
  select,flag=sectortable,clear;
  select,flag=sectortable,column=name,s,r11,r12,r13,r14,r15,r16,
  r21,r22,r23,r24,r25,r26,r31,r32,r33,r34,r35,r36,r41,r42,r43,r44,r45,r46,
  r51,r52,r53,r54,r55,r56,r61,r62,r63,r64,r65,r66;
  twiss,sectormap,SECTORFILE=my_sector.tfs;
  select, flag=twiss, range=#s,column=name,s,RE;
  twiss, table=twiss,rmatrix,file=rmatrix.dat;
  ```

The new 2022 logical flags of the MAD-X with Space Charge *options* are:

- **sc_mytracksumm_maxlines (Default=4000008)**
  The set Value is printed as an Info.

  The default value comes from Npart(=2000)*Turns(=1000)*Safety(=2)+8. At the end of each RUN command, and after any checksum_restart file is written, the tracks from the last turn are written to the mytracksumm table (see, however the sc_mytracksumm_per_turns control below) and the total number of lines in the table are checked with a table_range call. If they exceed the maxlines limit the code stops with an error message.

  There is an additional pre-compiled limit on the number of lines in the variable

- **mytracksumm_maxlineso (Default=16000008)**
  This is an additional pre-compiled limit on the number of lines which is defined in util.f90 within the module SCdat. If this line limit is exceeded the code stops with an error message.

- **sc_mytracksumm_per_turns (Default=1)**
  The set Value is printed as an Info.

  This option controls the frequency at which a mytracksumm table entry is made after the last turn of a RUN command. With the default value of 1 every last turn of a RUN command is entered. With a value of 2 only the last turn of every second RUN

command is entered and so on. Note that the first turn, turn 0, of a MAD-X execution is always written to the mytracksumm table. Redefinition of sc_mytracksumm_per_turns between RUN commands is allowed and takes immediate effect. A simple way to enter the final turn of a multi-turn RUN sequence into the mytracksumm table would be to add a final one turn RUN command.

- **sc_mytracksumm_start_particle (Default=1)**
  The set value is printed as an Info.

  This, and the following option, control the range of particle identifiers that are written to the mytracksumm table. At the start of the first RUN command the particle identifiers are renumbered incrementally from 1 to the number of particles either read from a checkpoint_restart file or a start_particles file. The start and end particle ranges are for this numbering convention though in the mytracksumm table the particle identifiers stored may be different from this original numbering in the case of a lost particle, for example if particle 1 is lost then the first particle in the mytracksumm table range from 1 will have the particle id of 2.

- **sc_mytracksumm_end_particle (Default=0)** The set value is printed as an Info.

  The default value of 0 sets the end particle to be the last one.

  This option defines the last particle in the range as above to be entered into the mytracksumm table. The default value of 0 sets the end particle to be the last one defined. In the case of a lost particle the end_particle value will be set to the minimum of the defined value and the remaining number of particles.

The second set of parameters/flags are either created automatically, are set in the **BEAM** command or are input to the **RUN** MAD-X command in the **TRACK** environment.

- **N_particles**, Number of Particles used in the MAD-X **BEAM** command as **Npart**:
  In the third step of the preparatory phase-I this parameter has to be set in the **spcharge_input_parameters.madx** MAD-X input file. Notice, that in our realization of MAD-X with Space Charge, it is supposed that **N_particles** is the number of particles in a coasting beam. It can be evaluated with the Laslett tune shift formula Eq. 31.5 (see Sec. 31.9.4) with a default value for $L_{\text{bunch}} = L_{\text{ring}}$, which ensures $B_f = 1$ for the coasting beam.

  For bunched Gaussian beams the Eq. 31.6 can be used. To this end B_f has to be calculated via Eq. 31.9 which can be achieved by simply setting the (effective) **Bunch_length** to $\sqrt{2\pi} \cdot \sigma_z$ in [m] in the **spcharge_input_parameters.madx** file.

- **Ex_spch, Ey_spch**, Un-normalized transverse Emittance:
  set also in the **spcharge_input_parameters.madx** MAD-X input file. As an example here the normalized emittances of the PS experiments transferred to the un-normalize ones:
  Ex_spch=3.5E-6/gamma/beta; Ey_spch=2.2E-6/gamma/beta;.

- **N_part_gain (DEFAULT = 1.)**, a **RUN** command attribute:
  is a supplementary factor for **N_particles** used as a knob for precise settings (or corrections) of tune shift values. For example, it can be used either for serial express

calculations with slightly different tune shifts or in order to compensate tune shift reductions in arcs with $(D > 0)$ for bunches with significant momentum spreads (dp¿0) as illustrated in the Appendix.

- **checkpnt_restart (DEFAULT = FALSE)**, a **RUN** command attribute:
  This flag set to **TRUE** allows to continue a previous run with bit-by-bit precision. All relevant data for such a run continuation is written in binary format to the file **checkpoint_restart.dat** at the end of each tracking run. See however limitations in case of lost particles as discussed above for the **exit_loss_turn** flag.

The following four **RUN** input parameters are not essential for SC calculations but are meant to detect particles outside of the bucket. Start coordinate input files with particles outside the bucket will not be accepted and the MAD-X run stops with an error. During the run a particle will be considered lost once it is outside the bucket. This has been found necessary to avoid the frequent appearance of NaNs during runs. Moreover, a proper error handling has been put into place.

- **sigma_z (DEFAULT = 0.)**: Longitudinal beam sigma in [m].

- **track_harmon (DEFAULT = 1)**: Harmonic Number: Originally this optional parameter has been introduced to safeguard the user from particles outside the bucket at the beginning of the run. However, in cases of frozen SC treatment without adaptive changes to the longitudinal "sigma_t", the value of "track_harmon" can be used to create any arbitrary initial "sigma_t" value.

- **deltap_rms (DEFAULT = 0.)**: RMS of deltap over p.

- **deltap_max (DEFAULT = 1.e20)**: Maximum value of deltap over p.

For the functioning of the code there are internal parameters that are listed here for a better understanding. None of the following can be modified from the input:

- **N_macro_max = 10000**, Maximum allowed macro-particle number: This limit is hardwired in the code, on request this value may be increased.

- **N_macro_surv**, Surviving Macro-Particle Number:

  The total surviving number of macro-particles is set automatically to the number of lines in the start coordinate file which the user has to provide. Fig. 31.1 shows an example of how such a file has to be set-up.

  In fact, in the tracking step some external tools have been provided to create a 3D Gaussian distribution using the adequate beam sigma values in the 3 planes.

  During run time this number is reduced after each particle loss.

- **z_factor**, Factor of the initial longitudinal sigma divided by the actual longitudinal sigma.

- **N_for_i**, Number of macro-particles found to be in the Gaussian distribution: this is calculated by a special routine [1] that also calculates the emittances from the distribution of macro-particles.

---

[1] The algorithm for the routine has been formulated by Yu. Alexahin.

- **rat_bb_n_ions**, Factor of the SC strength: $\frac{N\_for\_i}{N\_macro\_surv} \times$ **z_factor**. This factor is applied to the SC strength when **emittance_update = TRUE**.

```
start, x= 0.14983218580000000E-02,px=-0.33121724759999999E-03,y=-0.11548748040000000E-03,py= 0.30503126469999999E-04,t= 0.12096721505993868E+01,pt= 0.16448544546669575E-03;
start, x= 0.62788231440000003E-03,px= 0.25065244059999998E-03,y=-0.16341025319999999E-02,py=-0.11746341559999999E-03,t=-0.10313597706481812E+01,pt= 0.67674812799810343E-04;
start, x=-0.97863238140000002E-03,px=-0.16348000739999999E-03,y=-0.51148806819999995E-03,py= 0.86579932350000006E-04,t= 0.10477483145882251E+01,pt=-0.16975810426045928E-03;
start, x=-0.82378244570000003E-03,px=-0.20154635140000001E-03,y=-0.36092287940000002E-04,py= 0.24149527899999999E-03,t=-0.20860040960512372E+01,pt= 0.11105404364394674E-03;
start, x= 0.79818500939999990E-03,px=-0.21314146560000001E-03,y= 0.66062239059999997E-03,py= 0.61991739220000001E-04,t= 0.36667770021034642E+01,pt=-0.45987961056920845E-04;
start, x=-0.49967394320000005E-03,px= 0.11507667110000000E-03,y=-0.11496230799999999E-02,py= 0.40743458370000003E-04,t=-0.15084879553233497E+01,pt=-0.37478240925088973E-03;
start, x=-0.61602402359999995E-03,px=-0.26358948600000001E-03,y=-0.22802241529999998E-02,py=-0.16877376590000001E-05,t=-0.77797355411443136E+00,pt=-0.28264758307979893E-03;
start, x= 0.31463706659999998E-02,px= 0.13883746299999999E-03,y=-0.46963670589999997E-04,py=-0.59689797529999998E-04,t= 0.49455521049733214E+01,pt=-0.27020022913837018E-03;
start, x=-0.13300175589999999E-02,px= 0.22996079990000000E-03,y=-0.12173648130000001E-02,py=-0.13402779460000000E-03,t= 0.39895760987764302E+01,pt=-0.19355013661109319E-03;
start, x=-0.48308748419999998E-04,px=-0.47829904779999998E-04,y=-0.19689866569999998E-02,py=-0.89511112829999999E-04,t=-0.24702265528702472E+01,pt=-0.36667007847272852E-03;
start, x=-0.65039961749999999E-03,px=-0.10260695910000000E-04,y= 0.10590818669999999E-02,py= 0.40141094239999997E-05,t=-0.86829525416224962E+00,pt= 0.71084936675702332E-04;
start, x=-0.74244653049999999E-03,px= 0.37679298500000000E-03,y= 0.89246224959999996E-03,py= 0.26488643140000002E-03,t=-0.32074041084174438E+01,pt= 0.36881277515358384E-03;
start, x=-0.13045069929999999E-02,px= 0.23251911860000001E-03,y= 0.25459941800000000E-03,py= 0.90614624539999994E-04,t= 0.43473158328956339E-01,pt= 0.16382022967224968E-03;
start, x=-0.89955667830000002E-04,px= 0.93013026649999999E-04,y= 0.20503800880000001E-03,py= 0.80048677459999994E-04,t=-0.47069131512660398E+01,pt=-0.81012847740542183E-04;
start, x=-0.15095759600000000E-02,px=-0.57501016270000000E-04,y= 0.74448886199999999E-03,py= 0.70720491409999997E-05,t= 0.16673429120509520E+01,pt=-0.19677585217329777E-03;
start, x= 0.11966627289999999E-02,px= 0.60254506000000000E-04,y= 0.17796135559999999E-02,py= 0.84000257520000003E-04,t=-0.31234935813856888E+01,pt=-0.55005776381986029E-03;
start, x=-0.92961658730000005E-03,px=-0.99119440959999998E-04,y=-0.88180799100000002E-03,py= 0.11589705710000000E-03,t= 0.37609188428884605E+00,pt= 0.27177409192088536E-03;
start, x= 0.11007479900000000E-02,px=-0.90091533900000004E-04,y= 0.31501711249999999E-04,py= 0.12239831420000000E-04,t= 0.24290912211820350E+01,pt=-0.38168142779408036E-03;
start, x=-0.39283910179999999E-03,px= 0.17145427829999999E-03,y=-0.20917740470000000E-02,py=-0.87840491889999997E-05,t=-0.35871909933056751E+01,pt= 0.17158569639119149E-03;
start, x= 0.92155942709999997E-03,px=-0.12667228030000001E-04,y=-0.23336179550000001E-03,py=-0.26675543399999998E-03,t=-0.13196970874173803E+01,pt= 0.32724046094537943E-04;
start, x=-0.45705260750000001E-03,px= 0.39046576499999999E-04,y= 0.77553588789999999E-03,py= 0.20397313830000001E-03,t=-0.81193795187432183E+00,pt= 0.45631581882575209E-04;
```

Figure 31.1: Example of a start coordinates file.

31.9.2 Set-Up Example

In this section it will be described how the user's MAD-X lattice will be instrumented with thin SC kicks. As an arbitrary example we are using the CERN PS. It uses MAD-X internal routines and a set MAD-X control and MACRO files.

This set-up is documented at the web-site: Set-up of MAD-X with Space Charge

One finds the following directories:

1. **Manual**
   There, one will find updated versions of this manual.

2. **6D-Matched-Distribution**
   This MAD-X SC implementation requires 6D linearly matched distributions of a couple of 1'000 macro-particles. One finds all tools and instructions to prepare such distributions adapted to the user's machine case.

3. **Set-Up_Macros**

   At this location one finds the following directories:

   (a) **PS-example**
       This directory hold's a fully instantiated example run for the CERN PS. It has been run on the CERN lxplus system via the command, given that one has placed " mad/bin" in the LINUX $PATH environment:

       `madx make_machine_sequence.madx | tee out_1`

       The MAD-X executables with Space Charge can be downloaded at the web at:

       The relevant output file is "ring_seq_bb_spch_thin.madx". This is the thin lens PS lattice file instrumented with SC kicks at a particular tune working point. This file is one of essential input file for the SC simulation together with the 6D linearly matched distributions, as discussed above.

(b) **Generic-Macros**

In this directory one finds the generic MACROs that should work for any machine case. In Appendix 31.9.5 the input file "madx_spch_input.madx" is explained that allows the standard user to modify a title in the Set-Up procedure. More relevant is the main input file "make_machine_sequence.madx" which is discussed in Appendix 31.9.5. Basically, this file consist of a part 1 where the user is required to set-up and simplify the lattice of the particular machine case to be studied. This is followed by part 2 where, after some standard MAD-X commands, the user is required to specify the machine parameters like "kinetic_energy_gev" and 2 MACRO control parameters explained in the Appendix. Last, for reference of the purpose of the applied MACROs in Appendix 31.9.5 the file "main_make_spch.madx" is shown. The standard user will not be required to change anything in that file.

31.9.3 Run Example

In Fig. 31.2 the SC tracking part of a typical runs is depicted:

- In a first step the TFS tables **time_var_mul** for time varying multipoles are read-in (see Fig. 31.4). The first column has the element name, the second is the running number of different multipoles. In this case there are 3 different sextupoles. The next column gives the number of turns after which the multipole strength will be changed. If turn numbers are missing no change of the multipoles will be applied for those turn numbers. The following 20 columns are reserved for multipoles of up to 10th order. The normal coefficients are followed by the skew ones: i.e. normal dipole, vertical dipole, normal quadrupole, skew quadrupole, normal sextupole etc. In this example the normal sextupoles (fifth multipole column) hold values different from zero. The remaining 15 columns are not shown.

  Then the TFS table of phase trombone **time_var_pha** is read-in (see Fig. 31.5). It has the same structure as the TFS table for multipoles. However, in this case the 36 matrix elements are contained in this table (32 columns have been suppressed in the figure). In this example there is only one phase trombone.

  This example contains no time varying cavity. The **time_var_cav** file would just have the necessary first 3 columns and a single column with the desired **VOLT** in [MV].

- Notice that after any TWISS command one has to read-in the table **spch_bb_twiss_ini.dat**.

- A set of flags are needed to activate the SC treatment.

- The MAD-X **TRACK** environment is being opened and the start coordinates are being read-in.

- On the MAD-X **RUN** command special care has been taken to avoid numerical exception by defining the RF bucket.

- A run with a single turn is followed by a loop of runs with a TWISS parameter recalculation after every turn at all positions of the SC elements. Notice, that the table **spch_bb_twiss_ini.dat** has to be read-in after each TWISS command.

- Last, the tracking output is written out for the emittance evolution, the final TWISS parameters, the 6D coordinate printout and the record of lost particles respectively.

- Fig. 31.3 shows a **RUN EXAMPLE** available in the implementation after 2018 featuring the flags **sc_3d_kick**, **sc_3d_periodic=TRUE** ("periodic" mode) and **sc_3d_periodic=false** ("free" mode), see above for more details.

```
readtable, file="time_var_mul";
readtable, file="time_var_pha";
readtable, file="spch_bb_twiss_ini.dat";

option,bborbit=true;
option,bb_sxy_update=true;
option,bb_ultra_relati=true;
option,sc_chrom_fix=true;
option,-info,-warn,-echo;

ap_max:=0.5; !
TRACK, onepass, recloss, APERTURE;
 CALL, FILE="start_2k_particles.dat";
 deltap_rms=0.000555;
 deltap_max=5*deltap_rms;
 RUN, turns=1000, maxaper={ap_max,1,1,1,1000,1},
      n_part_gain=1,
      sigma_z=4.794126487, track_harmon=2,
      deltap_rms=deltap_rms, deltap_max=deltap_max;
 n=1;
 while (n < 10)
 {
   value n;
   twiss, table="spch_bb",file="spch_bb_twiss.dat";
   readtable, file="spch_bb_twiss.dat";
   RUN, turns=1, maxaper={ap_max,1,1,1,1000,1};
   n = n + 1;
 };
ENDTRACK;
write,table=bb6d_ixy,file="bb6d_ixy.txt";
write,table=tracksumm,file="tracksumm";
write,table=trackloss,file="trackloss";
STOP; !=============================================
```

Figure 31.2: A typical run example before the 2018 implementation, i.e. adaptive and purely frozen mode.

```
!=====================================================
!readtable, file="time_var_mul";
!readtable, file="time_var_pha";
!readtable, file="spch_bb_twiss_e.dat";
option,bborbit=true;
option,bb_sxy_update=true;
option,bb_ultra_relati=true;
option,sc_chrom_fix=true;
option,sc_3d_kick=true;
option,sc_3d_periodic=true;
!option,sc_3d_periodic=false;
option,-info,-warn,-echo;
ap_max:=0.5; !
select,flag=sectortable,column=name,s,r11,r12,r13,r14,r15,r16,r21,r22,r23,r24,
r25,r26,r31,r32,r33,r34,r35,r36,r41,r42,r43,r44,r45,r46,r51,r52,r53,r54,r55,
r56,r61,r62,r63,r64,r65,r66;
twiss,sectormap,SECTORFILE="my_sector.tfs";
select, flag=twiss, range=#s,column=name,s,RE;
twiss, table=twiss,rmatrix,file="rmatrix.dat";
 TRACK, onepass, recloss, APERTURE;█
 CALL, FILE="start_particles.dat";
 deltap_rms=0.00095;
 deltap_max=5*deltap_rms;
 RUN, turns=1000, maxaper={ap_max,1,1,1,1000,1},
      n_part_gain=1, checkpnt_restart,
      sigma_z=9.59489078280046, track_harmon=2,
      deltap_rms=deltap_rms, deltap_max=deltap_max;
 write,table=bb6d_ixy,    file="bb6d_ixy.txt";
 write,table=mytracksumm, file="tracksumm";
 write,table=mytrackloss, file="trackloss";
 n=1;
 while (n < 200000)
 {
   value n;
   select,flag=twiss,clear;
   select, flag=twiss, range=#s,column=name,s,RE;
   twiss, table=twiss,rmatrix,file=rmatrix.dat;
   RUN, turns=1000, maxaper={ap_max,1,1,1,1000,1};
   write,table=bb6d_ixy,    file="bb6d_ixy.txt";
   write,table=mytracksumm, file="tracksumm";
   write,table=mytrackloss, file="trackloss";
   n = n + 1;
 };
 ENDTRACK;
STOP; !=========================================
```

Figure 31.3: A typical run example after 2018 implementation, i.e. with the symplectic SC 3D kick and "periodic" or "free"(self-consistent) mode.

Figure 31.4 table:

@ NAME	%19s	"TIME_VAR_MUL"
@ TYPE	%08s	"TIME_VAR_MUL"
@ TITLE	%08s	"no-title"
@ ORIGIN	%19s	"MAD-X 3.04.00 Linux"
@ DATE	%08s	"28/08/12"
@ TIME	%08s	"00.59.05"

* NAME	NUMBER	TURN	dkn0	dks0	dkn1	dks1	dkn2
$ %s	%le	%le	%le	%le	%le	%le	%le
"sextres_no434"	1.	1.	0.0000000000000000000E+00	0.0000000000000000000E+00	0.0000000000000000000E+00	0.0000000000000000000E+00	0.0000000000000000000E+00
"sextres_no454"	2.	1.	0.0000000000000000000E+00	0.0000000000000000000E+00	0.0000000000000000000E+00	0.0000000000000000000E+00	0.0000000000000000000E+00
"sextres_no469"	3.	1.	0.0000000000000000000E+00	0.0000000000000000000E+00	0.0000000000000000000E+00	0.0000000000000000000E+00	0.0000000000000000000E+00
"sextres_no434"	1.	2.	0.0000000000000000000E+00	0.0000000000000000000E+00	0.0000000000000000000E+00	0.0000000000000000000E+00	0.1194000000000000010E-01
"sextres_no454"	2.	2.	0.0000000000000000000E+00	0.0000000000000000000E+00	0.0000000000000000000E+00	0.0000000000000000000E+00	0.1990000000000000005E-01
"sextres_no469"	3.	2.	0.0000000000000000000E+00	0.0000000000000000000E+00	0.0000000000000000000E+00	0.0000000000000000000E+00	0.1328900000000000005E-01
"sextres_no434"	1.	3.	0.0000000000000000000E+00	0.0000000000000000000E+00	0.0000000000000000000E+00	0.0000000000000000000E+00	0.7973400000000000028E-02
"sextres_no454"	2.	3.	0.0000000000000000000E+00	0.0000000000000000000E+00	0.0000000000000000000E+00	0.0000000000000000000E+00	0.6655999999999999593E-02
"sextres_no469"	3.	3.	0.0000000000000000000E+00	0.0000000000000000000E+00	0.0000000000000000000E+00	0.0000000000000000000E+00	0.3993599999999999993E-02
"sextres_no434"	1.	4.	0.0000000000000000000E+00	0.0000000000000000000E+00	0.0000000000000000000E+00	0.0000000000000000000E+00	0.1328900000000000005E-01
"sextres_no454"	2.	4.	0.0000000000000000000E+00	0.0000000000000000000E+00	0.0000000000000000000E+00	0.0000000000000000000E+00	0.1583999999999999984E-01
"sextres_no469"	3.	4.	0.0000000000000000000E+00	0.0000000000000000000E+00	0.0000000000000000000E+00	0.0000000000000000000E+00	0.2648999999999999784E-01
"sextres_no434"	1.	5.	0.0000000000000000000E+00	0.0000000000000000000E+00	0.0000000000000000000E+00	0.0000000000000000000E+00	0.1194000000000000010E-01
"sextres_no454"	2.	5.	0.0000000000000000000E+00	0.0000000000000000000E+00	0.0000000000000000000E+00	0.0000000000000000000E+00	0.1990000000000000005E-01
"sextres_no469"	3.	5.	0.0000000000000000000E+00	0.0000000000000000000E+00	0.0000000000000000000E+00	0.0000000000000000000E+00	0.3305700000000000030E-01
"sextres_no434"	1.	6.	0.0000000000000000000E+00	0.0000000000000000000E+00	0.0000000000000000000E+00	0.0000000000000000000E+00	0.1983419999999999997E-01
"sextres_no454"	2.	6.	0.0000000000000000000E+00	0.0000000000000000000E+00	0.0000000000000000000E+00	0.0000000000000000000E+00	0.1983419999999999997E-01
"sextres_no469"	3.	6.	0.0000000000000000000E+00	0.0000000000000000000E+00	0.0000000000000000000E+00	0.0000000000000000000E+00	0.3305700000000000030E-01

Figure 31.4: Example of a Time Variation Multipole TFS table.

Figure 31.5 table:

@ NAME	%19s	"TIME_VAR_PHA"
@ TYPE	%08s	"TIME_VAR_PHA"
@ TITLE	%08s	"no-title"
@ ORIGIN	%19s	"MAD-X 3.04.00 Linux"
@ DATE	%08s	"28/08/12"
@ TIME	%08s	"00.59.05"

* NAME	NUMBER	TURN	rm11	rm12	rm13	rm14
$ %s	%le	%le	%le	%le	%le	%le
"sect_matr_1"	1.	1.	0.1009338859545934945E+01	0.6827978492267716669E-01	0.0000000000000000000E+00	0.0000000000000000000E+00
"sect_matr_1"	1.	2.	0.1009642700233517110E+01	0.7050711017643174216E-01	0.0000000000000000000E+00	0.0000000000000000000E+00
"sect_matr_1"	1.	3.	0.1009946771486166298E+01	0.7273443308911869996E-01	0.0000000000000000000E+00	0.0000000000000000000E+00
"sect_matr_1"	1.	4.	0.1010250809207219999E+01	0.7496175358692443345E-01	0.0000000000000000000E+00	0.0000000000000000000E+00
"sect_matr_1"	1.	5.	0.1010544331042328556E+01	0.7711226756891784166E-01	0.0000000000000000000E+00	0.0000000000000000000E+00
"sect_matr_1"	1.	6.	0.1010848308282703542E+01	0.7933958310471363186E-01	0.0000000000000000000E+00	0.0000000000000000000E+00
"sect_matr_1"	1.	7.	0.1011141760980647796E+01	0.8149009215788290013E-01	0.0000000000000000000E+00	0.0000000000000000000E+00
"sect_matr_1"	1.	8.	0.1011445667900118086E+01	0.8371740244589135938E-01	0.0000000000000000000E+00	0.0000000000000000000E+00
"sect_matr_1"	1.	9.	0.1011739061224473668E+01	0.8586790629449053696E-01	0.0000000000000000000E+00	0.0000000000000000000E+00
"sect_matr_1"	1.	10.	0.1012032424345193940E+01	0.8801840748529533856E-01	0.0000000000000000000E+00	0.0000000000000000000E+00
"sect_matr_1"	1.	11.	0.1012336231697213766E+01	0.9024570941721876668E-01	0.0000000000000000000E+00	0.0000000000000000000E+00
"sect_matr_1"	1.	12.	0.1012629531042000608E+01	0.9239620499203053676E-01	0.0000000000000000000E+00	0.0000000000000000000E+00
"sect_matr_1"	1.	13.	0.1012922799044791977E+01	0.9454669770694436276E-01	0.0000000000000000000E+00	0.0000000000000000000E+00
"sect_matr_1"	1.	14.	0.1013216035696546783E+01	0.9669718749567078616E-01	0.0000000000000000000E+00	0.0000000000000000000E+00

Figure 31.5: Example of a Time Variation Phase Trombone TFS table.

31.9.4 Laslett Tune Shift Formulae

Here the formula as used in the file **spcharge_laslett_tune_shift.madx** in the third step of the preparatory phase-I, with the tunes from TWISS:

$$\nu_{x,y}, \tag{31.2}$$

averaged $\bar{\beta}$ value and length of the machine L_{ring} relate:

$$\bar{\beta}_{x,y} = \frac{L_{\text{ring}}}{2\pi\nu_{x,y}}, \tag{31.3}$$

and averaged $\bar{\sigma}$:

$$\bar{\sigma}_{x,y} = \sqrt{\epsilon_{x,y}\bar{\beta}_{x,y}}, \tag{31.4}$$

the SC tune-shift reads:

$$\Delta\nu_{x,y} = \frac{r_p N_p \cdot \bar{\beta}_{x,y}}{2\pi \cdot B_f \cdot \gamma^3\beta^2 \cdot \bar{\sigma}_{x,y}(\bar{\sigma}_x + \bar{\sigma}_y)}, \tag{31.5}$$

with B_f as the bunching factor.

For the particular case of bunched Gaussian beams the SC tune shift formula reads as follows:

$$\Delta\nu_{x,y} = \frac{r_p N_p}{(2\pi)^{\frac{3}{2}} \cdot \gamma^3\beta^2 \cdot \sigma_z} \cdot \oint \frac{\beta_{x,y}(s) \cdot ds}{\sigma_{x,y}(s)(\sigma_x(s) + \sigma_y(s))}. \tag{31.6}$$

After integration one gets:

$$\Delta\nu_{x,y} = \frac{r_p N_p \cdot \bar{\beta}_{x,y}}{(2\pi)^{\frac{3}{2}} \cdot \gamma^3\beta^2 \cdot \frac{\sigma_z}{L_{\text{ring}}} \cdot \bar{\sigma}_{x,y}(\bar{\sigma}_x + \bar{\sigma}_y)}. \tag{31.7}$$

The bunching factor can be defined as the ratio of the "effective length" $L_{\text{bunch}}^{\text{eff}}$ to the length of the ring L_{ring}.

$$B_f = \frac{L_{\text{bunch}}^{\text{eff}}}{L_{\text{ring}}}. \tag{31.8}$$

where the "effective" bunch length can be seen as the length of an ideal bunch with uniform distribution, which contains the same number of particles as a bunch with an arbitrary non-uniform distribution, e.g. a Gaussian bunch.

This implies that for Gaussian beams with the $L_{\text{bunch}}^{\text{eff}} = \sqrt{2\pi} \cdot \sigma_z$ the bunching factor B_f should read:

$$B_f = \sqrt{2\pi} \cdot \frac{\sigma_z}{L_{\text{ring}}}. \tag{31.9}$$

31.9.5 Beam Modes

Remarks on the transition from coasting to non-monochromatic and then to bunched beams.

Demonstrated with a numerical example for the FNAL debuncher [35]:

1.) For **coasting** 4D beam $N = 1.75 \cdot 10^{13}$ with a tune shift of $\Delta\nu = -0.0148$ there is a good agreement between the analytic Laslett's tune shift (at $B_f = 1$) and the numerical TWISS determination.

2.) For non-monochromatic beams in the debuncher, let's take into account the well-known Formulae for the beam size in the arcs with $(D > 0)$ for bunches with significant momentum spread [36]:

$$\sigma_{tot}^2 = \sigma_\beta^2 + [D_x(s)\sigma_p]^2. \tag{31.10}$$

With $(\sigma_p = 4.5 \cdot 10-3)$ and due to the large D_x the tune shift reduces to $(\Delta\nu = -0.0054)$. This value has been calculated numerically via MAD-X with both TWISS commands using Eq. 31.10 and TBT tracking over 1024 turns. Therefore, one should increase Npart for SC-elements by a factor of $N\_part\_gain = 2.7$, i.e. by $\frac{0.0148}{0.0054}$, to get back $(\Delta\nu = -0.0148)$ as for non-monochromatic beams.

3.) Number of particles in coasting and bunched beam producing the same Laslett tune shift:

$$\begin{aligned}
Npart(6D) &= Npart(4D, B_f = 1) * N\_part\_gain * B_f = \\
&Npart(4D, B_f = 1) * N\_part\_gain * (\sqrt{2\pi} * \sigma_z/L_{\text{ring}}) = \\
&1.75 \cdot 10^{13} * (2.7) * (2.5 * 12/505) = 2.8 \cdot 10^{12}.
\end{aligned} \tag{31.11}$$

General MAD-X Set-Up Input

The user may modify the input file "madx_spch_input.madx". In particular the **title_madx_script** entry maybe modified to the user's taste: it will appear in MAD-X output files during the Set-Up phase and does not alter the outcome otherwise. Changing the other two items is discouraged and should only be performed by experts since it will require various other changes in the generic MACROS.

```
*******************************************************************************
input file: madx_spch_input.madx:
*******************************
&input_filename
twiss_element_list_filename='twiss_element_list.txt'
                ! created in 1a_step
&END

&madx_job_title
title_madx_script='2021: PS Run'
&END

&ring_seq_filename
filename_ring_elements_seq='machine_sequence.madx'
                      ! ring sequence with SAVE, SEQUENCE=<>,file=<>.madx;
&END
*******************************************************************************
```

Preparing MAD-X Lattice for SC Kicks

The main input file "make_machine_sequence.madx" consist of a user defined simplified original lattice definition of the user's machine case. This part is followed by some generic MAD-X commands and a list of input parameters that the user needs to adapt to his case. This second part of the file is depicted below. Besides evident parameters like "kinetic_energy_gev" there are 2 relevant parameters:

1. **L_div_max**
   This parameter defines the maximum distance between 2 SC kicks in [m]. For the PS we have been using 1 [m] which has allowed for a dense enough coverage of SC kicks around the machine and about 1'100 kicks seems good enough. Obviously, one should not overdo the number of kicks because this number is directly proportional to the execution time of the run. As an example: for the PS 500'000 turns take about 6 weeks computing time on modern computer PCs.

2. **delta_nu_min**
   The MAD-X SC Set-Up procedure is turning on the strength of the SC kicks adiabatically to avoid potential TWISS command crashes because the strength of the SC kicks depend on the TWISS parameters itself. The **delta_nu_min** parameter controls the speed of the convergence. Typically we are using **delta_nu_min=0.25** but for the PS we needed to do it in finer steps: **delta_nu_min=0.07**.

```
********************************************************************************
input part of file: make_machine_sequence.madx:
**********************************************
!
! Create file=twiss_element_list.txt printing element list
 SELECT, flag=twiss,clear;
 SELECT, flag=twiss, column=name, keyword, L;
 TWISS, file=twiss_element_list.txt;

! Do former "2_step MAD-X SC internally
 option,sc_setup=true; ! required do not change
 TWISS;
 option,sc_setup=false; ! required do not change

! Write out the machine sequence as file=machine_sequence.madx
 SAVE,SEQUENCE=MACHINE,file=machine_sequence.madx;
 call,file=machine_sequence.madx;

!
!Machine Parameters: Example PS
!
!Parameters to be set
kinetic_energy_gev =   2.0 ;
Ex_norm=3.5E-6 ; ! Normalized horizontal emittance
Ey_norm=2.2E-6 ; ! Normalized vertical emittance
```

```
Bunch_length_4s=135e-9 ; ! 4 sigma bunch_length [ns]
NPART_tot=55e10 ; ! Total number of particles in the machine
L_div_max=1.; value, L_div_max;  ! Upper limit for segment length
delta_nu_min=0.07; ! Typical 0.25; ! FRS 02.09.2020 Constant for tune-shift of PS

L_ring:=table(summ,length);
value, L_ring;
rest_energy_gev =   pmass ;
total_energy_GeV:=kinetic_energy_GeV+rest_energy_GeV;
gamma=total_energy_GeV/rest_energy_gev;
beta=sqrt(1.-1./gamma^2);
value, gamma,beta;
Ex_spch:=Ex_norm/gamma/beta; Ey_spch:=Ey_norm/gamma/beta; ! [m] ! PS tracking
Bunch_length:= Bunch_length_4s/4.*clight*beta; ! PS 1 RMS bunchlength in [m]
value,Bunch_length, clight, beta;
NPART:=NPART_tot*L_ring/Bunch_length/sqrt(2*pi); ! PS new definition
value, NPART;
N_spch:=SC_count-1;
value, L_div_max,N_spch,SC_count,delta_nu_min;

! EXPERTS Usuage only! Needed for backwards compatibility mode
CALL, FILE=input_compatibility.madx; ! required do not change

! Do former "3_step" with generic MACROs
CALL, FILE=main_make_spch.madx; ! required do not change
stop;
*****************************************************************************
```

Adiabatic Switching on SC

The execution of the MACROs is controlled by the file "main_make_spch.madx". There is no need to modify this file by the standard user. It is just shown for a reference of the purpose of each individual MACRO.

```
*****************************************************************************
input file: main_make_spch.madx:
*****************************
! Start file: "main_make_spch.madx"
! Calling MACROS
option,info,warn,echo,bb_ultra_relati;
set,format="24.16e";
debug_print=0;

value,rest_energy_GeV,kinetic_energy_GeV,total_energy_GeV,clight,gamma,
beta,Bunch_length,N_particles,Z_particle,A_particle,r0_particle,prad,L_div_max;

CALL FILE=spcharge_macro_20090316.madx;
```

```
! backwards compatibility mode
!CALL FILE=spcharge_constants_and_formulae.madx; ! <= constants for space charge

! backwards compatibility mode
!CALL FILE=spcharge_input_parameters.madx; ! <=input parameters for space charge

Call FILE=spcharge_beam_command.madx;      ! BEAM command

CALL FILE=spcharge_ring_seq.madx;          ! <= read lattice sequence
                                           ! with universal name (file generated by CVF)

USE, PERIOD=ring; TWISS;

Call FILE=spcharge_beam_command.madx;      ! second BEAM command

! =====  Analytical  tune shift ===============================
CALL FILE=spcharge_laslett_tune_shift.madx; ! <= Analytical evaluation

debug_print=0;

CALL FILE=spcharge_matrix.madx;

debug_print=0;

CALL FILE=spcharge_bbkick.madx;

! MAKETHIN
CALL, FILE = spcharge_makethin.madx;

twiss;
emit;
stop;
! End file: "main_make_spch.madx"
! End MAD-X file "make_machine_sequence.madx"
```

****

Part V

PTC Commands

Chapter 32. PTC Set-up Parameters

The Polymorphic Tracking Code [37] of Etienne Forest 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.

Attention: 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 values are transferred from the MAD-X data structures to the PTC data structures. Any modification to the MAD-X data structure is unknown to PTC until the next call to PTC_CREATE_LAYOUT. For example, a deferred expression of MAD-X is only evaluated at the time of the PTC_CREATE_LAYOUT command and is ignored within PTC afterwards.

Several modules using the PTC code have been presently implemented in MAD-X. These MAD-X-PTC modules[38] are executed by the following commands: PTC_TWISS, PTC_NORMAL, PTC_TRACK, PTC_TRACK_LINE.

To perform calculations with these MAD-X-PTC commands, the PTC environment must be initialized, handled and turned off by special commands within the MAD-X input script.

32.1 Command Synopsis

A typical set of commands to invoke PTC is given below:

```
PTC_CREATE_UNIVERSE, SECTOR_NMUL_MAX= integer, SECTOR_NMUL= integer,
                     NTPSA= logical, SYMPRINT= logical;

PTC_CREATE_LAYOUT, TIME= logical, MODEL= integer,
                   METHOD= integer, NST= integer, EXACT= logical,
                   OFFSET_DELTAP= double, ERRORS_OUT= logical,
                   MAGNET_NAME= string, RESPLIT= logical,
                   THIN= double, XBEND= double,
                   EVEN = logical;
...
PTC_MOVE_TO_LAYOUT, INDEX= integer;
...
PTC_READ_ERRORS, OVERWRITE= logical;
...
PTC_ALIGN;
...
PTC_END;
```

32.2   **PTC_CREATE_UNIVERSE**

The `PTC_CREATE_UNIVERSE` command is required to set-up the `PTC` environment.

> PTC_CREATE_UNIVERSE, SECTOR_NMUL_MAX=integer, SECTOR_NMUL=integer,
>   NTPSA=logical, SYMPRINT=logical;

The attributes are:

SECTOR_NMUL_MAX a global variable in `PTC` needed for exact sector bends defining up to which order Maxwell's equation are solved (see [37] page 76-77). The value of `SECTOR_NMUL_MAX` must not be smaller than `SECTOR_NMUL` otherwise `MAD-X` stops with an error. If a negative value is passed than it is identified automatically by scanning the currently selected sequence.
(Default: -1)

SECTOR_NMUL a 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 $\leq$ `SECTOR_NMUL_MAX` are treated similar to *SixTrack*. If a negative value of `SECTOR_NMUL_MAX` is passed than it is also identified automatically. However, if multipolar parameters of the bends are to be modified inside the PTC universe, for example with `PTC_READ_ERRORS`, then this parameter needs to be set to a corresponding value. Please note that using large values (above 10) slows down the computations, so the smallest required value should be used. We recommended a minimum of MAX(`ORDER`+1,N+2) (where `ORDER` is the order of the map), however `order`+1+N would give the most accurate results.
(Default:-1)

NTPSA   invokes the Differential Algebra (DA) package written in C++ and kindly provided by Lingyun Yang (lyyang@lbl.gov).
Etienne Forest has written the wrapper to allow the use of both the legendary DA package written in Fortran by Martin Berz (default) and this new DA package of Lingyun Yang. It is expected that this DA package will allow for the efficient calculation of a large number of DA parameters.
(Default: false)

SYMPRINT   a flag to enable the printing of the check of symplecticity. It is recommended to leave this flag set to TRUE.
(Default: true)

32.3   **PTC_CREATE_LAYOUT**

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.

```
PTC_CREATE_LAYOUT, TIME=logical, MODEL=integer, METHOD=integer,
                   NST=integer, EXACT=logical, OFFSET_DELTAP=double,
                   ERRORS_OUT=logical, MAGNET_NAME=string,
                   RESPLIT=logical, THIN=double, XBEND=double,
                   EVEN=logical;
```

The attributes are:

TIME   a logical flag to control which coordinate system is being used.
               (Default= true)

               Please see TIME of PTC_SETSWITCH command for more details.

MODEL   an integer to switch between models:
               1 for "Drift-Kick-Drift"; (Default value)
               2 for "Matrix-Kick-Matrix" and
               3 for "Delta-Matrix-Kick-Matrix" (SixTrack-code model).
               Note: Model 1 usually will require more integration and slicing than model 2.

METHOD   the integration order: 2, 4, 6 or 8. This will either use Forest-Yoshida inte-
               gration or Boole's rule (See [37] Chapter K for more details). The optimum
               METHOD value corresponds to the value beyond which the studied properties no
               longer change (in conjunction with increasing slicing). METHOD has a greater
               effect on this convergence, while not increasing the amount of computations
               as much as slicing would to achieve the same convergence
               (Default: 2)

NST   the number of integration steps. (Default: 1)
               The body of each element is divided into NST equal slices and Forest-Yoshida
               integration is carried out on each slice. For best results NST should increase
               with strength and length of elements. The optimum NST value corresponds to
               the value beyond which the studied properties no longer change. However, for
               time consuming calculations the user may reduce NST. (See below the RESPLIT
               option for automatic adjustment.)
               This attribute sets the same NST value for all "thick" elements ($l > 0$) of a
               beam-line; however each individual element may also have its own NST value
               defined independently (see below).

EXACT   a logical flag to turn on calculations with an exact Hamiltonian, otherwise the
               expanded Hamiltonian is used.
               (Default: false)

OFFSET_DELTAP [**Beware: Expert attribute!**] provides relative momentum deviation of
               the reference particle (6D case ONLY). This option implies TOTALPATH=true.
               (Default: 0.0)

ERRORS_OUT   a logical flag to write-out multipolar errors in EFCOMP table format.
               (Default: false)
               Two tables are created and filled: "errors_field" contains only field errors,
               "errors_total" contains also desired field components, which can include the

strength of correctors. The choice of magnets is defined by the `MAGNET_NAME` attribute (see below). The tables can be [written](#) to file, and can be read back via the `ERRORS_IN` flag.
The `ERRORS_IN` flag has precedence over this `ERRORS_OUT` flag.

MAGNET_NAME a string giving a simple selection for the names of magnet to be used for an error write-out using the `ERRORS_OUT` flag (see above). The errors are recorded for all magnets with names starting with the exact string given here, which would be equivalent to the ??? regular expression.
(Default: nil)

RESPLIT a logical flag to apply the `PTC` resplit procedure. This is meant to create an "adaptive" setting of the `METHOD` and `NST` attributes according to the strengths of quadrupoles (using the `THIN` attribute) and dipoles (using the `XBEND` attribute). The `EVEN` attribute further controls the number of splits.
(Default: false)

THIN is the main `RESPLIT` attribute and is meant for splitting quadrupoles according to their strength. The default value `THIN=0.001` has shown in practice to work well without costing too much with respect of performance.

XBEND is an optional `RESPLIT` attribute and is meant for splitting dipoles. A value `XBEND=0.001` is also advisable for dipoles.
(Default: -1.0 for no splitting)

EVEN a logical switch to ensure even number of splits when using the `RESPLIT` procedure of `PTC`, which is particularly useful when one attempts to calculate `PTC_TWISS` with the `CENTER_MAGNETS` option, i.e. to calculate the `TWISS` parameters in the center of the element. Uneven number of splits is ensured with `EVEN=false`. (Default: true)

32.4 PTC_SETSWITCH

The `PTC_SETSWITCH` command allows to set some global `PTC` states and to configure the interface between `MAD-X` and `PTC`, adapting the behavior of the program to the needs.

```
PTC_SETSWITCH, DEBUGLEVEL=integer, MAPDUMP=integer, MADPRINT=logical,
               EXACT_MIS=logical, TOTALPATH=logical, RADIATION=logical,
               ENVELOPE=logical, STOCHASTIC=logical, MODULATION=logical,
               FRINGE=logical, TIME=logical, SEED=integer,
               MAXACCELERATION=logical, NOCAVITY=logical, NOCHARGE=logical;
```

The command parameters and switches are:

DEBUGLEVEL (Default: 1)
            Sets the level of debugging printout: 0 prints none, 4 prints everything

MAPDUMP (Default: 0)
         Sets the level of map dump printout in all tracking codes.
         0: None,

1: Order 0,
2: Order 1,
3: The full map.

MADPRINT   (Default: false)
Sets map dump printout format so that it can be read by MAD-NG

EXACT_MIS   (Default: false)
Switch ensures exact misalignment treatment.

TOTALPATH   (Default: false)
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.
This switch changes behaviour of the RF cavities, including RF multipoles and crab cavities. If it is false, the time of flight effect between the cavities (or the ring length) is ignored because the kick is proportional to
$sin(2\pi \cdot \text{FREQ} \cdot t/c + \text{LAG})$, where t is the time coodinate. So only distance from the synchronous particle plays a role. For example, changing ring length will not affect the cavity. On the other hand, it gurantees that the defined LAG is observed. Conversely, if TOTALPATH is true then t becomes the total time of flight so its effect is accounted for. In the case when cavity is detuned the closed orbit momentum will change and in ray tracking phase slippage from turn to turn will be seen. However, LAG of cavities needs to be carefully calculated because its phasing will depend on its position. Naturally, in cases with only one RF cavity the closed orbit search will automatically determine the offset.

RADIATION   (Default: false)
Sets the radiation switch/internal state of PTC. In PTC basically all the elements radiate including sextupoles, solenoids and orbit correctors.

ENVELOPE   (Default: false)
Sets the envelope switch/internal state of PTC. It allows to calculate dumping due to radiation and stochastic effects. Warning: this makes the tracking approximately twice slower, so low order should be used.

STOCHASTIC   (Default: false)
Sets the stochastic switch/internal state of PTC. It enables stochastic emission of photons in ray tracking, it only affects PTC_TRACK and PTC_TRACKLINE. The emission is calculated during map tracking therefore PTC_TWISS or PTC_NORMAL needs to be invoked before launching the tracking (also with RADIATION, ENVELOPE and STOCHASTIC set to true). Every tracked ray will receive the same stochastic kicks.

MODULATION   (Default: false)
Sets the modulation switch/internal state of PTC. It needs to be set to true to observe effect of AD dipoles.

FRINGE   (Default: false)
Sets the fringe switch/internal state of PTC.

If true the influence of the fringe fields is evaluated for quadrupole fringe fields based on the b_2 and a_2 components of the element.

Please note that currently fringe fields are always taken into account for some elements (e.g. traveling wave cavities) even if this flag is set to false. The detailed list of elements will be provided later, when the situation in this matter will be definitely settled.

TIME   (Default: true)
                If true, Selects time of flight (cT to be precise) rather than path length as the 6th variable of PTC, i.e. 5th of MAD-X.
                This option changes the canonical coordinate system depending whether the calculation is done in 5D or 6D:

    5D   if TIME is true, the fifth coordinate is PT, $p_t = \Delta E/p_0 c$
                                      if TIME is false, the fifth coordinate is DELTAP, $\delta_p = \Delta p/p_0$

    6D   if TIME is true, the MAD-X coordinate system $\{-ct,\ p_t\}$ is used.
                                      if TIME is false, the second PTC coordinate system {-pathlength, δ_p} is used.

                Note: at small energy ($\beta_0 \ll 1$), momentum-dependent variables like dispersion will depend strongly on the choice of the logical input variable TIME. In fact, the derivative ($\frac{\partial}{\partial \delta_p}$) and ($\frac{\partial}{\partial p_t}$) 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.

SEED   (Default: 123456789)
                Sets the seed of PTC random number generator, which is independent from the MADX generators.

MAXACCELERATION (Default: true)
                Switch to set cavities phases so the reference orbit is always on the crest, i.e. gains max energy.

NOCAVITY   (Default: false)
                Switch to turn off the RF cavities, with NOCAVITY=true, RF cavities are ignored.

NOCHARGE   (Default: true)
                Switch to ignore the charge of your beam. If set to false, the charge of the beam is taken into account.

32.5   PTC_MOVE_TO_LAYOUT

Several PTC layouts can be created within a single PTC-"universe". The layouts are automatically numbered with sequential integers by the MAD-X code. The PTC_MOVE_TO_LAYOUT command is used to activate a specific layout, and the next PTC commands will be applied to this active PTC layout until a new PTC layout is created or activated.

```
PTC_MOVE_TO_LAYOUT, INDEX=integer;
```

The only attribute is:

INDEX   is the numeric index of the PTC layout to be activated.
           (Default: 1)

32.6   PTC_READ_ERRORS

The PTC_READ_ERRORS command reads any number of **"errors_read"** table through the
READTABLE mechanism.

```
PTC_READ_ERRORS, OVERWRITE=logical;
```

The only attribute is

OVERWRITE   a flag to specify that the read-in errors overwrite previous errors instead of
            adding the read-in errors to existing errors, ie multipole components already
            present.
            (Default: false)

Note:
For calculations with exact flag set to true, SECTOR_NMUL parameter of PTC_CREATE_UNIVERSE
needs to be set to a value bigger than the highest order of an error in bending magnets.

Note:
Because of the way the table is read in memory, a warning will always be issued by default
in the form:

```
warning: string_from_table_row: row out of range: errors_read->name[1>=n+1<=n]
```

where **n** is the number of records read from the table. This warning has no consequence on
the errors read and the following calculation.
The warning is purely the result of the way that the reading loop is programmed with a break
based on the return value of the routine *string_from_table_row*. But if *string_from_table_row*
tries to read in a row (n+1) past the last row (n) of the table, it prints a warning before
returning a value that will effectively break the loop. Of course this will only happen if the
WARN option is true and this can be turned off with

```
OPTION, -WARN;
```

32.7   PTC_ALIGN

The PTC_ALIGN command is used to apply the MAD-X alignment errors to the current PTC
layout, and takes no attributes.

```
PTC_ALIGN;
```

32.8   PTC_END

The PTC_END command turns off the PTC environment, which releases all memory and returns
control to the MAD-X world proper.

```
PTC_END;
```

32.9   Additional Options for Physical Elements

For some of the `MAD-X` elements, additional attributes can be defined that are available to
`PTC` only. `PTC` also uses standard `MAD-X` attributes in a slightly different way.

```
SBEND | RBEND | QUADRUPOLE | SEXTUPOLE | OCTUPOLE | SOLENOID ,
        L=real, ...  , TILT=real, ...  , NST=integer, ...  ,
        KNL={real, real, real,...}, KSL={real, real, real,...};
```

These attributes are:

L   the length of the element.
              `PTC` treats bending magnets (`SBEND` or `RBEND`) as `MARKER` if their length is equal
              to zero.

NST   gives a specific `NST` values for a particular "thick" element ($L > 0$).

              For example RF cavities are represented in `MAD-X` by a single kick, while `PTC`
              splits the RF cavity into `NST` segments thereby taking into account properly the
              transit-time effects of the cavity. Specifying explicitly `NST=1` for RF cavity re-
              produces in `PTC` the approximate results of `MAD-X`, ignoring transit time effects.

KNL, KSL   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. KNL and KSL multipole coefficients are
              specified as the integrated value ($\int K ds$) of the field components along the
              magnet axis. The multipole components in `PTC` are spread over the length of
              thick elements. This is a considerable advantage of `PTC` input compared to
              `MAD-X` which allows only thin multipoles.

              KNL   is an array representing the normal multipole coefficients.
                            (Default: 0 m^{-1})

              KSL   is an array representing the skew multipole coefficients.
                            (Default: 0 m^{-1})

              A full range of additional multipole field errors can be additionally specified
              with the `EFCOMP` command. Errors are added to the above multipole fields on
              the bench.
              If you specify the corresponding multipole coefficient in a magnet that can
              accept a non integrated value, these are added together, i.e. $k_1 = k_1 + \mathtt{knl}[2]/\mathtt{l}$,
              where `knl[2]` is the value of the second element of the array knl and .

32.9.1   Additional Fringe Field Parameters

The following parameters are available for the following elements (Additional details can be
found in 11.3):

```
SBEND | RBEND | QUADRUPOLE | SEXTUPOLE | OCTUPOLE | SOLENOID ,
       FINT=real, FINTX=real, HGAP=real, FRNGMAX=integer,
       F1=real, F2=real, FRINGE=integer, ...;
```

FINT   (Default: 0)
                  The fringe field integral at entrance and exit of the bend.
                  This has been activated for all elements listed above, not just for `(S|R)BEND`, as in `MAD-X`.

FINTX   (Default: =FINT)
                  If defined and positive, the fringe field integral at the exit of the element, overriding `FINT` for the exit.
                  This allows to set different fringe field integrals at entrance (`FINT`) and exit (`FINTX`) of the element.
                  This has been activated for all elements listed above, not just for `(S|R)BEND`, as in `MAD-X`.

HGAP   (default: 0 m)
                  The half gap of the magnet.
                  This has been activated for all elements listed above, not just for `(S|R)BEND`, as in `MAD-X`.

FRNGMAX   (default: 2)
                  The maximum multipole component to be included in the multipolar fringe field.

F1   (default: 0)
                  The entry linear fringe field coefficient for quadrupoles, from SAD [39].

F2   (default: 0)
                  The exit linear fringe field coefficient for quadrupoles, from SAD [39].

FRINGE   (default: 0)
                  This is a bit mask that specifies the fringe field model to be used. The bits are defined as follows:

                  1: The dipole fringe field is performed using the `FINT`, `FINTX` and `HGAP` parameters. Note: It is not possible to remove this fringe from a bend element, as `PTC` will always use the dipole fringe field for bending magnets.

                  2: The multipole fringe field is performed, integrating up to the minimum of `FRNGMAX` and the highest multipole component specified.

                  4: Use the linear fringe field map for the quadrupole from the SAD code, using parameters `F1`, `F2` [39].

An example use of the `FRINGE`, would be to specify a dipole fringe and the multipolar fringe field, has `FRINGE=3`, but to specify the multipolar fringe field and the SAD fringe field map, use `FRINGE=6`.

32.9.2   Additional Pole Face Parameters

The following parameters are available for the following elements, not just for `(S|R)BEND`, as in `MAD-X` (Additional details can be found in 11.3):

```
SBEND | RBEND | QUADRUPOLE | SEXTUPOLE | OCTUPOLE ,
        ..., E1 = real, E2 = real, ..., H1 = real, H2 = real;
```

E1   (default: 0 rad)
                The entry pole face rotation angle.

E2   (default: 0 rad)
                The exit pole face rotation angle.

H1   (default: 0 m^{-1})
                The entry pole face curvature.

H2   (default: 0 m^{-1})
                The exit pole face curvature.

32.9.3   Additional Parameters for the `SBEND` and `RBEND` elements

The following parameters are available for the `SBEND` and `RBEND` elements:

```
SBEND | RBEND,
        ..., K0=real, K0S=real, K1=real, K1S=real,
       K2=real, K2S=real, K3=real, K3S=real, ...;
```

The philosophy of the implementation of these components is the same as described in 32.9, i.e. the multipole components (divided by the element length) are added to the strengths listed above.

32.9.4   Parameters for the `CAVITY` elements

The following parameter is available for the following elements:

```
RFCAVITY | RFMULTIPOLE | CRABCAVITY,
             ..., no_cavity_totalpath=logical, ...  ;
```

no␣cavity␣totalpath (default=false)

    If `true`, the total path length of the cavity is not taken into account. See `TOTALPATH` in 32.4 for more details. This parameter has a higher priority than `TOTALPATH` in 32.4 and therefore can override this value per element.

Note that the `PNL` and `PSL` parameters, used in `MAD-X`, are not considered in `PTC`.

Furthermore, the implementation of the `CRABCAVITY` element means that the following are equivalent:

```
CRABCAVITY , L=LENGTH, VOLT=V, LAG=lag,
         FREQ=freq, ...,;
RFMULTIPOLE, L=LENGTH, VOLT=0, LAG=(lag-0.25),
         FREQ=freq, ..., KNL={V/1000 * LENGTH};
```

Essentially, the voltage is set to zero and converted to a dipole compenet, and the lag is reduced.

32.9.5   Additional Parameter for `DRIFT` Element

```
DRIFT, L=real, ...  , TILT=real;
```

TILT   The DRIFT can also be tilted.

32.9.6   Additional Parameter for `MULTIPOLE` Element

```
MULTIPOLE, LRAD=real, KSI=real, ...  ;
```

LRAD   A fictitious length (default: 0 m), originally only used to compute synchrotron radiation effect.

KSI   The `KSI` (default: 0 rad)parameter is used to specify the integrated solenoid component. If `KSI` is specified, but `LRAD` is not, or 0, then this attribute is ignored and a warning is emmited.

32.10   The `RBEND` element in PTC

For the `RBEND` element, the `PTC` environment provides three types of this element:

```
RBEND, L=real, ...  , TRUE␣RBEND=logical, E1=real, E2=real, ...;
```

The attributes `TRUE␣RBEND`, `E1` and `E2` are used to decide which `RBEND` is used in `PTC`. The default is `TRUE␣RBEND=FALSE`.

CURVED RBEND (`TRUE␣RBEND=FALSE`)

    This is the normal rbend that is used in `MAD-X`, which is equivalent to an `SBEND` with an E1 and E2 set to `ANGLE`/2.

    Note: This means that additional multipole components are integrated in curved space, therefore also requires more computation and `SECTOR␣NMUL␣MAX` must be considered.

STRAIGHT RBEND (TRUE_RBEND=TRUE)

> This is equivalent to the LIKEMAD=TRUE case in [37] Subsection K.4.12 Figure 19.
>
> Note: This means that additional multipole components are integrated in straight space.

TRUE PARALLEL RBEND (TRUE_RBEND=TRUE & (E1 > 2*PI | E2 > 2*PI))

> This is equivalent to the LIKEMAD=FALSE case in [37] Subsection K.4.12 Figure 19.
>
> This RBEND element performs a translation, to ensure the orbit is restored, at the entry (if E1 > 2*PI) or exit (if E2 > 2*PI) of the element. If both E1 and E2 are greater than 2*PI, then an error is thrown.
>
> The sum of E1 and E2 will be the total bend angle and therefore if E(1|2) is greater than 2*PI, then, E(2|1) dictates the rotation on entry and exit of the element. This magnet is equivalent to the RBEND_STRAIGHT if E(1|2) > 2*PI and E(2|1) = ANGLE/2.

It is important to note that each of these different rbends will have different lengths, since the L parameter is the path length (LD). Therefore, the weighting of your multipole components will be different for each of these elements, which uses the true element length. We calculate the true element length as follows, where the input length of the user is L_D, the input ANGLE is θ and the α is the value of E(1|2) when E(2|1) > 2*PI.

CURVED RBEND $L = L_D$

STRAIGHT RBEND $L = \frac{L_D \sin \theta/2}{\theta/2}$

TRUE PARALLEL RBEND $L = \frac{L_D \sin \theta/2}{\theta/2} \cos\left(\theta/2 - \alpha\right)$

Chapter 33. Thick-Lens Tracking Module

The `PTC-TRACK` module [40, 38] is the symplectic thick-lens tracking facility in `MAD-X`. It is based on `PTC` library [37] written by E. Forest. 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.

Examples
Several examples can be found on the web at http:cern.chfrsmad-X_examplesptc_track.

33.1 Synopsis

A typical tracking job in `PTC` requires a number of commands to be issued:

```
PTC_CREATE_UNIVERSE;
PTC_CREATE_LAYOUT, MODEL=integer, METHOD=integer, NST=integer, [EXACT];
    ...
    PTC_START, X=real, PX=real, Y=real, PY=real, T=real, PT=real;
    PTC_START, FX=real,PHIX=real, FY=real,PHIY=real, FT=real,PHIT=real;
    ...
    PTC_OBSERVE, PLACE=string;
    ...
    PTC_TRACK, ...;
    ...
    PTC_TRACKLINE, ...;
    ...
    PTC_TRACK_END;
    ...
PTC_END;
```

33.2 PTC_START

To start particle tracking, a series of initial trajectory coordinates must be given with the `PTC_START` command; and as many commands as initial trajectories can be given.

`PTC_START` commands must appear before the `PTC_TRACK` command.

```
PTC_START, X=real, PX=real, Y=real, PY=real, T=real, PT=real,
           FX=real, PHIX=real, FY=real, PHIY=real, FT=real, PHIT=real;
```

The coordinates can be

`X, PX, Y, PY, T, PT` i.e. the standard canonical coordinates.
(Default: 0.0)

`FX, PHIX, FY, PHIY, FT, PHIT` i.e. the action-angle coordinates 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 a preceding BEAM command. F_z are expressed in number of r.m.s. beam sizes and Φ_z are expressed in radians.
(Default: 0.0)

Remarks
In the uncoupled case, the canonical and the action-angle variables are related with equations

$$z = F_z(E_z)^{1/2}cos(\Phi_z) \qquad p_z = F_z(E_z)^{1/2}sin(\Phi_z) \qquad (33.1)$$

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 canonical coordinates.

The use of the action-angle coordinates requires the option CLOSED_ORBIT in the PTC_TRACK command.

If the option CLOSED_ORBIT in the PTC_TRACK command is active (see above) 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.

33.3   PTC_OBSERVE

Besides 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.

```
PTC_OBSERVE, PLACE=string;
```

The only attribute is

PLACE   the name of observation point.
                 (Default: NULL)

Remarks
The first observation point at the beginning of the beam-line is marked as **"start"**.

It is *strongly* recommended to specify markers as observation points.

The data at observation points other than **"start"** can be produced in two different ways:

1. traditional element-by-element tracking. (See MAD-X thin tracking) which requires the option ELEMENT_BY_ELEMENT of PTC_TRACK to be active.

2. coordinate transformation from **"start"** to the respective observation points using high-order PTC transfer maps, which requires the option CLOSED_ORBIT of PTC_TRACK to be active, and the options RADIATION and ELEMENT_BY_ELEMENT of PTC_TRACK to be inactive.

33.4 PTC_TRACK

The `PTC_TRACK` command initiates trajectory tracking by entering the thick-lens tracking module.

The tracking can be done "element-by-element" or "turn-by-turn". In the first case particle coordinates are tracked from element to element, and if an element is an observation point than the coordinates are saved in a table. In the second case particle coordinates are always tracked over full turn (from start to start) and afterwards these coordinates are transformed using Transfer Map from the start to a given observation point. The order of the transfer map is defined by `NORMAL_NO` switch.

Tracking is done in parallel, i.e. the coordinates of all particles are transformed through each beam element, or over full turns.

A particle is lost if its trajectory is outside specified boundaries. In `PTC`, there is a continuous check that the particle trajectories stay within the aperture limits.

The Normal Form calculation is controlled through options of the `PTC_TRACK` command.

Remarks
If `TOTALPATH` is set to true than the 5th coordinate is total path length or time of flight and the PTC conversion is kept for the output tables and files. In particular, initial coordinates provided in `PTC_START` will appear with swapped sign.
Please also remember that as the total time or path length becomes larger as it becomes less precise and calculation gets more vulnerable to numerical instabilities. `PTC_TRACK` internally attempts to periodically subtract from it a value corresponding to common wavelenth of all oscillating elements, however, in cases when ratios of all the frequency pairs are not rational numbers than it can not be done.

```
PTC_TRACK, ICASE=integer, DELTAP=real, CLOSED_ORBIT=logical,
           ELEMENT_BY_ELEMENT=logical, TURNS=integer,
           DUMP=logical, ONETABLE=logical, RECLOSS=logical,
           MAXAPER=real_array,
           NORM_NO=integer, NORM_OUT=logical,
           FILE[=string], EXTENSION=string, FFILE=integer,
           X=real, PX=real, Y=real, PY=real, T=real, PT=real,
           RADIATION=logical, RADIATION_MODEL1=logical,
           RADIATION_ENERGY_LOSS=logical,
           RADIATION_QUAD=logical,
           BEAM_ENVELOPE=logical, SPACE_CHARGE=logical;
```

The attributes are:

ICASE   the user-defined dimensionality of the phase-space (4, 5 or 6).
                    `ICASE` has higher priority over other options. In particular:

> 1. RF cavities with non-zero voltage are ignored for `ICASE=4` or `ICASE=5`.
>
> 2. A non-zero `DELTAP` is ignored for `ICASE=4` or `ICASE=6`.

                    However, if an RF cavity has voltage set to zero and `ICASE=6` is specified, `PTC`

sets ICASE=4.
(Default: 4)

DELTAP   the relative momentum offset for reference closed orbit (used for 5D case
              ONLY).
              DELTAP is ignored for ICASE=6, but the option OFFSET_DELTAP of command
              PTC_CREATE_LAYOUT may be used, if the reference particle should have a mo-
              mentum offset.
              (Default: 0.0)

CLOSED_ORBIT a logical switch to activate the closed orbit calculation. This option must be
              used for closed rings only. This option allows to activate the Normal Form
              analysis, if required. With CLOSED_ORBIT=false, the sequence is treated as a
              transfer line.
              (Default: false)

ELEMENT_BY_ELEMENT a logical switch to activate the element-by-element tracking, from the
              default turn-by-turn tracking.
              (Default: false)

TURNS   number of turns to be tracked.
              (Default: 1)

DUMP   a logical flag to enforce writing particle coordinates to formatted text files.
              (Default: false)

ONETABLE   a logical switch to write all particle coordinates to a single file instead of sep-
              arate files.
              (Default: false)
              Tracks are reported at the defined observation points (see PTC_OBSERVE com-
              mand) plus at the end of each turn and at the start. In order to make the
              file more readable each reported location (file segment) is marked by comment
              line like
              #segment s_no nobsp npart ielem el_name,
              where s_no is sequential number of the reported segment, nobsp is total number
              of the user defined observation points, npart is number of particles reported
              at the observation point. ielem is index of the element and el_name is name
              of the element.

RECLOSS   flag to create in memory a table named "trackloss" containing the coordinates
              of lost particles.
              (Default: false)
              See TRACK section for the details.

MAXAPER   an array defining upper limits for particle coordinates, essentially defining the
              aperture to trigger particle loss.
              (Default: {0.1, 0.01, 0.1, 0.01, 1.0, 0.1})
              Please watch that the check is performed directly on the PTC variables, there-
              fore the last 2 variables change their meaning depending if TIME switch of
              PTC is set to true or false, see PTC_SETSWITCH command for more details.

If `TOTALPATH` is set to true then check on the 5th coordinate is not done. If `ELEMENT_BY_ELEMENT` is false then the check is performed only at the end of each turn (at the end of the layout).

NORM_NO order of the internally used Transfer Maps and Normal Forms.
`NORM_NO=1` makes them linear (always true for `MAD-X`).
The Transfer Maps are used to transform track coordinates from `START` to given observation point if `ELEMENT_BY_ELEMENT` is false. The Normal Form is used to transform coordinates between Cartesian and action-angle variables.
(Default: 1)

NORM_OUT a logical switch to transform canonical variables to action-angle variables.
(Default: false)

FILE if FILE is omitted, no output is written to file.
if FILE is present, track tables are printed, optionally to files with name constructed from the base filename specified.
The actual name of the output file is constructed from the baseline given with `FILE` to which are appended the strings ".obsnnnn" (where nnnn is the observation point index) and ".pnnnn" (where nnnn is now the particle number), unless the `ONETABLE` option is activated.
(Default: "track")

EXTENSION a string providing the filename extension for the track table files, e.g., txt, doc...
(Default: nil)

FFILE defines the periodicity **n** of the printout: coordinates are printed every n turns.
(Default: 1)

X, PX, Y, PY, T, PT the initial canonical coordinates for the closed orbit search.
(Default: 0.0)

RADIATION a logical flag to turn on the synchrotron radiation calculated by an internal procedure of `PTC`.
The option `RADIATION` has precedence over `RADIATION_MODEL1` when both are activated.
(Default: false)

RADIATION_MODEL1 a logical flag to turn on the synchrotron radiation according to the method given in [30]. This model simulates quantum excitation via a random number generator and tables for photon emission. It can be used only with the option `ELEMENT_BY_ELEMENT`.
(Default: false)

RADIATION_ENERGY_LOSS a logical flag to add back the average energy loss thereby taking only the quantum excitation into effect. It applies only when for `RADIATION_MODEL1` is active.
(Default: false)

RADIATION QUAD a logical flag to add the effect of synchrotron radiation in quadrupoles. It
supplements either model RADIATION or RADIATION MODEL1.
(Default: false)

BEAM ENVELOPE a logical switch to activate the calculation of the beam envelopes with PTC.
It requires the options RADIATION and ICASE=6.
(Default: false)

SPACE CHARGE [**under construction**]
a logical flag to activate the simulation of space charge forces between particles.
(Default: false)

33.5   PTC_TRACKLINE

The PTC_TRACKLINE command performs particle tracking that takes into account acceleration
in travelling 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_END.

All tracks created with PTC_START commands before 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 an observation point at the end of a sequence.

```
PTC_TRACK_LINE, TURNS=integer,
                ONETABLE=logical, FILE=string, EXTENSION=string,
                ROOTNTUPLE=logical,
                EVERYSTEP=logical, TABLEALLSTEPS=logical,
                GCS=logical;
```

The attributes are:

TURNS   number of turns to be tracked. If the layout of the machine is not closed, this
             value is forced to TURNS=1 by PTC
             (Default: 1)

ONETABLE   a logical switch to write all particle coordinates to a single file instead of
             separate files.
             (Default: false) [to be clarified]

FILE   if FILE is omitted, no output is written to file.
             if FILE is present, track tables are printed, optionally to files with name con-
             structed from the base filename specified.
             The actual name of the output file is constructed from tyhe baseline given with
             FILE to which are appended the strings ".obsnnnn" (where nnnn is the obser-
             vation point index) and ".pnnnn" (where nnnn is now the particle number),
             unless the ONETABLE option is activated.
             (Default: "track")

EXTENSION   a string providing the filename extension for the track table files, e.g., txt, doc...
(Default: nil)

ROOTNTUPLE   a logical switch to store data to ROOT file as ntuple. Accessible only if RPLOT plugin is available. i.e. only if MAD-X is dynamically linked and RPLOT plugin is present.
(Default: false)

EVERYSTEP   a logical switch to activate the recording of track parameters at every integration step. Normally tracking data are stored internally only at the end of each element. EVERYSTEP provides the user with finer data points. It implies usage of the so called node (thin) layout.
Track parameters are stored for each step in file "thintracking_ptc.txt". Storage of parameters in a table for each step might be very memory consuming. To switch it off use option TABLEALLSTEPS.
Collective effects can be taken into account only using this mode (this feature of PTC is not interfaced into MAD-X).
(Default: false)

TABLEALLSTEPS   (to be completed)
(Default: false)

GCS   a logical switch to store track parameters in Global Coordinate System - normally it starts at the entrance face of the first element.
(Default: false)

Plotting of track parameters (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, 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 the defined aperture for elements are immediately stopped.

Behavior of PTC calculations can be adapted with PTC_SETSWITCH command and with appropriate switches of PTC_CREATE_LAYOUT command.

33.6   PTC_TRACK_END

The PTC_TRACK_END command terminates the command lines related to the PTC_TRACK module.

```
PTC_TRACK_END;
```

The initial and final canonical coordinates are collected in the internal table "tracksumm", which can be written to file.

33.7   Choice of options

The following table facilitates the choice of the correct options for a number of typical tasks:

1. The tracking of a beam-line with default parameters.

2. Smilar to "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.

Option	case 1	case 2	case 3	case 4	case 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	-	-	-	-	-

Chapter 34. Ripken Optics Parameters

34.1 Introduction

The **PTC_TWISS module**[38] of MAD-X is based on the PTC code and is supplementary to the TWISS module of MAD-X. In PTC_TWISS the Twiss parameters are calculated according to the formalism of G. Ripken, developped in [41] and most accessible in [42].

PTC_TWISS tracks a special representation of the beam in three degrees of freedom. It works on the coupled lattice functions which are essentially the projections of the lattice functions for the eigen-modes on the three planes.

PTC_TWISS lists the projections of the ellipses of motion onto the three planes (x, p_x), (y, p_y), (t, p_t) expressed via 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 command also calculates the dispersion values D_1, ... ,D_4.

In MAD-X commands and tables, these parameters are denoted as `beta11, ..., beta33`, `alfa11, ..., alfa33`, `gama11, ..., gama33`, `mu1, ..., mu3`, `disp1, ..., disp4`, respectively.

The Ripken parametrization can be transformed into the Edwards-Teng parametrization (used in the module TWISS of MAD-X) using the formulae of Lebedev [43].

The parameters are noted as `betx, bety, alfx, alfy` and the coupling matrix: `R11, R12, R21` and `R22`. In absence of coupling, the following holds: `betx = beta11, bety = beta22, alfx = alfa11` and `alfy = alfa22`.

PTC_TWISS can also compute the $\Delta p/p_0$ (or $\Delta E/p_0 c$ if time=TRUE) dependency of the Twiss parameters. The column names `beta11p, ..., beta33p, alfa11p, ..., alfa33p, gama11p, ..., gama33p` denote the derivatives of the optics parameters with respect to $\Delta p/p_0$ (or $\Delta E/p_0 c$ if time=TRUE).

In order to evaluate the $\Delta p/p_0$-dependency of the Twiss parameters, the order (NO) of the map must set to at least 2.

The derivatives of the dispersion with respect to $\Delta p/p_0$ (or $\Delta E/p_0 c$ if time=TRUE) have column names: `disp1p, ..., disp4p`. Second and third order derivatives have respective column names: `disp1p2, ..., disp4p2` for the second order, and `disp1p3, ..., disp4p3` for the third order.

In addition, PTC computes the momentum compaction factor α_c (`ALPHA_C`) and time slip factor η_c. Higher order derivatives of compaction factor with respect to $\Delta p/p_0$ up to urder 4 (`ALPHA_C_P, ALPHA_C_P2` and `ALPHA_C_P3`) are computed f ICASE=56 and time=false, providing the order of calculation as defined by NO switch is high enough. The values appear in the header of the PTC_TWISS output file, and a value of -1000000 means the value has not been computed.

For clarification: in the 4-D case, there is the following correspondence between MAD-X and the Ripken's notations: `beta11` $\equiv \beta_{xI}$, `beta12` $\equiv \beta_{xII}$, `beta21` $\equiv \beta_{yI}$, `beta22` $\equiv \beta_{yII}$. In the uncoupled 4-D case, `beta11` is the same as the classical β_x (`betx`) and `beta22` is β_y (`bety`), while `beta12` and `beta21` are zero. in the coupled case all `betaNN` are non-zero and `beta11`, `beta22` are distinctively different from β_x, β_y, respectively.

`PTC_TWISS` also tracks the eigenvectors and prints them to Twiss table according to the SELECT command with `FLAG=ptc_twiss`. Either all 36 components or particular components of the eigenvectors can be selected with `EIGN` or `EIGNij`, 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 (Q1, Q2 and if `ICASE=6` with cavity, Qs) and chromaticities (for `NO`≥ 2).

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, disp1p,...,disp4p,
        disp1p2,...,disp4p2, disp1p3,...,disp4p3,
        [eign], eign11,...,eign16,...,eign61,...,eign66;
...
PTC_TWISS;
...
PTC_END;
```

34.2   PTC_TWISS

The `PTC_TWISS` command causes computation of the Twiss parameters in Ripken's style. It operates on the working beam line defined in the latest USE command.

Applications for the `PTC_TWISS` command are similar to the TWISS command of `MAD-X`. The `PTC_TWISS` can be applied to two basic tasks. It can calculate either a periodic solution or a solution with initial conditions.

```
 PTC_TWISS, ICASE=integer, DELTAP=real,
           CLOSED_ORBIT=logical, DELTAP_DEPENDENCY=logical,
           SLICE_MAGNETS=logical,
           RANGE=string, FILE[=filename], TABLE[=tabname],
           INITIAL_MATRIX_TABLE=logical, INITIAL_MATRIX_MANUAL=logical,
           INITIAL_MAP_MANUAL=logical, BETA0=string, MAPTABLE=logical,
           IGNORE_MAP_ORBIT=logical, RING_PARAMETERS=logical,
           NORMAL=logical, TRACKRDTS=logical,
           BETX=real, ALFX=real, MUX=real,
           BETY=real, ALFY=real, MUY=real,
           BETZ=real, ALFZ=real,
           DX=real, DPX=real, DY=real, DPY=real,
           X=real, PX=real, Y=real, PY=real, T=real, PT=real,
           RE11=real, RE12=real, ..., RE16=real,
           ...
           RE61=real, RE62=real, ..., RE66=real;
```

The attributes are:

ICASE the dimensionality of the phase-space (4, 5 or 6). (Default: 4)
Note that `ICASE` is internally set to 56 when attempting to set `ICASE=6` with no cavity, and `ICASE` is internally set to 4 when attempting to set `ICASE=6` with an RF cavity with zero voltage.

NO the order of the map. (Default: 1)
For evaluating the derivatives of the Twiss parameter w.r.t. $\Delta p/p_0$, e.g. for evaluating the chromaticities, the order must be at least equal to 2.

DELTAP relative momentum offset for reference closed orbit. (Default: 0.0)

CLOSED_ORBIT a logical switch to trigger the closed orbit calculation (applies to periodic solution ONLY).
(Default: false)

DELTAP_DEPENDENCY a logical switch to trigger the computation of the Twiss and dispersion derivatives w.r.t. $\Delta p/p_0$. Derivation formula assume that `ICASE` $\geq$ 5, so that $\Delta p/p_0$ is supplied as a parameter.
(Default: false)

SLICE_MAGNETS a logical switch to activate the evaluation of Twiss parameters at each integration step inside magnets, in addition to the end face. The number of slices is determined by the number of steps (`NST`) that can be separately defined for each element, or otherwise set by `NST` parameter when creating the PTC layout. Note that the orbit rms calculated in this mode counts as valid data points both the end of the previous element and the entrance of the current element. Since the first integration node is always at the entrance of the magnet (after position offset and fringe effects are calculated) which corresponds to the same s position (and usually optical functions) as the end of the previous element, the points at the interface between magnets are included twice in the

rms calculation.
(Default: false)

CENTER_MAGNETS a logical switch to activate the evaluation of Twiss parameters at the middle of each magnet. This relies on internal slicing and 'integration nodes' as determined by the number of steps (`NST`) selected when creating the PTC layout. This number is assumed to be even otherwise the program issues a warning.
(Default: false)

FILE   if the `FILE` attribute is omitted, no output is written to file.
                If the `FILE` attribute name is present, the optional attribute value argument is the name of the file for printing the `PTC_TWISS` output. The default file name is "ptc_twiss".
                (Default: false)

TABLE   if the `TABLE` attribute is omitted, no output is written to an internal table.
                If the `TABLE` attribute name is present, the optional attribute value argument is the name of the internal table for `PTC_TWISS` variables. The default table name is "ptc_twiss".
                (Default: false)

SUMMARY_FILE   if the `SUMMARY_FILE` attribute is omitted, no summary output is written to file.
                If the `SUMMARY_FILE` attribute name is present, the optional attribute value argument is the name of the file for printing the `PTC_TWISS_SUMMARY` table output. The default file name is "ptc_twiss_summary".
                (Default: false)

SUMMARY_TABLE   if the `SUMMARY_TABLE` attribute is omitted, no summary output is written to an internal table.
                If the `SUMMARY_TABLE` attribute name is present, the optional attribute value argument is the name of the internal summary table for `PTC_TWISS_SUMMARY` variables. The default table name is "ptc_twiss_summary".
                (Default: false)

RANGE   a string in RANGE format that specifies a segment of beam-line for the `PTC_TWISS` calculation.
                (Default: $\#S/\#E$)

INITIAL_MATRIX_TABLE a logical flag to trigger the reading of the transfer map from table named "map_table" created by a preceding `PTC_TWISS` or `PTC_NORMAL` command. The table can be also read before hand from files using a READTABLE command.
                (Default: false)

INITIAL_MATRIX_MANUAL a logical flag to trigger the use of the input variables `RE11, ...,RE66` as the transfer matrix.
                (Default: false)

RE11,..., RE66 values of the 6×6 transfer matrix.
                (Default: 6×6 unit matrix)

INITIAL_MAP_MANUAL a logical flag to trigger the use of an input map stored beforehand in file "fort.18", e.g. by a previous initial run of PTC_NORMAL).
(Default: false)

IGNORE_MAP_ORBIT a logical flag to ignore the orbit in the map and use the closed orbit instead if requested, or the orbit defined by the starting point specified with X, PX, Y, P, T, DT parameters otherwise.
(Default: false)

BETA0 the name of a BETA0 block containing the Twiss parameters to be used as input. When ICASE=6 or ICASE=56, this information must be complemented by supplying a value for BETZ on the PTC_TWISS command line.
(Default: beta0)

MAPTABLE a logical flag to save the one-turn-map to table "map_table". The one-turn-map can then be used as starting condition for a subsequent PTC_TWISS, see INITIAL_MATRIX_TABLE parameter above.
(Default: false)

BETX, ALFX, MUX, BETY, ALFY, MUY, BETZ, ALFZ, DX, DPX, DY, DPY :
Edwards and Teng [16] Twiss and dispersion parameters: $\beta_{x,y,z}$, $\alpha_{x,y,z}$, $\mu_{x,y}$, $D_{x,y}$, $D_{px,py}$.
(Default: 0)

RING_PARAMETERS a logical flag to force computation of ring parameters (γ_{tr}, α_c, etc.).
(Default: false)

X, PX, Y, PY, T, PT the canonical coordinates of the initial orbit.
(Default: 0.0)

NORMAL a logical flag that triggers saving of the normal form analysis results (the closed solution) into dedicated table called NONLIN. It is the same as ptc_normal. However, the nonlin table has format such that its coefficients can be accessed within the MADX script using table command. This permits, for example, to perform matching of these parameters or or value extraction to a variable. Also, all available orders are calculated and no select_ptc_normal is required. Currently the following parameters are calculated: tunes (and all their derivatives), dipsersions (and all their derivatives), eigen values, transfer map, resonance driving terms (generating function) and the pseudo Hamiltonian. If radiation and envelope is switched on with PTC_SETSWITCH then damping decrements, equilibrium emittances and beam sizes are calculated. Attention: the results are always the plain polynomial coefficients, which is different from ptc_normal nomenclature for some variables, which in turn always gives values of the partial derivatives. Therefore, in order to obtain values of the partial derivatives the respective factorials and binomial coefficients need to be factored out by the user. Also, if TIME=true the chromatic variables are NOT transformed into usual $\Delta p/p_0$ dependence and in this case they are related to $\Delta E/p_0 c$. For example, dispersions and chormaticities will be different by relativistic beta factor. Similarly, the T coordinate is time times the speed of

light instead of path length times the speed of light.

TRACKRDTS   a logical flag that triggers tracking of Resonance Driving Terms (RDTs). For each element all RDTs from order 3 up to value defined by switch `NO` are saved in `TWISSRDT` table. This table contains also auxiliary columns as s position, element names, strengths and lengths such that RDT values can be plotted the same way as values from `TWISS` table. RDTs are elements of the generating function (GFN) of the normalization transformation and for compatibility with `PTC_NORMAL` notation the columns are prefixed with GNF. Because they are complex numbers 3 values are saved: real part (GNFC), imaginary part (GNFS) and amplitude (GNFA). For example, RDT f_{300000} will be saved in columns `GNFC_3_0_0_0_0_0`, `GNFS_3_0_0_0_0_0` and `GNFA_3_0_0_0_0_0`. Currently the indexes can be only single digit numbers. If amplitude of an RDT is smaller than 1e-12, it is saved as zero. RDTs that have zero amplitude for all elements are not saved in the table.

34.3   Periodic Solution

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 attributes described in `PTC_TWISS` above.

```
PTC_TWISS, ICASE=integer, DELTAP=real, CLOSED_ORBIT=logical,
          RANGE=string, FILE[=string], TABLE[=string];
```

34.4   Evaluation of Twiss parameters inside magnets

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 attributes described in `PTC_TWISS` above may be selected.

```
PTC_TWISS, ICASE=integer, DELTAP=real, CLOSED_ORBIT=logical,
          RANGE=string, FILE[=string], TABLE[=string],
          SLICE_MAGNETS=logical;
```

Example:
An example is found in the PTC_TWISS Examples repository.

34.5   Solution with Initial Conditions

Initial conditions can be supplied in different ways. Naturally only one of the methods below can be used at a time, and they can not be mixed. In this mode it is assumed that the lattice is a line and no ring parameters are evaluated (their values are set to -1000000), unless `RING_PARAMETERS=true`, which forces computation of closed solution for the resulting map. If a closed solution does not exist, `PTC` reports an error and exits.

The following logic is programmed in `PTC` to identify the source of initial conditions:

```
IF      ( INITIAL_MATRIX_TABLE=true && (a map-table exists)) THEN
          use initial values from a Map-Table

ELSEIF  ( INITIAL_MAP_MANUAL=true ) THEN
          use initial values from a Given Map File

ELSEIF  ( INITIAL_MATRIX_MANUAL=true ) THEN
          use initial values from a Given Matrix

ELSEIF  ( BETA0 block is given ) THEN
          use initial values from a BETA0 block

ELSE
          use initial values from Given Twiss parameters

ENDIF
```

34.5.1 Initial Values from the Given Twiss Parameters

PTC_TWISS calculates a solution with initial conditions given by the Twiss parameters, which are explicitly typed as attributes to the command. This case is also limited to uncoupled motion of the preceding ring or beam-line.

```
PTC_TWISS, ICASE=integer, DELTAP=real,
          RANGE=string, FILE[=string], TABLE[=string],
          BETX=real, ALFX=real, MUX=real,
          BETY=real, ALFY=real, MUY=real,
          BETZ=real, ALFZ=real,
          DX=real, DPX=real, DY=real, DPY=real,
          X=real, PX=real, Y=real, PY=real, T=real, PT=real;
```

Example:
An example is found in the PTC_TWISS Examples in the folder "Example2".

34.5.2 Initial Values from a Map-Table

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 existing map-table in memory, as generated by a preceding PTC_NORMAL command.

```
PTC_TWISS, ICASE=integer, DELTAP=real,
          RANGE=string, FILE[=string], TABLE[=string],
          INITIAL_MATRIX_TABLE;
```

Example:
An example is found in the PTC_TWISS Examples in the folder "Example3".

34.5.3   Initial Values from a Map-File

`PTC_TWISS` calculates a solution with initial conditions given as a map-file (fort.18) obtained from a preceding ring or beam-line. It requires the input option `INITIAL_MAP_MANUAL` and an existing map-file in file "fort.18", as generated by a preceding `PTC_NORMAL` command.

```
PTC_TWISS, ICASE=integer, DELTAP=real,
           RANGE=string, FILE[=string], TABLE[=string],
           INITIAL_MAP_MANUAL;
```

Example:
An example is found in the `PTC_TWISS` Examples in the folder "Example3".

34.5.4   Initial Values from a Given Matrix

`PTC_TWISS` calculates a solution with initial conditions given by a matrix explicitly given as attribute to the command. It requires the option `INITIAL_MATRIX_MANUAL`. `MAD-X` expects a symplectic 6x6 transfer matrix as input.

```
PTC_TWISS, ICASE=integer, DELTAP=real,
           RANGE=string, FILE[=string], TABLE[=string],
           INITIAL_MATRIX_MANUAL,
           RE11=real, RE12=real, ...  , RE16=real,
           ...
           RE61=real, RE62=real, ...  , RE66=real;
```

Example:
An example is found in the `PTC_TWISS` Examples in the folder "Example4".

34.5.5   Initial Values from Twiss Parameters via BETA0-block

`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 `MAD-X` for a preceding ring or beam-line. This case is limited to uncoupled motion of the preceding machine.

```
PTC_TWISS, ICASE=integer, DELTAP=real,
           RANGE=string, FILE[=string], TABLE[=string],
           BETA0=string ;
```

Example:
An example is found in the `PTC_TWISS` Examples in the folder "Example1".

Chapter 35. Non-Linear Machine Parameters

The `PTC_NORMAL` module ([38] and [44]) of `MAD-X` 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 `MAD-8`. It allows to calculate dispersions, chromaticities, anharmonicities and Hamiltonian terms to very high order. In fact, the order is only limited by the available computer memory and computing time.

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 user requirements with a series of `SELECT_PTC_NORMAL` command. A special `MAD-X` table is dynamically built using those commands and is filled by a subsequent call to `PTC_NORMAL`.

Another essential advantage of this table is 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;
```

35.1 SELECT_PTC_NORMAL

The `SELECT_PTC_NORMAL` command selects the parameters to be calculated by a subsequent `PTC_NORMAL` command.

```
SELECT_PTC_NORMAL, DX=integer, DPX=integer, DY=integer, DPY=integer,
                   Q1=0, Q2=0, DQ1=integer, DQ2=integer,
                   ANHX=integerarray, ANHY=integerarray,
                   GNFU=integer,0,0, HAML=integer,0,0,
                   EIGN=integer,integer;
```

The attributes are:

`DX, DPX, DY, DPY` the dispersion paramaters specified as integer numbers specifying their order: $D_x^{(n)} = \partial^{(n)} x_{co}/\partial \delta_p^{(n)}$, $D_{px}^{(n)} = \partial^{(n)} px_{co}/\partial \delta_p^{(n)}$, $D_y^{(n)} = \partial^{(n)} y_{co}/\partial \delta_p^{(n)}$, $D_{py}^{(n)} = \partial^{(n)} py_{co}/\partial \delta_p^{(n)}$, where `co` is abbreviation of "closed orbit".

Q1, Q2   horizontal and vertical tune paramaters are fixed to order 0: $Q_1^{(0)}$, $Q_2^{(0)}$

DQ1, QD2   the tune derivative parameters specified as integer numbers specifying their order. $\partial^{(n)}Q_1/\partial\delta_p^{(n)}$, $\partial^{(n)}Q_2/\partial\delta_p^{(n)}$

ANHX, ANHY   the anharmonicities, each defined by three integer numbers: the order n_1 of ϵ_1, the order n_2 of ϵ_2 and the order n_3 of δ_p. $\partial^{(n_1+n_2+n_3)}Q_z/(\partial\epsilon_1^{(n_1)}\partial\epsilon_2^{(n_2)}\partial\delta_p^{(n_3)})$. For example, ANHX=2,0,0 represents second order in ϵ_1: $\partial^{(2)}Q_1/\partial\epsilon_1^{(2)}$. And ANHY=3,1,2 represents $\partial^{(6)}Q_2/(\partial\epsilon_1^{(3)}\partial\epsilon_2^{(1)}\partial\delta_p^{(2)})$.

EIGN   components of the eigenvectors at the end of the structure can be specified by two integers: the eigenvector number and the coordinate coded in the list: $\{x, p_x, y, p_y, t, p_t\}$:
                The pair n_1,n_2 defines the n_2-th component of the n_1-th eigenvector.

GNFU   The Generating Function can be specified by $\{n_1, n_2, n_3\}$. The positive and negative values of $n - 1$ define the order of upright or skew resonances respectively. The integers n_2 and n_3 are reserved for a future upgrade and must be set to 0.
                For example GNFU=-5,0,0 calculates all Generating Function terms for skew decapoles. In the output table the cosine, sine and amplitude coefficients are denoted as "GNFC", "GNFS" and "GNFA" respectively.

HAML   the Hamiltonian terms can be specified by $\{n_1, n_2, n_3\}$ The positive and negative values of n_1 define the order of upright or skew resonances, respectively. The integers n_2 and n_3 are reserved for a future upgrade and must be set to 0. For example, HAML=3,0,0 calculates all Hamiltonian terms for upright sextupoles. In the output table the cosine, sine and amplitude coefficients are denoted as "HAMC", "HAMS" and "HAMA" respectively.

Caution: if more than one order of terms is selected only the lower one is correct because higher orders contain "cross terms" from the lower ones.

35.2   PTC_NORMAL

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.

```
PTC_NORMAL, ICASE=integer, NORMAL=logical, CLOSED_ORBIT=logical,
            NO=integer, MAPTABLE=logical, DELTAP=double;
```

The attributes are:

ICASE   user-defined dimensionality of the phase-space (4, 5 or 6).
                (Default: 4)

NO   the order of the map.
                (Default: 1)

CLOSED_ORBIT a logical switch to turn on the closed orbit calculation.
(Default: false)

DELTAP relative momentum offset for reference closed orbit.
(Default: 0.0)

MAPTABLE a logical flag to activate the storage of the map-table in memory.
MAPTABLE=true and NO=1 creates the one-turn matrix which can be used by
the next PTC_TWISS command.
(Default: false)

NORMAL a logical flag to activate the calculation of the Normal Form.
(Default: false)

Example

The simple example is located on the Web-page for the PTC_NORMAL example.

Chapter 36. `MAD-X-PTC` Auxiliaries

This chapter documents the interface between `MAD-X` and `PTC` and the auxiliary commands available in the `PTC` library.

Available Commands

- PTC_SETSWITCH
- PTC_KNOB
- PTC_SETKNOBVALUE
- MATCH_WITHPTCKNOBS   (**Under Construction**)
- PTC_PRINTPARAMETRIC
- PTC_EPLACEMENT
- PTC_PRINTFRAMES
- PTC_SELECT
- PTC_SELECT_MOMENT
- PTC_MOMENTS   (**Under Construction**)
- PTC_DUMPMAPS
- PTC_SETCAVITIES

36.1 PTC_KNOB

```
PTC_KNOB, ELEMENTNAME=string,
          KN=integer{, integer}, KS=integer{, integer},
          EXACTMATCH=logical;
```

Sets knobs in PTC calculations. This is currently valid only in PTC_TWISS; PTC_NORMAL will follow.

Knobs appear as additional parameters of the phase space. Twiss functions are then obtained as functions of these additional parameters (Taylor series). Map elements may also be stored as functions of knobs. The PTC_SELECT command description shows how to request a given element to be stored as a Taylor series.

The parametric results can also be:

1. written to a file with PTC_PRINTPARAMETRIC.

2. plotted and studied using rviewer command (RPLOT plugin).

3. used to rapidly obtain approximate values of lattice functions for given values of knobs (PTC_SETKNOBVALUE). This feature is the foundation of a fast matching algorithm with PTC.

Command parameters and switches:

ELEMENTNAME   a string in range format (Default: NULL)
Specifies name of the element containing the knob(s) to be set.

KN,KS   list of integers (Default: ???)
Defines which order

EXACTMATCH   (Default: .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 also 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   ???

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: Knob values are matched to get requested lattice functions.

36.2 PTC_SETKNOBVALUE

The PTC_SETKNOBVALUE command sets a given knob value.

```
PTC_SETKNOBVALUE, ELEMENTNAME=string,
                  KN=integer{,integer}, KS=integer{,integer},
                  VALUE=real;
```

All values in the twiss table used by the last `PTC_TWISS` command and the columns specified with `PTC_SELECT, PARAMETRIC=true;` are reevaluated using the buffered parametric results.

The parameters of the command basically contain the fields that allow to identify uniquely the knob and the value to be set.

Command parameters and switches:

ELEMENTNAME   a string in range format that specifies the name of the element containing the knob to be set.
              (Default: none)

KN, KS   are lists of integers that define the knob.
              (Default: -1)

VALUE   specifies the value to which the knob is to be set.
              (Default: 0)

Example:
dog leg chicane: strength of dipole field component in quadrupoles is matched to obtain the required R56 value.

36.3 PTC_VARYKNOBS (Under Construction)

The `PTC_VARYKNOBS` command allows matching with `PTC` knobs.

```
PTC_VARYKNOB, INITIAL=string, ELEMENT=string,
              KN=integer{,integer}, KS=integer{,integer},
              EXACTMATCH=logical, TRUSTRANGE=real,
              STEP=real, LOWER=real, UPPER=real;
```

where the attributes are

INITIAL

ELEMENT

KN, KS

EXACTMATCH

TRUSTRANGE   defines the range over which the expansion is trusted
             (Default: 0.1)

STEP

LOWER

UPPER

This matching procedure takes advantage of the parametric results that are accessible with `PTC`. Namely, parameters occurring 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 which makes finding 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 the 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 is described below.

```
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 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_length`. This guarantees consistency in treatment of normal and skew dipole 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 component is zero and tilt and normal component are modified accordingly.

Example
dog leg chicane.

Algorithm

1. Buffer the key commands (`ptc_varyknob`, `constraint`, `ptc_setswitch`, `ptc_twiss`, `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.

36.4   PTC_PRINTPARAMETRIC

Prints parametric results obtained with PTC_TWISS after defining paramteters (knobs) with PTC_KNOB command (see The resulting optical functions are represented as polynomials of the knobs.

```
PTC_PRINTPARAMETRIC, FORMAT=string, FILENAME=string;
```

Command parameters and switches:

FORMAT   output file format. Valid entries are 'tex' and 'txt'.

FILENAME   the name of the output file.

36.5 PTC_EPLACEMENT

Places a given element at required position and orientation. All rotations are made around the front face of the element.

```
PTC_EPLACEMENT, RANGE=string, REFFRAME=string,
                X=real, Y=real, Z=real, PHI=real, THETA=real,
                ONLYPOSITION=logical, ONLYORIENTATION=logical,
                AUTOPLACEDOWNSTREAM=logical, SURVEYALL=logical;
```

Command parameters and switches:

RANGE   a string in range format that specifies the name of the element to be moved.

REFFRAME   defines the coordinate system with respect to which coordinates and angles are specified.
                Possible values are:

   gcs   global coordinate system (Default)

   current   current position

   previouselement end face of the previous element

X, Y, Z   shifs of the front face of the magnet.
                (Default: 0.0)

THETA   pitch angle, rotation around x axis.
                (Default: 0.0)

PHI   yaw angle, rotation around y axis.
                (Default: 0.0)

PSI   roll angle, rotation around s axis.
                (Default: 0.0)

ONLYPOSITION   a flag to perform only translation changes, and orientation of element is left unchanged.
                (Default: false)

ONLYORIENTATION a flag to perform only rotation changes and position of element is left unchanged.
                (Default: false)

AUTOPLACEDOWNSTREAM a logical flag: if true all elements downstream are placed at default positions with respect to the moved element; if false the rest of the layout stays untouched.
                (Default: true)

SURVEYALL   a logical flag used essentially for debugging. If true, an internal survey of the entire line is performed after element placement at new position and orientation.
                (Default: true)

Example

Dog leg chicane: position of quadrupoles is matched to obtain required `R566` value.

36.6 PTC_PRINTFRAMES

Print the `PTC` geometry of a layout to a specified file.

> PTC_PRINTFRAMES, FILE=filename, FORMAT=string;

Command parameters and switches:

FILE   specifies the name of the file.

FORMAT   specifies the format of geometry data. Currently two formats are accepted:

        text   prints a simple text file. (Default)

        rootmacro   creates ROOT macro that can be used to produce a 3D display of the geometry.

Example

Dog leg chicane with some elements displaced with help of `PTC_EPLACEMENT`.

36.7 PTC_SELECT

Selects a map element to be stored in a user-defined table, or stored as a function (Taylor series) of a defined knob. Both cases can be joined in a single `PTC_SELECT` command.

> PTC_SELECT, TABLE=tabname, COLUMN=string,string,
>   POLYNOMIAL=integer, MONOMIAL=string, PARAMETRIC=logical,
>   QUANTITY=string;

Command parameters and switches:

TABLE   the name of the table where values should be stored.

COLUMN   the name of the column in the table where values should be stored.

POLYNOMIAL   specifies the row of the map.

MONOMIAL   a string composed of digits that defines the monomials of the polynomial in `PTC` nomenclature. The length of the string should be equal to the number of variables and each digit corresponds to the exponent of the corresponding 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 other words it is R21.

PARAMETRIC   a logical switch. If true, and if any knobs is defined, the map element is stored as the parametric result.
                (Default: false)

QUANTITY   ??? is that the element referred above ??

To store map elements in a user-defined table and column, the table with the named columns should pre-exist the `PTC_SELECT` command.

To store map elements as a function of a defined knob, the `PARAMETRIC` attribute must be set to `true`.

Examples

dog leg chicane: strength of quads is matched to obtain required T112 value.

dog leg chicane: position of quads is matched to obtain required T566 value.

dog leg chicane: dipole and quadrupole strengths are matched with the help of knobs to obtain required momentum compaction and Twiss functions.

36.8 PTC_SELECT_MOMENT

Selects a moment to be stored in a user-defined table, or stored as a function (Taylor series) of a defined knob. Both cases can be joined in one command.

```
PTC_SELECT_MOMENT, TABLE=tabname, COLUMN=string,
                   MOMENT_S=string,string, MOMENT=integer,
                   PARAMETRIC= logical;
```

Command parameters and switches:

MOMENT_S   a list of coma separated strings, each composed of up to 6 digits defining the moment of a polynomial in `PTC` nomenclature:
the string `'ijklmn'`, where i,j,k,l,m,n are digits from 0 to 9, defines the moment $< x^i p_x{}^j y^k p_y{}^l \Delta T^m (\Delta p/p)^n >$.

For example, `MOMENT_S=100000` defines $< x^1 >$

Note that for input we always use `MAD-X` notation where dp/p is always the 6th coordinate. Internally to `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 denoted as the string `'000002'`, even in 5D case.

This notation allows to define more then one moment with a single command. In this case, the corresponding column names are built from the string arguments to `MOMENT_S` with a `mu` prefix. However, they are always extended to 6 digits, i.e. trailing 0's are automatically added.

For example, with `MOMENT_S=2`, defines $< x^2 >$ and the corresponding column name is `mu200000`.

This method does not allow to pass bigger numbers larger than 9. In order to define such a moment, see the attribute `MOMENT` below.

MOMENT   a list of up to 6 coma separated integers that define the moment:
$< x^i p_x{}^j y^k p_y{}^l \Delta T^m (\Delta p/p)^n >$ being defined as `MOMENT=i,j,k,l,m,n`
(Default: 0)

For example: `MOMENT=2` defines $< x^2 >$, `MOMENT=0,0,2` defines $< y^2 >$, `MOMENT=0,14,0,2` defines $< p_x{}^{14}p_y{}^2 >$, etc.

COLUMN   defines the name of the column where values should be stored. If not specified then it is automatically generated from the `MOMENT` definition:
$< x^i p_x{}^j y^k p_y{}^l \Delta T^m (\Delta p/p)^n > => $ `mu_i_j_k_l_m_n`
(where numbers are separated with underscores).
This attribute is ignored if `MOMENT_S` is specified.
(Default: none)

TABLE   specifies the name of the table where the calculated moments are stored.
(Default: moments)

PARAMETRIC   a logical flag to to store the element as a parametric result if a knob has been defined.
(Default: false)

To store a moment in a user-defined table and column, the table with the named columns should pre-exist the `PTC_SELECT_MOMENT` command.

To store a moment as a function of a defined knob, the `PARAMETRIC` attribute must be set to `true`.

Examples
from ATF2:

```
!Here is sigmax**2
ptc_select_moment, table = momord2,   moment_s= 2;

!Below are example how to encode other moments
ptc_select_moment, table = momord2,   moment_s= 20,11,02,002,0011,0002,00002;
ptc_select_moment, table = momord2xy, moment_s= 1010,0110,1001,0101,10001,
                                                 01001,00101,00011;
ptc_select_moment, table = momord4,   moment_s= 40,22,04, 004,0022,0004;
ptc_select_moment, table = momord6,   moment_s= 6;
```

36.9   PTC_MOMENTS

The command `PTC_MOMENTS` calculates the 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.

```
PTC_MOMENTS, NO=integer,
             XDISTR=string, YDISTR=string, ZDISTR=string;
```

The command parameters and switches are

NO   order of the calculation, maximally twice the order of the last twiss.
(Default: 1)

XDISTR, YDISTR, ZDISTR define the distribution in x, y and z dimension respectively and can take one of the following values:

> gauss   Gaussian distribution (Default)
>
> flat5   flat distribution in the first of variables (dp over p) of a given dimension and Delta Dirac in the second one (T)
>
> flat56   flat rectangular distribution

Examples
ATF2

36.10 PTC_DUMPMAPS

PTC_DUMPMAPS dumps the linear part of the map for each element of the layout into the specified file.

> PTC_DUMPMAPS, FILE=filename;

The only command parameter is:

FILE   the filename of the file to which the matrices are dumped.
(Default: ptcmaps)

36.11 PTC_SETCAVITIES

The PTC_SETCAVITIES command adjusts cavities and sets appropriate reference momenta for a layout containing travelling wave cavities.

> PTC_SETCAVITIES;

The main goal is to update the reference beam energy for the elements that follow a travelling wave cavity. PTC traces the synchronous particle, that is the particle that has all its parameters set to zero at the beginning of the layout under study.

When PTC reaches a cavity in the layout, the parameters of the cavity may be adjusted according to the user-defined MAXACCEL switch previously set in PTC_SETSWITCH.

If MAXACCEL=true the phase of the cavity is adjusted so it gives the maximum acceleration. The phase lag is then added to this adjusted phase.

If MAXACCEL=false the cavity parameters are left unchanged.

The synchronous particle is then tracked through the travelling wave cavity and the energy gain is calculated. This energy becomes the new reference energy for all elements downstream of the cavity.

This process is repeated at every cavity encountered further in the tracking trough the layout.

Parameters of the cavities are printed to a file named "twcavsettings.txt".

Attention:

in `PTC` the phase velocity of a cavity wave is always equal to the speed of light. Hence, if `PTC` internal state `TIME` is `true`, which is the most correct setting, the voltage seen by a particle is varying along the structure. If `TIME=false`, the tracked particle is assumed to propagate at the speed of light ($v = c$) and the particle moves synchronously with the wave front.

Attention:

For programming reasons, any element that changes the reference momentum, i.e. travelling wave cavities, must be followed by a marker. If a marker does not follow immediately each of these elements, `PTC` detects an error and stops the program. Hence two cavities cannot be placed one immediately after the other and a marker must be inserted in between.

Part VI

Trailing Material

Chapter 37. Known Differences to Other Programs

37.1 Definitions in `MAD-8`

`MAD-8` uses full 6-by-6 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 [45], `MAD-8` uses the relative energy error p/p_0, which is equal the relative momentum error `delta` $= \delta p/p_0$ multiplied by $\beta = v/c$.

As from Version 8.13, `MAD-8` used an additional **constant** momentum error delta_s in all optical calculations. The transfer maps contained the **exact** dependence upon this value; therefore the tunes for large deviations could be computed with high accuracy as opposed to previous versions.

The choice of canonical variables in `MAD-X` still leads to slightly different definitions of the lattice functions. In `MAD-X` the Courant-Snyder invariants mentioned in [3] take the form

$$W_x = \gamma_x x^2 - 2\alpha_x x p_x + \beta_x p_x^2$$

Comparison to the original form

$$W_x = \gamma_x x^2 - 2\alpha_x x x' + \beta_x x'^2$$

shows that the orbit functions cannot be the same.

A more detailed analysis, using $x' = p_x/(1+\delta)$ shows that all formulas can be made consistent by defining the MAD orbit functions (index M) with respect to the Courant and Snyder functions (index C) as follows:

$$\beta_{xM} = \beta_{xC} \times (1 + \delta),$$
$$\alpha_{xM} = \alpha_{xC},$$
$$\gamma_{xM} = \gamma_{xC}/(1 + \delta)$$

For constant δ_s along the beam line and $\delta = 0$, the lattice functions are the same. In a machine where δ 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-8` invariants have the advantage that they remain invariants along the beam line even for variable δ.

With the new method this problem occurs in Twiss module (see 20) only for non-constant δ.

37.2 Treatment of Energy Error in TWISS

It has been noted by Milutinovic and Ruggiero [46] that `MAD-8` returned tunes which are too low for non-zero δ. 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 `MAD-8` Version 8.13 onwards.

Chapter 38. `MAD-X` recipes and pitfalls

Find a loose collection of pitfalls that may be difficult to avoid in particular for new users but also experienced user might profit from this list.

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 calculation 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 Thinlens Tracking module presently suffers from this deficiency since it requires the true 6d closed orbit and not the approximate one as calculated by Twiss. In this context one has to mention that the coordinate system for the Twiss module is not x, px in the horizontal plane as the advertised canonical coordinates instead x, x' have been used (same for the vertical plane).

Be careful that for `TWISS` with the `CENTRE` attribute activated, *i.e.* looking inside the element, the closed orbit includes the misalignment of the element.

Dispersion for machines with small relativistic beta `MAD-X` uses the PT coordinate as the canonical momentum in the longitudinal plane. The derivative of e.g. dispersion is therefore not taken wrt delta-p over p but PT. Therefore one unfortunately finds the dispersion being divided by the relativistic beta which is annoying for low energy machines. PTC allows to change the coordinate system to delta-p over p with the "time=false" option of the `PTC_CREATE_LAYOUT` command which delivers the proper dispersion with the

Non-standard definition of DDX, DDPX, DDY, DDPY The `MAD-X` proper defintion of `DDX, DDPX, DDY, DDPY` is **not** the second order derivative with respect to deltap/p. In order to get the second order derivative you need to multiply the value by 2. The corresponding values from `PTC_NORMAL` and in `PTC_TWISS` are the proper derivaties to all orders.

Chromaticity calculation in presence of coupling Chromaticity calculations are typically in order and agree with `PTC` and other codes. However, it was recently discovered that in presences of coupling `MAD-X` simply seems to ignore coupling when the chromaticity is calculated. This is surprising since the eigentunes Q1, Q2 are properly calculated for a given (small!) dp/p. The issue is under investigation.

Field errors in thick elements Only a very limited number of field error components are considered in `TWISS` calculations for some thick elements. Find below a complete list of all those field error components that are taking into account for a particular thick element. It should be mentioned that `BEND`s also allow a skew quadrupole component K1s but NOT in the body of the magnet. It is only active in the edge effect for radiation (expert use only).

Magnet Type	Normal Field Components	Skew Field Components
Bend	Dipole	—
	Quadrupole	—
	Sextupole	—
HKicker	Dipole	—
VKicker	—	Dipole
Quadrupole	Quadrupole	Quadrupole
Sextupole	Sextupole	Sextupole
Octupole	Octupole	Octupole

MAD-X versus PTC The 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`.

When introducing a cavity with the `HARMON` attribute instead of the `FREQ` attribute (highly discouraged!) a problem arises for `PTC_TWISS` due to the fact that internally `HARMON` is transferred to `FREQ` too late. A simple `TWISS` command executed before `PTC` start-up will help. However, avoiding `HARMON` is advantageous.

SLOW attribute in matching The `SLOW` attribute enforces the old matching procedure and is considerably slower. Therefore we did not make it the default option. Recently a number of parameters, like `RE56`, have been added to the list of matchable parameters in the default and fast version. Nevertheless, some parameters are only available when using the `SLOW` attribute. Therefore it is advisable to check with the `SLOW` attribute if there are doubts about the matching procedure.

Validity of Twiss parameters The standard Teng-Edwards Twiss parameters suffer from a deficiency near full coupling: i.e. the "donuts" of linear motion in x-x' and y-y' phase space have no hole anymore. This means that all energy is transfered from one plane to the other. In this case the Twiss parameters and the coupling matrix (`R11, R12, R21, R22`) become large or even infinite or the beta functions might become negative. The Ripken-Mais Twiss parameters are always well defined (they are the "average" amplitude functions of their proper phase space region), i.e. at full coupling we have: beta11 $\sim$ beta12 and beta21 $\sim$ beta22. Using the `RIPKEN` flag `TWISS` calculates the Mais-Ripken

parameters via a transformation from the Teng-Edwards Twiss parameters. Obviously this fails when the Teng-Edwards Twiss parameters are ill defined. In this case one has to rely on `PTC_TWISS`.

MAKETHIN might invalidate a sequence Several `MAD-X` commands such as `EALIGN`, `EFCOMP` won't work on sequences produced by `MAKETHIN`. In order to use such commands on thin sequences, it is advisable to save the sequence on a file, and then re-load it.

Chapter 39. Relation between p_t and δ_p

In this chapter we establish the exact relation of p_t and δ_p and show how it can be approximated.

Note that for simplicity c has been set to 1 in this document. This does not change the outcome of the derivations.

The definition of p_t is the following:

$$p_t = \frac{E - E_0}{P_0} \tag{39.1}$$

, where E is the total energy of the particle, E_0 is the energy of the reference particle and P_0 is the momentum of the reference particle.

δ_p is defined as:

$$\delta_p = \frac{P - P_0}{P_0} \tag{39.2}$$

, where P is the momentum of the particle. We will use the following relation in several places in the following document: $E = \frac{P}{\beta} = \frac{P_0(1+\delta_p)}{\beta}$.

The following part is a derivation of the relation between p_t and δ_p in the general case.

$$E = \sqrt{P^2 + m_0^2} = \sqrt{P_0^2(1 + \delta_p)^2 + m_0^2} \tag{39.3}$$

, where m_0 is the rest mass of the particle. Rearranging equation 39.1 we can also write the energy as

$$E = P_0 pt + E_0 = P_0 pt + \frac{P_0}{\beta_0} \tag{39.4}$$

, where we used the fact that $E_0 = \frac{P_0}{\beta_0}$

Squaring equation 39.3 and equation 39.4 we can write:

$$P_0^2(1 + \delta_p)^2 + m_0^2 = (P_0 pt + \frac{P_0}{\beta_0})^2 = P_0^2 p_t^2 + \frac{2P_0^2 p_t}{\beta_0} + \frac{P_0^2}{\beta_0^2} \tag{39.5}$$

We note that we can write $\beta_0 = \frac{P_0}{E_0} = \frac{P_0}{\sqrt{P_0^2 + m_0^2}}$ which gives:

$$\frac{P_0^2}{\beta_0^2} = P_0^2 + m_0^2 \tag{39.6}$$

Substituting equation 39.6 in to equation 39.5 gives:

$$P_0^2(1 + \delta_p)^2 + m_0^2 = P_0^2 p_t^2 + \frac{2P_0^2 p_t}{\beta_0} + P_0^2 + m_0^2. \tag{39.7}$$

We can now cancel m_0 on both sides, divide by P_0^2 and then finally we take the square root and we obtain the relation:

$$1 + \delta_p = \sqrt{p_t^2 + \frac{2p_t}{\beta_0} + 1} \tag{39.8}$$

If we now want an approximation to the above formula we again take the square and subtract 1 from both side and we get:

$$2\delta_p + \delta_p^2 = p_t^2 + \frac{2p_t}{\beta_0} \tag{39.9}$$

In the normal cases $p_t << 1$ so $p_t >> p_t^2$. We then only use the leading order in p_t and δ_p, which gives us: $\frac{2p_t}{\beta_0} \approx 2\delta_p$

$$p_t \approx \beta_0 \delta_p \tag{39.10}$$

Chapter 40. Contributors to `MAD-X`

Lists are provided in alphabetical order.

Disclaimer: any omissions in these lists are accidental.

Feel free to contact mad support (mad@cern.ch) if you have been left out or some of your contributions are not listed.

40.1   MAD team

The list includes all the persons officially working on the `MAD-X` project.

Laurent Deniau: project manager since june 2011.
Andrea Latina: since october 2011.
Piotr Skowronski: since january 2012.
Irina Tecker: since january 2015.

40.2   Module keepers and contributors

This list includes all the volunteers who accepted to maintain and develop some of the `MAD-X` modules.

Fanouria Antoniou: ibs.
Helmut Burkhardt: makethin.
Riccardo De Maria: match2.
Laurent Deniau: survey.
Valery Kapin (FNAL, Chicago): ptc_track.
Emanuele Laface (ESS, Sweden): match.
Andrea Latina: trac, twiss.
Yannis Papaphilippou: dynap.
Ghislain Roy: aperture, error, cororbit, touschek, c6t, plot.
Piotr Skowronski: (ptc_trac), ptc_twiss, ptc_normal, ptc_module.
Irina Tecker: trac, twiss, documentation.
Rogelio Tomas: emit.

We are looking for volunteers, any help is welcome!

40.3   Special contributors

This list includes all the persons who have contributed exceptionally to the project in the past.

Frank Schmidt: project custodian from 2002 to 2011.
Etienne Forest: author of PTC & FPP (written in F90).
Hans Grote: author of the core (written in C).

40.4 Other contributors

The list below includes the persons who have contributed significantly to the `MAD-X` project in the past. Unless noted they were affiliated with CERN at the time of their contributions.

Ralph Assmann (emit)
Oliver Bruning (match)
Hans Grote (Core in C, plot)
Werner Herr (error, cororbit)
Bernard Jeanneret (aperture)
Alex Koschik (thintrack)
Nikolay Malitsky - BNL, New-York (sxf)
Eric McIntosh (memory leaks)
Jean Luc Nougaret (twiss, ptc_twiss, C/Fortran wrappers, bug tracker)
Thys Risselada (threader, closed orbit)
Frank Schmidt (c6t, twiss, sodd, ptc_twiss, ptc_module, ptc_normal)
Yipeng Sun (thintrack)
Frank Tecker (survey)
André Verdier (survey)
Lingyun Yang (tpsa)
Frank Zimmermann (dynap, touschek, ibs)

`MAD-9` contributors
Christoph Iselin is the author of MAD-9 and a major contributor to the Classic library.

`MAD-8` contributors
Hans Grote and Christoph Iselin are the authors of `MAD-8`.

Change Log

<p style="text-align:center">since version 5.02.00</p>

The following changes have been made to the documentation since August 15th, 2014 in version 5.02.02

The changes are indexed by date (most recent first) and provide the `MAD-X` version number where the change applies.

`2018-Jun-07` version 5.04.01
Added information about the SixTrack option long_names in the chapter: SixTrack.

`2018-Apr-23` version 5.04.01
Fix definition of `E1` and `E2` in Bending Magnet.

`2017-Oct-20` version 5.03.07
Added a comment about the relation between ONEPASS and BBORBIT for TRACK.

`2017-Oct-17` version 5.03.07
Fix a typo in the definition of the time in the cannonical variables.

`2017-Jun-27` version 5.03.07
Add `SELECT, flag=interpolate` option to select positions for intermediate values during `TWISS`.

`2017-May-29` version 5.03.06
Add p_s definition and (local) phase slip factor.
Move `PTC_SETSWITCH` description right after `PTC_CREATE_LAYOUT`.

`2017-Apr-11` version 5.03.00
Allow Track to specify a new seed for the quantum attribute.

`2017-Apr-11` version 5.03.00
New random number generator with independent streams.

`2016-Dec-20` version 5.02.13, r6049
Lrad and tilt added to monitors for users' convenience, but not used internally.

`2016-Dec-9` version 5.02.13, r6031, r6032
New command setvars_knob and fill_knob.

`2016-Nov-28` version 5.02.13, r6018
K1 implementation in thick sbend in track.

`2016-Nov-14` version 5.02.13, r6003
Field errors ksl0 added to kickers.

`2016-Sep-30` version 5.02.10, r5946
add flag to export all markers to Sixtrack

2016-Jun-15   version 5.02.10, r5909
              sigma matrix added to Twiss table: sig11...sig66.

2016-Apr-22   version 5.02.10, r5833
              the function TABINDEX has been introduced and the section has been reschuf-
              fled.

2016-Apr-19   version 5.02.10, r5816
              the defaults for BEAM attributes ET, SIGT, SIGE have been set to the values
              $10^{-3}, 1, 10^{-3}$ respectively (were $1, 0, 0$).

2016-Apr-07   version 5.02.10, r5810
              the attribute SECTORACC has been added to the twiss command for sec-
              tormap computation.

2016-Feb-14   version 5.02.08, r5673
              fixed a bug in EMIT whereby the coordinates of the orbit were mixed up
              (through Fortran equivalence statements) between the vertical and longitu-
              dinal plane, leading to wrong results in damping partition numbers and emit-
              tances. In all tests in our test-suite the difference turned out to be very small
              but other users might experience otherwise.

2015-Nov-03   version 5.02.07, r5484
              the keyword VERSION has been introduced.

2015-Sep-15   version 5.02.07, r5407
              Removed the precedence information in the BEAM command between the longi-
              tudinal emittance (ET) and beams sizes (SIGT and SIGE) since this is actually
              not implemented in the code.

2015-Sep-15   version 5.02.07, r5399
              A new option NOEXPR has been added to the SAVE command to allow saving
              sequences with only values and no expressions in variables and commands.

2015-Sep-04   version 5.02.07, r5380
              Change of the trailing message printed on output that no longer mentions the
              version number and version architecture. The same information can already
              be found in the header message.

2015-Aug-31   version 5.02.07, r5367
              TEAPOT is now the default style for MAKETHIN. The previous default style that
              was used when the STYLE attribute was not specified in the MAKETHIN com-
              mand has been given the name HYBRID and can still be used with the explicit
              STYLE=hybrid attribute.
              **All MAKETHIN commands where STYLE is not specified now use the
              TEAPOT style instead of the previously unnamed HYBRID style**

2015-Jul-15   version 5.02.06, r5336
              corrected a typo in equation 1.10 reported by Michael Severance (Stony Brook).
              The B_2 factor of the expansion of the B_x field was reading $(xy - \frac{h^3}{6}y^3 + \cdots)$
              and has been corrected to $(xy - \frac{h}{6}y^3 + \cdots)$.

`2015-Jun-09`   version 5.02.06, r5250
added a guard against negative sequence length and negative element lengths
at the time of sequence expansion triggered by a `USE` command, with `MAD-`
`X` then finishing with fatal error. No checks were performed so far on these
attributes, assuming that all length were positive.

`2015-Jun-05`   version 5.02.06, r5247
added the cardinal sine `SINC(x)` to the list of available operators in arithmetic
expressions.

`2015-Mar-31`   version 5.02.05, r5181
added a `SHRINK` command to remove rows at the end of an existing table.

`2015-Mar-11`   version 5.02.05, r5162
Major change to the definition of emittances in the `BEAM` command. For
historical reasons, there was a factor 4 in the relation between normalised
emittance ϵ_n and geometric emittance ϵ: $\epsilon_n = 4\beta\gamma\,\epsilon$, where β and γ are the
usual relativistic factors.
The common definition $\epsilon_n = \beta\gamma\,\epsilon$ is now used across all `MAD-X` modules.

The `APERTURE` command now gets the geometric emittances from values in-
put or calculated in the `BEAM` command; the attributes `EXN` and `EYN` of the
`APERTURE` command have been removed together with their default value of
`EXN = 2.75E-6` and `EYN = 2.75E-6` corresponding to the standard normalized
emittances for LHC beams in collisions.

`2015-Mar-10`   version 5.02.05, r5161
Major change to definition of `RACETRACK` aperture type. The `RACETRACK`
aperture now refers to a generalized shape with rounding of corner with el-
lipse instead of circle. The `APERTURE` array now takes four arguments for the
`RACETRACK` shape: maximum horizontal extent, maximum vertical extent, hor-
izontal semi-axis and vertical semi-axis of ellipse for rounding the corner.
**Note also that the definition of the first two arguments has changed
from horizontal and vertical offsets to horizontal and vertical maxi-
mum extensions.**

Removed also all references in the code and the manual to the `MARGUERITE`
aperture type (two `RECTCIRCLE`s crossing at right angle) that has been depre-
cated for some time already.

`2015-Feb-19`   version 5.02.05, r5143
added the `OCTAGON` in the list of predefined `APERTURE` types.

`2015-Feb-11`   version 5.02.05, r5128
clarified that the `NMASS` constant is the unified atomic mass unit and not the
neutron mass. None of the constants have changed in 2014 PDG publication
with respect to the 2012 version [47]. Updated the reference to PDG publica-
tions to include 2014 version [8].

`2015-Jan-28`   version 5.02.05, r5118
clarified in the definition of magnetic elements that the effect of defined mag-

netic strengths is always the same, irrespective of the `CHARGE` of the particles declared in the `BEAM` command. It is agreed in the literature that a positive quadrupole (positive K_1) focuses positive particles in the horizontal plane and defocuses negative particles in the same horizontal plane, for the same direction of propagation.

Currently `MAD-X` ignores the `CHARGE` attribute and focuses both positive and negative particles in the horizontal plane when going through a quadrupole with positive K_1.

THIS MAY CHANGE IN THE FUTURE TO CONFORM TO EXISTING CONVENTIONS

Electrostatic elements (`ELSEPARATOR`, `RFCAVITY`, `CRABCAVITY`, and the RF part of the `RFMULTIPOLE`) handle the `CHARGE` attribute appropriately and provide opposite effects for opposite charges travelling in the same direction.

2014-Dec-19 version 5.02.05, r5111
added the Gauss error function `ERF` and the complementary error function `ERFC` to the list of available operators in arithmetic expressions. Added documentation in the same section for the `FLOOR`, `CEIL` and `ROUND` functions that were already implemented.

2014-Dec-10 version 5.02.04, r5093 and r5101
clarified the global coordinate system figure 1.2 with colors and representations of projections of planes onto the horizontal Cartesian plane as well as intersections of local coordinate planes with horizontal Cartesian plane.

2014-Nov-25 version 5.02.04, r5092
removed the `GLOBAL` matching constraints `DDQ1, DDQ2` from the documentation since they are not implemented in the code.

2014-Nov-14 version 5.02.04, r5081
added a `COPYFILE` command. Changed the attribute name for the destination for the `RENAMEFILE` command from `NAME` to `TO`.

2014-Nov-13 version 5.02.04, r5078
fixed figure 25.3 where the *x*-axis was pointing in the wrong direction and the orientation of the element for positive `DPHI` was also not conforming to the text for the `EALIGN` command.

2014-Nov-13 version 5.02.04, r5080
documented a bug occurring when `LINE` or `MACRO` constructs appear within a `IF ... ELSEIF ... ELSE` or a `WHILE` construct. This bug will not be fixed now.
Clarified also that `IF ... ELSEIF ... ELSE` and `WHILE` constructs can be nested to at least six levels deep.

2014-Oct-14 version 5.02.03, r5013
fixed a documented feature of `SURVEY` where the first `KSL` component of thin `MULTIPOLE` elements, representing a vertical angle for a thin dipole, was not taken into account. Both `KNL` and `KSL` are now properly taken into account. Another change was to make `SURVEY` take into account the `RFMULTIPOLE` ele-

ments in the same way that it treats `MULTIPOLE` elements.

2014-Aug-27   version 5.02.03, r4947
               changed the behaviour of `FILL` to accept as parameter a row number equal to
               the current number of rows in the table plus one, with the effect of creating a
               new row and filling it.

2014-Aug-25   version 5.02.03, r4942 and r4943
               harmonized the behaviour of `FILL`, `SETVARS` and `SETVARS_LIN` with respect to
               negative row numbers, and updated the default values. Added documentation
               sections for `SETVARS` and `SETVARS_LIN` that were hitherto undocumented.

2014-Aug-18   version 5.02.03, r4932
               a single element can now be repeated in a beamline expansion: `2*S` and `-2*S`
               are of course identical (single elements are not reversed head to tail), and also
               equivalent to `2*(S)` and `-2*(S)` if `S` is a single element.
               Documentation updated; see 13.3

Bibliography

[1] Karl L. Brown. A First-and Second-Order Matrix Theory for the Design of Beam Transport Systems and Charged Particle Spectrometers. Technical Report SLAC-75, SLAC, 1972. Revision 3.

[2] Karl L. Brown, David C. Carey, Christopher Iselin, and Franck Rothacker. TRANSPORT - A Computer Program for Designing Charged Particle Beam Transport Systems. Technical Report CERN 73-16, revised as CERN 80-04, CERN, 1980.

[3] E. D. Courant and H. S. Snyder. Theory of the alternating gradient synchrotron. *Annals of Physics*, 3:1–48, 1958.

[4] Brian W. Montague. Linear Optics for Improved Chromaticity Correction. Technical Report LEP Note 165, CERN, 1979.

[5] C. Carli and K. Skoufaris. On the evolution of the chromatic functions around an ip computed by mad-x. Technical report, CERN, Geneva, Mar 2022.

[6] R. H. Helm, M. J. Lee, P.L. Morton, and M. Sands. Evaluation of synchrotron radiation integrals. *IEEE Trans. Nucl. Sci.*, (SLAC-PUB-1193):900–901, 1973. PDF.

[7] J. M. Jowett. Introductory statistical mechanics for electron storage rings. *AIP Conf. Proc.*, (SLAC-PUB-4033):864–970, 1986. PDF.

[8] K.A. Olive et al. (Particle Data Group). The Review of Particle Physics. *Chin. Phys. C*, 38:090001, 2014.

[9] V. Danilov and S. Nagaitsev. ??? *Phys. Rev. ST Accel. Beams*, 13:084002, 2010.

[10] Philippe Defert, Ph. Hofmann, and R. Keyser. The Table File System, the C Interfaces. Technical Report LAW Note 9, CERN, 1989.

[11] Frank Schmidt. SixTrack, User's Reference Manual. Technical Report SL/95-56 (AP), CERN, 1994.

[12] SixTrack website. http://cern.ch/sixtrack.

[13] Frank Schmidt. SODD: A computer code to calculate detuning and distortion function terms in first and second order. Technical Report CERN SL/Note 99-009 (AP), CERN, 1999.

[14] Mark Hayes and Frank Schmidt. Run Environment for SixTrack. Technical Report LHC Project Note 300, CERN, July 2002.

[15] Hans Grote, J. Holt, N. Malitsky, Fulvia Pilat, Richard Talmann, and C. G. Trahern. SXF (Standard eXchange Format): definition, syntax, examples. Technical Report RHIC/AP/155, BNL, August 1998.

[16] D. A. Edwards and L. C. Teng. Parametrisation of linear coupled motion in periodic systems. *IEEE Trans. on Nucl. Sc.*, 20:885, 1973.

[17] F. James. MINUIT, A package of programs to minimise a function of n variables, compute the covariance matrix, and find the true errors. Technical report, CERN, 1978. Program Library Code D507.

[18] A. Chao. Evaluation of beam distribution parameters in an electron storage ring. *Journal of Applied Physics*, 50:595–598, 1979.

[19] H. Burkhardt, R. De Maria, M. Giovannozzi, and T. Risselada. Improved TEAPOT method and tracking with thick quadrupoles for the LHC and its upgrade. In *Proceedings of the 2013 IPAC Conference*, number MOPWO027 in International Particle Accelerator Conference, 2013. http://accelconf.web.cern.ch/AccelConf/IPAC2013/papers/mopwo027.pdf.

[20] D. E. Knuth. *The Art of Computer Programming*, volume 2. Addison-Wesley, second edition edition, 1981. Semi-numerical Algorithms.

[21] Johann Bengtsson and John Irwin. Analytical Calculation of Smear and Tune Shift. Technical Report SSC-232, SSC, February 1990.

[22] Frank Schmidt. SODD: A physics Guide. Technical Report Beam Physics Note 60, CERN, 1999.

[23] E. T. d'Amico. SODD: another `MAD-X` module. Technical Report AB Note-2004-069 (ABP), CERN, 2004.

[24] A. Piwinski. The Touschek Effect in Strong Focusing Storage Rings. Technical Report DESY-98-179, DESY, 1998.

[25] Alex Chao and Maury Tigner, editors. *Handbook of Accelerator Physics and Engineering*. World Scientific, 1999.

[26] Matthew Sands. The Physics of Electron Storage Rings. Technical Report SLAC-121, UC-28, SLAC, University of California, 1970.

[27] J. D. Bjorken and S. K. Mtingwa. Intrabeam Scattering. Technical Report FERMILAB-Pub-82/47-THY, FNAL, July 1982.

[28] M. Conte and Michel Martini. ??? *Particle Accelerators*, 17:1, 1985.

[29] Fanouria Antoniou and Frank Zimmermann. To be published. Technical report, CERN, 2012.

[30] G. J. Roy. A new method for the simulation of synchrotron radiation in particle tracking codes. *Nuclear Instruments & Methods in Phys. Res.*, A298:128–133, 1990.

[31] Helmut Burkhardt. Monte Carlo Generation of the Energy Spectrum of Synchrotron Radiation. Technical Report CERN-OPEN-2007-018, Jun 2007.

[32] R. Bartolini, A. Bazzani, M. Giovannozzi, W. Scandale, and E. Todesco. Tune evaluation in simulations and experiments. Technical Report CERN SL/95-84 (AP), CERN, 1995.

[33] Valery Kapin and Frank Schmidt. MADX-SC Flag Description. Technical Report CERN-ACC-NOTE-2013-0036, Nov 2013.

[34] Frank Schmidt and Yuri Alexahin. New SC Algorithm for MAD-X. Technical report, CERN, Geneva, Oct 2018.

[35] Valery Kapin. Space charge simulation using madx with account of longitudinal motion. *FNAL Beamsdoc-3582 v2*, 2011.

[36] Mario Conte and William W MacKay. *An Introduction to the Physics of Particle Accelerators*. WORLD SCIENTIFIC, 2nd edition, 2008.

[37] Etienne Forest, Frank Schmidt, and Eric McIntosh. Introduction to the Polymorphic Tracking Code. Technical Report KEK Report 2002-3, CERN-SL-2002-044-AP, KEK, CERN, July 2002. http://ccdb4fs.kek.jp/cgi-bin/img/allpdf?200302020, http://cern.ch/madx/doc/sl-2002-044.pdf.

[38] Frank Schmidt. MAD-X PTC Integration. Number MPPE012 in Particle Accelerator Conference, page 1772, Knoxville, USA, 2005. http://cern.ch/madx/doc/MPPE012.pdf.

[39] KEK. Elements/QUAD, SADWiki, Nov 2022. https://www-kekb.kek.jp/Documentation/SAD/?Elements%2FQUAD.

[40] V. Kapin and F. Schmidt. PTC modules for MAD-X code. Technical report, CERN, 2006. To be published.

[41] Gerhard Ripken. Untersuchungen zur Strahlführung und Stabilität der Teilchenbewegung in Beschleunigern und Storage-Ringen unter strenger Berücksichtigung einer Kopplung der Betatronschwingungen. Technical Report DESY Internal Report R1-70/4, DESY, 1970.

[42] Gerhard Ripken and Ferdinand Willeke. Methods of Beam Optics. Technical report, DESY, 1988.

[43] V. A. Lebedev and S. A. Bogacsz. Betatron motion with coupling of horizontal and vertical degrees of motion. Technical report, Thomas Jefferson National Accelerator Facility, 2010. http://cern.ch/madx/doc/1748-0221_5_10_P10010.pdf.

[44] E. T.d'Amico. Nonlinear parameters from PTC. Technical Report MAD-X Meeting 7, CERN, November 2004.

[45] A. Dragt. Lectures on Nonlinear Orbit Dynamics. In *1981 Summer School on High Energy Particle Accelerators*. American Institute of Physics, July 1982.

[46] J. Milutinovic and A. G. Ruggiero. Comparison of Accelerator Codes for a RHIC Lattice. Technical Report AD/AP/TN-9, BNL, 1988.

[47] J. Beringer et al. (Particle Data Group). The Review of Particle Physics. *Physical Review*, D86:010001, 2012.

Index