

MAD-NG Reference Manual (DRAFT)

Methodical Accelerator Design

Version 0.9 – 2020.10

Laurent Deniau.

Accelerator Beam Physics,
CERN, Meyrin, Switzerland.

Abstract

The Methodical Accelerator Design – Next Generation application is an all-in-one standalone versatile tool for particle accelerator design, modeling, and optimization, and for beam dynamics and optics studies. Its general purpose scripting language is based on the simple yet powerful Lua programming language (with a few extensions) and embeds the state-of-art Just-In-Time compiler LuaJIT. Its physics is based on symplectic integration of differential maps made out of GTPSA (Generalized Truncated Power Series). The physics of the transport maps and the normal form analysis were both strongly inspired by the PTC/FPP library from E. Forest. MAD-NG development started in 2016 by the author as a side project of MAD-X, hence MAD-X users should quickly become familiar with its ecosystem, e.g. lattices definition.

<http://cern.ch/mad>

Keywords

Methodical Accelerator Design; Accelerator beam physics; Scientific computing; JIT compiler; C and Lua programming.

Contents

I	General	12
1	Introduction	13
1	Presentation	13
2	Installation	13
2.1	Releases version	14
3	Interactive Mode	14
4	Batch Mode	15
5	Online Help	15
2	Scripting	17
1	Lua and LuaJIT	17
2	Lua primer	17
2.1	Variables	18
2.2	Control flow	18
2.3	Functions	19
2.4	Tables	19
2.5	Methods	20
3	Extensions	21
3.1	Line comment	21
3.2	Unary plus	21
3.3	Local in table	21
3.4	Lambda function	22
3.5	Deferred expression	23
3.6	Ranges	23
3.7	Lua syntax and extensions	24
4	Types	24
4.1	Value vs reference	25
5	Concepts	26
6	Ecosystem	26
3	Objects	28
1	Creation	28
1.1	Constructors	28
1.2	Incomplete objects	29
1.3	Classes	29
1.4	Identification	29

1.5	Customizing creation	30
2	Inheritance	30
2.1	Reading attributes	30
2.2	Writing attributes	31
2.3	Class instances	31
2.4	Examples	31
3	Attributes	32
4	Methods	33
5	Metamethods	36
6	Flags	37
7	Environments	37
4	Beams	39
1	Attributes	39
2	Methods	40
3	Metamethods	41
4	Particles database	41
5	Particle charges	41
6	Examples	42
5	Elements	43
1	Taxonomy	43
2	Attributes	45
3	Methods	46
4	Metamethods	47
5	Elements	48
5.1	SBend	48
5.2	RBend	48
5.3	Quadrupole	49
5.4	Sextupole	49
5.5	Octupole	50
5.6	Decapole	50
5.7	Dodecapole	50
5.8	Solenoid	50
5.9	Multipole	50
5.10	TKicker	50
5.11	Kicker, HKicker, VKicker	51
5.12	Monitor, HMonitor, VMonitor	51
5.13	RFCavity	51

5.14	RFMultipole	51
5.15	ElSeparator	52
5.16	Wiggler	52
5.17	BeamBeam	52
5.18	GenMap	52
5.19	SLink	52
5.20	Translate	53
5.21	XRotation, YRotation, SRotation	53
5.22	ChangeRef	53
5.23	ChangeNrj	53
6	Flags	53
7	Fringe fields	54
8	Sub-elements	54
9	Aperture	54
10	Misalignment	56
6	Sequence	58
1	Attributes	58
2	Methods	58
3	Metamethods	62
4	Sequences creation	63
5	Elements positions	63
6	Elements selections	64
7	Indexes, names and counts	64
8	Iterators and ranges	65
9	Examples	66
9.1	FODO cell	66
9.2	SPS compact description	67
9.3	Installing elements I	68
9.4	Installing elements II	68
7	MTable	70
1	Attributes	70
2	Methods	71
3	Metamethods	75
4	MTables creation	76
5	Rows selections	76
6	Indexes, names and counts	77
7	Iterators and ranges	78

8	Examples	79
8.1	Creating a MTable	79
8.2	Extending a MTable	79
8	MADX	81
1	Environment	81
2	Importing Sequences	81
3	Converting Scripts	81
4	Converting Macros	81
II	Commands	82
9	Introduction	83
10	Survey	84
1	Command synopsis	84
2	Survey mtable	87
3	Geometrical tracking	88
3.1	Slicing	88
3.2	Sub-elements	89
4	Examples	89
11	Track	90
1	Command synopsis	90
2	Track mtable	94
3	Dynamical tracking	95
3.1	Slicing	95
3.2	Sub-elements	96
3.3	Particles status	96
4	Examples	97
12	Cofind	98
1	Command synopsis	98
2	Cofind mtable	102
3	Examples	103
13	Twiss	104
1	Command synopsis	104
2	Twiss mtable	109
3	Tracking linear normal form	112

4	Examples	112
14	Match	113
1	Command synopsis	113
2	Environment	114
3	Command	115
4	Variables	116
5	Constraints	117
6	Objective	119
7	Algorithms	121
7.1	Stopping criteria	122
7.2	Objective function	123
7.3	Derivatives	123
8	Console output	124
9	Modules	124
9.1	LSopt	124
9.2	NLOpt	125
10	Examples	127
10.1	Matching tunes and chromaticity	127
10.2	Matching interaction point	129
10.3	Fitting data	130
10.4	Fitting data with derivatives	131
10.5	Minimizing function	132
15	Correct	134
1	Command synopsis	134
2	Correct mtable	136
3	Examples	137
16	Emit	138
17	Plot	139
1	Command synopsis	139
III	Physics	140
18	Introduction	141
1	Local reference system	141
2	Global reference system	141

19	Geometric Maps	143
20	Dynamic Maps	144
21	Integrators	145
22	Orbit	146
1	Closed Orbit	146
23	Optics	147
24	Normal Forms	148
25	Misalignments	149
26	Aperture	150
27	Radiation	151
IV	Modules	152
28	Introduction	153
29	Types	154
30	Constants	155
31	Generic Utilities	156
32	Generic Math	157
33	Range	158
34	Complex	159
35	Matrix	160
36	GTPSA	161
37	DA Map	162
38	Generic Physics	163
39	External modules	164

V	Programming	165
40	Introduction	166
41	MAD environment	167
42	Tests	168
1	Adding Tests	168
43	Elements	169
1	Adding Elements	169
44	Commands	170
1	Adding Commands	170
45	Modules	171
1	Adding Modules	171
2	Embedding Modules	171
46	Using C FFI	172
VI	Appendix	173
47	GitHub Repository	174
48	Contributors	175
49	Bibliography	176
50	Index	177

List of Figures

2.1	MAD-NG ecosystem and status.	27
3.1	Object model and inheritance.	32
5.1	Reference system for a sector bending magnet.	48
5.2	Reference system for a rectangular bending magnet.	49
5.3	Displacements in the (x, s) plane.	56
5.4	Displacements in the (y, s) plane.	56
5.5	Displacements in the (x, y) plane.	57
10.1	Survey command synopsis	84
10.2	Geometrical tracking with slices.	88
11.1	Track command synopsis	91
11.2	Dynamical tracking with slices.	96
12.1	Cofind command synopsis	99
13.1	Twiss command synopsis	105
14.1	Match command synopsis	113
14.2	Match command summary output	125
14.3	Match command intermediate output	126
14.4	Fitting data	130
14.5	Fitting data with noise	131
14.6	Fitting data with derivatives	132
15.1	Correct command synopsis	134
17.1	Plot command synopsis	139
18.1	Local Reference System	141
18.2	Global Reference System	142

List of Tables

2.1	Operators precedence	24
14.1	Match returned status	115
14.2	Match default weights list	119
14.3	List of least squares methods	127
14.4	List of non-linear local methods	127
14.5	List of non-linear global methods	128

Foreword

This textbook is in a very draft version...

Part I is under review and completion, chapters have been completed up to ch4 so far...

Temporary abbreviations used in the text:

TBD	To Be Documented
TBR	To Be Reviewed
NYI	Not Yet Implemented
Todo	To Do

Part I
General

Chapter 1. Introduction

1 Presentation

The Methodical Accelerator Design – Next Generation application is an all-in-one standalone versatile tool for particle accelerator design, modeling, and optimization, and for beam dynamics and optics studies. Its general purpose scripting language is based on the simple yet powerful Lua programming language (with a few extensions) and embeds the state-of-art Just-In-Time compiler LuaJIT. Its physics is based on symplectic integration of differential maps made out of GTPSA (Generalized Truncated Power Series). The physics of the transport maps and the normal form analysis were both strongly inspired by the PTC/FPP library from E. Forest. MAD-NG development started in 2016 by the author as a side project of MAD-X, hence MAD-X users should quickly become familiar with its ecosystem, e.g. lattices definition.

MAD-NG is free open-source software, distributed under the GNU General Public License v3.¹ The source code, units tests,² integration tests, and examples are all available on its [Github repository](#), including the [documentation](#) and its \LaTeX source. For convenience, the binaries and few examples are also made available from the [releases repository](#) located on the AFS shared file system at CERN.

2 Installation

Download the binary corresponding to your platform from the [releases repository](#) and install it in a local directory. Update (or check) that the PATH environment variable contains the path to your local directory or prefix `mad` with this path to run it. Rename the application from `mad-arch-v.m.n` to `mad` and make it executable with the command `'chmod u+x mad'` on Unix systems or add the `.exe` extension on Windows.

```
$ ./mad -h
usage: ./mad [options]... [script [args]...].
Available options are:
  -e chunk    Execute string 'chunk'.
  -l name     Require library 'name'.
  -b ...      Save or list bytecode.
  -j cmd      Perform JIT control command.
  -O[opt]     Control JIT optimizations.
  -i          Enter interactive mode after executing 'script'.
  -q          Do not show version information.
  -M          Do not load MAD environment.
  -Mt[=num]  Set initial MAD trace level to 'num'.
  -MT[=num]  Set initial MAD trace level to 'num' and location.
  -E          Ignore environment variables.
  --         Stop handling options.
  -          Execute stdin and stop handling options.
```

¹MAD-NG embeds the libraries [FFTW](#), [NFFT](#) and [NLopt](#) released under GNU (L)GPL too.

²MAD-NG has few thousands unit tests that do few millions checks, and it is constantly growing.

2.1 Releases version

MAD-NG releases are tagged on the Github repository and use mangled binary names on the releases repository, i.e. `mad-arch-v.m.n` where:

arch is the platform architecture for binaries among `linux`, `macos` and `windows`.

v is the **version** number, `0` meaning beta-version under active development.

m is the **major** release number corresponding to features completeness.

n is the **minor** release number corresponding to bug fixes.

3 Interactive Mode

To run MAD-NG in interactive mode, just typewrite its name on the Shell invite like any command-line tool. It is recommended to wrap MAD-NG with the [readline wrapper](#) `rlwrap` in interactive mode for easier use and commands history:

```
$ rlwrap ./mad
----- -- ----- | Methodical Accelerator Design
 / \ / \ / _ \ / _ \ | release: 0.9.0 (OSX 64)
 / __ / / /_ / / /_ / | support: http://cern.ch/mad
 /_/ /_/ /_/ /_/ /_/ / | licence: GPL3 (C) CERN 2016+
 | started: 2020-08-01 20:13:51

> print "hello world!"
"hello world!"
```

Here the application is assumed to be installed in the current directory `'.'` and the character `'>'` is the prompt waiting for user input in interactive mode. If you write an incomplete statement, the interpreter waits for its completion by issuing a different prompt:

```
> print          -- 1st level prompt, incomplete statement
>> "hello world!" -- 2nd level prompt, complete the statement
hello world!     -- execute
```

Typing the character `'='` right after the 1st level prompt is equivalent to call the **print** function:

```
> = "hello world!" -- 1st level prompt followed by =
hello world!      -- execute print "hello world!"
> = MAD.option.numfmt
% -.10g
```

To quit the application typewrite `Ctrl+D` to send EOF (end-of-file) on the input,³ or `Ctrl+\` to send the SIGQUIT (quit) signal, or `Ctrl+C` to send the stronger SIGINT (interrupt) signal. If the application is stalled or looping for ever, typewriting a single `Ctrl+\` or `Ctrl+C` twice will stop it:

```
> while true do end -- loop forever, 1st Ctrl+C doesn't stop it
pending interruption in VM! (next will exit) -- 2nd Ctrl+C
interrupted! -- application stopped

> while true do end -- loop forever, a single Ctrl+\ does stop it
Quit: 3 -- Signal 3 caught, application stopped
```

³Note that sending `Ctrl+D` twice from MAD-NG invite will quit both MAD-NG and its parent Shell...

In interactive mode, each line input is run in its own *chunk*⁴, which also rules variables scopes. Hence **local** variables are not visible between chunks, i.e. input lines. The simple solutions are either to use global variables or to enclose local statements into the same chunk delimited by the **do ... end** keywords:

```
> local a = "hello"
> print(a.." world!")
stdin:1: attempt to concatenate global 'a' (a nil value)
stack traceback:
stdin:1: in main chunk
[C]: at 0x01000325c0

> do                -- 1st level prompt, open the chunk
>> local a = "hello" -- 2nd level prompt, waiting for statement completion
>> print(a.." world!") -- same chunk, local 'a' is visible
>> end              -- close and execute the chunk
hello world!
> print(a)         -- here 'a' is an unset global variable
nil
> a = "hello"      -- set global 'a'
> print(a.." world!") -- works but pollutes the global environment
hello world!
```

4 Batch Mode

To run MAD-NG in batch mode, just run it in the shell with files as arguments on the command line:

```
$ ./mad [mad options] myscript1.mad myscript2.mad ...
```

where the scripts contains programs written in the MAD-NG programming language (see [Scripting](#)).

5 Online Help

MAD-NG is equipped with an online help system⁵ useful in interactive mode to quickly search for information displayed in the man-like Unix format :

```
> help()
Related topics:
MADX, aperture, beam, cmatrix, cofind, command, complex, constant, correct,
ctpsa, cvector, dynmap, element, filesys, geomap, gfunc, gmath, gphys, gplot,
gutil, hook, lfun, linspace, logrange, logspace, match, matrix, mflow,
monomial, mtable, nlogrange, nrange, object, operator, plot, range, reflect,
regex, sequence, strict, survey, symint, symintc, tostring, totable, tpsa,
track, twiss, typeid, utest, utility, vector.

> help "MADX"
NAME
MADX environment to emulate MAD-X workspace.
```

⁴A chunk is the unit of execution in Lua (see [Lua 5.2 §3.3.2](#)).

⁵The online help is far incomplete and will be completed, updated and revised as the application evolves.

SYNOPSIS

```
local lhcb1 in MADX
```

DESCRIPTION

This module provide the function 'load' that read MADX sequence and optics files and load them in the MADX global variable. If it does not exist, it will create the global MADX variable as an object and load into it all elements, constants, and math functions compatible with MADX.

RETURN VALUES

The MADX global variable.

EXAMPLES

```
MADX:open()  
-- inline definition  
MADX:close()
```

SEE ALSO

element, object.

Complementary to the help function, the function show displays the type and value of variables, and if they have attributes, the list of their names in the lexicographic order:

```
> show "hello world!"  
:string: hello world!  
> show(MAD.option)  
:table: MAD.option  
colwidth           :number: 18  
hdrwidth           :number: 18  
intfmt             :string: % -10d  
madxenv            :boolean: false  
nocharge           :boolean: false  
numfmt             :string: % -.10g  
ptcmodel           :boolean: false  
strfmt            :string: % -25s
```


Chapter 2. Scripting

The choice of the scripting language for MAD-NG was sixfold: the *simplicity* and the *completeness* of the programming language, the *portability* and the *efficiency* of the implementation, and its easiness to be *extended* and *embedded* in an application. In practice, very few programming languages and implementations fulfill these requirements, and Lua and his Just-In-Time (JIT) compiler LuaJIT were not only the best solutions but almost the only ones available when the development of MAD-NG started in 2016.

1 Lua and LuaJIT

The easiest way to shortly describe these choices is to cite their authors.

*“Lua is a powerful, efficient, lightweight, embeddable scripting language. It supports procedural programming, object-oriented programming, functional programming, data-driven programming, and data description. Lua combines simple procedural syntax with powerful data description constructs based on associative arrays and extensible semantics. Lua is dynamically typed and has automatic memory management with incremental garbage collection, making it ideal for configuration, scripting, and rapid prototyping.”*¹

*“LuaJIT is widely considered to be one of the fastest dynamic language implementations. It has outperformed other dynamic languages on many cross-language benchmarks since its first release in 2005 — often by a substantial margin — and breaks into the performance range traditionally reserved for offline, static language compilers.”*²

Lua and LuaJIT are free open-source software, distributed under the very liberal MIT license.

MAD-NG embeds a patched version of LuaJIT 2.1, a very efficient implementation of Lua 5.2.³ Hence, the scripting language of MAD-NG is Lua 5.2 with some extensions detailed in the next section, and used for both, the development of most parts of the application, and as the user scripting language. There is no strong frontier between these two aspects of the application, giving full access and high flexibility to the experienced users. The filename extension of MAD-NG scripts is `.mad`.

Learning Lua is easy and can be achieved within a few hours. The following links should help to quickly become familiar with Lua and LuaJIT:

- [Lua](#) website.
- [Lua 5.2](#) manual for MAD-NG (30 p. PDF).
- [Lua 5.0](#) free online book (old).
- [LuaJIT](#) website.
- [LuaJIT](#) wiki.
- [LuaJIT 2.1](#) documentation.
- [LuaJIT 2.1](#) on GitHub.

2 Lua primer

The next subsections introduce the basics of the Lua programming language with syntax highlights, namely variables, control flow, functions, tables and methods.⁴

¹This text is taken from the "What is Lua?" section of the Lua website.

²This text is taken from the "Overview" section of the LuaJIT website.

³The ENV feature of Lua 5.2 is not supported and will never be according to M. Pall.

⁴This primer was adapted from the blog "Learn Lua in 15 minutes" by T. Neylon.

2.1 Variables

```
n = 42 -- All numbers are doubles, but the JIT may specialize them.
-- IEEE-754 64-bit doubles have 52 bits for storing exact int values;
-- machine precision is not a problem for ints < 1e16.

s = 'walternate' -- Immutable strings like Python.
t = "double-quotes are also fine"
u = [[ Double brackets
      start and end
      multi-line strings.]]
v = "double-quotes \z

      are also fine" -- \z eats next whitespaces
t, u, v = nil -- Undefines t, u, v.
-- Lua has multiple assignments and nil completion.
-- Lua has garbage collection.

-- Undefined variables return nil. This is not an error:
foo = anUnknownVariable -- Now foo = nil.
```

2.2 Control flow

```
-- Blocks are denoted with keywords like do/end:
while n < 50 do
  n = n + 1 -- No ++ or += type operators.
end

-- If clauses:
if n > 40 then
  print('over 40')
elseif s ~= 'walternate' then -- ~= is not equals.
  -- Equality check is == like Python; ok for strs.
  io.write('not over 40\n') -- Defaults to stdout.
else
  -- Variables are global by default.
  thisIsGlobal = 5 -- Camel case is common.
  -- How to make a variable local:
  local line = io.read() -- Reads next stdin line.
  -- String concatenation uses the .. operator:
  print('Winter is coming, '..line)
end

-- Only nil and false are falsy; 0 and '' are true!
aBoolValue = false
if not aBoolValue then print('was false') end

-- 'or' and 'and' are short-circuited.
-- This is similar to the a?b:c operator in C/js:
ans = aBoolValue and 'yes' or 'no' --> ans = 'no'

-- numerical for begin, end[, step] (end included)
```

```

revSum = 0
for j = 100, 1, -1 do revSum = revSum + j end

```

2.3 Functions

```

function fib(n)
  if n < 2 then return 1 end
  return fib(n - 2) + fib(n - 1)
end

```

-- Closures and anonymous functions are ok:

```

function adder(x)
  -- The returned function is created when adder is
  -- called, and captures the value of x:
  return function (y) return x + y end
end

```

```

a1 = adder(9)
a2 = adder(36)
print(a1(16)) --> 25
print(a2(64)) --> 100

```

-- Returns, func calls, and assignments all work with lists
-- that may be mismatched in length.
-- Unmatched receivers get nil; unmatched senders are discarded.

```

x, y, z = 1, 2, 3, 4
-- Now x = 1, y = 2, z = 3, and 4 is thrown away.

```

```

function bar(a, b, c)
  print(a, b, c)
  return 4, 8, 15, 16, 23, 42
end

```

```

x, y = bar('zaphod') --> prints "zaphod nil nil"
-- Now x = 4, y = 8, values 15,..,42 are discarded.

```

-- Functions are first-class, may be local/global.
-- These are the same:

```

function f(x) return x * x end
f = function (x) return x * x end

```

-- And so are these:

```

local function g(x) return math.sin(x) end
local g; g = function (x) return math.sin(x) end
-- the 'local g' decl makes g-self-references ok.

```

```

-- Calls with one string param don't need parens:
print 'hello' -- Works fine.

```

2.4 Tables

-- Tables = Lua's only compound data structure;

```

-- they are associative arrays, i.e. hash-lookup dicts;
-- they can be used as lists, i.e. sequence of non-nil values.

-- Dict literals have string keys by default:
t = {key1 = 'value1', key2 = false, ['key.3'] = true }

-- String keys looking as identifier can use dot notation:
print(t.key1, t['key.3']) -- Prints 'value1 true'.
-- print(t.key.3)         -- Error, needs explicit indexing by string
t.newKey = {}             -- Adds a new key/value pair.
t.key2 = nil             -- Removes key2 from the table.

-- Literal notation for any (non-nil) value as key:
u = {'@!#' = 'qbert', [{}] = 1729, [6.28] = 'tau'}
print(u[6.28]) -- prints "tau"

-- Key matching is basically by value for numbers
-- and strings, but by identity for tables.
a = u['@!#'] -- Now a = 'qbert'.
b = u[{}]    -- We might expect 1729, but it's nil:

-- A one-table-param function call needs no parens:
function h(x) print(x.key1) end
h{key1 = 'Sonmi~451'} -- Prints 'Sonmi~451'.

for key, val in pairs(u) do -- Table iteration.
  print(key, val)
end

-- List literals implicitly set up int keys:
l = {'value1', 'value2', 1.21, 'gigawatts'}
for i,v in ipairs(l) do -- List iteration.
  print(i,v,l[i])       -- Indices start at 1 !
end
print("length=", #l)    -- # is defined only for sequence.
-- A 'list' is not a real type, l is just a table
-- with consecutive integer keys, treated as a list,
-- i.e. l = {[1]='value1', [2]='value2', [3]=1.21, [4]='gigawatts'}
-- A 'sequence' is a list with non-nil values.

```

2.5 Methods

```

-- Methods notation:
-- function tblname:fn(...) is the same as
--   function tblname.fn(self, ...) with self being the table.
-- calling tblname:fn(...) is the same as
--   tblname.fn(tblname, ...) here self becomes the table.
t = { disp=function(s) print(s.msg) end, -- Method 'disp'
      msg="Hello world!" }
t:disp() -- Prints "Hello world!"
function t:setmsg(msg) self.msg=msg end -- Add a new method 'setmsg'

```

```
t:setmsg "Good bye!"
t:disp() -- Prints "Good bye!"
```

3 Extensions

The aim of the extensions patches applied to the embedded LuaJIT in MAD-NG is to extend the Lua syntax in handy directions, like for example to support the deferred expression operator. A serious effort has been put to develop a Domain Specific Language (DSL) embedded in Lua using these extensions and the native language features to mimic as much as possible the syntax of MAD-X in the relevant aspects of the language, like the definition of elements, lattices or commands, and ease the transition of MAD-X users.

Bending and extending a programming language like Lua to embed a DSL is more general and challenging than creating a freestanding DSL like in MAD-X. The former is compatible with the huge codebase written by the Lua community, while the latter is a highly specialized niche language. The chosen approach attempts to get the best of the two worlds.

3.1 Line comment

The line comment operator `!` is valid in MAD-NG, but does not exist in Lua:⁵

```
local a = 1      ! this remaining part is a comment
local b = 2      -- line comment in Lua
```

3.2 Unary plus

The unary plus operator `+` is valid in MAD-NG, but does not exist in Lua:⁵

```
local a = +1     -- syntax error in Lua
local b = +a     -- syntax error in Lua
```

3.3 Local in table

The local `in` table syntax provides a convenient way to retrieve values from a *mappable* and avoid error-prone repetitions of attributes names. The syntax is as follows:

```
local sin, cos, tan in math      -- syntax error in Lua
local a, b, c in { a=1, b=2, c=3 }
! a, b, c in { a=1, b=2, c=3 }  -- invalid with global variables
```

which is strictly equivalent to the Lua code:

```
local sin, cos, tan = math.sin, math.cos, math.tan
local tbl = { a=1, b=2, c=3 }
local a, b, c = tbl.a, tbl.b, tbl.c
! local sin, cos, tan = math.cos, math.sin, math.tan  -- nasty typo
```

The JIT has many kinds of optimization to improve a lot the execution speed of the code, and these work much better if variables are declared `local` with minimal lifespan. *This language extension is of first importance for writing fast clean code!*

⁵This feature was introduced to ease the automatic translation of lattices from MAD-X to MAD-NG.

3.4 Lambda function

The lambda function syntax is pure syntactic sugar for function definition and therefore fully compatible with the Lua semantic. The following definitions are all semantically equivalent:

```

local f = function(x) return x^2 end -- Lua syntax
local f = \x x^2 -- most compact form
local f = \x -> x^2 -- most common form
local f = \ (x) -> x^2 -- for readability
local f = \ (x) -> (x^2) -- less compact form
local f = \x (x^2) -- uncommon valid form
local f = \ (x) x^2 -- uncommon valid form
local f = \ (x) (x^2) -- uncommon valid form

```

The important point is that no space must be present between the *lambda* operator `\` and the first formal parameter or the first parenthesis; the former will be considered as an empty list of parameters and the latter as an expressions list returning multiple values, and both will trigger a syntax error. For the sake of readability, it is possible without changing the semantic to add extra spaces anywhere in the definition, add an arrow operator `->`, or add parentheses around the formal parameter list, whether the list is empty or not.

The following examples show *lambda* functions with multiple formal parameters:

```

local f = function(x,y) return x+y end -- Lua syntax
local f = \x x+y -- most compact form
local f = \x,y -> x+y -- most common form
local f = \x, y -> x + y -- aerial style

```

The lambda function syntax supports multiple return values by enclosing the list of returned expressions within (not optional!) parentheses:

```

local f = function(x,y) return x+y, x-y end -- Lua syntax
local f = \x,y(x+y,x-y) -- most compact form
local f = \x,y -> (x+y,x-y) -- most common form

```

Extra surrounding parentheses can also be added to disambiguate false multiple return values syntax:

```

local f = function(x,y) return (x+y)/2 end -- Lua syntax
local f = \x,y -> ((x+y)/2) -- disambiguation: single value returned
! local f = \x,y -> (x+y)/2 -- invalid syntax at '/'

local f = function(x,y) return (x+y)*(x-y) end -- Lua syntax
local f = \x,y -> ((x+y)*(x-y)) -- disambiguation: single value returned
! local f = \x,y -> (x+y)*(x-y) -- invalid syntax at '*'

```

It is worth understanding the error message that invalid syntaxes above would report,

```
file:line: attempt to perform arithmetic on a function value.
```

as it is a bit subtle and needs some explanations: the *lambda* is syntactically closed at the end of the returned expression `(x+y)`, and the following operations `/` or `*` are considered as being outside the *lambda* definition, that is applied to the freshly created function itself...

Finally, the *lambda* function syntax supports full function syntax (for consistency) using the *fat* arrow operator `=>` in place of the arrow operator:

```
local c = 0
```

```

local f = function(x) c=c+1 return x^2 end    -- Lua syntax
local f = \x => c=c+1 return x^2 end        -- most compact form

```

The fat arrow operator requires the **end** keyword to close syntactically the *lambda* function, and the **return** keyword to return values (if any), as in Lua functions definitions.

3.5 Deferred expression

The deferred expression operator **:=** is semantically equivalent to a *lambda* function without argument. It is syntactically valid only inside *table* constructors (see [Lua 5.2 §3.4.8](#)):⁶

```

local var = 10
local fun = \-> var
! local fun := var  -- invalid syntax outside table constructors
local tbl = { v1 := var, v2 =\-> var, v3 = var }
print(tbl.v1(), tbl.v2(), tbl.v3, fun()) -- display: 10 10 10 10
var = 20
print(tbl.v1(), tbl.v2(), tbl.v3, fun()) -- display: 20 20 10 20

```

The deferred expressions hereabove have to be explicitly called to retrieve their values, because they are defined in a *table*. It is a feature of the object model making the deferred expressions behaving like values. Still, it is possible to support deferred expressions as values in a raw *table*, i.e. a table without metatable, using the deferred function from the [typeid](#) module:

```

local deferred in MAD.typeid
local var = 10
local tbl = deferred { v1 := var, v2 =\-> var, v3 = var }
print(tbl.v1, tbl.v2, tbl.v3) -- display: 10 10 10
var = 20
print(tbl.v1, tbl.v2, tbl.v3) -- display: 20 20 10

```

3.6 Ranges

The ranges are created from pairs or triplets of concatenated numbers:⁷

```

start..stop..step  -- order is the same as numerical 'for'
start..stop        -- default step is 1
3..4               -- spaces are not needed around concat operator
3..4..0.1          -- floating numbers are handled
4..3..-0.1         -- negative steps are handled
stop..start..-step -- operator precedence

```

The default value for unspecified step is 1. The Lua syntax has been modified to accept concatenation operator without surrounding spaces for convenience.

Ranges are *iterable* and *lengthable* so the following code excerpt is valid:

```

local rng = 3..4..0.1
print(#rng) -- display: 11
for i,v in ipairs(rng) do print(i,v) end

```

More details on ranges can be found in the [Range](#) module, especially about the range and logrange constructors that may adjust *step* to ensure precise loops and iterators behaviors with floating-point numbers.

⁶This feature was introduced to ease the automatic translation of lattices from MAD-X to MAD-NG.

⁷This is the only feature of MAD-NG that is incompatible with the semantic of Lua.

3.7 Lua syntax and extensions

The operator precedence (see [Lua 5.2 §3.4.7](#)) is recapped and extended in [Table 2.1](#) with their precedence level (on the left) from lower to higher priority and their associativity (on the right).

Table 2.1: Operators precedence with priority and associativity.

1:	or	left
2:	and	left
3:	< > <= >= ~= ==	left
4:	..	right
5:	+ - (binary)	left
6:	* / %	left
7:	not # - + (unary)	left
8:	^	right
9:	. [] () (call)	left

The *string* literals, *table* constructors, and *lambda* definitions can be combined with function calls (see [Lua 5.2 §3.4.9](#)) advantageously like in the object model to create objects in a similar way to MAD-X. The following function calls are semantically equivalent by pairs:

! with parentheses	! without parentheses
<code>func('hello world!')</code>	<code>func 'hello world!'</code>
<code>func("hello world!")</code>	<code>func "hello world!"</code>
<code>func([[hello world!]])</code>	<code>func [[hello world!]]</code>
<code>func({... <i>fields</i> ...})</code>	<code>func {... <i>fields</i> ...}</code>
<code>func(\x -> x^2)</code>	<code>func \x -> x^2</code>
<code>func(\x,y -> (x+y,x-y))</code>	<code>func \x,y -> (x+y,x-y)</code>

4 Types

MAD-NG is based on Lua, a dynamically typed programming language that provides the following *basic types* often italicized in this textbook:

- nil* The type of the value `nil`. Uninitialized variables, unset attributes, mismatched arguments, mismatched return values etc, have `nil` values.
- boolean* The type of the values `true` and `false`.
- number* The type of IEEE 754 double precision floating point numbers. They are exact for integers up to $\pm 2^{53}$ ($\approx \pm 10^{16}$). Value like `0`, `1`, `1e3`, `1e-3` are numbers.
- string* The type of character strings. Strings are “internalized” meaning that two strings with the same content compare equal and share the same memory address:
`a="hello"; b="hello"; print(a==b) --display: true.`
- table* The type of tables, see [Lua 5.2 §3.4.8](#) for details. In this textbook, the following qualified types are used to distinguish between two kinds of special use of tables:

- A *list* is a table used as an array, that is a table indexed by a *continuous* sequence of integers starting from 1 where the length operator `#` has defined behavior.⁸
- A *set* is a table used as a dictionary, that is a table indexed by keys — strings or other types — or a *sparse* sequence of integers where the length operator `#` has undefined behavior.

<i>function</i>	The type of functions, see Lua 5.2 §3.4.10 for details. In this textbook, the following qualified types are used to distinguish between few kinds of special use of functions: <ul style="list-style-type: none"> – A <i>lambda</i> is a function defined with the <code>\</code> syntax. – A <i>functor</i> is an object⁹ that behaves like a function. – A <i>method</i> is a function called with the <code>:</code> syntax and its owner as first argument. A <i>method</i> defined with the <code>:</code> syntax has an implicit first argument named <code>self</code>.¹⁰
<i>thread</i>	The type of coroutines, see Lua 5.2 §2.6 for details.
<i>userdata</i>	The type of raw pointers with memory managed by Lua, and its companion <i>lightuserdata</i> with memory managed by the host language, usually C. They are mainly useful for interfacing Lua with its C API, but MAD-NG favors the faster FFI ¹¹ extension of LuaJIT.
<i>cdata</i>	The type of C data structures that can be defined, created and manipulated directly from Lua as part of the FFI ¹¹ extension of LuaJIT. The numeric ranges, the complex numbers, the (complex) matrices, and the (complex) GTPSA are <i>cdata</i> fully compatible with the embedded C code that operates them.

This textbook uses also some extra terms in place of types:

<i>value</i>	An instance of any type.
<i>reference</i>	A valid memory location storing some <i>value</i> .
<i>logical</i>	A <i>value</i> used by control flow, where <code>nil</code> \equiv <code>false</code> and <i>anything-else</i> \equiv <code>true</code> .

4.1 Value vs reference

The types *nil*, *boolean* and *number* have a semantic by *value*, meaning that variables, arguments, return values, etc., hold their instances directly. As a consequence, any assignment makes a copy of the *value*, i.e. changing the original value does not change the copy.

The types *function*, *table*, *thread*, *userdata* and *cdata* have a semantic by *reference*, meaning that variables, arguments, return values, etc., do not store their instances directly but a *reference* to them. As a consequence, any assignment makes a copy of the *reference* and the instance becomes shared, i.e. references have a semantic by *value* but changing the original value does change the copy.¹²

The types *string*, *complex cdata* and *numeric (log)range cdata* have a hybrid semantic. In practice these types have a semantic by *reference*, but they behave like types with semantic by *value* because their instances are immutable, and therefore sharing them is safe.

⁸The Lua community uses the term *sequence* instead of *list*, which is confusing in the context of MAD-NG.

⁹Here the term “object” is used in the Lua sense, not as an object from the object model of MAD-NG.

¹⁰This *hidden* methods argument is named `self` in Lua and Python, or `this` in Java and C++.

¹¹FFI stands for Foreign Function Interface, an acronym well known in high-level languages communities.

¹²References semantic in Lua is similar to pointers semantic in C, see ISO/IEC 9899:1999 §6.2.5.

The instances of the (basic) types with semantic by *reference* are called “objects” in Lua, and should not be confused with the instances of the *object* (non-basic) type introduced by the object model of MAD-NG.

5 Concepts

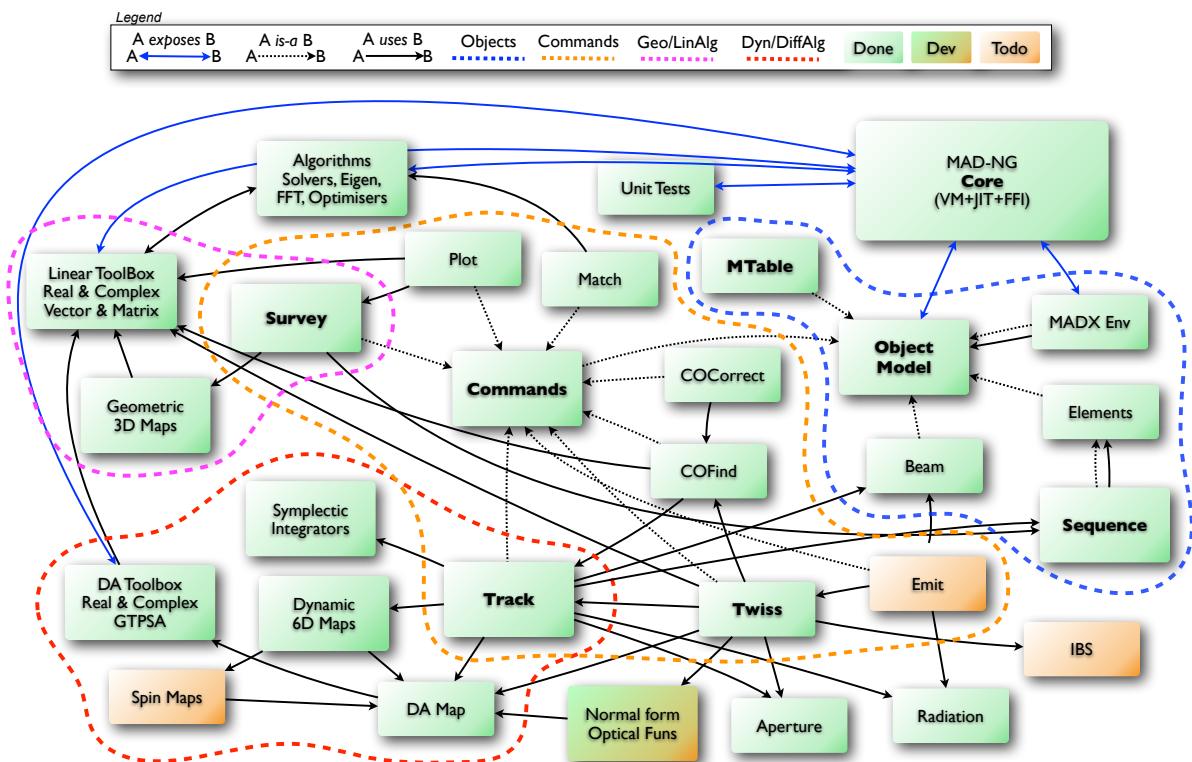
The concepts are natural extensions of types that concentrate more on behavior of objects⁹ than on types. MAD-NG introduces many concepts to validate objects passed as argument before using them. The main concepts used in this textbook are listed below, see the [typeid](#) module for more concepts:

<i>lengthable</i>	An object that can be sized using the length operator #. Strings, lists, vectors and ranges are examples of <i>lengthable</i> objects.
<i>indexable</i>	An object that can be indexed using the square bracket operator []. Tables, vectors and ranges are examples of <i>indexable</i> objects.
<i>iterable</i>	An object that can be iterated with a loop over indexes or a generic for with the <code>ipairs</code> iterator. Lists, vectors and ranges are examples of <i>iterable</i> objects.
<i>mappable</i>	An object that can be iterated with a loop over keys or a generic for with the <code>pairs</code> iterator. Sets and objects (from the object model) are examples of <i>mappable</i> objects.
<i>callable</i>	An object that can be called using the call operator (). Functions and functors are examples of <i>callable</i> objects.

6 Ecosystem

The Figure 2.1 shows a schematic representation of the ecosystem of MAD-NG, which should help the users to understand the relationship between the different components of the application. The dashed lines are grouping the items (e.g. modules) by topics while the arrows are showing interdependencies between them and the colors their status.

Figure 2.1: MAD-NG ecosystem and status.



Chapter 3. Objects

The object model is of key importance as it implements many features used extensively by objects like beam, sequence, mtable, all the commands, all the elements, and the MADX environment. The aim of the object model is to extend the scripting language with concepts like objects, inheritance, methods, metamethods, deferred expressions, commands and more.

In computer science, the object model of MAD-NG is said to implement the concepts of prototypical objects, single inheritance and dynamic lookup of attributes:

- A *prototypical object* is an object created from a prototype,¹ named its parent.
- *Single inheritance* specifies that an object has only one direct parent.
- *Dynamic lookup* means that undefined attributes are searched in the parents at *each* read.

A prototype represents the default state and behavior, and new objects can reuse part of the knowledge stored in the prototype by inheritance, or by defining how the new object differs from the prototype. Because any object can be used as a prototype, this approach holds some advantages for representing default knowledge, and incrementally and dynamically modifying them.

1 Creation

The creation of a new object requires to hold a reference to its parent, i.e. the prototype, which indeed will create the child and return it as if it were returned from a function:

```
local object in MAD
local obj = object { }
```

The special *root object* object from the MAD environment is the parent of *all* objects, including elements, sequences, TFS tables and commands. It provides by inheritance the methods needed to handle objects, environments, and more. In this minimalist example, the created object has `object` as parent, so it is the simplest object that can be created.

It is possible to name immutably an object during its creation:

```
local obj = object 'myobj' { }
print(obj.name) -- display: myobj
```

Here,² `obj` is the variable holding the object while the *string* `'myobj'` is the name of the object. It is important to distinguish well the variable that holds the *object* from the object's name that holds the *string*, because they are very often named the same.

It is possible to define attributes during object creation or afterward:

```
local obj = object 'myobj' { a=1, b='hello' }
obj.c = { d=5 } -- add a new attribute c
print(obj.name, obj.a, obj.b, obj.c.d) -- display: myobj 1 hello 5
```

1.1 Constructors

The previous object creation can be done equivalently using the prototype as a constructor:

```
local obj = object('myobj',{ a=1, b='hello' })
```

¹Objects are not clones of prototypes, they share states and behaviors with their parents but do not hold copies.

²This syntax for creating objects eases the lattices translation from MAD-X to MAD-NG.

An object constructor expects two arguments, an optional *string* for the name, and a required *table* for the attributes placeholder, optionally filled with initial attributes. The table is used to create the object itself, so it cannot be reused to create a different object:

```
local attr = { a=1, b='hello' }
local obj1 = object('obj1',attr) -- ok
local obj2 = object('obj2',attr) -- runtime error, attr is already used.
```

The following objects creations are all semantically equivalent but use different syntax that may help to understand the creation process and avoid runtime errors:

```
-- named objects:
local nobj = object 'myobj' { } -- two stages creation.
local nobj = object 'myobj' ({ }) -- idem.
local nobj = object('myobj') { } -- idem.
local nobj = object('myobj')({ }) -- idem.
local nobj = object('myobj', { }) -- one stage creation.
-- unnamed objects:
local uobj = object { } -- one stage creation.
local uobj = object ({ }) -- idem.
local uobj = object() { } -- two stages creation.
local uobj = object()({ }) -- idem.
local uobj = object(nil,{ }) -- one stage creation.
```

1.2 Incomplete objects

The following object creation shows how the two stage form can create an incomplete object that can only be used to complete its construction:

```
local obj = object 'myobj' -- obj is incomplete, table is missing
print(obj.name) -- runtime error.
obj = obj { } -- now obj is complete.
print(obj.name) -- display: myobj
```

Any attempt to use an incomplete object will trigger a runtime error with a message like:

```
file:line: forbidden read access to incomplete object.
```

or

```
file:line: forbidden write access to incomplete object.
```

depending on the kind of access.

1.3 Classes

An object used as a prototype to create new objects becomes a *class*, and a class cannot change, add, remove or override its methods and metamethods. This restriction ensures the behavioral consistency between the children after their creation. An object qualified as *final* cannot create instances and therefore cannot become a class.

1.4 Identification

The object module extends the `typeid` module with the `is_object(a)` function, which returns true if its argument `a` is an object, false otherwise:

```

local is_object in MAD.typeid
print(is_object(object), is_object(object{}), is_object{}))
-- display: true true false

```

It is possible to know the objects qualifiers using the appropriate methods:

```

print(object:is_class(), object:is_final(), object:is_readonly())
-- display: true false true

```

1.5 Customizing creation

During the creation process of objects, the metamethod `__init(self)` is invoked if it exists, with the newly created object as its sole argument to let the parent finalize or customize its initialization *before* it is returned. This mechanism is used by commands to run their `:exec()` *method* during their creation.

2 Inheritance

The object model allows to build tree-like inheritance hierarchy by creating objects from classes, themselves created from other classes, and so on until the desired hierarchy is modeled. The example below shows an excerpt of the taxonomy of the elements as implemented by the `element` module, with their corresponding depth levels in comment:

```

local object in MAD -- depth level 1
local element = object {...} -- depth level 2

local drift_element = element {...} -- depth level 3
local instrument = drift_element {...} -- depth level 4
local monitor = instrument {...} -- depth level 5
local hmonitor = monitor {...} -- depth level 6
local vmonitor = monitor {...} -- depth level 6

local thick_element = element {...} -- depth level 3
local tkicker = thick_element {...} -- depth level 4
local kicker = tkicker {...} -- depth level 5
local hkicker = kicker {...} -- depth level 6
local vicker = kicker {...} -- depth level 6

```

2.1 Reading attributes

Reading an attribute not defined in an object triggers a recursive dynamic lookup along the chain of its parents until it is found or the root object is reached. Reading an object attribute defined as a *function* automatically evaluates it with the object passed as the sole argument and the returned value is forwarded to the reader as if it were the attribute's value. When the argument is not used by the function, it becomes a *deferred expression* that can be defined directly with the operator `:=` as explained in section 3.5. This feature allows to use attributes holding values and functions the same way and postpone design decisions, e.g. switching from simple value to complex calculations without impacting the users side with calling parentheses at every use.

The following example is similar to the second example of the section 3.5, and it must be clear that `fun` must be explicitly called to retrieve the value despite that its definition is the same as the attribute `v2`.

```

local var = 10
local fun = \-> var -- here := is invalid

```

```

local obj = object { v1 := var, v2 =\-> var, v3 = var }
print(obj.v1, obj.v2, obj.v3, fun()) -- display: 10 10 10 10
var = 20
print(obj.v1, obj.v2, obj.v3, fun()) -- display: 20 20 10 20

```

2.2 Writing attributes

Writing to an object uses direct access and does not involve any lookup. Hence setting an attribute with a non-nil value in an object hides his definition inherited from the parents, while setting an attribute with nil in an object restores the inheritance lookup:

```

local obj1 = object { a=1, b='hello' }
local obj2 = obj1 { a=\s-> s.b..' world' }
print(obj1.a, obj2.a) -- display: 1 hello world
obj2.a = nil
print(obj1.a, obj2.a) -- display: 1 1

```

This property is extensively used by commands to specify their attributes default values or to rely on other commands attributes default values, both being overridable by the users.

It is forbidden to write to a read-only objects or to a read-only attributes. The former can be set using the `:readonly method`, while the latter corresponds to attributes with names that start by `__`, i.e. two underscores.

2.3 Class instances

To determine if an object is an instance of a given class, use the `:is_instanceOf method`:

```

local hmonitor, instrument, element in MAD.element
print(hmonitor:is_instanceOf(instrument)) -- display: true

```

To get the list of *public* attributes of an instance, use the `:get_varkeys method`:

```

for _,a in ipairs(hmonitor:get_varkeys()) do print(a) end
for _,a in ipairs(hmonitor:get_varkeys(object)) do print(a) end
for _,a in ipairs(hmonitor:get_varkeys(instrument)) do print(a) end
for _,a in ipairs(element:get_varkeys()) do print(a) end

```

The code snippet above lists the names of the attributes set by:

- the object `hmonitor` (only).
- the objects in the hierachy from `hmonitor` to `object` included.
- the objects in the hierachy from `hmonitor` to `instrument` included.
- the object `element` (only), the root of all elements.

2.4 Examples

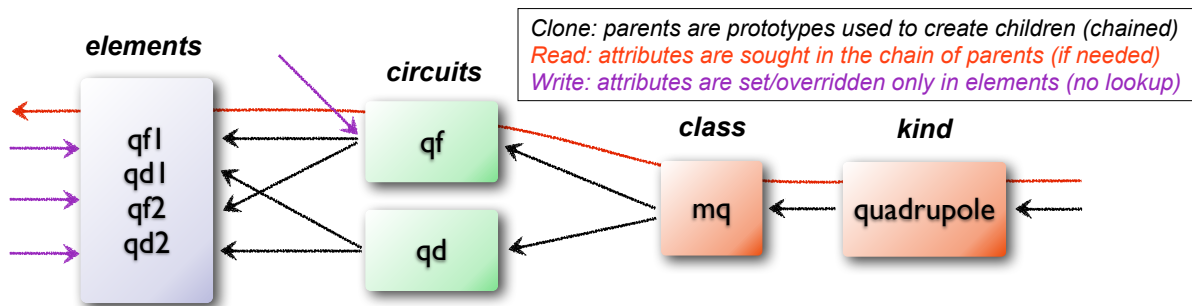
The Figure 3.1 summarizes inheritance and attributes lookup with arrows and colors, which are reproduced by the example hereafter:

```

local element, quadrupole in MAD.element -- kind
local mq = quadrupole 'mq' { l = 2.1 } -- class
local qf = mq 'qf' { k1 = 0.05 } -- circuit
local qd = mq 'qd' { k1 = -0.06 } -- circuit

```

Figure 3.1: Object model and inheritance.



```

local qf1 = qf      'qf1' {}          -- element
... -- more elements
print(qf1.k1)      -- display: 0.05 (lookup)
qf.k1 = 0.06        -- update strength of 'qf' circuit
print(qf1.k1)      -- display: 0.06 (lookup)
qf1.k1 = 0.07       -- set strength of 'qf1' element
print(qf.k1, qf1.k1) -- display: 0.06 0.07 (no lookup)
qf1.k1 = nil       -- cancel strength of 'qf1' element
print(qf1.k1, qf1.l) -- display: 0.06 2.1 (lookup)
print(#element:get_varkeys()) -- display: 33 (may vary)

```

The element `quadrupole` provided by the `element` module is the father of the objects created on its left. The *black arrows* show the user defined hierarchy of object created from and linked to the `quadrupole`. The main `quadrupole mq` is a user class representing the physical element, e.g. defining a length, and used to create two new classes, a focusing quadrupole `qf` and a defocusing quadrupole `qd` to model the circuits, e.g. hold the strength of elements connected in series, and finally the real individual elements `qf1`, `qd1`, `qf2` and `qd2` that will populate the sequence. A tracking command will request various attributes when crossing an element, like its length or its strength, leading to lookup of different depths in the hierarchy along the *red arrow*. A user may also write or overwrite an attribute at different level in the hierarchy by accessing directly to an element, as shown by the *purple arrows*, and mask an attribute of the parent with the new definitions in the children. The construction shown in this example follows the *separation of concern* principle and it is still highly reconfigurable despite that it does not contain any deferred expression or lambda function.

3 Attributes

New attributes can be added to objects using the dot operator `.` or the indexing operator `[]` as for tables. Attributes with non-*string* keys are considered as private. Attributes with *string* keys starting by two underscores are considered as private and read-only, and must be set during creation:

```

mq.comment = "Main Arc Quadrupole"
print(qf1.comment)    -- displays: Main Arc Quadrupole
qf.__k1 = 0.01        -- error
qf2 = qf { __k1=0.01 } -- ok

```

The root object provides the following attributes:

name A *lambda* returning the *string* `__id`.

parent A *lambda* returning a *reference* to the parent *object*.

Warning: the following private and read-only attributes are present in all objects as part of the object model and should *never be used, set or changed*; breaking this rule would lead to an *undefined behavior*:

__id	A <i>string</i> holding the object's name set during its creation. (default: nil).
__flg	A <i>number</i> holding the object's flags.
__var	A <i>table</i> holding the object's variables, i.e. attributes storing pairs of (<i>key</i> , <i>value</i>).
__env	A <i>table</i> holding the object's environment.
__index	A <i>reference</i> to the object's parent variables.

4 Methods

New methods can be added to objects but not classes, using the `:set_methods(set)` *method* with `set` being the *set* of methods to add as in the following example:

```
sequence :set_methods {
  name_of  = name_of,
  index_of = index_of,
  range_of = range_of,
  length_of = length_of,
  ...
}
```

where the keys are the names of the added methods and their values must be a *callable* accepting the object itself, i.e. `self`, as their first argument.

The root object provides the following methods:

is_final	A <i>method</i> () returning a <i>boolean</i> telling if the object is final, i.e. cannot have instance.
is_class	A <i>method</i> () returning a <i>boolean</i> telling if the object is a <i>class</i> , i.e. had/has an instance.
is_readonly	A <i>method</i> () returning a <i>boolean</i> telling if the object is read-only, i.e. attributes cannot be changed.
is_instanceOf	A <i>method</i> (cls) returning a <i>boolean</i> telling if <code>self</code> is an instance of <code>cls</code> .
set_final	A <i>method</i> ([a]) returning <code>self</code> set as final if <code>a ~= false</code> or non-final.
set_readonly	A <i>method</i> ([a]) returning <code>self</code> set as read-only if <code>a ~= false</code> or read-write.
same	A <i>method</i> ([name]) returning an empty clone of <code>self</code> and named after the <i>string</i> <code>name</code> (default: nil).
copy	A <i>method</i> ([name]) returning a copy of <code>self</code> and named after the <i>string</i> <code>name</code> (default: nil). The private attributes are not copied, e.g. the final, class or read-only qualifiers are not copied.
get_varkeys	A <i>method</i> ([cls]) returning both, the <i>list</i> of the non-private attributes of <code>self</code> down to <code>cls</code> (default: <code>self</code>) included, and the <i>set</i> of their keys in the form of pairs (<i>key</i> , <i>key</i>).
get_variables	A <i>method</i> (lst, [set], [noeval]) returning a <i>set</i> containing the pairs (<i>key</i> , <i>value</i>) of the attributes listed in <code>lst</code> . If <code>set</code> is provided, it will be used to store the pairs. If <code>noeval == true</code> , the functions are not evaluated. The full <i>list</i> of attributes can be retrieved from <code>get_varkeys</code> . Shortcut <code>getvar</code> .

set_variables A *method* (set, [override]) returning self with the attributes set to the pairs (*key*, *value*) contained in set. If override == false, the read-only attributes (with *key* starting by "__") cannot be updated. If override == "init", the read-only qualifier of self is ignored. Shortcut setvar.

copy_variables A *method* (set, [lst], [override]) returning self with the attributes listed in lst set to the pairs (*key*, *value*) contained in set. If lst is not provided, it is replaced by self.__attr. If set is an *object* and lst.noeval exists, it is used as the list of attributes to copy without function evaluation.³ If override == false, the read-only attributes (with *key* starting by "__") cannot be updated. If override == "init", the read-only qualifier of self is ignored. Shortcut cpyvar.

wrap_variables A *method* (set, [override]) returning self with the attributes wrapped by the pairs (*key*, *value*) contained in set, where the *value* must be a *callable* (a) that takes the attribute (as a callable) and returns the wrapped *value*. If override == false, the read-only attributes (with *key* starting by "__") cannot be updated. If override == "init", the read-only qualifier of self is ignored.

The following example shows how to convert the length *l* of an RBEND from cord to arc,⁴ keeping its strength *k0* to be computed on the fly:

```

local cord2arc in MAD.gmath
local rbend    in MAD.element
local printf  in MAD.utility
local rb = rbend 'rb' { angle=pi/10, l=2, k0=\s s.angle/s.l }
printf("l=%.5f, k0=%.5f\n", rb.l, rb.k0) -- l=2.00000, k0=0.15708
rb:wrap_variables { l=\s cord2arc(l(),s.angle) } -- RBARC
printf("l=%.5f, k0=%.5f\n", rb.l, rb.k0) -- l=2.00825, k0=0.15643
rb.angle = pi/20 -- update angle
printf("l=%.5f, k0=%.5f\n", rb.l, rb.k0) -- l=2.00206, k0=0.07846

```

The method converts non-*callable* attributes into callables automatically to simplify the user-side, i.e. *l()* can always be used as a *callable* whatever its original form was. At the end, *k0* and *l* are computed values and updating *angle* affects both as expected.

clear_variables A *method* () returning self after setting all non-private attributes to nil.

clear_array A *method* () returning self after setting the array slots to nil, i.e. clear the *list* part.

clear_all A *method* () returning self after clearing the object except its private attributes.

set_parent A *method* (newp, [override]) returning self after changing its parent to a new parent sharing the same direct parent. If override == "init", the read-only qualifier of self is ignored.

The following example shows how to change the parent of an object:

```

local hmonitor, vmonitor, monitor in MAD.element
local bpm = hmonitor 'bpm' { l=0.2 }
print(bpm.parent.name) -- display: hmonitor
bpm:set_parent(vmonitor) -- hmon. and vmon. have the same parent.
print(bpm.parent.name) -- display: vmonitor
!bpm:set_parent(monitor) -- error

```

³This feature is used to setup a command from another command, e.g. track from twiss.

⁴This approach is safer than the volatile option RBARC of MAD-X.

The last line would trigger an error as `vmonitor` and `monitor` don't share the same parent.

- set_methods** A *method* (`set`, [`override`]) returning `self` with the methods set to the pairs (*key*, *value*) contained in `set`, where *key* must be a *string* (the method's name) and *value* must be a *callable* (the method itself). If `override == false`, the read-only methods (with *key* starting by `"__"`) cannot be updated. If `override == "init"`, the class qualifier of `self` is ignored, and it is the user's responsibility to ensure the behavioral consistency across children, i.e. no method was added yet to a child.
- set_metamethods** A *method* (`set`, [`override`]) returning `self` with the attributes set to the pairs (*key*, *value*) contained in `set`, where *key* must be a *string* (the metamethod's name) and *value* must be a *callable* (the metamethod itself). If `override ~= true`, the metamethods cannot be updated.
- insert** A *method* (`[idx]`, `a`) returning `self` after inserting `a` at the position `idx` (default: `#self+1`) and shifting up the items at positions `idx...`
- remove** A *method* (`[idx]`) returning the *value* removed at the position `idx` (default: `#self`) and shifting down the items at positions `idx...`
- move** A *method* (`idx1`, `idx2`, `idxto`, [`dst`]) returning the destination object `dst` (default: `self`) after moving the items from `self` at positions `idx1..idx2` to `dst` at positions `idxto...`. The destination range can overlap with the source range.
- sort** A *method* (`[cmp]`) returning `self` after sorting in-place its *list* part using the ordering *callable* `cmp(ai, aj)` (default: `"<"`), which must define a partial order over the items. The sorting algorithm is not stable.
- bsearch** A *method* (`a`, [`cmp`], [`low`], [`high`]) returning the lowest index `idx` in the range specified by `low..high` (default: `1..#self`) from the **ordered** *list* of `self` that compares `true` with item `a` using the *callable* `cmp(a, self[idx])` (default: `"<="` for ascending, `">="` for descending), or `high+1`. In the presence of multiple equal items, `"<="` (resp. `">="`) will return the index of the first equal item while `"<"` (resp. `">"`) the index next to the last equal item for ascending (resp. descending) order.⁵
- lsearch** A *method* (`a`, [`cmp`], [`low`], [`high`]) returning the lowest index `idx` in the range specified by `low..high` (default: `1..#self`) from the *list* of `self` that compares `true` with item `a` using the *callable* `cmp(a, self[idx])` (default: `"=="`), or `high+1`. In the presence of multiple equal items in an ordered *list*, `"<="` (resp. `">="`) will return the index of the first equal item while `"<"` (resp. `">"`) the index next to the last equal item for ascending (resp. descending) order.⁵
- get_flags** A *method* (`()`) returning the flags of `self`. The flags are not inherited nor copied.
- set_flags** A *method* (`flags`) returning `self` after setting the flags determined by `flags`.
- clear_flags** A *method* (`flags`) returning `self` after clearing the flags determined by `flags`.
- test_flags** A *method* (`flags`) returning a *boolean* telling if all the flags determined by `flags` are set.
- open_env** A *method* (`[ctx]`) returning `self` after opening an environment, i.e. a global scope, using `self` as the context for `ctx` (default: `1`). The argument `ctx` must be either a *function* or a *number* defining a call level ≥ 1 .

⁵`bsearch` and `lsearch` stand for binary (ordered) search and linear (unordered) search respectively.

close_env	A <i>method</i> () returning <code>self</code> after closing the environment linked to it. Closing an environment twice is safe.
load_env	A <i>method</i> (loader) returning <code>self</code> after calling the loader, i.e. a compiled chunk, using <code>self</code> as its environment. If the loader is a <i>string</i> , it is interpreted as the filename of a script to load, see functions <code>load</code> and <code>loadfile</code> in Lua 5.2 §6.1 for details.
dump_env	A <i>method</i> () returning <code>self</code> after dumping its content on the terminal in the rough form of pairs (<i>key</i> , <i>value</i>), including content of table and object <i>value</i> , useful for debugging environments.
is_open_env	A <i>method</i> () returning a <i>boolean</i> telling if <code>self</code> is an open environment.
raw_len	A <i>method</i> () returning the <i>number</i> of items in the <i>list</i> part of the object. This method should not be confused with the native <i>function</i> <code>rawlen</code> .
raw_get	A <i>method</i> (key) returning the <i>value</i> of the attribute <i>key</i> without <i>lambda</i> evaluation nor inheritance lookup. This method should not be confused with the native <i>function</i> <code>rawget</code> .
raw_set	A <i>method</i> (key, val) setting the attribute <i>key</i> to the <i>value</i> <i>val</i> , bypassing all guards of the object model. This method should not be confused with the native <i>function</i> <code>rawset</code> . Warning: use this dangerous method at your own risk!
var_get	A <i>method</i> (key) returning the <i>value</i> of the attribute <i>key</i> without <i>lambda</i> evaluation.
var_val	A <i>method</i> (key, val) returning the <i>value</i> <i>val</i> of the attribute <i>key</i> with <i>lambda</i> evaluation. This method is the complementary of <code>var_get</code> , i.e. <code>__index</code> \equiv <code>var_val</code> \circ <code>var_get</code> .
dumpobj	A <i>method</i> ([fname], [cls], [patt]) return <code>self</code> after dumping its non-private attributes in file <i>fname</i> (default: <code>stdout</code>) in a hierarchical form down to <i>cls</i> . If the <i>string</i> <i>patt</i> is provided, it filters the names of the attributes to dump. If <i>fname</i> \equiv <code>'-'</code> , the dump is returned as a <i>string</i> in place of <code>self</code> .

5 Metamethods

New metamethods can be added to objects but not classes, using the `:set_metamethods(set)` *method* with *set* being the *set* of metamethods to add as in the following example:

```
sequence :set_metamethods {
  __len      = len_mm,
  __index    = index_mm,
  __newindex = newindex_mm,
  ...
}
```

where the keys are the names of the added metamethods and their values must be *callable* accepting the object itself, i.e. `self`, as their first argument.

The root object provides the following metamethods:

__init	A <i>metamethod</i> () called to finalize <code>self</code> before returning from the constructor.
__same	A <i>metamethod</i> () similar to the <i>method</i> <code>same</code> .

__copy	A <i>metamethod</i> () similar to the <i>method</i> copy.
__parent	A <i>metamethod</i> () similar to the <i>attribute</i> parent.
__len	A <i>metamethod</i> () called by the length operator # to return the size of the <i>list</i> part of self.
__call	A <i>metamethod</i> ([name], tbl) called by the call operator () to return an instance of self created from name and tbl, i.e. using self as a constructor.
__index	A <i>metamethod</i> (key) called by the indexing operator [key] to return the <i>value</i> of an attribute determined by <i>key</i> after having performed <i>lambda</i> evaluation and inheritance lookup.
__newindex	A <i>metamethod</i> (key, val) called by the assignment operator [key]=val to create new attributes for the pairs (<i>key</i> , <i>value</i>).
__pairs	A <i>metamethod</i> () called by the <i>pairs function</i> to return an iterator over the non-private attributes of self.
__ipairs	A <i>metamethod</i> () called by the <i>ipairs function</i> to return an iterator over the <i>list</i> part of self.
__tostring	A <i>metamethod</i> () called by the <i>tostring function</i> to return a <i>string</i> describing succinctly self.

The following attributes are stored with metamethods in the metatable, but have different purposes:

__obj	A unique private <i>reference</i> that characterizes objects.
__metatable	A <i>reference</i> to the metatable itself protecting against modifications.

6 Flags

The object model uses *flags* to qualify objects, like *class-object*, *final-object* and *readonly-object*. The difference with *boolean* attributes is that flags are *not* inherited nor copied. The flags of objects are managed by the methods `:get_flags`, `:set_flags`, `:clear_flags` and `:test_flags`. Methods like `:is_class`, `:is_final` and `:is_readonly` are roughly equivalent to call the method `:test_flags` with the corresponding (private) flag as argument. Note that functions from the `typeid` module that check for types or kinds, like `is_object` or `is_beam`, never rely on flags because types and kinds are not qualifiers.

From the technical point of view, flags are encoded into a 32-bit integer and the object model uses the protected bits 29-31, hence bits 0-28 are free of use. Object flags can be used and extended by other modules introducing their own flags, like the `element` module that relies on bits 0-4 and used by many commands. In practice, the bit index does not need to be known and should not be used directly but through its name to abstract its value.

7 Environments

The object model allows to transform an object into an environment; in other words, a global workspace for a given context, i.e. scope. Objects-as-environments are managed by the methods `open_env`, `close_env`, `load_env`, `dump_env` and `is_open_env`. Things defined in this workspace will be stored in the object, and accessible from outside using the standard ways to access object attributes:

```

local object in MAD
local one = 1
local obj = object { a:=one } -- obj with 'a' defined
-- local a = 1                -- see explication below

obj:open_env()                -- open environment
b = 2                         -- obj.b defined
c =\ -> a.."":"..b          -- obj.c defined
obj:close_env()              -- close environment

print(obj.a, obj.b, obj.c)    -- display: 1  2  1:2
one = 3
print(obj.a, obj.b, obj.c)    -- display: 3  2  3:2
obj.a = 4
print(obj.a, obj.b, obj.c)    -- display: 4  2  4:2

```

Uncommenting the line `local a = 1` would change the last displayed column to 1:2 for the three prints because the *lambda* defined for `obj.c` would capture the local `a` as it would exist in its scope. As seen hereabove, once the environment is closed, the object still holds the variables as attributes.

The MADX environment is an object that relies on this powerful feature to load MAD-X lattices, their settings and their "business logic", and provides functions, constants and elements to mimic the behavior of the global workspace of MAD-X to some extent:

```

MADX:open_env()
mq_k1 = 0.01                  -- mq.k1 is not a valid identifier!
MQ = QUADRUPOLE {l=1, k1:=MQ_K1} -- MADX environment is case insensitive
MADX:close_env()            -- but not the attributes of objects!
local mq in MADX
print(mq.k1)                -- display: 0.01
MADX.MQ_K1 = 0.02
print(mq.k1)                -- display: 0.02

```

Note that MAD-X workspace is case insensitive and everything is "global" (no scope, namespaces), hence the `quadrupole` element has to be directly available inside the MADX environment. Moreover, the MADX object adds the method `load` to extend `load_env` and ease the conversion of MAD-X lattices. For more details see chapter 8.

Chapter 4. Beams

The beam object is the *root object* of beams that store information relative to particles and particle beams. It also provides a simple interface to the particles and nuclei database.

The beam module extends the `typeid` module with the `is_beam` function, which returns `true` if its argument is a beam object, `false` otherwise.

1 Attributes

The beam *object* provides the following attributes:

particle	A <i>string</i> specifying the name of the particle. (default: "positron").
mass	A <i>number</i> specifying the energy-mass of the particle [GeV]. (default: <code>emass</code>).
charge	A <i>number</i> specifying the charge of the particle in [q] unit of <code>qelect</code> . ¹ (default: 1).
spin	A <i>number</i> specifying the spin of the particle. (default: 0).
emrad	A <i>lambda</i> returning the electromagnetic radius of the particle [m], $emrad = krad_GeV * charge^2 / mass$ where $krad_GeV = 10^{-9} (4\pi\epsilon_0)^{-1} q$.
aphot	A <i>lambda</i> returning the average number of photon emitted per bending unit, $aphot = kpht_GeV * charge^2 * betgam$ where $kpht_GeV = \frac{5}{2\sqrt{3}} krad_GeV (\hbar c)^{-1}$.
energy	A <i>number</i> specifying the particle energy [GeV]. (default: 1).
pc	A <i>lambda</i> returning the particle momentum times the speed of light [GeV], $pc = (energy^2 - mass^2)^{\frac{1}{2}}$.
beta	A <i>lambda</i> returning the particle relativistic $\beta = \frac{v}{c}$, $beta = (1 - (mass/energy)^2)^{\frac{1}{2}}$.
gamma	A <i>lambda</i> returning the particle Lorentz factor $\gamma = (1 - \beta^2)^{-\frac{1}{2}}$, $gamma = energy/mass$.
betgam	A <i>lambda</i> returning the product $\beta\gamma$, $betgam = (gamma^2 - 1)^{\frac{1}{2}}$.
pc2	A <i>lambda</i> returning pc^2 , avoiding the square root.
beta2	A <i>lambda</i> returning $beta^2$, avoiding the square root.
betgam2	A <i>lambda</i> returning $betgam^2$, avoiding the square root.
brho	A <i>lambda</i> returning the magnetic rigidity [T.m], $brho = GeV_c * pc / charge $ where $GeV_c = 10^9 / c$.
ex	A <i>number</i> specifying the horizontal emittance ϵ_x [m]. (default: 1).
ey	A <i>number</i> specifying the vertical emittance ϵ_y [m]. (default: 1).
et	A <i>number</i> specifying the longitudinal emittance ϵ_t [m]. (default: 1e-3).

¹The `qelect` value is defined in the `constant` module.

exn	A <i>lambda</i> returning the normalized horizontal emittance [m], exn = ex * betgam.
eyn	A <i>lambda</i> returning the normalized vertical emittance [m], eyn = ey * betgam.
etn	A <i>lambda</i> returning the normalized longitudinal emittance [m], etn = et * betgam.
nbunch	A <i>number</i> specifying the number of particle bunches in the machine. (default: 0).
npart	A <i>number</i> specifying the number of particles per bunch. (default: 0).
sigt	A <i>number</i> specifying the bunch length in $c\sigma_t$. (default: 1).
sige	A <i>number</i> specifying the relative energy spread in σ_E/E [GeV]. (default: 1e-3).

The beam *object* also implements a special protect-and-update mechanism for its attributes to ensure consistency and precedence between the physical quantities stored internally:

- The following attributes are *read-only*, i.e. writing to them triggers an error:
mass, charge, spin, emrad, aphot.
- The following attributes are *read-write*, i.e. hold values, with their accepted numerical ranges:
particle, energy > mass,
ex > 0, ey > 0, et > 0,
nbunch > 0, npart > 0, sigt > 0, sige > 0.
- The following attributes are *read-update*, i.e. setting these attributes update the energy, with their accepted numerical ranges:
pc > 0, 0.9 > beta > 0, gamma > 1, betgam > 0.1, brho > 0,
pc2, beta2, betgam2.
- The following attributes are *read-update*, i.e. setting these attributes update the emittances ex, ey, and et respectively, with their accepted numerical ranges:
exn > 0, eyn > 0, etn > 0.

2 Methods

The beam object provides the following methods:

- new_particle** A *method* (particle, mass, charge, [spin]) creating new particles or nuclei and store them in the particles database. The arguments specify in order the new particle's name, energy-mass [GeV], charge [q], and spin (default: 0). These arguments can also be grouped into a *table* with same attribute names as the argument names and passed as the solely argument.
- set_variables** A *method* (set) returning self with the attributes set to the pairs (*key*, *value*) contained in set. This method overrides the original one to implement the special protect-and-update mechanism, but the order of the updates is undefined. It also creates new particle on-the-fly if the mass and the charge are defined, and then select it. Shortcut setvar.
- showdb** A *method* ([file]) displaying the content of the particles database to file (default: io.stdout).

3 Metamethods

The beam object provides the following metamethods:

- __init** A *metamethod* () returning *self* after having processed the attributes with the special protect-and-update mechanism, where the order of the updates is undefined. It also creates new particle on-the-fly if the mass and the charge are defined, and then select it.
- __newindex** A *metamethod* (key, val) called by the assignment operator [key]=val to create new attributes for the pairs (key, value) or to update the underlying physical quantity of the beam objects.

The following attribute is stored with metamethods in the metatable, but has different purpose:

- __beam** A unique private *reference* that characterizes beams.

4 Particles database

The beam *object* manages the particles database, which is shared by all beam instances. The default set of supported particles is:

```
electron, positron, proton, antiproton, neutron, antineutron, ion, muon,
antimuon, deuteron, antideuteron, negmuon (=muon), posmuon (=antimuon).
```

New particles can be added to the database, either explicitly using the `new_particle` method, or by creating or updating a beam *object* and specifying all the attributes of a particle, i.e. particle's name, charge, mass, and (optional) spin:

```
local beam in MAD
local nmass, pmass, mumass in MAD.constant

-- create a new particle
beam:new_particle{ particle='mymuon', mass=mumass, charge=-1, spin=1/2 }

-- create a new beam and a new nucleus
local pbbeam = beam { particle='pb208', mass=82*pmass+126*nmass, charge=82 }
```

The particles database can be displayed with the `showdb` method at any time from any beam:

```
beam:showdb() -- check that both, mymuon and pb208 are in the database.
```

5 Particle charges

The physics of MAD-NG is aware of particle charges. To enable the compatibility with codes like MAD-X that ignores the particle charges, the global option `nocharge` can be used to control the behavior of created beams as shown by the following example:

```
local beam, option in MAD
local beam1 = beam { particle="electron" } -- beam with negative charge
print(beam1.charge, option.nocharge)      -- display: -1 false

option.nocharge = true                    -- disable particle charges
local beam2 = beam { particle="electron" } -- beam with negative charge
print(beam2.charge, option.nocharge)      -- display: 1 true
```

```
-- beam1 was created before nocharge activation...
print(beam1.charge, option.nocharge)      -- display: -1 true
```

This approach ensures consistency of beams behavior during their entire lifetime.²

6 Examples

The following code snippet creates the LHC lead beams made of bare nuclei $^{208}\text{Pb}^{82+}$:

```
local beam in MAD
local lhcb1, lhcb2 in MADX
local nmass, pmass, amass in MAD.constant
local pbmass = 82*pmass+126*nmass

-- attach a new beam with a new particle to lhcb1 and lhcb2.
lhcb1.beam = beam 'Pb208' { particle='pb208', mass=pbmass, charge=82 }
lhcb2.beam = lhcb1.beam -- let sequences share the same beam...

-- print Pb208 nuclei energy-mass in GeV and unified atomic mass.
print(lhcb1.beam.mass, lhcb1.beam.mass/amass)
```

²The option rbarc in MAD-X is too volatile and does not ensure such consistency...

Chapter 5. Elements

The element object is the *root object* of all elements used to model particle accelerators, including sequences and drifts. It provides most default values inherited by all elements.

The element module extends the `typeid` module with the `is_element` function, which returns `true` if its argument is an element object, `false` otherwise.

1 Taxonomy

The classes defined by the element module are organized according to the kinds and the roles of their instances. The classes defining the kinds are:

thin	The <i>thin</i> elements have zero-length and their physics does not depend on it, i.e. the attribute <code>l</code> is discarded or forced to zero in the physics.
thick	The <i>thick</i> elements have a length and their physics depends on it. Elements like <code>sband</code> , <code>rbend</code> , <code>quadrupole</code> , <code>solenoid</code> , and <code>elseparator</code> trigger a runtime error if they have zero-length. Other thick elements will accept to have zero-length for compatibility with MAD-X ¹ , but their physics will have to be adjusted. ²
drift	The <i>drift</i> elements have a length with a drift-like physics if $l \geq \text{minlen}$ ³ otherwise they are discarded or ignored. Any space between elements with a length $l \geq \text{minlen}$ are represented by an <i>implicit</i> drift created on need by the <code>s</code> -iterator of sequences and discarded afterward.
patch	The <i>patch</i> elements have zero-length and the purpose of their physics is to change the reference frame.
extrn	The <i>extern</i> elements are never part of sequences. If they are present in a sequence definition, they are expanded and replaced by their content, i.e. stay external to the lattice.
specl	The <i>special</i> elements have special roles like <i>marking</i> places (i.e. <code>maker</code>) or <i>branching</i> sequences (i.e. <code>slink</code>).

These classes are not supposed to be used directly, except for extending the hierarchy defined by the element module and schematically reproduced hereafter to help users understanding:

```
thin_element = element 'thin_element' { is_thin    = true }
thick_element = element 'thick_element' { is_thick   = true }
drift_element = element 'drift_element' { is_drift   = true }
patch_element = element 'patch_element' { is_patch   = true }
extrn_element = element 'extrn_element' { is_extern  = true }
specl_element = element 'specl_element' { is_special = true }

sequence      = extrn_element 'sequence'  { }
assembly     = extrn_element 'assembly'   { }
bline        = extrn_element 'bline'      { }
```

¹In MAD-X, zero-length sextupole and octupole are valid but may have surprising effects...

²E.g. zero-length sextupole must define their strength with `kn[3]` instead of `k2` to have the expected effect.

³By default `minlen` = 10^{-12} m.

```

marker      = specL_element 'marker'    { }
slink       = specL_element 'slink'     { }

drift       = drift_element 'drift'     { }
collimator  = drift_element 'collimator' { }
instrument   = drift_element 'instrument' { }
placeholder = drift_element 'placeholder' { }

sbend       = thick_element 'sbend'     { }
rbend       = thick_element 'rbend'     { }
quadrupole  = thick_element 'quadrupole' { }
sextupole   = thick_element 'sextupole' { }
octupole    = thick_element 'octupole'  { }
decapole    = thick_element 'decapole'  { }
dodecapole  = thick_element 'dodecapole' { }
solenoid    = thick_element 'solenoid'  { }
tkicker     = thick_element 'tkicker'   { }
wiggler     = thick_element 'wiggler'   { }
elseparator = thick_element 'elseparator' { }
rfcavity    = thick_element 'rfcavity'  { }
genmap      = thick_element 'genmap'    { }

beambeam    = thin_element  'beambeam'  { }
multipole   = thin_element  'multipole'  { }

xrotation   = patch_element 'xrotation' { }
yrotation   = patch_element 'yrotation' { }
srotation   = patch_element 'srotation' { }
translate   = patch_element 'translate'  { }
changeref   = patch_element 'changeref'  { }
changedir   = patch_element 'changedir'  { }
changenrj   = patch_element 'changenrj'  { }

-- specializations
rfmultipole = rfcavity      'rfmultipole' { }
crabcavity  = rfmultipole  'crabcavity'  { }

monitor     = instrument    'monitor'     { }
hmonitor    = monitor       'hmonitor'    { }
vmonitor    = monitor       'vmonitor'    { }

kicker      = tkicker       'kicker'      { }
hkicker     = kicker        'hkicker'     { }
vkicker     = kicker        'vkicker'     { }

```

All the classes above, including `element`, define the attributes `kind = name` and `is_name = true` where `name` correspond to the class name. These attributes help to identify the kind and the role of an element as shown in the following code excerpt:

```

local drift, hmonitor, sequence in MAD.element
local dft = drift    {}

```

```

local bpm = hmonitor {}
local seq = sequence {}
print(dft.kind)           -- display: drift
print(dft.is_drift)      -- display: true
print(dft.is_drift_element) -- display: true
print(bpm.kind)         -- display: hmonitor
print(bpm.is_hmonitor)  -- display: true
print(bpm.is_monitor)   -- display: true
print(bpm.is_instrument) -- display: true
print(bpm.is_drift_element) -- display: true
print(bpm.is_element)   -- display: true
print(bpm.is_drift)     -- display: true
print(bpm.is_thick_element) -- display: nil (not defined = false)
print(seq.kind)         -- display: sequence
print(seq.is_element)   -- display: true
print(seq.is_extrn_element) -- display: true
print(seq.is_thick_element) -- display: nil (not defined = false)

```

2 Attributes

The element *object* provides the following attributes:

- l** A *number* specifying the physical length of the element on the design orbit [m]. (default: 0).
- lrad** A *number* specifying the field length of the element on the design orbit considered by the radiation [m]. (default: `lrad = \s -> s.l`).
- angle** A *number* specifying the bending angle α of the element [rad]. A positive angle represents a bend to the right, i.e. a $-y$ -rotation towards negative x values. (default: 0).
- tilt** A *number* specifying the physical tilt of the element [rad]. All the physical quantities defined by the element are in the tilted frame, except `misalign` that comes first when tracking through an element, see the [track](#) command for details. (default: 0).
- model** A *string* specifying the integration model "DKD" or "TKT" to use when tracking through the element and overriding the command attribute, see the [track](#) command for details. (default: `cmd.model`).
- method** A *number* specifying the integration order 2, 4, 6, or 8 to use when tracking through the element and overriding the command attribute, see the [track](#) command for details. (default: `cmd.method`).
- nslice** A *number* specifying the number of slices or a *list* of increasing relative positions or a *callable* (`elm`, `mflw`, `lw`) returning one of the two previous kind of positions specification to use when tracking through the element and overriding the command attribute, see the [survey](#) or the [track](#) commands for details. (default: `cmd.nslice`).
- refpos** A *string* holding one of "entry", "centre" or "exit", or a *number* specifying a position in [m] from the start of the element, all of them resulting in an offset to subtract to the `at` attribute to find the s -position of the element entry when inserted in a sequence, see [elements positions](#) for details. (default: `nil` \equiv `seq.refer`).

- aperture** A *mappable* specifying aperture attributes, see [Aperture](#) for details. (default: {kind='circle', 1}).
- apertype** A *string* specifying the aperture type, see [Aperture](#) for details. (default: \s -> s.aperture.kind or 'circle').⁴
- misalign** A *mappable* specifying misalignment attributes, see [Misalignment](#) for details. (default: nil).

The `thick_element` *object* adds the following multipolar and fringe fields attributes:

- kn1, ks1** A *list* specifying respectively the **multipolar** and skew integrated strengths of the element [m^{-i+1}]. (default: {}).
- dkn1, dks1** A *list* specifying respectively the multipolar and skew integrated strengths errors of the element [m^{-i+1}]. (default: {}).
- e1, e2** A *number* specifying respectively the horizontal angle of the pole faces at entry and exit of the element [rad]. A positive angle goes toward inside the element, see [Figures 5.1](#) and [5.2](#). (default: 0).
- h1, h2** A *number* specifying respectively the horizontal curvature of the pole faces at entry and exit of the element [m^{-1}]. A positive curvature goes toward inside the element. (default: 0).
- hgap** A *number* specifying half of the vertical gap at the center of the pole faces of the element [m]. (default: 0).
- fint** A *number* specifying the fringe field integral at entrance of the element. (default: 0).
- fintx** A *number* specifying the fringe field integral at exit of the element. (default: fint).
- fringe** A *number* specifying the bitmask to activate fringe fields of the element, see [Flags](#) for details. (default: 0).
- fringemax** A *number* specifying the maximum order for multipolar fringe fields of the element. (default: 2).
- kill_ent_fringe** A *logical* specifying to kill the entry fringe fields of the element. (default: false).
- kill_exi_fringe** A *logical* specifying to kill the exit fringe fields of the element. (default: false).
- f1, f2** A *number* specifying quadrupolar fringe field first and second parameter of SAD. (default: 0).

3 Methods

The `element` *object* provides the following methods:

- select** A *method* ([flg]) to select the element for the flags flg (default: selected).
- deselect** A *method* ([flg]) to deselect the element for the flags flg (default: selected).
- is_selected** A *method* ([flg]) to test the element for the flags flg (default: selected).

⁴This attribute was introduced to ease the translation of MAD-X sequences and may disappear in some future.

- is_disabled** A *method* () to test if the element is *disabled*, which is equivalent to call the method `is_selected(disabled)`.
- is_observed** A *method* () to test if the element is *observed*, which is equivalent to call the method `is_selected(observed)`.
- is_implicit** A *method* () to test if the element is *implicit*, which is equivalent to call the method `is_selected(implicit)`.

The `drift_element` and `thick_element` objects provide the following extra methods, see [sub-elements](#) for details about the `sat` attribute:

- index_sat** A *method* (`sat`, [`cmp`]) returning the lowest index `idx` (starting from 1) of the first sub-element with a relative position that compares `true` with the *number* `sat` using the optional *callable* `cmp(sat, self[idx].sat)` (default: "`=="`"), or `#self+1`. In the presence of multiple equal positions, "`<="`" (resp. "`>="`") will return the lowest index of the position while "`<`" (resp. "`>`") the lowest index next to the position for ascending (resp. descending) order.
- insert_sat** A *method* (`elm`, [`cmp`]) returning the element after inserting the zero-length sub-element `elm` at the index determined by `:index_sat(elm.sat, [cmp])` using the optional *callable* `cmp` (default: "`<`").
- replace_sat** A *method* (`elm`) returning the replaced sub-element found at the index determined by `:index_sat(elm.sat)` by the new zero-length sub-element `elm`, or `nil`.
- remove_sat** A *method* (`sat`) returning the removed sub-element found at the index determined by `:index_sat(sat)`, or `nil`.

4 Metamethods

The `element` object provides the following metamethods:

- __len** A *metamethod* () overloading the length operator `#` to return the number of subelements in the *list* part of the element.
- __add** A *metamethod* (`obj`) overloading the binary operator `+` to build a `bline` object from the juxtaposition of two elements.
- __mul** A *metamethod* (`n`) overloading the binary operator `*` to build a `bline` object from the repetition of an element `n` times, i.e. one of the two operands must be a *number*.
- __unm** A *metamethod* (`n`) overloading the unary operator `-` to build a `bline` object from the turning of an element, i.e. reflect the element.
- __tostring** A *metamethod* () returning a *string* built from the element information, e.g. `print(monitor 'bpm' {})` display the *string* `":monitor: 'bpm' memory-address"`.

The operators overloading of elements allows to unify sequence and beamline definitions in a consistent and simple way, noting that sequence and `bline` are (external) elements too.

The following attribute is stored with metamethods in the metatable, but has different purpose:

- __elem** A unique private *reference* that characterizes elements.

5 Elements

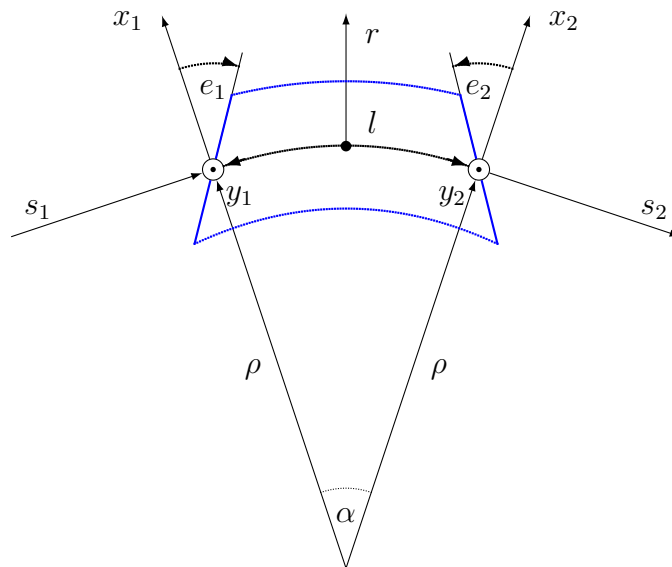
Some elements define new attributes or override the default values provided by the *root object* element. The following subsections describe the elements supported by MAD-NG.

5.1 SBend

The sbend element is a sector bending magnet with a curved reference system as shown in Figure 5.1, and defines or overrides the following attributes:

- k0** A number specifying the dipolar strength of the element [m^{-1}].
(default: $k_0 = \backslash s \rightarrow s.\text{angle}/s.l$).^{5,6}
- k0s** A number specifying the dipolar skew strength of the element [m^{-1}]. (default: 0).
- k1, k1s** A number specifying respectively the quadrupolar and skew strengths of the element [m^{-2}]. (default: 0).
- k2, k2s** A number specifying respectively the sextupolar and skew strengths of the element [m^{-3}]. (default: 0).
- fringe** Set to flag `fringe.bend` to activate the fringe fields by default, see [Flags](#) for details.

Figure 5.1: Reference system for a sector bending magnet.



5.2 RBend

The rbend element is a rectangular bending magnet with a straight reference system as shown in Figure 5.2, and defines or overrides the following attributes:

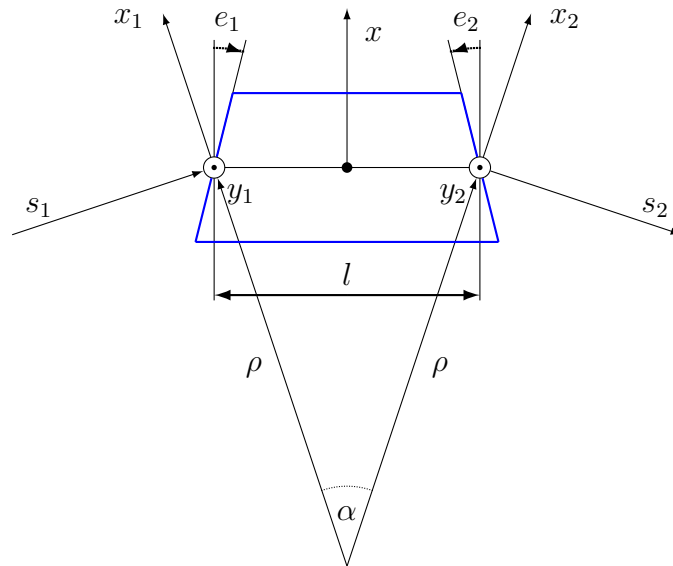
- k0** A number specifying the dipolar strength of the element [m^{-1}].
(default: $k_0 = \backslash s \rightarrow s.\text{angle}/s.l$).^{5,6}

⁵By default bending magnets are ideal bends, that is $\text{angle} = k_0 * l$.

⁶For compatibility with MAD-X.

- k0s** A number specifying the dipolar skew strength of the element [m^{-1}]. (default: 0).
- k1, k1s** A number specifying respectively the quadrupolar and skew strengths of the element [m^{-2}]. (default: 0).
- k2, k2s** A number specifying respectively the sextupolar and skew strengths of the element [m^{-3}]. (default: 0).
- fringe** Set to flag `fringe.bend` to activate the fringe fields by default, see [Flags](#) for details.
- true_r bend** A logical specifying if this `rbend` element behaves like (`false`) a `sbend` element with parallel pole faces, i.e. $e_1 = e_2 = \alpha/2$ in Figure 5.1, or like (`true`) a rectangular bending magnet with a straight reference system as shown in Figure 5.2. (default: `false`).⁶

Figure 5.2: Reference system for a rectangular bending magnet.



5.3 Quadrupole

The quadrupole element is a straight focusing element and defines the following attributes:

- k0, k0s** A number specifying respectively the dipolar and skew strengths of the element [m^{-1}]. (default: 0).
- k1, k1s** A number specifying respectively the quadrupolar and skew strengths of the element [m^{-2}]. (default: 0).
- k2, k2s** A number specifying respectively the sextupolar and skew strengths of the element [m^{-3}]. (default: 0).

5.4 Sextupole

The sextupole element is a straight element and defines the following attributes:

- k2, k2s** A number specifying respectively the sextupolar and skew strengths of the element [m^{-3}]. (default: 0).

5.5 Octupole

The octupole element is a straight element and defines the following attributes:

k3, k3s A *number* specifying respectively the octupolar and skew strengths of the element [m^{-4}]. (default: 0).

5.6 Decapole

The decapole element is a straight element and defines the following attributes:

k4, k4s A *number* specifying respectively the decapolar and skew strength of the element [m^{-5}]. (default: 0).

5.7 Dodecapole

The dodecapole element is a straight element and defines the following attributes:

k5, k5s A *number* specifying respectively the dodecapolar and skew strength of the element [m^{-6}]. (default: 0).

5.8 Solenoid

The solenoid element defines the following attributes:

ks, ksi A *number* specifying respectively the strength [rad/m] and the integrated strength [rad] of the element. A positive value points toward positive s . (default: 0).

5.9 Multipole

The multipole element is a thin element and defines the following attributes:

kn l , ks l A *list* specifying respectively the multipolar and skew integrated strengths of the element [m^{-i+1}]. (default: {}).

dkn l , dks l A *list* specifying respectively the multipolar and skew integrated strengths errors of the element [m^{-i+1}]. (default: {}).

5.10 TKicker

The tkicker element is the *root object* of kickers and defines or overrides the following attributes:

hkick A *number* specifying the horizontal strength of the element [m^{-1}]. By convention, a kicker with a positive horizontal strength kicks in the direction of the reference orbit, e.g. $\text{hkick} \equiv -\text{kn}l[1]$. (default: 0).

vkick A *number* specifying the vertical strength of the element [m^{-1}]. By convention, a kicker with a positive vertical strength kicks toward the reference orbit, e.g. $\text{vkick} \equiv \text{ks}l[1]$. (default: 0).

method Set to 2 if `ptcmodel` is not set to enforce pure momentum kick and avoid dipolar strength integration that would introduce dispersion.

5.11 Kicker, HKicker, VKicker

The `kicker` element inheriting from the `tkicker` element, is the *root object* of kickers involved in the orbit correction and defines the following attributes:

chkick, cvkick A *number* specifying respectively the horizontal and vertical correction strength of the element set by the `correct` command [m^{-1}]. (default: 0).

The `hkicker` (horizontal kicker) and `vkicker` (vertical kicker) elements define the following attribute:

kick A *number* specifying the strength of the element in its main direction [m^{-1}]. (default: 0).

5.12 Monitor, HMonitor, VMonitor

The `monitor` element is the root object of monitors involved in the orbit correction and defines the following attributes:

mredx, mredy A *number* specifying respectively the readout x,y -offset error of the element [m]. The offset is added to the beam position during orbit correction (after scaling). (default: 0).

mresx, mresy A *number* specifying respectively the readout x,y -scaling error of the element. The scale factor multiplies the beam position by $1+mres$ (before offset) during orbit correction.⁷ (default: 0).

The `hmonitor` (horizontal monitor) and `vmonitor` (vertical monitor) elements are specialisations inheriting from the `monitor` element.

5.13 RFCavity

The `rfcavity` element defines the following attributes:

volt A *number* specifying the peak RF voltage of the element [MV]. (default: 0).

freq A *number* specifying a non-zero RF frequency of the element [MHz]. (default: 0).

lag A *number* specifying the RF phase lag of the element in unit of 2π . (default: 0).

harmon A *number* specifying the harmonic number of the element if `freq` is zero. (default: 0).

n_bessel A *number* specifying the transverse focussing effects order of the element. (default: 0).

totalpath A *logical* specifying if the `totalpath` must be used in the element. (default: `true`).

5.14 RFMultipole

The `rfmultipole` element defines the following attributes:

pnl, psl A *list* specifying respectively the multipolar and skew phases of the element [rad]. (default: {}).

d pnl, d psl A *list* specifying respectively the multipolar and skew phases errors of the element [rad]. (default: {}).

⁷This definition comes from MAD-X default zeroed values such that undefined attribute gives a scale of 1.

5.15 ElSeparator

The `elseparator` element defines the following attributes:

- ex, ey** A *number* specifying respectively the electric field x,y -strength of the element [MV/m]. (default: 0).
- exl, eyl** A *number* specifying respectively the integrated electric field x,y -strength of the element [MV]. (default: 0).

5.16 Wiggler

The `wiggler` element defines the following attributes: NYI, TBD

5.17 BeamBeam

The `beambeam` element defines the following attributes: NYI, TBD

5.18 GenMap

The `genmap` element defines the following attributes:⁸

- damap** A *damap* used for thick integration.
- update** A *callable* (`elm`, `mflw`, `lw`) invoked before each step of thick integration to update the *damap*. (default: `nil`)
- nslice** A *number* specifying the number of slices or a *list* of increasing relative positions or a *callable* (`elm`, `mflw`, `lw`) returning one of the two previous kind of positions specification to use when tracking through the element and overriding the command attribute, see the [survey](#) or the [track](#) commands for details. (default: 1).

5.19 SLink

The `slink` element defines the following attributes:⁹

- sequence** A *sequence* to switch to right after exiting the element. (default: `nil`)
- range** A *range* specifying the span over the sequence to switch to, as expected by the sequence method `:siter`. (default: `nil`).
- nturn** A *number* specifying the number of turn to track the sequence to switch to, as expected by the sequence method `:siter`. (default: `nil`).
- dir** A *number* specifying the s -direction of the tracking of the sequence to switch to, as expected by the sequence method `:siter`. (default: `nil`).
- update** A *callable* (`elm`, `mflw`) invoked before retrieving the other attributes when entering the element. (default: `nil`)

⁸This element is a generalization of the `matrix` element of MAD-X, to use with care!

⁹This element allows to switch between sequences during tracking, kind of `if-then-else` for tracking.

5.20 Translate

The `translate` element is a patch element and defines the following attributes:

dx, dy, ds A *number* specifying respectively x,y,s -translation of the reference frame [m]. (default: 0)

5.21 XRotation, YRotation, SRotation

The `xrotation` (rotation around x -axis), `yrotation` (rotation around y -axis) and `srotation` (rotation around s -axis) elements are patches element and define the following attribute:

angle A *number* specifying the rotation angle around the axis of the element [rad]. (default: 0).

5.22 ChangeRef

The `changeref` element is a patch element and defines the following attributes:

dx, dy, ds A *number* specifying respectively x,y,s -translation of the reference frame [m]. (default: 0)

dtheta, dphi, dps A *number* specifying respectively $y,-x,s$ -rotation of the reference frame applied in this order after any translation [rad]. (default: 0)

5.23 ChangeNrg

The `changenrg` element is a patch element and defines the following attributes:

dnrj A *number* specifying the change by δ_E of the reference beam energy [GeV]. The special *string* value "pt" asks to reset the variable p_t to zero and move the change as a δ_E to the particle's owner beam, cloning the reference beam if needed. (default: 0)

6 Flags

The `element` module exposes the following *object* flags through `MAD.element.flags` to use in conjunction with the methods `select` and `deselect`:¹⁰

none All bits zero.

selected Set if the element has been selected.

disabled Set if the element has been disabled, e.g. for orbit correction.

observed Set if the element has been selected for observation, e.g. for output to TFS table. The `$end` markers are selected for observation by default, and commands with the `observe` attribute set to 0 discard this flag and consider all elements as selected for observation.

implicit Set if the element is implicit, like the temporary *implicit* drifts created on-the-fly by the sequence `s-iterator` with indexes at half integers. This flag is used by commands with the `implicit` attribute.

layout Set if the element `angle` must be used by layout plot. This flag is useful to plot multiple sequence layouts around interaction points, like `lhcb1` and `lhcb2` around `IP1` and `IP5`.

¹⁰Remember that flags are *not* inherited nor copied as they are qualifying the object itself.

7 Fringe fields

The `element` module exposes the following flags through `MAD.element.flags.fringe` to *control* the elements fringe fields through their attribute `fringe`, or to *restrict* the activated fringe fields with the commands attribute `fringe`:¹¹

none	All bits zero.
bend	Control the element fringe fields for bending fields.
mult	Control the element fringe fields for multipolar fields up to <code>fringemax</code> order.
rfcav	Control the element fringe fields for rfcavity fields.
qsad	Control the element fringe fields for multipolar fields with extra terms for quadrupolar fields for compatibility with SAD.
comb	Control the element fringe fields for combined bending and multipolar fields.
combqs	Control the element fringe fields for combined bending and multipolar fields with extra terms for quadrupolar fields for compatibility with SAD.

The *element* `thick_element` provides a dozen of attributes to parametrize the aforementioned fringe fields. Note that in some future, part of these attributes may be grouped into a *mappable* to ensure a better consistency of their parametrization.

8 Sub-elements

An element can have sub-elements stored in its *list* part, hence the length operator `#` returns the number of them, and so far sub-elements must have zero-length.¹² The attribute `sat` of sub-elements, i.e. `read sub-at`, is interpreted as their relative position inside the enclosing main element, that is a fractional of the length of the latter. Therefore, the positions of the sub-elements will vary proportionally to the length of their main element. The sub-elements are only considered and valid in the `drift_element` and `thick_element` kinds that have the methods `:index_sat`, `:insert_sat`, `:remove_sat`, and `:replace_sat` to manage sub-elements from their `sat` attribute. The sequence method `:install` updates the `sat` attribute of the elements installed as sub-elements if the *logical* `elements.subelem` is enabled, i.e. when the *s*-position determined by the `at`, `from` and `refpos` attributes falls inside an element already installed in the sequence and not inside an *implicit* drift.

9 Aperture

All the apertures are *mappable* defined by the following attributes in the tilted frame of an element, see the `track` command for details:

kind	A <i>string</i> specifying the aperture shape. (no default).
tilt	A <i>number</i> specifying the tilt angle of the aperture [rad]. (default: 0).
xoff, yoff	A <i>number</i> specifying the transverse <i>x, y</i> -offset of the aperture [m]. (default: 0).
maper	A <i>mappable</i> specifying a larger aperture encompassing a <code>polygon</code> aperture to use before checking the polygon itself to speed up the test. The attributes <code>tilt</code> , <code>xoff</code> and <code>yoff</code> are ignored and superseded by the ones of the <code>polygon</code> aperture. (default: <code>nil</code>).

¹¹Those flags are *not* object flags, but fringe fields flags.

¹²It is possible that `thick_element` will be allowed in some future.

The supported aperture shapes are listed hereafter. The parameters defining the shapes are expected to be in the *list* part of the apertures and defines the top-right sector shape, except for the polygon:

square	A square shape with one parameter defining the side half-length. It is the default aperture check with limits set to 1.
rectangle	A rectangular shape with two parameters defining the x,y -half lengths (default: 1 [m]).
circle	A circular shape with one parameter defining the radius.
ellipse	An elliptical shape with two parameters defining the x,y -radii. (default: 1 [m]).
rectcircle	A rectangular shape intersected with a circular shape with three parameters defining the x,y -half lengths and the radius. (default: 1 [m]).
rectellipse	A rectangular shape intersected with an elliptical shape with four parameters defining the x,y -half lengths and the x,y -radii.
racetrack	A rectangular shape with corners rounded by an elliptical shape with four parameters defining the x,y -half lengths and the corners x,y -radii.
octagon	A rectangular shape with corners truncated by a triangular shape with four parameters defining the x,y -half lengths and the triangle x,y -side lengths. An octagon can model hexagon or diamond shapes by equating the triangle lengths to the rectangle half-lengths.
polygon	A polygonal shape defined by two vectors v_x and v_y holding the vertices coordinates. The polygon does not need to be convex, simple or closed, but in the latter case it will be closed automatically by joining the first and the last vertices.
bbox	A 6D bounding box with six parameters defining the upper limits of the absolute values of the six coordinates.

The following example defines new classes with three different aperture definitions:

```

local quadrupole in MAD.element
local mq = quadrupole 'mq' { l=1,                                -- new class
  aperture = { kind='racetrack',
               tilt=pi/2, xoff=1e-3, yoff=5e-4,                -- attributes
               0.06,0.06,0.01,0.01 }                          -- parameters
}
local mqdiam = quadrupole 'mqdiam' { l=1,                       -- new class
  aperture = { kind='octagon', xoff=1e-3, yoff=1e-3,           -- attributes
               0.06,0.04,0.06,0.04 }                          -- parameters
}
local mqpoly = quadrupole 'mqpoly' { l=1,                       -- new class
  aperture = { kind='polygon', tilt=pi/2, xoff=1e-3, yoff=1e-3, -- attributes
               vx=vector{0.05, ...}, vy=vector{0, ...},      -- parameters
               aper={kind='square', 0.05}                      -- 2nd aperture
}

```

10 Misalignment

The misalignments are *mappable* defined at the entry of an element by the following attributes, see the [track](#) command for details:

- dx, dy, ds** A number specifying the x,y,s -displacement at the element entry [m], see Figures 5.3 and 5.4. (default: 0).
- dtheta** A number specifying the y -rotation angle (azimuthal) at the element entry [rad], see Figure 5.3. (default: 0).
- dphi** A number specifying the $-x$ -rotation angle (elevation) at the entry of the element [rad], see Figure 5.4. (default: 0).
- dpsi** A number specifying the s -rotation angle (roll) at the element entry [rad], see Figure 5.5. (default: 0).

Two kinds of misalignments are available for an element and summed beforehand:

- The *absolute* misalignments of the element versus its local reference frame, and specified by its `misalign` attribute. These misalignments are always considered.
- The *relative* misalignments of the element versus a given sequence, and specified by the `method:misalign` of sequence. These misalignments can be considered or not depending of command settings.

Figure 5.3: Displacements in the (x, s) plane.

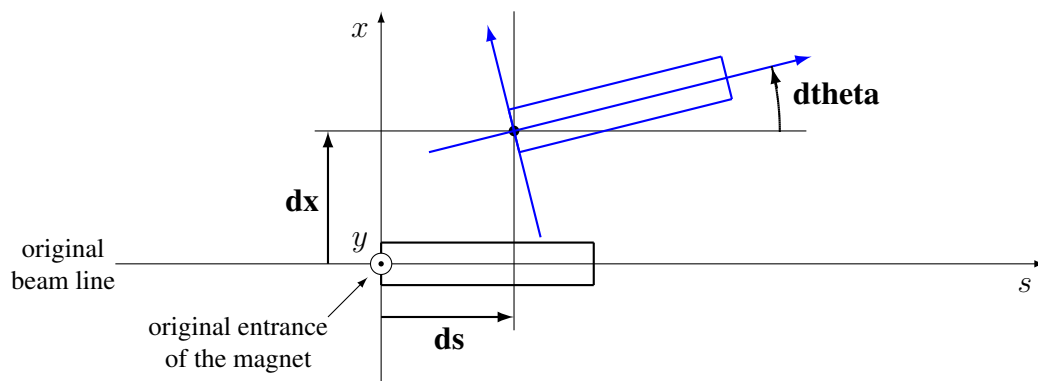


Figure 5.4: Displacements in the (y, s) plane.

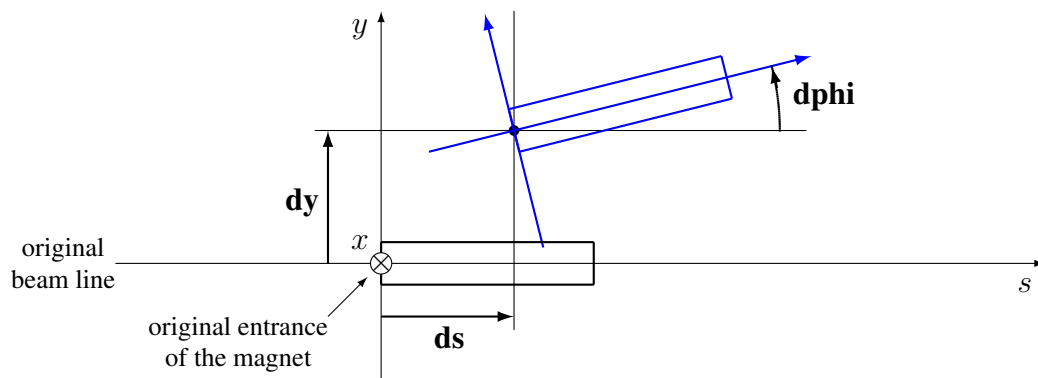
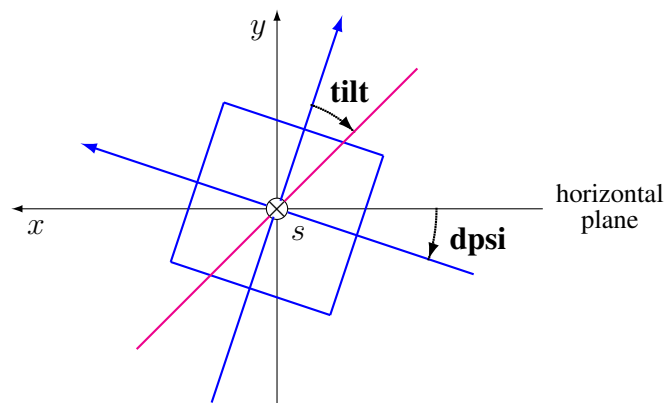


Figure 5.5: Displacements in the (x, y) plane.



Chapter 6. Sequence

The MAD Sequences are objects convenient to describe accelerators lattices built from a *list* of elements with increasing *s*-positions. The sequences are also containers that provide fast access to their elements by referring to their indexes, *s*-positions, or (mangled) names, or by running iterators constrained with ranges and predicates.

The sequence object is the *root object* of sequences that store information relative to lattices.

The sequence module extends the `typeid` module with the `is_sequence` function, which returns `true` if its argument is a sequence object, `false` otherwise.

1 Attributes

The sequence object provides the following attributes:

l	A <i>number</i> specifying the length of the sequence [m]. A <code>nil</code> will be replaced by the computed lattice length. A value greater or equal to the computed lattice length will be used to place the <code>\$end</code> marker. Other values will raise an error. (default: <code>nil</code>).
dir	A <i>number</i> holding one of 1 (forward) or -1 (backward) and specifying the direction of the sequence. ¹ (default: 1)
refer	A <i>string</i> holding one of "entry", "centre" or "exit" to specify the default reference position in the elements to use for their placement. An element can override it with its <code>refpos</code> attribute, see element positions for details. (default: <code>nil</code> \equiv "centre").
owner	A <i>logical</i> specifying if an <i>empty</i> sequence is a view with no data (<code>owner \neq true</code>), or a sequence holding data (<code>owner $=$ true</code>). (default: <code>nil</code>)
minlen	A <i>number</i> specifying the minimal length [m] to generate <i>implicit</i> drifts between elements in <i>s</i> -iterators generated by the method <code>:siter</code> . This attribute is automatically set to 10^{-6} m when a sequence is created within the MADX environment. (default: <code>nil</code>)
beam	An attached beam. (default: <code>nil</code>)

Warning: the following private and read-only attributes are present in all sequences and should *never be used, set or changed*; breaking this rule would lead to an *undefined behavior*:

__dat	A <i>table</i> containing all the private data of sequences.
__cycle	A <i>reference</i> to the element registered with the <code>:cycle</code> method. (default: <code>nil</code>)

2 Methods

The sequence object provides the following methods:

elem	A <i>method</i> (<code>idx</code>) returning the element stored at the positive index <code>idx</code> in the sequence, or <code>nil</code> .
spos	A <i>method</i> (<code>idx</code>) returning the <i>s</i> -position at the entry of the element stored at the positive index <code>idx</code> in the sequence, or <code>nil</code> .

¹This is equivalent to the MAD-X `bv` flag.

- upos** A *method* (*idx*) returning the *s*-position at the user-defined *refpos* offset of the element stored at the positive index *idx* in the sequence, or *nil*.
- ds** A *method* (*idx*) returning the length of the element stored at the positive index *idx* in the sequence, or *nil*.
- align** A *method* (*idx*) returning a *set* specifying the misalignment of the element stored at the positive index *idx* in the sequence, or *nil*.
- index** A *method* (*idx*) returning a positive index, or *nil*. If *idx* is negative, it is reflected versus the size of the sequence, e.g. `-1` becomes `#self`, the index of the `$end` marker.
- name_of** A *method* (*idx*, [*ref*]) returning a *string* corresponding to the (mangled) name of the element at the index *idx* or *nil*. An element name appearing more than once in the sequence will be mangled with an absolute count, e.g. `mq[3]`, or a relative count versus the optional reference element *ref* determined by `:index_of`, e.g. `mq{-2}`.
- index_of** A *method* (*a*, [*ref*], [*dir*]) returning a *number* corresponding to the positive index of the element determined by the first argument or *nil*. If *a* is a *number* (or a *string* representing a *number*), it is interpreted as the *s*-position of an element and returned as a second *number*. If *a* is a *string*, it is interpreted as the (mangled) name of an element as returned by `:name_of`. Finally, *a* can be a *reference* to an element to search for. The argument *ref* (default: *nil*) specifies the reference element determined by `:index_of(ref)` to use for relative *s*-positions, for decoding mangled names with relative counts, or as the element to start searching from. The argument *dir* (default: 1) specifies the direction of the search with values 1 (forward), -1 (backward), or 0 (no direction). The `dir=0` case may return an index at half-integer if *a* is interpreted as an *s*-position pointing to an *implicit drift*.
- range_of** A *method* ([*rng*], [*ref*], [*dir*]) returning three *numbers* corresponding to the positive indexes *start* and *end* of the range and its direction *dir*, or *nil* for an empty range. If *rng* is omitted, it returns `1, #self, 1`, or `#self, 1, -1` if *dir* is negative. If *rng* is a *number* or a *string* with no `'/'` separator, it is interpreted as both *start* and *end* and determined by `index_of`. If *rng* is a *string* containing the separator `'/'`, it is split in two *strings* interpreted as *start* and *end*, both determined by `:index_of`. If *rng* is a *list*, it will be interpreted as { *start*, *end*, [*ref*], [*dir*] }, both determined by `:index_of`, unless *ref* equals `'idx'` then both are determined by `:index` (i.e. a *number* is interpreted as an index instead of a *s*-position). The arguments *ref* (default: *nil*) and *dir* (default: 1) are forwarded to all invocations of `:index_of` with a higher precedence than ones in the *list* *rng*, and a runtime error is raised if the method returns *nil*, i.e. to disambiguate between a valid empty range and an invalid range.
- length_of** A *method* ([*rng*], [*ntrn*], [*dir*]) returning a *number* specifying the length of the range optionally including *ntrn* extra turns (default: 0), and calculated from the indexes returned by `:range_of([rng], nil, [dir])`.
- iter** A *method* ([*rng*], [*ntrn*], [*dir*]) returning an iterator over the sequence elements. The optional range is determined by `:range_of(rng, [dir])`, optionally including *ntrn* turns (default: 0). The optional direction *dir* specifies the forward 1 or the backward -1 direction of the iterator. If *rng* is not provided and the mtable is cycled, the *start* and *end* indexes are determined by `:index_of(self.__cycle)`. When used with a generic **for** loop, the iterator returns at each element: its index, the element itself, its *s*-position over the running loop and its signed length depending on the direction.

- siter** A *method* ([rng], [ntrn], [dir]) returning an *s*-iterator over the sequence elements. The optional range is determined by `:range_of([rng], nil, [dir])`, optionally including `ntrn` turns (default: 0). The optional direction `dir` specifies the forward 1 or the backward -1 direction of the iterator. When used with a generic **for** loop, the iterator returns at each iteration: its index, the element itself or an *implicit* drift, its *s*-position over the running loop and its signed length depending on the direction. Each *implicit* drift is built on-the-fly by the iterator with a length equal to the gap between the elements surrounding it and a half-integer index equal to the average of their indexes. The length of *implicit* drifts is bounded by the maximum between the sequence attribute `minlen` and the `minlen` from the [constant](#) module.
- foreach** A *method* (act, [rng], [sel], [not]) returning the sequence itself after applying the action `act` on the selected elements. If `act` is a *set* representing the arguments in the packed form, the missing arguments will be extracted from the attributes `action`, `range`, `select` and `default`. The action `act` must be a *callable* (`elm, idx, [midx]`) applied to an element passed as first argument and its index as second argument, the optional third argument being the index of the main element in case `elm` is a sub-element. The optional range is used to generate the loop iterator `:iter([rng])`. The optional selector `sel` is a *callable* (`elm, idx, [midx]`) predicate selecting eligible elements for the action using the same arguments. The selector `sel` can be specified in other ways, see [element selections](#) for details. The optional *logical* `not` (default: `false`) indicates how to interpret default selection, as *all* or *none*, depending on the semantic of the action.²
- select** A *method* ([flg], [rng], [sel], [not]) returning the sequence itself after applying the action `:select([flg])` to the elements using `:foreach(act, [rng], [sel], [not])`. By default sequence have all their elements deselected with only the `$end` marker observed.
- deselect** A *method* ([flg], [rng], [sel], [not]) returning the sequence itself after applying the action `:deselect([flg])` to the elements using `:foreach(act, [rng], [sel], [not])`. By default sequence have all their elements deselected with only the `$end` marker observed.
- filter** A *method* ([rng], [sel], [not]) returning a *list* containing the positive indexes of the elements determined by `:foreach(filt_act, [rng], [sel], [not])`, and its size. The *logical* `sel.subelem` specifies to select sub-elements too, and the *list* may contain non-integer indexes encoding their main element index added to their relative position, i.e. `midx.sat`. The builtin *function* `math.modf(num)` allows to retrieve easily the main element `midx` and the sub-element `sat`, e.g. `midx,sat = math.modf(val)`.
- install** A *method* (elm, [rng], [sel], [cmp]) returning the sequence itself after installing the elements in the *list* `elm` at their [element positions](#); unless `from="selected"` is defined meaning multiple installations at positions relative to each element determined by the method `:filter([rng], [sel], true)`. The *logical* `sel.subelem` is ignored. If the arguments are passed in the packed form, the extra attribute `elements` will be used as a replacement for the argument `elm`. The *logical* `elm.subelem` specifies to install elements with zero-length and computed *s*-position falling inside sequence elements as sub-elements, and set their `sat` attribute accordingly. The optional *callable* `cmp(elm spos, spos[idx])` (default: `"<"`) is used to search for the *s*-position of the installation, where equal *s*-position are installed after (i.e. before with `"<="`), see `bsearch` from the [utility](#) module for details. The *implicit* drifts are checked after each element installation.

²For example, the `:remove` method needs `not=true` to *not* remove all elements if no selector is provided.

- replace** A *method* (*elm*, [*rng*], [*sel*]) returning the *list* of replaced elements by the elements in the *list* *elm* placed at their [element positions](#), and the *list* of their respective indexes, both determined by `:filter([rng], [sel], true)`. The *list* *elm* cannot contain instances of sequence or *bline* elements and will be recycled as many times as needed to replace all selected elements. If the arguments are passed in the packed form, the extra attribute elements will be used as a replacement for the argument *elm*. The *logical* `sel.subelem` specifies to replace selected sub-elements too and set their `sat` attribute to the same value. The *implicit* drifts are checked only once all elements have been replaced.
- remove** A *method* ([*rng*], [*sel*]) returning the *list* of removed elements and the *list* of their respective indexes, both determined by `:filter([rng], [sel], true)`. The *logical* `sel.subelem` specifies to remove selected sub-elements too.
- move** A *method* ([*rng*], [*sel*]) returning the sequence itself after updating the [element positions](#) at the indexes determined by `:filter([rng], [sel], true)`. The *logical* `sel.subelem` is ignored. The elements must keep their order in the sequence and surrounding *implicit* drifts are checked only once all elements have been moved.³
- misalign** A *method* (*algn*, [*rng*], [*sel*]) returning the sequence itself after setting the [element misalignments](#) from *algn* at the indexes determined by `:filter([rng], [sel], true)`. If *algn* is a *mappable*, it will be used to misalign the filtered elements. If *algn* is a *iterable*, it will be accessed using the filtered elements indexes to retrieve their specific misalignment. If *algn* is a *callable* (*idx*), it will be invoked for each filtered element with their index as solely argument to retrieve their specific misalignment.
- reflect** A *method* ([*name*]) returning a new sequence from the sequence reversed, and named from the optional *string* *name* (default: `self.name..'_'_rev'`).
- cycle** A *method* (*a*) returning the sequence itself after checking that *a* is a valid reference using `:index_of(a)`, and storing it in the `__cycle` attribute, itself erased by the methods editing the sequence like `:install`, `:replace`, `:remove`, `:share`, and `:unique`.
- share** A *method* (*seq2*) returning the *list* of elements removed from the *seq2* and the *list* of their respective indexes, and replaced by the elements from the sequence with the same name when they are unique in both sequences.
- unique** A *method* ([*fmt*]) returning the sequence itself after replacing all non-unique elements by new instances sharing the same parents. The optional *fmt* must be a *callable* (*name*, *cnt*, *idx*) that returns the mangled name of the new instance build from the element name, its count *cnt* and its index *idx* in the sequence. If the optional *fmt* is a *string*, the mangling *callable* is built by binding *fmt* as first argument to the function `string.format` from the standard library, see [Lua 5.2 §6.4](#) for details.
- publish** A *method* (*env*, [*keep*]) returning the sequence after publishing all its elements in the environment *env*. If the *logical* *keep* is `true`, the method will preserve existing elements from being overridden. This method is automatically invoked with `keep=true` when sequences are created within the MADX environment.
- copy** A *method* ([*name*], [*owner*]) returning a new sequence from a copy of `self`, with the optional *name* and the optional attribute *owner* set. If the sequence is a view, so will be the copy unless `owner == true`.

³Updating directly the positions attributes of an element has no effect.

- is_view** A *method* () returning `true` if the sequence is a view over another sequence data, `false` otherwise.
- set_readonly** Set the sequence as read-only, including its columns.
- save_flags** A *method* ([*flgs*]) saving the flags of all the elements to the optional *iterable* *flgs* (default: {}) and return it.
- restore_flags** A *method* (*flgs*) restoring the flags of all the elements from the *iterable* *flgs*. The indexes of the flags must match the indexes of the elements in the sequence.
- dumpseq** A *method* ([*fil*], [*info*]) displaying on the optional file *fil* (default: `io.stdout`) information related to the position and length of the elements. Useful to identify negative drifts and badly positioned elements. The optional argument *info* indicates to display extra information like elements misalignments.
- check_sequ** A *method* () checking the integrity of the sequence and its dictionary, for debugging purpose only.

3 Metamethods

The sequence object provides the following metamethods:

- __len** A *metamethod* () called by the length operator `#` to return the size of the sequence, i.e. the number of elements stored including the "`$start`" and "`$end`" markers.
- __index** A *metamethod* (*key*) called by the indexing operator [*key*] to return the *value* of an attribute determined by *key*. The *key* is interpreted differently depending on its type with the following precedence:
1. A *number* is interpreted as an element index and returns the element or `nil`.
 2. Other *key* types are interpreted as *object* attributes subject to object model lookup.
 3. If the *value* associated with *key* is `nil`, then *key* is interpreted as an element name and returns either the element or an *iterable* on the elements with the same name.⁴
 4. Otherwise returns `nil`.
- __newindex** A *metamethod* (*key*, *val*) called by the assignment operator [*key*]=*val* to create new attributes for the pairs (*key*, *value*). If *key* is a *number* specifying the index or a *string* specifying the name of an existing element, the following error is raised:
- ```
"invalid sequence write access (use replace method)"
```
- \_\_init** A *metamethod* () called by the constructor to compute the elements positions.<sup>5</sup>
- \_\_copy** A *metamethod* () similar to the `:copy method`.

The following attribute is stored with metamethods in the metatable, but has different purpose:

- \_\_sequ** A unique private *reference* that characterizes sequences.

<sup>4</sup>An *iterable* supports the length operator `#`, the indexing operator `[]`, and generic for loops with `ipairs`.

<sup>5</sup>MAD-NG does not have a MAD-X like "USE" command to finalize this computation.

## 4 Sequences creation

During its creation as an *object*, a sequence can defined its attributes as any object, and the *list* of its elements that must form a *sequence* of increasing *s*-positions. When subsequences are part of this *list*, they are replaced by their respective elements as a sequence *element* cannot be present inside other sequences. If the length of the sequence is not provided, it will be computed and set automatically. During their creation, sequences compute the *s*-positions of their elements as described in the section [element positions](#), and check for overlapping elements that would raise a “negative drift” runtime error.

The following example shows how to create a sequence form a *list* of elements and subsequences:

```

local sequence, drift, marker in MAD.element
local df, mk = drift 'df' {l=1}, marker 'mk' {}
local seq = sequence 'seq' {
 df 'df1' {}, mk 'mk1' {},
 sequence {
 sequence { mk 'mk0' {} },
 df 'df.s' {}, mk 'mk.s' {}
 },
 df 'df2' {}, mk 'mk2' {},
} :dumpseq()

-- display
sequence: seq, l=3
idx kind name l dl spos upos uds
001 marker $start 0.000 0 0.000 0.000 0.000
002 drift df1 1.000 0 0.000 0.500 0.500
003 marker mk1 0.000 0 1.000 1.000 0.000
004 marker mk0 0.000 0 1.000 1.000 0.000
005 drift df.s 1.000 0 1.000 1.500 0.500
006 marker mk.s 0.000 0 2.000 2.000 0.000
007 drift df2 1.000 0 2.000 2.500 0.500
008 marker mk2 0.000 0 3.000 3.000 0.000
009 marker $end 0.000 0 3.000 3.000 0.000

```

## 5 Elements positions

A sequence looks at the following attributes of an element, including sub-sequences, when installing it, *and only at that time*, to determine its position:

- at**            A *number* holding the position in [m] of the element in the sequence relative to the position specified by the *from* attribute.
- from**        A *string* holding one of "start", "prev", "next", "end" or "selected", or the (mangled) name of another element to use as the reference position, or a *number* holding a position in [m] from the start of the sequence. (default: "start" if  $at \geq 0$ , "end" if  $at < 0$ , and "prev" otherwise)
- refpos**      A *string* holding one of "entry", "centre" or "exit", or the (mangled) name of a sequence sub-element to use as the reference position, or a *number* specifying a position [m] from the start of the element, all of them resulting in an offset to subtract to the *at* attribute to find the *s*-position of the element entry. (default: `nil`  $\equiv$  `self.refer`).

**shared** A *logical* specifying if an element is used at different positions in the same sequence definition, i.e. shared multiple times, through temporary instances to store the many `at` and `from` attributes needed to specify its positions. Once built, the sequence will drop these temporary instances in favor of their common parent, i.e. the original shared element.

**Warning:** The `at` and `from` attributes are not considered as intrinsic properties of the elements and are used only once during installation. Any reuse of these attributes is the responsibility of the user, including the consistency between `at` and `from` after updates.

## 6 Elements selections

The element selection in sequence use predicates in combination with iterators. The sequence iterator manages the range of elements where to apply the selection, while the predicate says if an element in this range is illegible for the selection. In order to ease the use of methods based on the `:foreach` method, the selector predicate `sel` can be built from different types of information provided in a *set* with the following attributes:

**flag** A *number* interpreted as a flags mask to pass to the element method `:is_selected`. It should not be confused with the flags passed as argument to methods `:select` and `:deselect`, as both flags can be used together but with different meanings!

**pattern** A *string* interpreted as a pattern to match the element name using `string.match` from the standard library, see [Lua 5.2 §6.4](#) for details.

**class** An *element* interpreted as a *class* to pass to the element method `:is_instanceOf`.

**table** A *mappable* interpreted as a *set* used to select the elements by their name, i.e. the built predicate will use `tbl[elm.name]` as a *logical*.

**select** A *callable* interpreted as the selector itself, which allows to build any kind of predicate or to complete the restrictions already built above.

**subelem** A *boolean* indicating to include or not the sub-elements in the scanning loop. The predicate and the action receive the sub-element and its sub-index as first and second argument, and the main element index as third argument.

All these attributes are used in the aforementioned order to incrementally build predicates that are combined with logical conjunctions, i.e. `and`'ed, to give the final predicate used by the `:foreach` method. If only one of these attributes is needed, it is possible to pass it directly in `sel`, not as an attribute in a *set*, and its type will be used to determine the kind of predicate to build. For example, `self:foreach(act, monitor)` is equivalent to `self:foreach{action=act, class=monitor}`.

## 7 Indexes, names and counts

Indexing a sequence triggers a complex look up mechanism where the arguments will be interpreted in various ways as described in the `:__index` metamethod. A *number* will be interpreted as a relative slot index in the list of elements, and a negative index will be considered as relative to the end of the sequence, i.e. `-1` is the `$end` marker. Non-*number* will be interpreted first as an object key (can be anything), looking for sequence methods or attributes; then as an element name if nothing was found.

If an element exists but its name is not unique in the sequence, an *iterable* is returned. An *iterable* supports the length `#` operator to retrieve the number of elements with the same name, the indexing



operator `[]` waiting for a count  $n$  to retrieve the  $n$ -th element from the start with that name, and the iterator `ipairs` to use with generic for loops.

The returned *iterable* is in practice a proxy, i.e. a fake intermediate object that emulates the expected behavior, and any attempt to access the proxy in another manner should raise a runtime error.

**Warning:** The indexing operator `[]` interprets a *number* as a (relative) element index as the method `:index`, while the method `:index_of` interprets a *number* as a (relative) element  $s$ -position `[m]`.

The following example shows how to access to the elements through indexing and the *iterable*:

```

local sequence, drift, marker in MAD.element
local seq = sequence {
 drift 'df' { id=1 }, marker 'mk' { id=2 },
 drift 'df' { id=3 }, marker 'mk' { id=4 },
 drift 'df' { id=5 }, marker 'mk' { id=6 },
}
print(seq[1].name) -- display: $start (start marker)
print(seq[-1].name) -- display: $end (end marker)

print(#seq.df, seq.df[3].id) -- display: 3 5
for _,e in ipairs(seq.df) do io.write(e.id," ") end -- display: 1 3 5
for _,e in ipairs(seq.mk) do io.write(e.id," ") end -- display: 2 4 6

-- print name of drift with id=3 in absolute and relative to id=6.
print(seq:name_of(4)) -- display: df[2] (2nd df from start)
print(seq:name_of(2, -2)) -- display: df{-3} (3rd df before last mk)

```

The last two lines of code display the name of the same element but mangled with absolute and relative counts.

## 8 Iterators and ranges

Ranging a sequence triggers a complex look up mechanism where the arguments will be interpreted in various ways as described in the `:range_of` method, itself based on the methods `:index_of` and `:index`. The number of elements selected by a sequence range can be computed by the `:length_of` method, which accepts an extra *number* of turns to consider in the calculation.

The sequence iterators are created by the methods `:iter` and `:siter`, and both are based on the `:range_of` method as mentioned in their descriptions and includes an extra *number* of turns as for the method `:length_of`, and a direction `1` (forward) or `-1` (backward) for the iteration. The `:siter` differs from the `:iter` by its loop, which returns not only the sequence elements but also *implicit* drifts built on-the-fly when a gap  $> 10^{-12}$  m is detected between two sequence elements. Such implicit drift have half-integer indexes and make the iterator “continuous” in  $s$ -positions.

The method `:foreach` uses the iterator returned by `:iter` with a range as its sole argument to loop over the elements where to apply the predicate before executing the action. The methods `:select`, `:deselect`, `:filter`, `:install`, `:replace`, `:remove`, `:move`, and `:misalign` are all based directly or indirectly on the `:foreach` method. Hence, to iterate backward over a sequence range, these methods have to use either its *list* form or a numerical range. For example the invocation `seq:foreach(\e -> print(e.name), {-2, 2, 'idx', -1})` will iterate backward over the entire sequence `seq` excluding the `$start` and `$end` markers, while the invocation `seq:foreach(\e -> print(e.name), 5..2..-1)` will iterate backward over the elements with  $s$ -positions sitting in the interval `[2, 5]` m.

The tracking commands `survey` and `track` use the iterator returned by `:siter` for their main loop, with their `range`, `nturn` and `dir` attributes as arguments. These commands also save the iterator states in

their mflw to allow the users to run them nstep by nstep, see commands [survey](#) and [track](#) for details.

The following example shows how to access to the elements with the `:foreach` method:

```

local sequence, drift, marker in MAD.element
local observed in MAD.element.flags
local seq = sequence {
 drift 'df' { id=1 }, marker 'mk' { id=2 },
 drift 'df' { id=3 }, marker 'mk' { id=4 },
 drift 'df' { id=5 }, marker 'mk' { id=6 },
}

local act = \e -> print(e.name,e.id)
seq:foreach(act, "df[2]/mk[3]")
-- display:
df 3
mk 4
df 5
mk 6

seq:foreach{action=act, range="df[2]/mk[3]", class=marker}
-- display: markers at ids 4 and 6
seq:foreach{action=act, pattern="^[^$]"}
-- display: all elements except $start and $end markers
seq:foreach{action=\e -> e:select(observed), pattern="mk"}
-- same as: seq:select(observed, {pattern="mk"})

local act = \e -> print(e.name, e.id, e:is_observed())
seq:foreach{action=act, range="#s/#e"}
-- display:
$start nil false
df 1 false
mk 2 true
df 3 false
mk 4 true
df 5 false
mk 6 true
$end nil true

```

## 9 Examples

### 9.1 FODO cell

The following example shows how to build a very simple FODO cell and an arc made of 10 FODO cells.

```

local sequence, sbend, quadrupole, sextupole, hkicker, vkicker, marker in MAD.element
local mkf = marker 'mkf' {}
local ang=2*math.pi/80
local fodo = sequence 'fodo' { refer='entry',
 mkf { at=0, shared=true }, -- mark the start of the fodo
 quadrupole 'qf' { at=0, l=1 , k1=0.3 },
 sextupole 'sf' { l=0.3, k2=0 },

```

```

hkicker 'hk' { l=0.2, kick=0 },
sbend 'mb' { at=2, l=2 , angle=ang },

quadrupole 'qd' { at=5, l=1 , k1=-0.3 },
sextupole 'sd' { l=0.3, k2=0 },
vkicker 'vk' { l=0.2, kick=0 },
sbend 'mb' { at=7, l=2 , angle=ang },
}
local arc = sequence 'arc' { refer='entry', 10*fodo }
fodo:dumpseq() ; print(fodo.mkf, mkf)
-- display:
sequence: fodo, l=9
idx kind name l dl spos upos uds
001 marker $start 0.000 0 0.000 0.000 0.000
002 marker mkf 0.000 0 0.000 0.000 0.000
003 quadrupole qf 1.000 0 0.000 0.000 0.000
004 sextupole sf 0.300 0 1.000 1.000 0.000
005 hkicker hk 0.200 0 1.300 1.300 0.000
006 sbend mb 2.000 0 2.000 2.000 0.000
007 quadrupole qd 1.000 0 5.000 5.000 0.000
008 sextupole sd 0.300 0 6.000 6.000 0.000
009 vkicker vk 0.200 0 6.300 6.300 0.000
010 sbend mb 2.000 0 7.000 7.000 0.000
011 marker $end 0.000 0 9.000 9.000 0.000
marker: 'mkf' 0x01015310e8 marker: 'mkf' 0x01015310e8 -- same marker

```

## 9.2 SPS compact description

The following dummy example shows a compact definition of the SPS mixing elements, beam lines and sequence definitions. The elements are zero-length, so the lattice is too.

```

local drift, sbend, quadrupole, bline, sequence in MAD.element

-- elements (empty!)
local ds = drift 'ds' {}
local dl = drift 'dl' {}
local dm = drift 'dm' {}
local b1 = sbend 'b1' {}
local b2 = sbend 'b2' {}
local qf = quadrupole 'qf' {}
local qd = quadrupole 'qd' {}

-- subsequences
local pf = bline 'pf' {qf,2*b1,2*b2,ds} -- #: 6
local pd = bline 'pd' {qd,2*b2,2*b1,ds} -- #: 6
local p24 = bline 'p24' {qf,dm,2*b2,ds,pd} -- #: 11 (5+6)
local p42 = bline 'p42' {pf,qd,2*b2,dm,ds} -- #: 11 (6+5)
local p00 = bline 'p00' {qf,dl,qd,dl} -- #: 4
local p44 = bline 'p44' {pf,pd} -- #: 12 (6+6)
local insert = bline 'insert' {p24,2*p00,p42} -- #: 30 (11+2*4+11)
local super = bline 'super' {7*p44,insert,7*p44} -- #: 198 (7*12+30+7*12)

```

```

-- final sequence
local SPS = sequence 'SPS' {6*super} -- # = 1188 (6*198)

-- check number of elements and length
print(#SPS, SPS.l) -- display: 1190 0 (no element length provided)

```

### 9.3 Installing elements I

The following example shows how to install elements and subsequences in an empty initial sequence:

```

local sequence, drift in MAD.element
local seq = sequence "seq" { l=16, refer="entry", owner=true }
local sseq1 = sequence "sseq1" {
 at=5, l=6 , refpos="centre", refer="entry",
 drift "df1'" {l=1, at=-4, from="end"},
 drift "df2'" {l=1, at=-2, from="end"},
 drift "df3'" { at= 5 },
}
local sseq2 = sequence "sseq2" {
 at=14, l=6, refpos="exit", refer="entry",
 drift "df1''" { l=1, at=-4, from="end"},
 drift "df2''" { l=1, at=-2, from="end"},
 drift "df3''" { at= 5 },
}
seq:install {
 drift "df1" {l=1, at=1},
 sseq1, sseq2,
 drift "df2" {l=1, at=15},
} :dumpseq()

-- display:
sequence: seq, l=16
idx kind name l dl spos upos uds
001 marker $start 0.000 0 0.000 0.000 0.000
002 drift df1 1.000 0 1.000 1.000 0.000
003 drift df1' 1.000 0 4.000 4.000 0.000
004 drift df2' 1.000 0 6.000 6.000 0.000
005 drift df3' 0.000 0 7.000 7.000 0.000
006 drift df1'' 1.000 0 10.000 10.000 0.000
007 drift df2'' 1.000 0 12.000 12.000 0.000
008 drift df3'' 0.000 0 13.000 13.000 0.000
009 drift df2 1.000 0 15.000 15.000 0.000
010 marker $end 0.000 0 16.000 16.000 0.000

```

### 9.4 Installing elements II

The following more complex example shows how to install elements and subsequences in a sequence using a selection and the packed form for arguments:

```

local mk = marker "mk" { }

```

```

local seq = sequence "seq" { l = 10, refer="entry",
 mk "mk1" { at = 2 },
 mk "mk2" { at = 4 },
 mk "mk3" { at = 8 },
}
local sseq = sequence "sseq" { l = 3 , at = 5, refer="entry",
 drift "df1'" { l = 1, at = 0 },
 drift "df2'" { l = 1, at = 1 },
 drift "df3'" { l = 1, at = 2 },
}
seq:install {
 class = mk,
 elements = {
 drift "df1" { l = 0.1, at = 0.1, from="selected" },
 drift "df2" { l = 0.1, at = 0.2, from="selected" },
 drift "df3" { l = 0.1, at = 0.3, from="selected" },
 sseq,
 drift "df4" { l = 1, at = 9 },
 }
}

seq:dumpseq()
-- display:
sequence: seq, l=10
idx kind name l dl spos upos uds
001 marker $start 0.000 0 0.000 0.000 0.000
002 marker mk1 0.000 0 2.000 2.000 0.000
003 drift df1 0.100 0 2.100 2.100 0.000
004 drift df2 0.100 0 2.200 2.200 0.000
005 drift df3 0.100 0 2.300 2.300 0.000
006 marker mk2 0.000 0 4.000 4.000 0.000
007 drift df1 0.100 0 4.100 4.100 0.000
008 drift df2 0.100 0 4.200 4.200 0.000
009 drift df3 0.100 0 4.300 4.300 0.000
010 drift df1' 1.000 0 5.000 5.000 0.000
011 drift df2' 1.000 0 6.000 6.000 0.000
012 drift df3' 1.000 0 7.000 7.000 0.000
013 marker mk3 0.000 0 8.000 8.000 0.000
014 drift df1 0.100 0 8.100 8.100 0.000
015 drift df2 0.100 0 8.200 8.200 0.000
016 drift df3 0.100 0 8.300 8.300 0.000
017 drift df4 1.000 0 9.000 9.000 0.000
018 marker $end 0.000 0 10.000 10.000 0.000

```

# Chapter 7. MTable

The MAD Tables (MTables) — also named Table File System (TFS) — are objects convenient to store, read and write a large amount of heterogeneous information organized as columns and header. The MTables are also containers that provide fast access to their rows, columns, and cells by referring to their indexes, or some values of the designated reference column, or by running iterators constrained with ranges and predicates.

The `mtable` object is the *root object* of the TFS tables that store information relative to tables.

The `mtable` module extends the `typeid` module with the `is_mtable` function, which returns `true` if its argument is a `mtable` object, `false` otherwise.

## 1 Attributes

The `mtable` object provides the following attributes:

|                 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
|-----------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>type</b>     | A <i>string</i> specifying the type of the <code>mtable</code> (often) set to the name of the command that created it, like <code>survey</code> , <code>track</code> or <code>twiss</code> . (default: <code>'user'</code> ).                                                                                                                                                                                                                                                                                 |
| <b>title</b>    | A <i>string</i> specifying the title of the <code>mtable</code> (often) set to the attribute <code>title</code> of the command that created it. (default: <code>'no-title'</code> ).                                                                                                                                                                                                                                                                                                                          |
| <b>origin</b>   | A <i>string</i> specifying the origin of the <code>mtable</code> . (default: <code>"MAD version os arch"</code> ).                                                                                                                                                                                                                                                                                                                                                                                            |
| <b>date</b>     | A <i>string</i> specifying the date of creation of the <code>mtable</code> . (default: <code>"day/month/year"</code> ).                                                                                                                                                                                                                                                                                                                                                                                       |
| <b>time</b>     | A <i>string</i> specifying the time of creation of the <code>mtable</code> . (default: <code>"hour:min:sec"</code> ).                                                                                                                                                                                                                                                                                                                                                                                         |
| <b>refcol</b>   | A <i>string</i> specifying the name of the reference column used to build the dictionary of the <code>mtable</code> , and to mangle values with counts. (default: <code>nil</code> ).                                                                                                                                                                                                                                                                                                                         |
| <b>header</b>   | A <i>list</i> specifying the augmented attributes names (and their order) used by default for the header when writing the <code>mtable</code> to files. Augmented meaning that the <i>list</i> is concatenated to the <i>list</i> held by the parent <code>mtable</code> during initialization. (default: <code>{'name', 'type', 'title', 'origin', 'date', 'time', 'refcol'}</code> ).                                                                                                                       |
| <b>column</b>   | A <i>list</i> specifying the augmented columns names (and their order) used by default for the columns when writing the <code>mtable</code> to files. Augmented meaning that the <i>list</i> is concatenated to the <i>list</i> held by the parent <code>mtable</code> during initialization. (default: <code>nil</code> ).                                                                                                                                                                                   |
| <b>novector</b> | A <i>logical</i> specifying to not convert ( <code>novector == true</code> ) columns containing only numbers to vectors during the insertion of the second row. The attribute can also be a <i>list</i> specifying the columns names to remove from the specialization. If the <i>list</i> is empty or <code>novector ~= true</code> , all numeric columns will be converted to vectors, and support all methods and operations from the <a href="#">linear algebra</a> module. (default: <code>nil</code> ). |
| <b>owner</b>    | A <i>logical</i> specifying if an <i>empty</i> <code>mtable</code> is a view with no data ( <code>owner ~= true</code> ), or a <code>mtable</code> holding data ( <code>owner == true</code> ). (default: <code>nil</code> ).                                                                                                                                                                                                                                                                                 |
| <b>reserve</b>  | A <i>number</i> specifying an estimate of the maximum number of rows stored in the <code>mtable</code> . If the value is underestimated, the <code>mtable</code> will still expand on need. (default: <code>8</code> ).                                                                                                                                                                                                                                                                                       |

**Warning:** the following private and read-only attributes are present in all `mtables` and should *never be used, set or changed*; breaking this rule would lead to an *undefined behavior*:

|                |                                                                                                                                                                                                                   |
|----------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>__dat</b>   | A <i>table</i> containing all the private data of mtables.                                                                                                                                                        |
| <b>__seq</b>   | A <i>sequence</i> attached to the mtable by the <code>survey</code> and <code>track</code> commands and used by the methods receiving a <i>reference</i> to an element as argument. (default: <code>nil</code> ). |
| <b>__cycle</b> | A <i>reference</i> to the row registered with the <code>:cycle</code> method. (default: <code>nil</code> ).                                                                                                       |

## 2 Methods

The `mtable` object provides the following methods:

|                 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
|-----------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>nrow</b>     | A <i>method</i> ( <code>()</code> ) returning the <i>number</i> of rows in the mtable.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| <b>ncol</b>     | A <i>method</i> ( <code>()</code> ) returning the <i>number</i> of columns in the mtable.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| <b>ngen</b>     | A <i>method</i> ( <code>()</code> ) returning the <i>number</i> of columns generators in the mtable. The <i>number</i> of columns with data is given by <code>:ncol() - :ngen()</code> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| <b>colname</b>  | A <i>method</i> ( <code>idx</code> ) returning the <i>string</i> name of the <code>idx</code> -th column in the mtable or <code>nil</code> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| <b>colnames</b> | A <i>method</i> ( <code>[lst]</code> ) returning the <i>list</i> <code>lst</code> (default: <code>{}</code> ) filled with all the columns names of the mtable.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| <b>index</b>    | A <i>method</i> ( <code>idx</code> ) returning a positive index, or <code>nil</code> . If <code>idx</code> is negative, it is reflected versus the size of the mtable, e.g. <code>-1</code> becomes <code>#self</code> , the index of the last row.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| <b>name_of</b>  | A <i>method</i> ( <code>idx, [ref]</code> ) returning a <i>string</i> corresponding to the (mangled) <i>value</i> from the reference column of the row at the index <code>idx</code> , or <code>nil</code> . A row <i>value</i> appearing more than once in the reference column will be mangled with an absolute count, e.g. <code>mq[3]</code> , or a relative count versus the reference row determined by <code>:index_of(ref)</code> , e.g. <code>mq{-2}</code> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| <b>index_of</b> | A <i>method</i> ( <code>a, [ref], [dir]</code> ) returning a <i>number</i> corresponding to the positive index of the row determined by the first argument or <code>nil</code> . If <code>a</code> is a <i>number</i> (or a <i>string</i> representing a <i>number</i> ), it is interpreted as the index of the row and returned as a second <i>number</i> . If <code>a</code> is a <i>string</i> , it is interpreted as the (mangled) <i>value</i> of the row in the reference column as returned by <code>:name_of</code> . Finally, <code>a</code> can be a <i>reference</i> to an element to search for <b>if</b> the mtable has both, an attached sequence, and a column named <code>'eidx'</code> mapping the indexes of the elements to the attached sequence. <sup>1</sup> The argument <code>ref</code> (default: <code>nil</code> ) specifies the reference row determined by <code>:index_of(ref)</code> to use for relative indexes, for decoding mangled values with relative counts, or as the reference row to start searching from. The argument <code>dir</code> (default: <code>1</code> ) specifies the direction of the search with values <code>1</code> (forward), <code>-1</code> (backward), or <code>0</code> (no direction), which correspond respectively to the rounding methods <code>ceil</code> , <code>floor</code> and <code>round</code> from the <code>gmath</code> module. |
| <b>range_of</b> | A <i>method</i> ( <code>[rng], [ref], [dir]</code> ) returning three <i>numbers</i> corresponding to the positive indexes <i>start</i> and <i>end</i> of the range and its direction <i>dir</i> (default: <code>1</code> ), or <code>nil</code> for an empty range. If <code>rng</code> is omitted, it returns <code>1, #self, 1</code> , or <code>#self, 1, -1</code> if <code>dir</code> is negative. If <code>rng</code> is a <i>number</i> or a <i>string</i> with no <code>'/'</code> separator, it is interpreted as <i>start</i> and <i>end</i> , both determined by <code>:index_of</code> . If <code>rng</code> is a <i>string</i> containing the separator <code>'/'</code> , it is split in two <i>strings</i> interpreted as <i>start</i> and <i>end</i> , both determined by <code>:index_of</code> . If <code>rng</code> is a <i>list</i> , it will be interpreted as <code>{ start, end, [ref], [dir] }</code> , both determined by <code>:index_of</code> . The arguments <code>ref</code> and <code>dir</code> are forwarded to all invocations of <code>:index_of</code> .                                                                                                                                                                                                                                                                                                                   |

<sup>1</sup>These information are usually provided by the command creating the `mtable`, like `survey` and `track`.

with a higher precedence than ones in the *list* `rng`, and a runtime error is raised if the method returns `nil`, i.e. to disambiguate between a valid empty range and an invalid range.

- length\_of** A *method* (`[rng]`, `[ntrn]`, `[dir]`) returning a *number* specifying the length of the range optionally including `ntrn` extra turns (default: 0), and calculated from the indexes returned by `:range_of([rng], nil, [dir])`.
- get** A *method* (`row`, `col`, `[cnt]`) returning the *value* stored in the mtable at the cell (`row`, `col`), or `nil`. If `row` is a not a row index determined by `:index(row)`, it is interpreted as a (mangled) *value* to search in the reference column, taking into account the count `cnt` (default: 1). If `col` is not a column index, it is interpreted as a column name.
- set** A *method* (`row`, `col`, `val`, `[cnt]`) returning the mtable itself after updating the cell (`row`, `col`) to the value `val`, or raising an error if the cell does not exist. If `row` is a not a row index determined by `:index(row)`, it is interpreted as a (mangled) *value* to search in the reference column, taking into account the count `cnt` (default: 1). If `col` is not a column index, it is interpreted as a column name.
- getcol** A *method* (`col`) returning the column `col`, or `nil`. If `col` is not a column index, it is interpreted as a column name.
- setcol** A *method* (`col`, `val`) returning the mtable itself after updating the column `col` with the values of `val`, or raising an error if the column does not exist. If `col` is not a column index, it is interpreted as a column name. If the column is a generator, so must be `val` or an error will be raised. If the column is not a generator and `val` is a *callable* (`ri`), it will be invoked with the row index `ri` as its sole argument, using its returned value to update the column cell. Otherwise `val` must be an *iterable* or an error will be raised. If the column is already a specialized *vector*, the *iterable* must provide enough numbers to fill it entirely as `nil` is not a valid value.
- inscol** A *method* (`[ref]`, `col`, `val`, `[nvec]`) returning the mtable itself after inserting the column data `val` with the *string* name `col` at index `ref` (default: `:ncol()+1`). If `ref` is not a column index, it is interpreted as a column name. If `val` is a *callable* (`ri`), it will be added as a column generator. Otherwise `val` must be an *iterable* or an error will be raised. The *iterable* will used to fill the new column that will be specialized to a *vector* if its first value is a *number* and `nvec` `~= true` (default: `nil`).
- addcol** A *method* (`col`, `val`, `[nvec]`) equivalent to `:inscol(nil, col, val, [nvec])`.
- remcol** A *method* (`col`) returning the mtable itself after removing the column `col`, or raising an error if the column does not exist. If `col` is not a column index, it is interpreted as a column name.
- rencol** A *method* (`col`, `new`) returning the mtable itself after renaming the column `col` to the *string* `new`, or raising an error if the column does not exist. If `col` is not a column index, it is interpreted as a column name.
- getrow** A *method* (`row`, `[ref]`) returning the *mappable* (proxy) of the row determined by the method `:index_of(row, [ref])`, or `nil`.
- setrow** A *method* (`row`, `val`, `[ref]`) returning the mtable itself after updating the row at index determined by `:index_of(row, [ref])` using the values provided by the *mappable* `val`, which can be a *list* iterated as pairs of (*index*, *value*) or a *set* iterated as pairs



of *(key, value)* with *key* being the column names, or a combination of the two. An error is raised if the column does not exist.

- insrow** A *method* (*row*, *val*, [*ref*]) returning the mtable itself after inserting a new row at index determined by `:index_of(row, [ref])` and filled with the values provided by the *mappable* *val*, which can be a *list* iterated as pairs of *(index, value)* or a *set* iterated as pairs of *(key, value)* with *key* being the column names or a combination of the two.
- addrow** A *method* (*val*) equivalent to `:insrow(#self+1, val)`.
- remrow** A *method* (*row*, [*ref*]) returning the mtable itself after removing the row determined by the method `:index_of(row, [ref])`, or raising an error if the row does not exist.
- swprow** A *method* (*row1*, *row2*, [*ref1*], [*ref2*]) returning the mtable itself after swapping the content of the rows, both determined by the method `:index_of(row, [ref])`, or raising an error if one of the row does not exist.
- clrrow** A *method* (*row*, [*ref*]) returning the mtable itself after clearing the row determined by the method `:index_of(row, [ref])`, or raising an error if the row does not exist; where clearing the row means to set *vector* value to 0 and *nil* otherwise.
- clear** A *method* () returning the mtable itself after clearing all the rows, i.e. `#self == 0`, with an opportunity for new columns specialization.
- iter** A *method* (*[rng]*, [*ntrn*], [*dir*]) returning an iterator over the mtable rows. The optional range is determined by `:range_of([rng], [dir])`, optionally including *ntrn* turns (default: 0). The optional direction *dir* specifies the forward 1 or the backward -1 direction of the iterator. If *rng* is not provided and the mtable is cycled, the *start* and *end* indexes are determined by `:index_of(self.__cycle)`. When used with a generic **for** loop, the iterator returns at each rows the index and the row *mappable* (proxy).
- foreach** A *method* (*act*, [*rng*], [*sel*], [*not*]) returning the mtable itself after applying the action *act* on the selected rows. If *act* is a *set* representing the arguments in the packed form, the missing arguments will be extracted from the attributes *action*, *range*, *select* and *default*. The action *act* must be a *callable* (*row*, *idx*) applied to a row passed as first argument and its index as second argument. The optional range is used to generate the loop iterator `:iter([rng])`. The optional selector *sel* is a *callable* (*row*, *idx*) predicate selecting eligible rows for the action from the row itself passed as first argument and its index as second argument. The selector *sel* can be specified in other ways, see [row selections](#) for details. The optional *logical not* (default: false) indicates how to interpret default selection, as *all* or *none*, depending on the semantic of the action.<sup>2</sup>
- select** A *method* (*[rng]*, [*sel*], [*not*]) returning the mtable itself after selecting the rows using `:foreach(sel_act, [rng], [sel], [not])`. By default mtable have all their rows deselected, the selection being stored as *boolean* in the column at index 0 and named *is\_selected*.
- deselect** A *method* (*[rng]*, [*sel*], [*not*]) returning the mtable itself after deselecting the rows using `:foreach(desel_act, [rng], [sel], [not])`. By default mtable have all their rows deselected, the selection being stored as *boolean* in the column at index 0 and named *is\_selected*.

<sup>2</sup>For example, the `:remove` method needs `not=true` to *not* remove all rows if no selector is provided.

- filter** A *method* ([rng], [sel], [not]) returning a *list* containing the positive indexes of the rows determined by :foreach(filt\_act, [rng], [sel], [not]), and its size.
- insert** A *method* (row, [rng], [sel]) returning the mtable itself after inserting the rows in the *list* row at the indexes determined by :filter([rng], [sel], true). If the arguments are passed in the packed form, the extra attribute rows will be used as a replacement for the argument row, and if the attribute where="after" is defined then the rows will be inserted after the selected indexes. The insertion scheme depends on the number  $R$  of rows in the *list* row versus the number  $S$  of rows selected by :filter;  $1 \times 1$  (one row inserted at one index),  $R \times 1$  ( $R$  rows inserted at one index),  $1 \times S$  (one row inserted at  $S$  indexes) and  $R \times S$  ( $R$  rows inserted at  $S$  indexes). Hence, the insertion schemes insert respectively 1,  $R$ ,  $S$ , and  $\min(R, S)$  rows.
- remove** A *method* ([rng], [sel]) returning the mtable itself after removing the rows determined by :filter([rng], [sel], true).
- sort** A *method* (cmp, [rng], [sel]) returning the mtable itself after sorting the rows at the indexes determined by :filter([rng], [sel], true) using the ordering *callable* cmp(row1, row2). The arguments row1 and row2 are *mappable* (proxies) referring to the current rows being compared and providing access to the columns values for the comparison.<sup>3</sup> The argument cmp can be specified in a compact ordering form as a *string* that will be converted to an ordering *callable* by the function str2cmp from the [utility](#) module. For example, the *string* "-y,x" will be converted by the method to the following *lambda*  $\lambda r1, r2 \rightarrow r1.y > r2.y$  **or**  $r1.y == r2.y$  **and**  $r1.x < r2.x$ , where  $y$  and  $x$  are the columns used to sort the mtable in descending (-) and ascending (+) order respectively. The compact ordering form is not limited in the number of columns and avoids making mistakes in the comparison logic when many columns are involved.
- cycle** A *method* (a) returning the mtable itself after checking that a is a valid reference using :index\_of(a), and storing it in the \_\_cycle attribute, itself erased by the methods editing the mtable like :insert, :remove or :sort.
- copy** A *method* ([name], [owner]) returning a new mtable from a copy of self, with the optional name and the optional attribute owner set. If the mtable is a view, so will be the copy unless owner == true.
- is\_view** A *method* () returning true if the mtable is a view over another mtable data, false otherwise.
- set\_readonly** Set the mtable as read-only, including the columns and the rows proxies.
- read** A *method* ([filename]) returning a new instance of self filled with the data read from the file determined by openfile(filename, 'r', {'.tfs', '.txt', '.dat'}) from the [utility](#) module. This method can read columns containing the data types *nil*, *boolean*, *number*, *complex number*, (numerical) *range*, and (quoted) *string*. The header can also contain tables saved as *string* and decoded with *function* str2tbl from the [utility](#) module.
- write** A *method* ([filename], [clst], [hlst], [rsel]) returning the mtable itself after writing its content to the file determined by openfile(filename, 'w', {'.tfs', '.txt', '.dat'}) from the [utility](#) module. The columns to write and their order is determined by clst or self.column (default: nil  $\equiv$  all columns). The attributes to write

---

<sup>3</sup>A *mappable* supports the length operator #, the indexing operator [], and generic for loops with pairs.

in the header and their order is determined by `hlst` or `self.header`. The *logical* `rsel` indicates to save all rows or only rows selected by the `:select` method (`rsel == true`). This method can write columns containing the data types *nil*, *boolean*, *number*, *complex number*, (numerical) *range*, and (quoted) *string*. The header can also contain tables saved as *string* and encoded with *function* `tbl2str` from the `utility` module.

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
|--------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>print</b>       | A <i>method</i> ( <code>[clst]</code> , <code>[hlst]</code> , <code>[rsel]</code> ) equivalent to <code>:write(nil, [clst], [hlst], [rsel])</code> .                                                                                                                                                                                                                                                                                                  |
| <b>save_sel</b>    | A <i>method</i> ( <code>[sel]</code> ) saving the rows selection to the optional <i>iterable</i> <code>sel</code> (default: <code>{}</code> ) and return it.                                                                                                                                                                                                                                                                                          |
| <b>restore_sel</b> | A <i>method</i> ( <code>sel</code> ) restoring the rows selection from the <i>iterable</i> <code>sel</code> . The indexes of <code>sel</code> must match the indexes of the rows in the <code>mtable</code> .                                                                                                                                                                                                                                         |
| <b>make_dict</b>   | A <i>method</i> ( <code>[col]</code> ) returning the <code>mtable</code> itself after building the rows dictionary from the values of the reference column determined by <code>col</code> (default: <code>refcol</code> ) for fast row access. If <code>col</code> is not a column index, it is interpreted as a column name except for the special name 'none' that disables the rows dictionary and reset <code>refcol</code> to <code>nil</code> . |
| <b>check_mtbl</b>  | A <i>method</i> ( <code></code> ) checking the integrity of the <code>mtable</code> and its dictionary (if any), for debugging purpose only.                                                                                                                                                                                                                                                                                                          |

### 3 Metamethods

The `mtable` object provides the following metamethods:

|                   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
|-------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>__len</b>      | A <i>metamethod</i> ( <code></code> ) called by the length operator <code>#</code> to return the number of rows in the <code>mtable</code> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| <b>__add</b>      | A <i>metamethod</i> ( <code>val</code> ) called by the plus operator <code>+</code> returning the <code>mtable</code> itself after appending the row <code>val</code> at its end, similar to the <code>:addrow</code> method.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| <b>__index</b>    | A <i>metamethod</i> ( <code>key</code> ) called by the indexing operator <code>[key]</code> to return the <i>value</i> of an attribute determined by <i>key</i> . The <i>key</i> is interpreted differently depending on its type with the following precedence: <ol style="list-style-type: none"> <li>1. A <i>number</i> is interpreted as a row index and returns an <i>iterable</i> on the row (proxy) or <code>nil</code>.</li> <li>2. Other <i>key</i> types are interpreted as <i>object</i> attributes subject to object model lookup.</li> <li>3. If the <i>value</i> associated with <i>key</i> is <code>nil</code>, then <i>key</i> is interpreted as a column name and returns the column if it exists, otherwise...</li> <li>4. If <i>key</i> is not a column name, then <i>key</i> is interpreted as a value in the reference column and returns either an <i>iterable</i> on the row (proxy) determined by this value or an <i>iterable</i> on the rows (proxies) holding this non-unique value.<sup>4</sup></li> <li>5. Otherwise returns <code>nil</code>.</li> </ol> |
| <b>__newindex</b> | A <i>metamethod</i> ( <code>key</code> , <code>val</code> ) called by the assignment operator <code>[key]=val</code> to create new attributes for the pairs ( <i>key</i> , <i>value</i> ). If <i>key</i> is a <i>number</i> or a value specifying a row in the reference column or a <i>string</i> specifying a column name, the following error is raised:                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |

<sup>4</sup>An *iterable* supports the length operator `#`, the indexing operator `[]`, and generic for loops with `ipairs`.

```
"invalid mtable write access (use 'set' methods)"
```

**\_\_init**      A *metamethod* () called by the constructor to build the mtable from the column names stored in its *list* part and some attributes, like *owner*, *reserve* and *novector*.

**\_\_copy**      A *metamethod* () similar to the *method* *copy*.

The following attribute is stored with metamethods in the metatable, but has different purpose:

**\_\_mtbl**      A unique private *reference* that characterizes mtables.

## 4 MTables creation

During its creation as an *object*, a mtable can defined its attributes as any object, and the *list* of its column names, which will be cleared after its initialization. Any column name in the *list* that is enclosed by braces is designated as the reference column for the dictionary that provides fast row indexing, and the attribute *refcol* is set accordingly.

Some attributes are considered during the creation by the *metamethod* *\_\_init*, like *owner*, *reserve* and *novector*, and some others are initialized with defined values like *type*, *title*, *origin*, *date*, *time*, and *refcol*. The attributes *header* and *column* are concatenated with the the parent ones to build incrementing *list* of attributes names and columns names used by default when writing the mtable to files, and these lists are not provided as arguments.

The following example shows how to create a mtable form a *list* of column names add rows:

```
local mtable in MAD
local tbl = mtable 'mytable' {

 {'name'}, 'x', 'y' } -- column 'name' is the refcol
+ { 'p11', 1.1, 1.2 }
+ { 'p12', 2.1, 2.2 }
+ { 'p13', 2.1, 3.2 }
+ { 'p11', 3.1, 4.2 }
print(tbl.name, tbl.refcol, tbl:getcol'name')
-- display: mytable name mtable reference column: 0x010154cd10
```

**Pitfall:** When a column is named 'name', it must be explicitly accessed, e.g. with the *:getcol* method, as the indexing operator *[]* gives the precedence to object's attributes and methods. Hence, *tbl.name* returns the table name 'mytable', not the column 'name'.

## 5 Rows selections

The row selection in mtable use predicates in combination with iterators. The mtable iterator manages the range of rows where to apply the selection, while the predicate says if a row in this range is illegible for the selection. In order to ease the use of methods based on the *:foreach* method, the selector predicate *sel* can be built from different types of information provided in a *set* with the following attributes:

**selected**      A *boolean* compared to the rows selection stored in column '*is\_selected*'.

**pattern**      A *string* interpreted as a pattern to match the *string* in the reference column, which must exist, using *string.match* from the standard library, see [Lua 5.2 §6.4](#) for details. If the reference column does not exist, it can be built using the *:make\_dict* method.

|               |                                                                                                                                                        |
|---------------|--------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>kind</b>   | A <i>string</i> compared to the values in the column <i>kind</i> , which must exist.                                                                   |
| <b>select</b> | A <i>callable</i> interpreted as the selector itself, which allows to build any kind of predicate or to complete the restrictions already built above. |

All these attributes are used in the aforementioned order to incrementally build predicates that are combined with logical conjunctions, i.e. and'ed, to give the final predicate used by the `:foreach` method. If only one of these attributes is needed, it is possible to pass it directly in `sel`, not as an attribute in a *set*, and its type will be used to determine the kind of predicate to build. For example, `tbl:foreach(act, "^MB")` is equivalent to `tbl:foreach{action=act, pattern="^MB"}`.

## 6 Indexes, names and counts

Indexing a *mtable* triggers a complex look up mechanism where the arguments will be interpreted in various ways as described in the metamethod `__index`. A *number* will be interpreted as a relative row index in the list of rows, and a negative index will be considered as relative to the end of the *mtable*, i.e. `-1` is the last row. Non-*number* will be interpreted first as an object key (can be anything), looking for *mtable* methods or attributes; then as a column name or as a row *value* in the reference column if nothing was found.

If a row exists but its *value* is not unique in the reference column, an *iterable* is returned. An *iterable* supports the length `#` operator to retrieve the number of rows with the same *value*, the indexing operator `[]` waiting for a count *n* to retrieve the *n*-th row from the start with that *value*, and the iterator `ipairs` to use with generic for loops.

The returned *iterable* is in practice a proxy, i.e. a fake intermediate object that emulates the expected behavior, and any attempt to access the proxy in another manner should raise a runtime error.

**Note:** Compared to the sequence, the indexing operator `[]` and the method `:index_of` of the *mtable* always interprets a *number* as a (relative) row index. To find a row from a *s*-position `[m]` in the *mtable* if the column exists, use the functions `lsearch` or `bsearch` (if they are monotonic) from the [utility](#) module.

The following example shows how to access to the rows through indexing and the *iterable*:

```

local mtable in MAD
local tbl = mtable { {'name'}, 'x', 'y' } -- column 'name' is the refcol
 + { 'p11', 1.1, 1.2 }
 + { 'p12', 2.1, 2.2 }
 + { 'p13', 2.1, 3.2 }
 + { 'p11', 3.1, 4.2 }
print(tbl[1].y) -- display: 1.2
print(tbl[-1].y) -- display: 4.2

print(#tbl.p11, tbl.p12.y, tbl.p11[2].y) -- display: 2 2.2 4.2
for _,r in ipairs(tbl.p11) do io.write(r.x," ") end -- display: 1.1 3.1
for _,v in ipairs(tbl.p12) do io.write(v, " ") end -- display: 'p12' 2.1 2.2

-- print name of point with name p11 in absolute and relative to p13.
print(tbl:name_of(4)) -- display: p11[2] (2nd p11 from start)
print(tbl:name_of(1, -2)) -- display: p11{-1} (1st p11 before p13)

```

The last two lines of code display the name of the same row but mangled with absolute and relative counts.

## 7 Iterators and ranges

Ranging a mtable triggers a complex look up mechanism where the arguments will be interpreted in various ways as described in the method `:range_of`, itself based on the methods `:index_of` and `:index`. The number of rows selected by a mtable range can be computed by the `:length_of` method, which accepts an extra *number* of turns to consider in the calculation.

The mtable iterators are created by the method `:iter`, based on the method `:range_of` as mentioned in its description and includes an extra *number* of turns as for the method `:length_of`, and a direction 1 (forward) or -1 (backward) for the iteration.

The method `:foreach` uses the iterator returned by `:iter` with a range as its sole argument to loop over the rows where to apply the predicate before executing the action. The methods `:select`, `:deselect`, `:filter`, `:insert`, and `:remove` are all based directly or indirectly on the `:foreach` method. Hence, to iterate backward over a mtable range, these methods have to use either its *list* form or a numerical range. For example the invocation `tbl:foreach(\r -> print(r.name), {-2, 2, nil, -1})` will iterate backward over the entire mtable excluding the first and last rows, equivalently to the invocation `tbl:foreach(\r -> print(r.name), -2..2..-1)`.

The following example shows how to access to the rows with the `:foreach` method:

```

local mtable in MAD
local tbl = mtable { {'name'}, 'x', 'y' }
 + { 'p11', 1.1, 1.2 }
 + { 'p12', 2.1, 2.2 }
 + { 'p13', 2.1, 3.2 }
 + { 'p11', 3.1, 4.2 }

local act = \r -> print(r.name, r.y)
tbl:foreach(act, -2..2..-1)
-- display: p13 3.2
! p12 2.2
tbl:foreach(act, "p11[1]/p11[2]")
-- display: p11 1.2
! p12 2.2
! p13 3.2
! p11 4.2
tbl:foreach{action=act, range="p11[1]/p13"}
-- display: p11 1.2
! p12 2.2
! p13 3.2
tbl:foreach{action=act, pattern="^[1]$"}
-- display: p12 2.2
! p13 3.2
local act = \r -> print(r.name, r.y, r.is_selected)
tbl:select{pattern="p.1"}:foreach{action=act, range="1/-1"}
-- display: p11 1.2 true
! p12 2.2 nil
! p13 3.2 nil
! p11 4.2 true

```

## 8 Examples

### 8.1 Creating a MTable

The following example shows how the track command, i.e. self hereafter, creates its MTable:

```

local header = { -- extra attributes to save in track headers
 'direction', 'observe', 'implicit', 'misalign', 'deltap', 'lost' }

local function make_mtable (self, range, nosave)
 local title, dir, observe, implicit, misalign, deltap, savemap in self
 local sequ, nrow = self.sequence, nosave and 0 or 16

 return mtable(sequ.name, { -- keep column order!
 type='track', title=title, header=header,
 direction=dir, observe=observe, implicit=implicit, misalign=misalign,
 deltap=deltap, lost=0, range=range, reserve=nrow, __seq=sequ,
 {'name'}, 'kind', 's', 'l', 'id', 'x', 'px', 'y', 'py', 't', 'pt',
 'slc', 'turn', 'tdir', 'eidx', 'status', savemap and '__map' or nil })
end

```

### 8.2 Extending a MTable

The following example shows how to extend the MTable created by a twiss command with the elements tilt, angle and integrated strengths from the attached sequence:

```

-- The prelude creating the sequence seq is omitted.
local tws = twiss { sequence=seq, method=4, cofind=true }

local is_integer in MAD.typeid
tws:addcol('angle', \ri => -- add angle column
 local idx = tws[ri].eidx
 return is_integer(idx) and tws.__seq[idx].angle or 0 end)
: addcol('tilt', \ri => -- add tilt column
 local idx = tws[ri].eidx
 return is_integer(idx) and tws.__seq[idx].tilt or 0 end)

for i=1,6 do -- add kil and kisl columns
tws:addcol('k'..i-1..'l', \ri =>
 local idx = tws[ri].eidx
 if not is_integer(idx) then return 0 end -- implicit drift
 local elm = tws.__seq[idx]
 return (elm['k'..i-1] or 0)*elm.l + ((elm.knl or {})[i] or 0)
end)
: addcol('k'..i-1..'sl', \ri =>
 local idx = tws[ri].eidx
 if not is_integer(idx) then return 0 end -- implicit drift
 local elm = tws.__seq[idx]
 return (elm['k'..i-1..'s'] or 0)*elm.l + ((elm.ksl or {})[i] or 0)
end)
end

local cols = {'name', 'kind', 's', 'l', 'angle', 'tilt',

```

```
'x', 'px', 'y', 'py', 't', 'pt',
'beta11', 'beta22', 'alfa11', 'alfa22', 'mu1', 'mu2', 'dx', 'ddx',
'k1l', 'k2l', 'k3l', 'k4l', 'k1sl', 'k2sl', 'k3sl', 'k4sl'}
```

```
tws:write("twiss", cols) -- write header and columns to file twiss.tfs
```

Hopefully, the [physics](#) module provides the *function* `melmcol(mtbl, cols)` to achieve the same task easily:

```
-- The prelude creating the sequence seq is omitted.
```

```
local tws = twiss { sequence=seq, method=4, cofind=true }
```

```
-- Add element properties as columns
```

```
local melmcol in MAD.gphys
```

```
local melmcol(tws, {'angle', 'tilt', 'k1l', 'k2l', 'k3l', 'k4l',
 'k1sl', 'k2sl', 'k3sl', 'k4sl'})
```

```
-- write TFS table
```

```
tws:write("twiss", {
 'name', 'kind', 's', 'l', 'angle', 'tilt',
 'x', 'px', 'y', 'py', 't', 'pt',
 'beta11', 'beta22', 'alfa11', 'alfa22', 'mu1', 'mu2', 'dx', 'ddx',
 'k1l', 'k2l', 'k3l', 'k4l', 'k1sl', 'k2sl', 'k3sl', 'k4sl'})
```



# **Chapter 8. MADX**

- 1 Environment**
- 2 Importing Sequences**
- 3 Converting Scripts**
- 4 Converting Macros**

# **Part II**

## **Commands**

## **Chapter 9. Introduction**

# Chapter 10. Survey

The survey command provides a simple interface to the *geometric* tracking code.<sup>1</sup> The geometric tracking can be used to place the elements of a sequence in the [global reference system](#).

## 1 Command synopsis

Figure 10.1: Synopsis of the survey command with default setup.

```
mtbl, mflw [, eidx] = survey {
 sequence=sequ, -- sequence (required)
 range=nil, -- range of tracking (or sequence.range)
 dir=1, -- s-direction of tracking (1 or -1)
 s0=0, -- initial s-position offset [m]
 X0=0, -- initial coordinates x, y, z [m]
 A0=0, -- initial angles theta, phi, psi [rad] or matrix W0
 nturn=1, -- number of turns to track
 nstep=-1, -- number of elements to track
 nslice=1, -- number of slices (or weights) for each element
 implicit=false, -- slice implicit elements too (e.g. plots)
 misalign=false, -- consider misalignment
 save=true, -- create mtable and save results
 title=nil, -- title of mtable (default seq.name)
 observe=0, -- save only in observed elements (every n turns)
 savesel=fnil, -- save selector (predicate)
 savemap=false, -- save the orientation matrix W in the column __map
 atentry=fnil, -- action called when entering an element
 atslice=fnil, -- action called after each element slices
 atexit=fnil, -- action called when exiting an element
 atsave=fnil, -- action called when saving in mtable
 atdebug=fnil, -- action called when debugging the element maps
 info=nil, -- information level (output on terminal)
 debug=nil, -- debug information level (output on terminal)
 usrdef=nil, -- user defined data attached to the mflow
 mflow=nil, -- mflow, exclusive with other attributes except nstep
}
```

The survey command format is summarized in Figure 10.1, including the default setup of the attributes. The survey command supports the following attributes:

**sequence**     The *sequence* to survey. (no default, required).  
Example: sequence = lhcb1.

**range**        A *range* specifying the span of the sequence survey. If no range is provided, the command looks for a range attached to the sequence, i.e. the attribute seq.range. (default: nil).  
Example: range = "S.DS.L8.B1/E.DS.R8.B1".

---

<sup>1</sup>MAD-NG implements only two tracking codes denominated the *geometric* and the *dynamic* tracking.

|                 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
|-----------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>dir</b>      | The <i>s</i> -direction of the tracking: 1 forward, -1 backward. (default: 1).<br>Example: <code>dir = -1</code> .                                                                                                                                                                                                                                                                                                                                                                                          |
| <b>s0</b>       | A <i>number</i> specifying the initial <i>s</i> -position offset. (default: 0 [m]).<br>Example: <code>s0 = 5000</code> .                                                                                                                                                                                                                                                                                                                                                                                    |
| <b>X0</b>       | A <i>mappable</i> specifying the initial coordinates { <i>x, y, z</i> }. (default: 0 [m]).<br>Example: <code>X0 = { x=100, y=-50 }</code>                                                                                                                                                                                                                                                                                                                                                                   |
| <b>A0</b>       | A <i>mappable</i> specifying the initial angles theta, phi and psi or an orientation <i>matrix</i> $W_0$ . <sup>2</sup> (default: 0 [rad]).<br>Example: <code>A0 = { theta=deg2rad(30) }</code>                                                                                                                                                                                                                                                                                                             |
| <b>nturn</b>    | A <i>number</i> specifying the number of turn to track. (default: 1).<br>Example: <code>nturn = 2</code> .                                                                                                                                                                                                                                                                                                                                                                                                  |
| <b>nstep</b>    | A <i>number</i> specifying the number of element to track. A negative value will track all elements. (default: -1).<br>Example: <code>nstep = 1</code> .                                                                                                                                                                                                                                                                                                                                                    |
| <b>nslice</b>   | A <i>number</i> specifying the number of slices or an <i>iterable</i> of increasing relative positions or a <i>callable</i> ( <code>elm</code> , <code>mflw</code> , <code>lw</code> ) returning one of the two previous kind of positions to track in the elements. The arguments of the callable are in order, the current element, the tracked map flow, and the length weight of the step. This attribute can be locally overridden by the element. (default: 1).<br>Example: <code>nslice = 5</code> . |
| <b>implicit</b> | A <i>logical</i> indicating that implicit elements must be sliced too, e.g. for smooth plotting. (default: false).<br>Example: <code>implicit = true</code> .                                                                                                                                                                                                                                                                                                                                               |
| <b>misalign</b> | A <i>logical</i> indicating that misalignment must be considered. (default: false).<br>Example: <code>implicit = true</code> .                                                                                                                                                                                                                                                                                                                                                                              |
| <b>save</b>     | A <i>logical</i> specifying to create a <i>htable</i> and record tracking information at the observation points. The <code>save</code> attribute can also be a <i>string</i> specifying saving positions in the observed elements: "atentry", "atslice", "atexit" (i.e. true), "atbound" (i.e. entry and exit), "atbody" (i.e. slices and exit) and "atall". (default:true).<br>Example: <code>save = false</code> .                                                                                        |
| <b>title</b>    | A <i>string</i> specifying the title of the <i>htable</i> . If no title is provided, the command looks for the name of the sequence, i.e. the attribute <code>seq.name</code> . (default: nil).<br>Example: <code>title = "Survey around IP5"</code> .                                                                                                                                                                                                                                                      |
| <b>observe</b>  | A <i>number</i> specifying the observation points to consider for recording the tracking information. A zero value will consider all elements, while a positive value will consider selected elements only, checked with method <code>:is_observed</code> , every <code>observe &gt; 0</code> turns. (default: 0).<br>Example: <code>observe = 1</code> .                                                                                                                                                   |
| <b>savesel</b>  | A <i>callable</i> ( <code>elm</code> , <code>mflw</code> , <code>lw</code> , <code>islc</code> ) acting as a predicate on selected elements for observation, i.e. the element is discarded if the predicate returns false. The arguments are in order, the current element, the tracked map flow, the length weight of the slice and the                                                                                                                                                                    |

---

<sup>2</sup>An orientation matrix can be obtained from the 3 angles with `W=matrix(3):rotzxy(-phi, theta, psi)`.

- slice index. (default: `fnil`)  
Example: `savesel = \e -> mylist[e.name] ~= nil.`
- savemap** A *logical* indicating to save the orientation matrix `W` in the column `__map` of the *htable*. (default: `false`).  
Example: `savemap = true.`
- atentry** A *callable* (`elm`, `mflw`, `0`, `-1`) invoked at element entry. The arguments are in order, the current element, the tracked map flow, zero length and the slice index `-1`. (default: `fnil`).  
Example: `atentry = myaction.`
- atslice** A *callable* (`elm`, `mflw`, `lw`, `islc`) invoked at element slice. The arguments are in order, the current element, the tracked map flow, the length weight of the slice and the slice index. (default: `fnil`).  
Example: `atslice = myaction.`
- atexit** A *callable* (`elm`, `mflw`, `0`, `-2`) invoked at element exit. The arguments are in order, the current element, the tracked map flow, zero length and the slice index `-2`. (default: `fnil`).  
Example: `atexit = myaction.`
- atsave** A *callable* (`elm`, `mflw`, `lw`, `islc`) invoked at element saving steps, by default at exit. The arguments are in order, the current element, the tracked map flow, the length weight of the slice and the slice index. (default: `fnil`).  
Example: `atsave = myaction.`
- atdebug** A *callable* (`elm`, `mflw`, `lw`, [`msg`], [...]) invoked at the entry and exit of element maps during the integration steps, i.e. within the slices. The arguments are in order, the current element, the tracked map flow, the length weight of the integration step and a *string* specifying a debugging message, e.g. "`map_name:0`" for entry and "`:1`" for exit. If the level `debug`  $\geq 2$  and `atdebug` is not specified, the default *function* `mdump` is used. In some cases, extra arguments could be passed to the method. (default: `fnil`).  
Example: `atdebug = myaction.`
- info** A *number* specifying the information level to control the verbosity of the output on the console. (default: `nil`).  
Example: `info = 2.`
- debug** A *number* specifying the debug level to perform extra assertions and to control the verbosity of the output on the console. (default: `nil`).  
Example: `debug = 2.`
- usrdef** Any user defined data that will be attached to the tracked map flow, which is internally passed to the elements method `:survey` and to their underlying maps. (default: `nil`).  
Example: `usrdef = { myvar=somevalue }.`
- mflow** A *mflow* containing the current state of a survey command. If a map flow is provided, all attributes are discarded except `nstep`, `info` and `debug`, as the command was already set up upon its creation. (default: `nil`).  
Example: `mflow = mflow0.`

The survey command returns the following objects in this order:

|             |                                                                                                                                                                                       |
|-------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>mtbl</b> | A <i>mtable</i> corresponding to the TFS table of the survey command.                                                                                                                 |
| <b>mflw</b> | A <i>mflow</i> corresponding to the map flow of the survey command.                                                                                                                   |
| <b>eidx</b> | An optional <i>number</i> corresponding to the last surveyed element index in the sequence when <code>nstep</code> was specified and stopped the command before the end of the range. |

## 2 Survey mtable

The survey command returns a *mtable* where the information described hereafter is the default list of fields written to the TFS files.<sup>3</sup>

The header of the *mtable* contains the fields in the default order:

|                  |                                                                                         |
|------------------|-----------------------------------------------------------------------------------------|
| <b>name</b>      | The name of the command that created the <i>mtable</i> , e.g. "survey".                 |
| <b>type</b>      | The type of the <i>mtable</i> , i.e. "survey".                                          |
| <b>title</b>     | The value of the command attribute <code>title</code> .                                 |
| <b>origin</b>    | The origin of the application that created the <i>mtable</i> , e.g. "MAD 1.0.0 OSX 64". |
| <b>date</b>      | The date of the creation of the <i>mtable</i> , e.g. "27/05/20".                        |
| <b>time</b>      | The time of the creation of the <i>mtable</i> , e.g. "19:18:36".                        |
| <b>refcol</b>    | The reference <i>column</i> for the <i>mtable</i> dictionary, e.g. "name".              |
| <b>direction</b> | The value of the command attribute <code>dir</code> .                                   |
| <b>observe</b>   | The value of the command attribute <code>observe</code> .                               |
| <b>implicit</b>  | The value of the command attribute <code>implicit</code> .                              |
| <b>misalign</b>  | The value of the command attribute <code>misalign</code> .                              |
| <b>range</b>     | The value of the command attribute <code>range</code> . <sup>4</sup>                    |
| <b>__seq</b>     | The <i>sequence</i> from the command attribute <code>sequence</code> . <sup>5</sup>     |

The core of the *mtable* contains the columns in the default order:

|              |                                                                           |
|--------------|---------------------------------------------------------------------------|
| <b>name</b>  | The name of the element.                                                  |
| <b>kind</b>  | The kind of the element.                                                  |
| <b>s</b>     | The <i>s</i> -position at the end of the element slice.                   |
| <b>l</b>     | The length from the start of the element to the end of the element slice. |
| <b>angle</b> | The angle from the start of the element to the end of the element slice.  |
| <b>tilt</b>  | The tilt of the element.                                                  |
| <b>x</b>     | The global coordinate <i>x</i> at the <i>s</i> -position.                 |

<sup>3</sup>The output of *mtable* in TFS files can be fully customized by the user.

<sup>4</sup>This field is not saved in the TFS table by default.

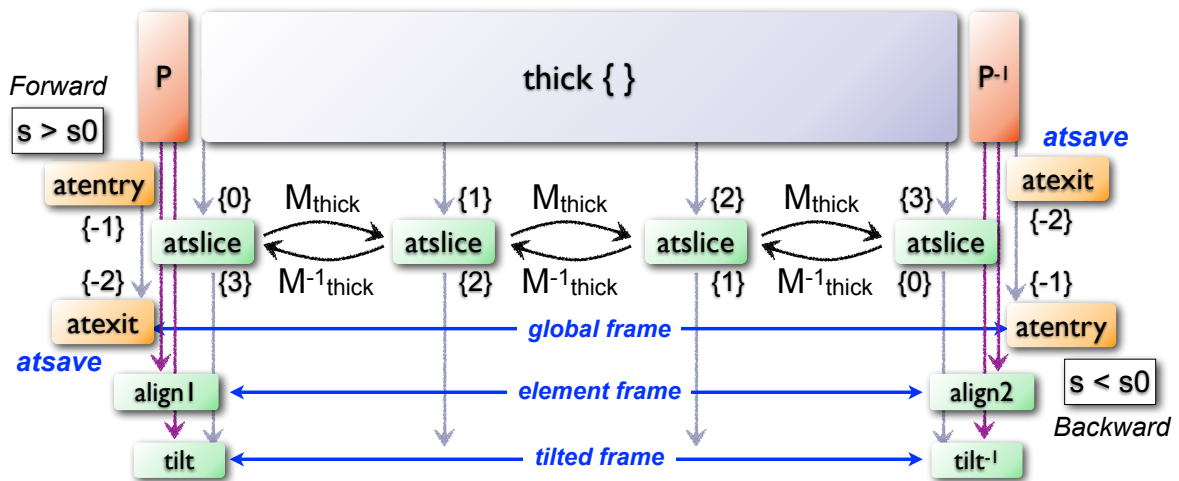
<sup>5</sup>Fields and columns starting with two underscores are protected data and never saved to TFS files.

|              |                                                                  |
|--------------|------------------------------------------------------------------|
| <b>y</b>     | The global coordinate $y$ at the $s$ -position.                  |
| <b>z</b>     | The global coordinate $z$ at the $s$ -position.                  |
| <b>theta</b> | The global angle $\theta$ at the $s$ -position.                  |
| <b>phi</b>   | The global angle $\phi$ at the $s$ -position.                    |
| <b>psi</b>   | The global angle $\psi$ at the $s$ -position.                    |
| <b>slc</b>   | The slice number ranging from $-2$ to $nslice$ .                 |
| <b>turn</b>  | The turn number.                                                 |
| <b>tdir</b>  | The $t$ -direction of the tracking in the element.               |
| <b>eidx</b>  | The index of the element in the sequence.                        |
| <b>__map</b> | The orientation <i>matrix</i> at the $s$ -position. <sup>5</sup> |

### 3 Geometrical tracking

The Figure 10.2 presents the scheme of the geometrical tracking through an element sliced with  $nslice=3$ . The actions `atentry` (index  $-1$ ), `atslice` (indexes  $0..3$ ), and `atexit` (index  $-2$ ) are reversed between the forward tracking ( $dir=1$  with increasing  $s$ -position) and the backward tracking ( $dir=-1$  with decreasing  $s$ -position). By default, the action `atsave` is attached to the exit slice, and hence it is also reversed in the backward tracking.

Figure 10.2: Geometrical tracking with slices.



#### 3.1 Slicing

The slicing can take three different forms:

- A *number* of the form  $nslice=N$  that specifies the number of slices with indexes  $0..N$ . This defines a uniform slicing with slice length  $l_{slice} = l_{elem}/N$ .
- An *iterable* of the form  $nslice=\{lw_1, lw_2, \dots, lw_N\}$  with  $\sum_i lw_i = 1$  that specifies the fraction of length of each slice with indexes  $0..N$  where  $N=\#nslice$ . This defines a non-uniform slicing with a slice length of  $l_i = lw_i \times l_{elem}$ .



- A *callable* (`elm`, `mflw`, `lw`) returning one of the two previous forms of slicing. The arguments are in order, the current element, the tracked map flow, and the length weight of the step, which should allow to return a user-defined element-specific slicing.

The surrounding  $P$  and  $P^{-1}$  maps represent the patches applied around the body of the element to change the frames, after the `atentry` and before the `atexit` actions:

- The misalignment of the element to move from the *global frame* to the *element frame* if the command attribute `misalign` is set to `true`.
- The tilt of the element to move from the element frame to the *titled frame* if the element attribute `tilt` is non-zero. The `atslice` actions take place in this frame.

These patches do not change the global frame per se, but they may affect the way that other components change the global frame, e.g. the tilt combined with the angle of a bending element.

### 3.2 Sub-elements

The `survey` command takes sub-elements into account, mainly for compatibility with the `track` command. In this case, the slicing specification is taken between sub-elements, e.g. 3 slices with 2 sub-elements gives a final count of 9 slices. It is possible to adjust the number of slices between sub-elements with the third form of slicing specifier, i.e. by using a callable where the length weight argument is between the current (or the end of the element) and the last sub-elements (or the start of the element).

## 4 Examples

TODO

# Chapter 11. Track

The track command provides a simple interface to the *dynamic* tracking code.<sup>1</sup> The dynamic tracking can be used to track the particles in the [local reference system](#) while running through the elements of a sequence. The particles coordinates can be expressed in the [global reference system](#) by changing from the local to the global frames using the information delivered by the [survey](#) command.

## 1 Command synopsis

The track command format is summarized in Figure 11.1, including the default setup of the attributes. The track command supports the following attributes:

|                 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
|-----------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>sequence</b> | The <i>sequence</i> to track. (no default, required).<br>Example: <code>sequence = lhcb1</code> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| <b>beam</b>     | The reference <i>beam</i> for the tracking. If no beam is provided, the command looks for a beam attached to the sequence, i.e. the attribute <code>seq.beam</code> . <sup>2</sup> (default: <code>nil</code> ).<br>Example: <code>beam = beam 'lhcb1' { beam-attributes }</code> .                                                                                                                                                                                                                                                                                                                                                                                                |
| <b>range</b>    | A <i>range</i> specifying the span of the sequence track. If no range is provided, the command looks for a range attached to the sequence, i.e. the attribute <code>seq.range</code> . (default: <code>nil</code> ).<br>Example: <code>range = "S.DS.L8.B1/E.DS.R8.B1"</code> .                                                                                                                                                                                                                                                                                                                                                                                                    |
| <b>dir</b>      | The <i>s</i> -direction of the tracking: 1 forward, -1 backward. (default: 1).<br>Example: <code>dir = -1</code> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| <b>s0</b>       | A <i>number</i> specifying the initial <i>s</i> -position offset. (default: 0 [m]).<br>Example: <code>s0 = 5000</code> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| <b>X0</b>       | A <i>mappable</i> (or a list of <i>mappable</i> ) specifying initial coordinates $\{x, px, y, py, t, pt\}$ , <i>damap</i> , or <i>beta block</i> for each tracked object, i.e. particle or <i>damap</i> . The <i>beta blocks</i> are converted to <i>damaps</i> , while the coordinates are converted to <i>damaps</i> only if <i>mapdef</i> is specified, but both will use <i>mapdef</i> to setup the <i>damap</i> constructor. Each tracked object may also contain a <i>beam</i> to override the reference beam, and a <i>logical nosave</i> to discard this object from being saved in the <i>htable</i> . (default: 0).<br>Example: <code>X0 = { x=1e-3, px=-1e-5 }</code> . |
| <b>00</b>       | A <i>mappable</i> specifying initial coordinates $\{x, px, y, py, t, pt\}$ of the reference orbit around which <i>X0</i> definitions take place. (default: 0).<br>Example: <code>00 = { x=1e-4, px=-2e-5, y=-2e-4, py=1e-5 }</code> .                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| <b>deltap</b>   | A <i>number</i> (or list of <i>number</i> ) specifying the initial $\delta_p$ to convert (using the beam) and add to the <i>pt</i> of each tracked particle or <i>damap</i> . (default: <code>nil</code> ).<br>Example: <code>s0 = 5000</code> .                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| <b>nturn</b>    | A <i>number</i> specifying the number of turn to track. (default: 1).<br>Example: <code>nturn = 2</code> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| <b>nstep</b>    | A <i>number</i> specifying the number of element to track. A negative value will track all elements. (default: -1).<br>Example: <code>nstep = 1</code> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |

---

<sup>1</sup>MAD-NG implements only two tracking codes denominated the *geometric* and the *dynamic* tracking.

<sup>2</sup>Initial coordinates *X0* may override it by providing per particle or *damap* beam.

**Figure 11.1:** Synopsis of the track command with default setup.

```

mtbl, mflw [, eid] = track {
 sequence=sequ, -- sequence (required)
 beam=nil, -- beam (or sequence.beam, required)
 range=nil, -- range of tracking (or sequence.range)
 dir=1, -- s-direction of tracking (1 or -1)
 s0=0, -- initial s-position offset [m]
 X0=0, -- initial coordinates (or damap(s), or beta block(s))
 O0=0, -- initial coordinates of reference orbit
 deltap=nil, -- initial deltap(s)
 nturn=1, -- number of turns to track
 nstep=-1, -- number of elements to track
 nslice=1, -- number of slices (or weights) for each element
 mapdef=false, -- setup for damap (or list of, true => {})
 method=2, -- method or order for integration (1 to 8)
 model='TKT', -- model for integration ('DKD' or 'TKT')
 ptcmodel=nil, -- use strict PTC thick model (override option)
 implicit=false, -- slice implicit elements too (e.g. plots)
 misalign=false, -- consider misalignment
 fringe=true, -- enable fringe fields (see element.flags.fringe)
 radiate=false, -- radiate at slices
 totalpath=false, -- variable 't' is the totalpath
 save=true, -- create mtable and save results
 title=nil, -- title of mtable (default seq.name)
 observe=1, -- save only in observed elements (every n turns)
 savesel=fnil, -- save selector (predicate)
 savemap=false, -- save damap in the column __map
 atentry=fnil, -- action called when entering an element
 atslice=fnil, -- action called after each element slices
 atexit=fnil, -- action called when exiting an element
 ataper=fnil, -- action called when checking for aperture
 atsave=fnil, -- action called when saving in mtable
 atdebug=fnil, -- action called when debugging the element maps
 info=nil, -- information level (output on terminal)
 debug=nil, -- debug information level (output on terminal)
 usrdef=nil, -- user defined data attached to the mflow
 mflow=nil, -- mflow, exclusive with other attributes except nstep
}

```

**nslice** A *number* specifying the number of slices or an *iterable* of increasing relative positions or a *callable* (`elm`, `mflw`, `lw`) returning one of the two previous kind of positions to track in the elements. The arguments of the callable are in order, the current element, the tracked map flow, and the length weight of the step. This attribute can be locally overridden by the element. (default: 1).

Example: `nslice = 5`.

**mapdef** A *logical* or a *damap* specification as defined by the [DAmapping](#) module to track DA maps

instead of particles coordinates. A value of `true` is equivalent to invoke the *damap* constructor with `{}` as argument. This attribute allows to track DA maps instead of particles. (default: `nil`).

Example: `mapdef = { xy=2, pt=5 }`.

|                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
|------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>method</b>    | A <i>number</i> specifying the order of integration from 1 to 8, or a <i>string</i> specifying a special method of integration. Odd orders are rounded to the next even order to select the corresponding Yoshida or Boole integration schemes. The special methods are <code>simple</code> (equiv. to DKD order 2), <code>collim</code> (equiv. to MKM order 2), and <code>teapot</code> (Teapot splitting order 2). (default: 2).<br>Example: <code>method = 'teapot'</code> .                                                   |
| <b>model</b>     | A <i>string</i> specifying the integration model, either <code>'DKD'</code> for <i>Drift-Kick-Drift</i> thin lens integration or <code>'TKT'</code> for <i>Thick-Kick-Thick</i> thick lens integration. <sup>3</sup> (default: <code>'TKT'</code> )<br>Example: <code>model = 'DKD'</code> .                                                                                                                                                                                                                                       |
| <b>ptcmodel</b>  | A <i>logical</i> indicating to use strict PTC model. <sup>4</sup> (default: <code>nil</code> )<br>Example: <code>ptcmodel = true</code> .                                                                                                                                                                                                                                                                                                                                                                                          |
| <b>implicit</b>  | A <i>logical</i> indicating that implicit elements must be sliced too, e.g. for smooth plotting. (default: <code>false</code> ).<br>Example: <code>implicit = true</code> .                                                                                                                                                                                                                                                                                                                                                        |
| <b>misalign</b>  | A <i>logical</i> indicating that misalignment must be considered. (default: <code>false</code> ).<br>Example: <code>misalign = true</code> .                                                                                                                                                                                                                                                                                                                                                                                       |
| <b>fringe</b>    | A <i>logical</i> indicating that fringe fields must be considered or a <i>number</i> specifying a bit mask to apply to all elements fringe flags defined by the element module. The value <code>true</code> is equivalent to the bit mask <code>-1</code> , i.e. allow all elements (default) fringe fields. (default: <code>true</code> ).<br>Example: <code>fringe = false</code> .                                                                                                                                              |
| <b>radiate</b>   | A <i>logical</i> enabling or disabling the radiation or a <i>string</i> specifying the type of radiation: <code>'average'</code> or <code>'quantum'</code> . The value <code>true</code> is equivalent to <code>'average'</code> . The value <code>'quantum+photon'</code> enables the tracking of emitted photons. (default: <code>false</code> ).<br>Example: <code>radiate = 'quantum'</code> .                                                                                                                                 |
| <b>totalpath</b> | A <i>logical</i> indicating to use the totalpath for the fifth variable <code>'t'</code> instead of the local path. (default: <code>false</code> ).<br>Example: <code>totalpath = true</code> .                                                                                                                                                                                                                                                                                                                                    |
| <b>save</b>      | A <i>logical</i> specifying to create a <i>mtable</i> and record tracking information at the observation points. The <code>save</code> attribute can also be a <i>string</i> specifying saving positions in the observed elements: <code>"atentry"</code> , <code>"atslice"</code> , <code>"atexit"</code> (i.e. <code>true</code> ), <code>"atbound"</code> (i.e. entry and exit), <code>"atbody"</code> (i.e. slices and exit) and <code>"atall"</code> . (default: <code>true</code> ).<br>Example: <code>save = false</code> . |
| <b>title</b>     | A <i>string</i> specifying the title of the <i>mtable</i> . If no title is provided, the command looks for the name of the sequence, i.e. the attribute <code>seq.name</code> . (default: <code>nil</code> ).<br>Example: <code>title = "track around IP5"</code> .                                                                                                                                                                                                                                                                |

<sup>3</sup>The TKT scheme (Yoshida) is automatically converted to the MKM scheme (Boole) when appropriate.

<sup>4</sup>In all cases, MAD-NG uses PTC `setup time=true`, `exact=true`.

- observe** A *number* specifying the observation points to consider for recording the tracking information. A zero value will consider all elements, while a positive value will consider selected elements only, checked with method `:is_observed`, every `observe > 0` turns. (default: 1).  
Example: `observe = 1`.
- savesel** A *callable* (`elm, mflw, lw, islc`) acting as a predicate on selected elements for observation, i.e. the element is discarded if the predicate returns `false`. The arguments are in order, the current element, the tracked map flow, the length weight of the slice and the slice index. (default: `fnil`)  
Example: `savesel = \e -> mylist[e.name] ~= nil`.
- savemap** A *logical* indicating to save the `damap` in the column `__map` of the *mtable*. (default: `false`).  
Example: `savemap = true`.
- atentry** A *callable* (`elm, mflw, 0, -1`) invoked at element entry. The arguments are in order, the current element, the tracked map flow, zero length and the slice index `-1`. (default: `fnil`).  
Example: `atentry = myaction`.
- atslice** A *callable* (`elm, mflw, lw, islc`) invoked at element slice. The arguments are in order, the current element, the tracked map flow, the length weight of the slice and the slice index. (default: `fnil`).  
Example: `atslice = myaction`.
- atexit** A *callable* (`elm, mflw, 0, -2`) invoked at element exit. The arguments are in order, the current element, the tracked map flow, zero length and the slice index `-2`. (default: `fnil`).  
Example: `atexit = myaction`.
- ataper** A *callable* (`elm, mflw, lw, islc`) invoked at element aperture checks, by default at last slice. The arguments are in order, the current element, the tracked map flow, the length weight of the slice and the slice index. If a particle or a `damap` hits the aperture, then its `status = "lost"` and it is removed from the list of tracked items. (default: `fnil`).  
Example: `ataper = myaction`.
- atsave** A *callable* (`elm, mflw, lw, islc`) invoked at element saving steps, by default at exit. The arguments are in order, the current element, the tracked map flow, the length weight of the slice and the slice index. (default: `fnil`).  
Example: `atsave = myaction`.
- atdebug** A *callable* (`elm, mflw, lw, [msg], [...]`) invoked at the entry and exit of element maps during the integration steps, i.e. within the slices. The arguments are in order, the current element, the tracked map flow, the length weight of the integration step and a *string* specifying a debugging message, e.g. `"map_name:0"` for entry and `":1"` for exit. If the level `debug ≥ 2` and `atdebug` is not specified, the default *function* `mdump` is used. In some cases, extra arguments could be passed to the method. (default: `fnil`).  
Example: `atdebug = myaction`.
- info** A *number* specifying the information level to control the verbosity of the output on the console. (default: `nil`).  
Example: `info = 2`.

|               |                                                                                                                                                                                                                                                                                                                                          |
|---------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>debug</b>  | A <i>number</i> specifying the debug level to perform extra assertions and to control the verbosity of the output on the console. (default: <code>nil</code> ).<br>Example: <code>debug = 2</code> .                                                                                                                                     |
| <b>usrdef</b> | Any user defined data that will be attached to the tracked map flow, which is internally passed to the elements method <code>:track</code> and to their underlying maps. (default: <code>nil</code> ).<br>Example: <code>usrdef = { myvar=somevalue }</code> .                                                                           |
| <b>mflow</b>  | A <i>mflow</i> containing the current state of a <code>track</code> command. If a map flow is provided, all attributes are discarded except <code>nstep</code> , <code>info</code> and <code>debug</code> , as the command was already set up upon its creation. (default: <code>nil</code> ).<br>Example: <code>mflow = mflow0</code> . |

The `track` command returns the following objects in this order:

|             |                                                                                                                                                                                      |
|-------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>mtbl</b> | A <i>htable</i> corresponding to the TFS table of the <code>track</code> command.                                                                                                    |
| <b>mflw</b> | A <i>mflow</i> corresponding to the map flow of the <code>track</code> command.                                                                                                      |
| <b>eidx</b> | An optional <i>number</i> corresponding to the last tracked element index in the sequence when <code>nstep</code> was specified and stopped the command before the end of the range. |

## 2 Track mtable

The `track` command returns a *htable* where the information described hereafter is the default list of fields written to the TFS files.<sup>5</sup>

The header of the *htable* contains the fields in the default order:

|                  |                                                                                         |
|------------------|-----------------------------------------------------------------------------------------|
| <b>name</b>      | The name of the command that created the <i>htable</i> , e.g. "track".                  |
| <b>type</b>      | The type of the <i>htable</i> , i.e. "track".                                           |
| <b>title</b>     | The value of the command attribute <code>title</code> .                                 |
| <b>origin</b>    | The origin of the application that created the <i>htable</i> , e.g. "MAD 1.0.0 OSX 64". |
| <b>date</b>      | The date of the creation of the <i>htable</i> , e.g. "27/05/20".                        |
| <b>time</b>      | The time of the creation of the <i>htable</i> , e.g. "19:18:36".                        |
| <b>refcol</b>    | The reference <i>column</i> for the <i>htable</i> dictionary, e.g. "name".              |
| <b>direction</b> | The value of the command attribute <code>dir</code> .                                   |
| <b>observe</b>   | The value of the command attribute <code>observe</code> .                               |
| <b>implicit</b>  | The value of the command attribute <code>implicit</code> .                              |
| <b>misalign</b>  | The value of the command attribute <code>misalign</code> .                              |
| <b>deltap</b>    | The value of the command attribute <code>deltap</code> .                                |
| <b>lost</b>      | The number of lost particle(s) or damap(s).                                             |
| <b>range</b>     | The value of the command attribute <code>range</code> . <sup>6</sup>                    |
| <b>—seq</b>      | The <i>sequence</i> from the command attribute <code>sequence</code> . <sup>7</sup>     |

<sup>5</sup>The output of *htable* in TFS files can be fully customized by the user.

<sup>6</sup>This field is not saved in the TFS table by default.

<sup>7</sup>Fields and columns starting with two underscores are protected data and never saved to TFS files.

The core of the *mtable* contains the columns in the default order:

|               |                                                                           |
|---------------|---------------------------------------------------------------------------|
| <b>name</b>   | The name of the element.                                                  |
| <b>kind</b>   | The kind of the element.                                                  |
| <b>s</b>      | The <i>s</i> -position at the end of the element slice.                   |
| <b>l</b>      | The length from the start of the element to the end of the element slice. |
| <b>id</b>     | The index of the particle or damap as provided in X0.                     |
| <b>x</b>      | The local coordinate <i>x</i> at the <i>s</i> -position.                  |
| <b>px</b>     | The local coordinate $p_x$ at the <i>s</i> -position.                     |
| <b>y</b>      | The local coordinate <i>y</i> at the <i>s</i> -position.                  |
| <b>py</b>     | The local coordinate $p_y$ at the <i>s</i> -position.                     |
| <b>t</b>      | The local coordinate <i>t</i> at the <i>s</i> -position.                  |
| <b>pt</b>     | The local coordinate $p_t$ at the <i>s</i> -position.                     |
| <b>slc</b>    | The slice index ranging from -2 to <code>nslice</code> .                  |
| <b>turn</b>   | The turn number.                                                          |
| <b>tdir</b>   | The <i>t</i> -direction of the tracking in the element.                   |
| <b>eidx</b>   | The index of the element in the sequence.                                 |
| <b>status</b> | The status of the particle or damap.                                      |
| <b>__map</b>  | The damap at the <i>s</i> -position. <sup>3</sup>                         |

### 3 Dynamical tracking

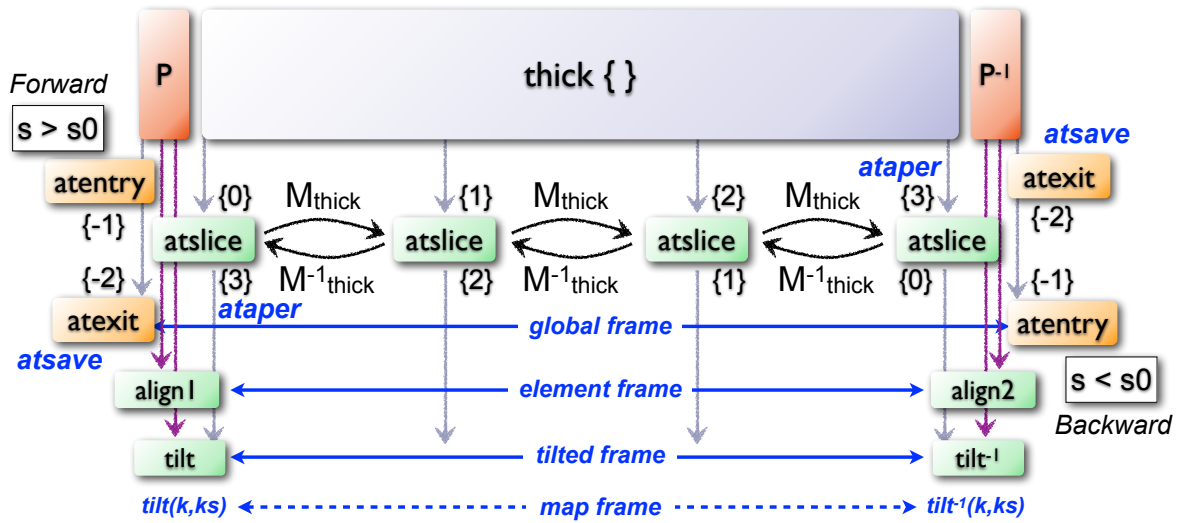
The Figure 11.2 presents the scheme of the dynamical tracking through an element sliced with `nslice=3`. The actions `atentry` (index -1), `atslice` (indexes 0..3), and `atexit` (index -2) are reversed between the forward tracking (`dir=1` with increasing *s*-position) and the backward tracking (`dir=-1` with decreasing *s*-position). By default, the action `atsave` is attached to the exit slice and the action `ataper` is attached to the last slice just before exit, i.e. to the last `atslice` action in the tilted frame, and hence they are also both reversed in the backward tracking.

#### 3.1 Slicing

The slicing can take three different forms:

- A *number* of the form `nslice=N` that specifies the number of slices with indexes 0..N. This defines a uniform slicing with slice length  $l_{\text{slice}} = l_{\text{elem}}/N$ .
- An *iterable* of the form `nslice={lw1,lw2,...,lwN}` with  $\sum_i lw_i = 1$  that specifies the fraction of length of each slice with indexes 0..N where  $N=\#\text{nslice}$ . This defines a non-uniform slicing with a slice length of  $l_i = lw_i \times l_{\text{elem}}$ .
- A *callable* (`elm`, `mflw`, `lw`) returning one of the two previous forms of slicing. The arguments are in order, the current element, the tracked map flow, and the length weight of the step, which should allow to return a user-defined element-specific slicing.

Figure 11.2: Dynamical tracking with slices.



The surrounding  $P$  and  $P^{-1}$  maps represent the patches applied around the body of the element to change the frames, after the `atentry` and before the `atexit` actions:

- The misalignment of the element to move from the *global frame* to the *element frame* if the command attribute `misalign` is set to `true`.
- The tilt of the element to move from the *element frame* to the *tilted frame* if the element attribute `tilt` is non-zero. The `atslice` actions take place in this frame.

The *map frame* is specific to some maps while tracking through the body of the element. In principle, the *map frame* is not visible to the user, only to the integrator. For example, a quadrupole with both  $k_1$  and  $k_{1s}$  defined will have a *map frame* tilted by the angle  $\alpha = -\frac{1}{2} \tan^{-1} \frac{k_{1s}}{k_1}$  attached to its thick map, i.e. the focusing matrix handling only  $\tilde{k}_1 = \sqrt{k_1^2 + k_{1s}^2}$ , but not to its thin map, i.e. the kick from all multipoles (minus  $k_1$  and  $k_{1s}$ ) expressed in the *tilted frame*, during the integration steps.

### 3.2 Sub-elements

The `track` command takes sub-elements into account. In this case, the slicing specification is taken between sub-elements, e.g. 3 slices with 2 sub-elements gives a final count of 9 slices. It is possible to adjust the number of slices between sub-elements with the third form of slicing specifier, i.e. by using a callable where the length weight argument is between the current (or the end of the element) and the last sub-elements (or the start of the element).

### 3.3 Particles status

The `track` command initializes the map flow with particles or damaps or both, depending on the attributes `X0` and `mapdef`. The `status` attribute of each particle or damap will be set to one of "Xset", "Bset" and "Mset" to track the origin of its initialization: coordinates, beta block, or damap. After the tracking, some particles or damaps may have the status "lost" and their number being recorded in the counter `lost` from TFS table header. Other commands like `cofind` or `twiss` may add extra tags to the status value, like "stable", "unstable" and "singular".



**4 Examples**

TODO

# Chapter 12. Cofind

The `cofind` command (i.e. closed orbit finder) provides a simple interface to find a closed orbit using the Newton algorithm on top of the `track` command.

## 1 Command synopsis

The `cofind` command format is summarized in Figure 12.1, including the default setup of the attributes. Most of these attributes are set to `nil` by default, meaning that `cofind` relies on the `track` command defaults. The `cofind` command supports the following attributes:

|                 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
|-----------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>sequence</b> | The <i>sequence</i> to track. (no default, required).<br>Example: <code>sequence = lhcb1</code> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| <b>beam</b>     | The reference <i>beam</i> for the tracking. If no beam is provided, the command looks for a beam attached to the sequence, i.e. the attribute <code>seq.beam</code> . <sup>1</sup> (default: <code>nil</code> ).<br>Example: <code>beam = beam 'lhcb1' { beam-attributes }</code> .                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| <b>range</b>    | A <i>range</i> specifying the span of the sequence track. If no range is provided, the command looks for a range attached to the sequence, i.e. the attribute <code>seq.range</code> . (default: <code>nil</code> ).<br>Example: <code>range = "S.DS.L8.B1/E.DS.R8.B1"</code> .                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| <b>dir</b>      | The <i>s</i> -direction of the tracking: 1 forward, -1 backward. (default: <code>nil</code> ).<br>Example: <code>dir = -1</code> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| <b>s0</b>       | A <i>number</i> specifying the initial <i>s</i> -position offset. (default: <code>nil</code> ).<br>Example: <code>s0 = 5000</code> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| <b>X0</b>       | A <i>mappable</i> (or a list of <i>mappable</i> ) specifying initial coordinates $\{x, px, y, py, t, pt\}$ , <i>damap</i> , or <i>beta block</i> for each tracked object, i.e. <i>particle</i> or <i>damap</i> . The <i>beta blocks</i> are converted to <i>damaps</i> , while the coordinates are converted to <i>damaps</i> only if <code>mapdef</code> is specified, but both will use <code>mapdef</code> to setup the <i>damap</i> constructor. Each tracked object may also contain a <i>beam</i> to override the reference beam, and a <i>logical</i> <code>nosave</code> to discard this object from being saved in the <code>htable</code> . (default: <code>nil</code> ).<br>Example: <code>X0 = { x=1e-3, px=-1e-5 }</code> . |
| <b>00</b>       | A <i>mappable</i> specifying initial coordinates $\{x, px, y, py, t, pt\}$ of the reference orbit around which <code>X0</code> definitions take place. (default: <code>0</code> ).<br>Example: <code>00 = { x=1e-4, px=-2e-5, y=-2e-4, py=1e-5 }</code> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| <b>deltap</b>   | A <i>number</i> (or list of <i>number</i> ) specifying the initial $\delta_p$ to convert (using the beam) and add to the <code>pt</code> of each tracked particle or <i>damap</i> . (default: <code>nil</code> ).<br>Example: <code>s0 = 5000</code> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| <b>nturn</b>    | A <i>number</i> specifying the number of turn to track. (default: <code>nil</code> ).<br>Example: <code>nturn = 2</code> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| <b>nstep</b>    | A <i>number</i> specifying the number of element to track. A negative value will track all elements. (default: <code>nil</code> ).<br>Example: <code>nstep = 1</code> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |

---

<sup>1</sup>Initial coordinates `X0` may override it by providing a beam per particle or *damap*.

Figure 12.1: Synopsis of the cofind command with default setup.

```

mtbl, mflw = cofind {
 sequence=sequ, -- sequence (required)
 beam=nil, -- beam (or sequence.beam, required)
 range=nil, -- range of tracking (or sequence.range)
 dir=nil, -- s-direction of tracking (1 or -1)
 s0=nil, -- initial s-position offset [m]
 X0=nil, -- initial coordinates (or damap, or beta block)
 O0=nil, -- initial coordinates of reference orbit
 deltap=nil, -- initial deltap(s)
 nturn=nil, -- number of turns to track
 nslice=nil, -- number of slices (or weights) for each element
 mapdef=true, -- setup for damap (or list of, true => {})
 method=nil, -- method or order for integration (1 to 8)
 model=nil, -- model for integration ('DKD' or 'TKT')
 ptcmodel=nil, -- use strict PTC thick model (override option)
 implicit=nil, -- slice implicit elements too (e.g. plots)
 misalign=nil, -- consider misalignment
 fringe=nil, -- enable fringe fields (see element.flags.fringe)
 radiate=nil, -- radiate at slices
 totalpath=nil, -- variable 't' is the totalpath
 save=false, -- create mtable and save results
 title=nil, -- title of mtable (default seq.name)
 observe=nil, -- save only in observed elements (every n turns)
 savesel=nil, -- save selector (predicate)
 savemap=nil, -- save damap in the column __map
 atentry=nil, -- action called when entering an element
 atslice=nil, -- action called after each element slices
 atexit=nil, -- action called when exiting an element
 ataper=nil, -- action called when checking for aperture
 atsave=nil, -- action called when saving in mtable
 atdebug=fnil, -- action called when debugging the element maps
 cotol=1e-8, -- closed orbit tolerance (i.e. |dX|)
 coiter=20, -- maximum number of iterations
 codiff=1e-10, -- finite differences step for jacobian
 comethod='newton', -- default algorithm
 Xl=0, -- optional final coordinates translation
 info=nil, -- information level (output on terminal)
 debug=nil, -- debug information level (output on terminal)
 usrdef=nil, -- user defined data attached to the mflow
 mflow=nil, -- mflow, exclusive with other attributes
}

```

**nslice**

A *number* specifying the number of slices or an *iterable* of increasing relative positions or a *callable* (elm, mflw, lw) returning one of the two previous kind of positions to track in the elements. The arguments of the callable are in order, the current element, the tracked map flow, and the length weight of the step. This attribute can be locally

- overridden by the element. (default: nil).  
Example: `nslice = 5`.
- mapdef** A *logical* or a *damap* specification as defined by the [DAmap](#) module to track DA maps instead of particles coordinates. A value of `true` is equivalent to invoke the *damap* constructor with `{}` as argument. A value of `false` or `nil` disable the use of *damaps* and force `cofind` to replace each particles or *damaps* by seven particles to approximate their Jacobian by finite difference. (default: `true`).  
Example: `mapdef = { xy=2, pt=5 }`.
- method** A *number* specifying the order of integration from 1 to 8, or a *string* specifying a special method of integration. Odd orders are rounded to the next even order to select the corresponding Yoshida or Boole integration schemes. The special methods are `simple` (equiv. to DKD order 2), `collim` (equiv. to MKM order 2), and `teapot` (Teapot splitting order 2). (default: `nil`).  
Example: `method = 'teapot'`.
- model** A *string* specifying the integration model, either `'DKD'` for *Drift-Kick-Drift* thin lens integration or `'TKT'` for *Thick-Kick-Thick* thick lens integration.<sup>2</sup> (default: `nil`)  
Example: `model = 'DKD'`.
- ptcmodel** A *logical* indicating to use strict PTC model.<sup>3</sup> (default: `nil`)  
Example: `ptcmodel = true`.
- implicit** A *logical* indicating that implicit elements must be sliced too, e.g. for smooth plotting. (default: `nil`).  
Example: `implicit = true`.
- misalign** A *logical* indicating that misalignment must be considered. (default: `nil`).  
Example: `misalign = true`.
- fringe** A *logical* indicating that fringe fields must be considered or a *number* specifying a bit mask to apply to all elements fringe flags defined by the element module. The value `true` is equivalent to the bit mask `-1`, i.e. allow all elements (default) fringe fields. (default: `nil`).  
Example: `fringe = false`.
- radiate** A *logical* enabling or disabling the radiation or the *string* specifying the *'average'* type of radiation. The value `true` is equivalent to *'average'* and the value *'quantum'* is converted to *'average'*. (default: `nil`).  
Example: `radiate = 'average'`.
- totalpath** A *logical* indicating to use the totalpath for the fifth variable *'t'* instead of the local path. (default: `nil`).  
Example: `totalpath = true`.
- save** A *logical* specifying to create a *htable* and record tracking information at the observation points. The `save` attribute can also be a *string* specifying saving positions in the observed elements: `"atentry"`, `"atslice"`, `"atexit"` (i.e. `true`), `"atbound"` (i.e. entry and exit), `"atbody"` (i.e. slices and exit) and `"atall"`. (default: `false`).  
Example: `save = false`.

<sup>2</sup>The TKT scheme (Yoshida) is automatically converted to the MKM scheme (Boole) when appropriate.

<sup>3</sup>In all cases, MAD-NG uses PTC setup `time=true`, `exact=true`.

- title** A *string* specifying the title of the *mtable*. If no title is provided, the command looks for the name of the sequence, i.e. the attribute `seq.name`. (default: `nil`).  
Example: `title = "track around IP5"`.
- observe** A *number* specifying the observation points to consider for recording the tracking information. A zero value will consider all elements, while a positive value will consider selected elements only, checked with method `:is_observed`, every `observe > 0` turns. (default: `nil`).  
Example: `observe = 1`.
- savesel** A *callable* (`elm, mflw, lw, islc`) acting as a predicate on selected elements for observation, i.e. the element is discarded if the predicate returns `false`. The arguments are in order, the current element, the tracked map flow, the length weight of the slice and the slice index. (default: `nil`)  
Example: `savesel = \e -> mylist[e.name] ~= nil`.
- savemap** A *logical* indicating to save the `damap` in the column `__map` of the *mtable*. (default: `nil`).  
Example: `savemap = true`.
- atentry** A *callable* (`elm, mflw, 0, -1`) invoked at element entry. The arguments are in order, the current element, the tracked map flow, zero length and the slice index `-1`. (default: `nil`).  
Example: `atentry = myaction`.
- atslice** A *callable* (`elm, mflw, lw, islc`) invoked at element slice. The arguments are in order, the current element, the tracked map flow, the length weight of the slice and the slice index. (default: `nil`).  
Example: `atslice = myaction`.
- atexit** A *callable* (`elm, mflw, 0, -2`) invoked at element exit. The arguments are in order, the current element, the tracked map flow, zero length and the slice index `-2`. (default: `nil`).  
Example: `atexit = myaction`.
- ataper** A *callable* (`elm, mflw, lw, islc`) invoked at element aperture checks, by default at last slice. The arguments are in order, the current element, the tracked map flow, the length weight of the slice and the slice index. If a particle or a `damap` hits the aperture, then its `status = "lost"` and it is removed from the list of tracked items. (default: `fnil`).  
Example: `ataper = myaction`.
- atsave** A *callable* (`elm, mflw, lw, islc`) invoked at element saving steps, by default at exit. The arguments are in order, the current element, the tracked map flow, the length weight of the slice and the slice index. (default: `nil`).  
Example: `atsave = myaction`.
- atdebug** A *callable* (`elm, mflw, lw, [msg], [...]`) invoked at the entry and exit of element maps during the integration steps, i.e. within the slices. The arguments are in order, the current element, the tracked map flow, the length weight of the integration step and a *string* specifying a debugging message, e.g. `"map_name:0"` for entry and `":1"` for exit. If the level `debug ≥ 2` and `atdebug` is not specified, the default *function* `mdump` is used. In some cases, extra arguments could be passed to the method. (default: `fnil`).  
Example: `atdebug = myaction`.

|                 |                                                                                                                                                                                                                                                                                                                      |
|-----------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>cotol</b>    | A <i>number</i> specifying the closed orbit tolerance. If all coordinates update of a particle or a damap are smaller than <code>cotol</code> , then it is tagged as "stable". (default: $1e-8$ ).<br>Example: <code>cotol = 1e-6</code> .                                                                           |
| <b>coiter</b>   | A <i>number</i> specifying the maximum number of iteration. If this threshold is reached, all the remaining tracked objects are tagged as "unstable". (default: 20).<br>Example: <code>coiter = 5</code> .                                                                                                           |
| <b>codiff</b>   | A <i>number</i> specifying the finite difference step to approximate the Jacobian when damaps are disabled. If <code>codiff</code> is larger than $100 \times \text{cotol}$ , it will be adjusted to $\text{cotol}/100$ and a warning will be emitted. (default: $1e-10$ ).<br>Example: <code>codiff = 1e-8</code> . |
| <b>comethod</b> | A <i>string</i> specifying the method to use for searching the closed orbit. So far, the only supported method is the default. (default: 'newton').<br>Example: <code>comethod = 'mymethod'</code> .                                                                                                                 |
| <b>X1</b>       | A <i>mappable</i> specifying the coordinates <code>{x, px, y, py, t, pt}</code> to <i>subtract</i> to the final coordinates of the particles or the damaps. (default: 0).<br>Example: <code>X1 = { t=100, pt=10 }</code> .                                                                                           |
| <b>info</b>     | A <i>number</i> specifying the information level to control the verbosity of the output on the console. (default: nil).<br>Example: <code>info = 2</code> .                                                                                                                                                          |
| <b>debug</b>    | A <i>number</i> specifying the debug level to perform extra assertions and to control the verbosity of the output on the console. (default: nil).<br>Example: <code>debug = 2</code> .                                                                                                                               |
| <b>usrdef</b>   | Any user defined data that will be attached to the tracked map flow, which is internally passed to the elements method <code>:track</code> and to their underlying maps. (default: nil).<br>Example: <code>usrdef = { myvar=somevalue }</code> .                                                                     |
| <b>mflow</b>    | A <i>mflow</i> containing the current state of a track command. If a map flow is provided, all attributes are discarded except <code>nstep</code> , <code>info</code> and <code>debug</code> , as the command was already set up upon its creation. (default: nil).<br>Example: <code>mflow = mflow0</code> .        |

The `cofind` command stops when all particles or damap are tagged as "stable", "unstable", "singular" or "lost". The `cofind` command returns the following objects in this order:

|             |                                                                                                                                                                                            |
|-------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>mtbl</b> | A <i>mtable</i> corresponding to the TFS table of the track command where the status column may also contain the new values "stable", "unstable" or "singular".                            |
| <b>mflw</b> | A <i>mflow</i> corresponding to the map flow of the track command. The particles or damaps status are tagged and ordered by "stable", "unstable", "singular", "lost" and <code>id</code> . |

## 2 Cofind mtable

The `cofind` command returns the track *mtable* unmodified except for the status column. The tracked objects `id` will appear once per iteration at the `$end` marker, and other defined observation points if any, until they are removed from the list of tracked objects.

**3 Examples**

TODO

# Chapter 13. Twiss

The `twiss` command provides a simple interface to compute the optical functions around an orbit on top of the `track` command, and the `cofind` command if the search for closed orbits is requested.

## 1 Command synopsis

The `twiss` command format is summarized in Figure 13.1, including the default setup of the attributes. Most of these attributes are set to `nil` by default, meaning that `twiss` relies on the `track` and the `cofind` commands defaults. The `twiss` command supports the following attributes:

|                 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
|-----------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>sequence</b> | The <i>sequence</i> to track. (no default, required).<br>Example: <code>sequence = lhcb1</code> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| <b>beam</b>     | The reference <i>beam</i> for the tracking. If no beam is provided, the command looks for a beam attached to the sequence, i.e. the attribute <code>seq.beam</code> . <sup>1</sup> (default: <code>nil</code> ).<br>Example: <code>beam = beam 'lhcb1' { beam-attributes }</code> .                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| <b>range</b>    | A <i>range</i> specifying the span of the sequence track. If no range is provided, the command looks for a range attached to the sequence, i.e. the attribute <code>seq.range</code> . (default: <code>nil</code> ).<br>Example: <code>range = "S.DS.L8.B1/E.DS.R8.B1"</code> .                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| <b>dir</b>      | The <i>s</i> -direction of the tracking: 1 forward, -1 backward. (default: <code>nil</code> ).<br>Example: <code>dir = -1</code> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| <b>s0</b>       | A <i>number</i> specifying the initial <i>s</i> -position offset. (default: <code>nil</code> ).<br>Example: <code>s0 = 5000</code> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| <b>X0</b>       | A <i>mappable</i> (or a list of <i>mappable</i> ) specifying initial coordinates $\{x, px, y, py, t, pt\}$ , <i>damap</i> , or <i>beta block</i> for each tracked object, i.e. <i>particle</i> or <i>damap</i> . The <i>beta blocks</i> are converted to <i>damaps</i> , while the coordinates are converted to <i>damaps</i> only if <code>mapdef</code> is specified, but both will use <code>mapdef</code> to setup the <i>damap</i> constructor. Each tracked object may also contain a <i>beam</i> to override the reference beam, and a <i>logical</i> <code>nosave</code> to discard this object from being saved in the <i>htable</i> . (default: <code>0</code> ).<br>Example: <code>X0 = { x=1e-3, px=-1e-5 }</code> . |
| <b>00</b>       | A <i>mappable</i> specifying initial coordinates $\{x, px, y, py, t, pt\}$ of the reference orbit around which <i>X0</i> definitions take place. (default: <code>0</code> ).<br>Example: <code>00 = { x=1e-4, px=-2e-5, y=-2e-4, py=1e-5 }</code> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| <b>deltap</b>   | A <i>number</i> (or list of <i>number</i> ) specifying the initial $\delta_p$ to convert (using the beam) and add to the <i>pt</i> of each tracked particle or <i>damap</i> . (default: <code>nil</code> ).<br>Example: <code>s0 = 5000</code> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| <b>chrom</b>    | A <i>logical</i> specifying to calculate the chromatic functions by finite different using an extra $\delta_p = 1e-6$ . (default: <code>false</code> ).<br>Example: <code>chrom = true</code> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| <b>coupling</b> | A <i>logical</i> specifying to calculate the optical functions for coupling terms in the normalized forms. (default: <code>false</code> ).<br>Example: <code>chrom = true</code> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |

---

<sup>1</sup>Initial coordinates *X0* may override it by providing a beam per particle or *damap*.



**Figure 13.1:** Synopsis of the twiss command with default setup.

```

mtbl, mflow [, eidX] = twiss {
 sequence=sequ, -- sequence (required)
 beam=nil, -- beam (or sequence.beam, required)
 range=nil, -- range of tracking (or sequence.range)
 dir=nil, -- s-direction of tracking (1 or -1)
 s0=nil, -- initial s-position offset [m]
 X0=nil, -- initial coordinates (or damap(s), or beta block(s))
 O0=nil, -- initial coordinates of reference orbit
 deltap=nil, -- initial deltap(s)
 chrom=false, -- compute chromatic functions by finite difference
 coupling=false, -- compute optical functions for non-diagonal modes
 nturn=nil, -- number of turns to track
 nstep=nil, -- number of elements to track
 nslice=nil, -- number of slices (or weights) for each element
 mapdef=true, -- setup for damap (or list of, true => {})
 method=nil, -- method or order for integration (1 to 8)
 model=nil, -- model for integration ('DKD' or 'TKT')
 ptcmodel=nil, -- use strict PTC thick model (override option)
 implicit=nil, -- slice implicit elements too (e.g. plots)
 misalign=nil, -- consider misalignment
 fringe=nil, -- enable fringe fields (see element.flags.fringe)
 radiate=nil, -- radiate at slices
 totalpath=nil, -- variable 't' is the totalpath
 save=true, -- create mtable and save results
 title=nil, -- title of mtable (default seq.name)
 observe=0, -- save only in observed elements (every n turns)
 savesel=nil, -- save selector (predicate)
 savemap=nil, -- save damap in the column __map
 atentry=nil, -- action called when entering an element
 atslice=nil, -- action called after each element slices
 atexit=nil, -- action called when exiting an element
 ataper=nil, -- action called when checking for aperture
 atsave=nil, -- action called when saving in mtable
 atdebug=fnil, -- action called when debugging the element maps
 cofind=false, -- search for closed orbit (default is false!)
 cotol=nil, -- closed orbit tolerance (i.e. |dX|)
 coiter=nil, -- maximum number of iterations
 codiff=nil, -- finite differences step for jacobian
 comethod=nil, -- default algorithm
 X1=nil, -- optional final coordinates translation
 info=nil, -- information level (output on terminal)
 debug=nil, -- debug information level (output on terminal)
 usrdef=nil, -- user defined data attached to the mflow
 mflow=nil, -- mflow, exclusive with other attributes
}

```

|                 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
|-----------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>nturn</b>    | A <i>number</i> specifying the number of turn to track. (default: <code>nil</code> ).<br>Example: <code>nturn = 2</code> .                                                                                                                                                                                                                                                                                                                                                                                                  |
| <b>nstep</b>    | A <i>number</i> specifying the number of element to track. A negative value will track all elements. (default: <code>nil</code> ).<br>Example: <code>nstep = 1</code> .                                                                                                                                                                                                                                                                                                                                                     |
| <b>nslice</b>   | A <i>number</i> specifying the number of slices or an <i>iterable</i> of increasing relative positions or a <i>callable</i> ( <code>elm</code> , <code>mflw</code> , <code>lw</code> ) returning one of the two previous kind of positions to track in the elements. The arguments of the callable are in order, the current element, the tracked map flow, and the length weight of the step. This attribute can be locally overridden by the element. (default: <code>nil</code> ).<br>Example: <code>nslice = 5</code> . |
| <b>mapdef</b>   | A <i>logical</i> or a <i>damap</i> specification as defined by the <a href="#">DAmapping</a> module to track DA maps instead of particles coordinates. A value of <code>true</code> is equivalent to invoke the <i>damap</i> constructor with <code>{}</code> as argument. A value of <code>false</code> or <code>nil</code> will be internally forced to <code>true</code> for the tracking of the normalized forms. (default: <code>true</code> ).<br>Example: <code>mapdef = { xy=2, pt=5 }</code> .                     |
| <b>method</b>   | A <i>number</i> specifying the order of integration from 1 to 8, or a <i>string</i> specifying a special method of integration. Odd orders are rounded to the next even order to select the corresponding Yoshida or Boole integration schemes. The special methods are <code>simple</code> (equiv. to DKD order 2), <code>collim</code> (equiv. to MKM order 2), and <code>teapot</code> (Teapot splitting order 2). (default: <code>nil</code> ).<br>Example: <code>method = 'teapot'</code> .                            |
| <b>model</b>    | A <i>string</i> specifying the integration model, either <code>'DKD'</code> for <i>Drift-Kick-Drift</i> thin lens integration or <code>'TKT'</code> for <i>Thick-Kick-Thick</i> thick lens integration. <sup>2</sup> (default: <code>nil</code> )<br>Example: <code>model = 'DKD'</code> .                                                                                                                                                                                                                                  |
| <b>ptcmodel</b> | A <i>logical</i> indicating to use strict PTC model. <sup>3</sup> (default: <code>nil</code> )<br>Example: <code>ptcmodel = true</code> .                                                                                                                                                                                                                                                                                                                                                                                   |
| <b>implicit</b> | A <i>logical</i> indicating that implicit elements must be sliced too, e.g. for smooth plotting. (default: <code>nil</code> ).<br>Example: <code>implicit = true</code> .                                                                                                                                                                                                                                                                                                                                                   |
| <b>misalign</b> | A <i>logical</i> indicating that misalignment must be considered. (default: <code>nil</code> ).<br>Example: <code>misalign = true</code> .                                                                                                                                                                                                                                                                                                                                                                                  |
| <b>fringe</b>   | A <i>logical</i> indicating that fringe fields must be considered or a <i>number</i> specifying a bit mask to apply to all elements fringe flags defined by the element module. The value <code>true</code> is equivalent to the bit mask <code>-1</code> , i.e. allow all elements (default) fringe fields. (default: <code>nil</code> ).<br>Example: <code>fringe = false</code> .                                                                                                                                        |
| <b>radiate</b>  | A <i>logical</i> enabling or disabling the radiation or the <i>string</i> specifying the <code>'average'</code> type of radiation during the closed orbit search. The value <code>true</code> is equivalent to <code>'average'</code> and the value <code>'quantum'</code> is converted to <code>'average'</code> . (default: <code>nil</code> ).<br>Example: <code>radiate = 'average'</code> .                                                                                                                            |

---

<sup>2</sup>The TKT scheme (Yoshida) is automatically converted to the MKM scheme (Boole) when appropriate.

<sup>3</sup>In all cases, MAD-NG uses PTC setup `time=true`, `exact=true`.

- totalpath** A *logical* indicating to use the totalpath for the fifth variable 't' instead of the local path. (default: nil).  
Example: totalpath = true.
- save** A *logical* specifying to create a *mtable* and record tracking information at the observation points. The save attribute can also be a *string* specifying saving positions in the observed elements: "atentry", "atslice", "atexit" (i.e. true), "atbound" (i.e. entry and exit), "atbody" (i.e. slices and exit) and "atall". (default: false).  
Example: save = false.
- title** A *string* specifying the title of the *mtable*. If no title is provided, the command looks for the name of the sequence, i.e. the attribute seq.name. (default: nil).  
Example: title = "track around IP5".
- observe** A *number* specifying the observation points to consider for recording the tracking information. A zero value will consider all elements, while a positive value will consider selected elements only, checked with method :is\_observed, every observe > 0 turns. (default: nil).  
Example: observe = 1.
- savesel** A *callable* (elm, mflw, lw, islc) acting as a predicate on selected elements for observation, i.e. the element is discarded if the predicate returns false. The arguments are in order, the current element, the tracked map flow, the length weight of the slice and the slice index. (default: fnil)  
Example: savesel = \e -> mylist[e.name] ~= nil.
- savemap** A *logical* indicating to save the damap in the column \_\_map of the *mtable*. (default: nil).  
Example: savemap = true.
- atentry** A *callable* (elm, mflw, 0, -1) invoked at element entry. The arguments are in order, the current element, the tracked map flow, zero length and the slice index -1. (default: fnil).  
Example: atentry = myaction.
- atslice** A *callable* (elm, mflw, lw, islc) invoked at element slice. The arguments are in order, the current element, the tracked map flow, the length weight of the slice and the slice index. (default: fnil).  
Example: atslice = myaction.
- atexit** A *callable* (elm, mflw, 0, -2) invoked at element exit. The arguments are in order, the current element, the tracked map flow, zero length and the slice index -2. (default: fnil).  
Example: atexit = myaction.
- ataper** A *callable* (elm, mflw, lw, islc) invoked at element aperture checks, by default at last slice. The arguments are in order, the current element, the tracked map flow, the length weight of the slice and the slice index. If a particle or a damap hits the aperture, then its status = "lost" and it is removed from the list of tracked items. (default: fnil).  
Example: ataper = myaction.
- atsave** A *callable* (elm, mflw, lw, islc) invoked at element saving steps, by default at exit. The arguments are in order, the current element, the tracked map flow, the length weight

of the slice and the slice index. (default: `fnil`).

Example: `atsave = myaction`.

- atdebug** A *callable* (`elm`, `mflow`, `lw`, [`msg`], [...]) invoked at the entry and exit of element maps during the integration steps, i.e. within the slices. The arguments are in order, the current element, the tracked map flow, the length weight of the integration step and a *string* specifying a debugging message, e.g. "`map_name:0`" for entry and "`:1`" for exit. If the level `debug`  $\geq 2$  and `atdebug` is not specified, the default *function* `mdump` is used. In some cases, extra arguments could be passed to the method. (default: `fnil`).  
Example: `atdebug = myaction`.
- cofind** A *logical* indicating to search for the closed orbit. (default: `false`).  
Example: `cofind = true`.
- cotol** A *number* specifying the closed orbit tolerance. If all coordinates update of a particle or a damap are smaller than `cotol`, then it is tagged as "`stable`". (default: `1e-8`).  
Example: `cotol = 1e-6`.
- coiter** A *number* specifying the maximum number of iteration. If this threshold is reached, all the remaining tracked objects are tagged as "`unstable`". (default: `20`).  
Example: `coiter = 5`.
- codiff** A *number* specifying the finite difference step to approximate the Jacobian when damaps are disabled. If `codiff` is larger than  $100 \times \text{cotol}$ , it will be adjusted to `cotol/100` and a warning will be emitted. (default: `1e-10`).  
Example: `codiff = 1e-8`.
- comethod** A *string* specifying the method to use for searching the closed orbit. So far, the only supported method is the default. (default: `'newton'`).  
Example: `comethod = 'mymethod'`.
- X1** A *mappable* specifying the coordinates `{x, px, y, py, t, pt}` to *subtract* to the final coordinates of the particles or the damaps. (default: `0`).  
Example: `X1 = { t=100, pt=10 }`.
- info** A *number* specifying the information level to control the verbosity of the output on the console. (default: `nil`).  
Example: `info = 2`.
- debug** A *number* specifying the debug level to perform extra assertions and to control the verbosity of the output on the console. (default: `nil`).  
Example: `debug = 2`.
- usrdef** Any user defined data that will be attached to the tracked map flow, which is internally passed to the elements method `:track` and to their underlying maps. (default: `nil`).  
Example: `usrdef = { myvar=somevalue }`.
- mflow** A *mflow* containing the current state of a `track` command. If a map flow is provided, all attributes are discarded except `nstep`, `info` and `debug`, as the command was already set up upon its creation. (default: `nil`).  
Example: `mflow = mflow0`.

The `twiss` command returns the following objects in this order:

|             |                                                                                                                                                                                      |
|-------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>mtbl</b> | A <i>mtable</i> corresponding to the augmented TFS table of the <code>track</code> command with the <code>twiss</code> command columns.                                              |
| <b>mflw</b> | A <i>mflow</i> corresponding to the augmented map flow of the <code>track</code> command with the <code>twiss</code> command data.                                                   |
| <b>eidx</b> | An optional <i>number</i> corresponding to the last tracked element index in the sequence when <code>nstep</code> was specified and stopped the command before the end of the range. |

## 2 Twiss mtable

The `twiss` command returns a *mtable* where the information described hereafter is the default list of fields written to the TFS files.<sup>4</sup>

The header of the *mtable* contains the fields in the default order:<sup>5</sup>

|                  |                                                                                         |
|------------------|-----------------------------------------------------------------------------------------|
| <b>name</b>      | The name of the command that created the <i>mtable</i> , e.g. "track".                  |
| <b>type</b>      | The type of the <i>mtable</i> , i.e. "track".                                           |
| <b>title</b>     | The value of the command attribute <code>title</code> .                                 |
| <b>origin</b>    | The origin of the application that created the <i>mtable</i> , e.g. "MAD 1.0.0 OSX 64". |
| <b>date</b>      | The date of the creation of the <i>mtable</i> , e.g. "27/05/20".                        |
| <b>time</b>      | The time of the creation of the <i>mtable</i> , e.g. "19:18:36".                        |
| <b>refcol</b>    | The reference <i>column</i> for the <i>mtable</i> dictionary, e.g. "name".              |
| <b>direction</b> | The value of the command attribute <code>dir</code> .                                   |
| <b>observe</b>   | The value of the command attribute <code>observe</code> .                               |
| <b>implicit</b>  | The value of the command attribute <code>implicit</code> .                              |
| <b>misalign</b>  | The value of the command attribute <code>misalign</code> .                              |
| <b>deltap</b>    | The value of the command attribute <code>deltap</code> .                                |
| <b>lost</b>      | The number of lost particle(s) or damap(s).                                             |
| <b>chrom</b>     | The value of the command attribute <code>chrom</code> .                                 |
| <b>coupling</b>  | The value of the command attribute <code>coupling</code> .                              |
| <b>length</b>    | The <i>s</i> -length of the tracked design orbit.                                       |
| <b>q1</b>        | The tunes of mode 1.                                                                    |
| <b>q2</b>        | The tunes of mode 2.                                                                    |
| <b>q3</b>        | The tunes of mode 3.                                                                    |
| <b>alfap</b>     | The momentum compaction factor $\alpha_p$ .                                             |

<sup>4</sup>The output of *mtable* in TFS files can be fully customized by the user.

<sup>5</sup>The fields from `name` to `lost` are set by the `track` command.

|                |                                                                                     |
|----------------|-------------------------------------------------------------------------------------|
| <b>etap</b>    | The phase slip factor $\eta_p$ .                                                    |
| <b>gammatr</b> | The energy gamma transition $\gamma_{tr}$ .                                         |
| <b>synch1</b>  | The first synchrotron radiation integral.                                           |
| <b>synch_1</b> | NYI The first synchrotron radiation integral.                                       |
| <b>synch_2</b> | NYI The second synchrotron radiation integral.                                      |
| <b>synch_3</b> | NYI The third synchrotron radiation integral.                                       |
| <b>synch_4</b> | NYI The fourth synchrotron radiation integral.                                      |
| <b>synch_5</b> | NYI The fifth synchrotron radiation integral.                                       |
| <b>range</b>   | The value of the command attribute <code>range</code> . <sup>6</sup>                |
| <b>__seq</b>   | The <i>sequence</i> from the command attribute <code>sequence</code> . <sup>7</sup> |

The core of the *mtable* contains the columns in the default order:<sup>8</sup>

|               |                                                                           |
|---------------|---------------------------------------------------------------------------|
| <b>name</b>   | The name of the element.                                                  |
| <b>kind</b>   | The kind of the element.                                                  |
| <b>s</b>      | The <i>s</i> -position at the end of the element slice.                   |
| <b>l</b>      | The length from the start of the element to the end of the element slice. |
| <b>id</b>     | The index of the particle or damap as provided in X0.                     |
| <b>x</b>      | The local coordinate <i>x</i> at the <i>s</i> -position .                 |
| <b>px</b>     | The local coordinate $p_x$ at the <i>s</i> -position.                     |
| <b>y</b>      | The local coordinate <i>y</i> at the <i>s</i> -position.                  |
| <b>py</b>     | The local coordinate $p_y$ at the <i>s</i> -position.                     |
| <b>t</b>      | The local coordinate <i>t</i> at the <i>s</i> -position.                  |
| <b>pt</b>     | The local coordinate $p_t$ at the <i>s</i> -position.                     |
| <b>slc</b>    | The slice index ranging from -2 to <code>nslice</code> .                  |
| <b>turn</b>   | The turn number.                                                          |
| <b>tdir</b>   | The <i>t</i> -direction of the tracking in the element.                   |
| <b>eidx</b>   | The index of the element in the sequence.                                 |
| <b>status</b> | The status of the particle or damap.                                      |
| <b>alfall</b> | The optical function $\alpha$ of mode 1 at the <i>s</i> -position.        |

<sup>6</sup>This field is not saved in the TFS table by default.

<sup>7</sup>Fields and columns starting with two underscores are protected data and never saved to TFS files.

<sup>8</sup>The column from `name` to `status` are set by the `track` command.

|               |                                                               |
|---------------|---------------------------------------------------------------|
| <b>beta11</b> | The optical function $\beta$ of mode 1 at the $s$ -position.  |
| <b>gama11</b> | The optical function $\gamma$ of mode 1 at the $s$ -position. |
| <b>mu1</b>    | The phase advance $\mu$ of mode 1 at the $s$ -position.       |
| <b>dx</b>     | The dispersion function of $x$ at the $s$ -position.          |
| <b>dpx</b>    | The dispersion function of $p_x$ at the $s$ -position.        |
| <b>alfa22</b> | The optical function $\alpha$ of mode 2 at the $s$ -position. |
| <b>beta22</b> | The optical function $\beta$ of mode 2 at the $s$ -position.  |
| <b>gama22</b> | The optical function $\gamma$ of mode 2 at the $s$ -position. |
| <b>mu2</b>    | The phase advance $\mu$ of mode 2 at the $s$ -position.       |
| <b>dy</b>     | The dispersion function of $y$ at the $s$ -position.          |
| <b>dpy</b>    | The dispersion function of $p_y$ at the $s$ -position.        |
| <b>alfa33</b> | The optical function $\alpha$ of mode 3 at the $s$ -position. |
| <b>beta33</b> | The optical function $\beta$ of mode 3 at the $s$ -position.  |
| <b>gama33</b> | The optical function $\gamma$ of mode 3 at the $s$ -position. |
| <b>mu3</b>    | The phase advance $\mu$ of mode 3 at the $s$ -position.       |
| <b>—map</b>   | The damp at the $s$ -position. <sup>7</sup>                   |

The `chrom` attribute will add the following fields to the `mtable` header:

|            |                                                                   |
|------------|-------------------------------------------------------------------|
| <b>dq1</b> | The chromatic derivative of tunes of mode 1, i.e. chromaticities. |
| <b>dq2</b> | The chromatic derivative of tunes of mode 2, i.e. chromaticities. |
| <b>dq3</b> | The chromatic derivative of tunes of mode 3, i.e. chromaticities. |

The `chrom` attribute will add the following columns to the `mtable`:

|             |                                                                                    |
|-------------|------------------------------------------------------------------------------------|
| <b>dmu1</b> | The chromatic derivative of the phase advance of mode 1 at the $s$ -position.      |
| <b>ddx</b>  | The chromatic derivative of the dispersion function of $x$ at the $s$ -position.   |
| <b>ddpx</b> | The chromatic derivative of the dispersion function of $p_x$ at the $s$ -position. |
| <b>wx</b>   | The chromatic amplitude function of mode 1 at the $s$ -position.                   |
| <b>phix</b> | The chromatic phase function of mode 1 at the $s$ -position.                       |
| <b>dmu2</b> | The chromatic derivative of the phase advance of mode 2 at the $s$ -position.      |
| <b>ddy</b>  | The chromatic derivative of the dispersion function of $y$ at the $s$ -position.   |
| <b>ddpy</b> | The chromatic derivative of the dispersion function of $p_y$ at the $s$ -position. |
| <b>wy</b>   | The chromatic amplitude function of mode 2 at the $s$ -position.                   |

**phiy**            The chromatic phase function of mode 2 at the  $s$ -position.

The coupling attribute will add the following columns to the *mtable*:

**alfa12**        The optical function  $\alpha$  of coupling mode 1-2 at the  $s$ -position.

**beta12**        The optical function  $\beta$  of coupling mode 1-2 at the  $s$ -position.

**gama12**       The optical function  $\gamma$  of coupling mode 1-2 at the  $s$ -position.

**alfa13**        The optical function  $\alpha$  of coupling mode 1-3 at the  $s$ -position.

**beta13**        The optical function  $\beta$  of coupling mode 1-3 at the  $s$ -position.

**gama13**       The optical function  $\gamma$  of coupling mode 1-3 at the  $s$ -position.

**alfa21**        The optical function  $\alpha$  of coupling mode 2-1 at the  $s$ -position.

**beta21**        The optical function  $\beta$  of coupling mode 2-1 at the  $s$ -position.

**gama21**       The optical function  $\gamma$  of coupling mode 2-1 at the  $s$ -position.

**alfa23**        The optical function  $\alpha$  of coupling mode 2-3 at the  $s$ -position.

**beta23**        The optical function  $\beta$  of coupling mode 2-3 at the  $s$ -position.

**gama23**       The optical function  $\gamma$  of coupling mode 2-3 at the  $s$ -position.

**alfa31**        The optical function  $\alpha$  of coupling mode 3-1 at the  $s$ -position.

**beta31**        The optical function  $\beta$  of coupling mode 3-1 at the  $s$ -position.

**gama31**       The optical function  $\gamma$  of coupling mode 3-1 at the  $s$ -position.

**alfa32**        The optical function  $\alpha$  of coupling mode 3-2 at the  $s$ -position.

**beta32**        The optical function  $\beta$  of coupling mode 3-2 at the  $s$ -position.

**gama32**       The optical function  $\gamma$  of coupling mode 3-2 at the  $s$ -position.

### 3 Tracking linear normal form

TODO

### 4 Examples

TODO



# Chapter 14. Match

The match command provides a unified interface to several optimizer. It can be used to match optics parameters (its main purpose), to fit data sets with parametric functions in the least-squares sense, or to find local or global minima of non-linear problems. Most local methods support bounds, equalities and inequalities constraints. The *least-squares* methods are custom variant of the Newton-Raphson and the Gauss-Newton algorithms implemented by the `LSopt` module. The local and global *non-linear* methods are relying on the `NLOpt` module, which interfaces the embedded `NLOpt` library that implements a dozen of well-known algorithms.

## 1 Command synopsis

**Figure 14.1:** Synopsis of the match command with default setup.

```
status, fmin, ncall = match {
 command = function or nil,
 variables = { variables-attributes,
 { variable-attributes },
 ...more variable definitions...
 { variable-attributes } },
 equalities = { constraints-attributes,
 { constraint-attributes },
 ...more equality definitions...
 { constraint-attributes } },
 inequalities = { constraints-attributes,
 { constraint-attributes },
 ...more inequality definitions...
 { constraint-attributes } },
 weights = { weights-list },
 objective = { objective-attributes },
 maxcall=nil, -- call limit
 maxtime=nil, -- time limit
 info=nil, -- information level (output on terminal)
 debug=nil, -- debug information level (output on terminal)
 usrdef=nil, -- user defined data attached to the environment
}
```

The match command format is summarized in Figure 14.1, including the default setup of the attributes. The match command supports the following attributes:

- command**     A *callable* (e) that will be invoked during the optimization process at each iteration. (default: nil).  
Example: `command := twiss { twiss-attributes }.`
- variables**    An *mappable* of single variable specification that can be combined with a *set* of specifications for all variables. (no default, required).  
Example: `variables = {{ var="seq.knobs.mq_k1" }}.`

- equalities** An *mappable* of single equality specification that can be combined with a *set* of specifications for all equalities. (default: {}).  
Example: `equalities = {{ expr=\t -> t.q1-64.295, name='q1' }}`.
- inequalities** An *mappable* of single inequality specification that can be combined with a *set* of specifications for all inequalities. (default: {}).  
Example: `inequalities = {{ expr=\t -> t.mq4.beta11-50 }}`.
- weights** A *mappable* of weights specification that can be used in the `kind` attribute of the constraints specifications. (default: {}).  
Example: `weights = { px=10 }`.
- objective** A *mappable* of specifications for the objective to minimize. (default: {}).  
Example: `objective = { method="LD_LMDIF", fmin=1e-10 }`.
- maxcall** A *number* specifying the maximum allowed calls of the command function or the objective function. (default: nil).  
Example: `maxcall = 100`.
- maxtime** A *number* specifying the maximum allowed time in seconds. (default: nil).  
Example: `maxtime = 60`.
- info** A *number* specifying the information level to control the verbosity of the output on the [console](#). (default: nil).  
Example: `info = 3`.
- debug** A *number* specifying the debug level to perform extra assertions and to control the verbosity of the output on the [console](#). (default: nil).  
Example: `debug = 2`.
- usrdef** Any user defined data that will be attached to the matching environment, which is passed as extra argument to all user defined functions in the `match` command. (default: nil).  
Example: `usrdef = { var=vector(15) }`.

The `match` command returns the following values in this order:

- status** A *string* corresponding to the status of the command or the stopping reason of the method. See [Table 14.1](#) for the list of supported status.
- fmin** A *number* corresponding to the best minimum reached during the optimization.
- ncall** The *number* of calls of the command function or the objective function.

## 2 Environment

The `match` command creates a matching environment, which is passed as argument to user's functions invoked during an iteration. It contains some useful attributes that can be read or changed during the optimization process (with care):

- ncall** The current *number* of calls of the command and/or the objective functions.
- dtime** A *number* reporting the current elapsed time.
- stop** A *logical* stopping the `match` command immediately if set to `true`.

**Table 14.1:** List of status (*string*) returned by the match command.

| status        | Meaning                                                              |
|---------------|----------------------------------------------------------------------|
| SUCCESS       | Generic success (NLopt only, unlikely).                              |
| FMIN          | fmin <i>criteria</i> is fulfilled by the objective function.         |
| FTOL          | tol or rtol <i>criteria</i> are fulfilled by the objective function. |
| XTOL          | tol or rtol <i>criteria</i> are fulfilled by the variables step.     |
| MAXCALL       | maxcall <i>criteria</i> is reached.                                  |
| MAXTIME       | maxtime <i>criteria</i> is reached.                                  |
| ROUNDOFF      | Round off limited iteration progress, results may still be useful.   |
| STOPPED       | Termination forced by user, i.e. env.stop = true.                    |
| <i>Errors</i> |                                                                      |
| FAILURE       | Generic failure (NLopt only, unlikely).                              |
| INVALID_ARGS  | Invalid argument (NLopt only, unlikely).                             |
| OUT_OF_MEMORY | Ran out of memory (NLopt only, unlikely).                            |

- info**            The current information level  $\geq 0$ .
- debug**            The current debugging level  $\geq 0$ .
- usrdef**            The `usrdef` attribute of the match command or `nil`.
- command**            The `command` attribute of the match command or `nil`.
- variables**            The `variables` attribute of the match command.
- equalities**            The `equalities` attribute of the match command or `{}`.
- inequalities**            The `inequalities` attribute of the match command or `{}`.
- weights**            The `weights` attribute of the match command or `{}`.

### 3 Command

The attribute `command` (default: `nil`) must be a *callable* (`e`) that will be invoked with the matching environment as first argument during the optimization, right after the update of the `variables` to their new values, and before the evaluation of the `constraints` and the `objective` function. (default: `nil`).

`command = function or nil,`

The value returned by `command` is passed as the first argument to all constraints. If this return value is `nil`, the match command considers the current iteration as invalid. Depending on the selected method, the optimizer can start a new iteration or stop.

A typical `command` definition for matching optics is a function that calls a `twiss` command<sup>1</sup>:

```
command := mchklost(twiss { twiss-attributes })
```

where the function `mchklost` surrounding the `twiss` command checks if the returned `mtable` (i.e. the `twiss` table) has lost particles and returns `nil` instead:

<sup>1</sup>Here, the function (i.e. the deferred expression) ignores the matching environment passed as first argument.

```
mchklost = \mt -> mt.lost == 0 and mt or nil
```

The function `mchklost`<sup>2</sup> is useful to avoid that all constraints do the check individually.

## 4 Variables

The attribute `variables` (no default, required) defines the variables that the command `match` will update while trying to minimize the objective function.

```
variables = { variables-attributes,
 { variable-attributes },
 ...more variable definitions...
 { variable-attributes } },
```

The *variable-attributes* is a set of attributes that specify a single variable:

- var**            A *string* specifying the identifier (and indirection) needed to reach the variable from the user's scope where the `match` command is defined. (default: `nil`).  
Example: `var = "lhcb1.mq_12l4_b1.k1"`.
- name**           A *string* specifying the name of the variable to display when the `info` level is positive. (default: `var`).  
Example: `name = "MQ.12L4.B1->k1"`.
- min**            A *number* specifying the lower bound for the variable. (default: `-inf`).  
Example: `min = -4`.
- max**            A *number* specifying the upper bound for the variable. (default: `+inf`).  
Example: `max = 10`.
- sign**           A *logical* enforcing the sign of the variable by moving `min` or `max` to zero depending on the sign of its initial value. (default: `false`).  
Example: `sign = true`.
- slope**          A *number* enforcing ([LSopt](#) methods only) with its sign the variation direction of the variable, i.e. positive will only increase and negative will only decrease. (default: `0`).  
Example: `slope = -1`.
- step**           A small positive *number* used to approximate the derivatives using the [finite difference](#) method. If the value is not provided, the command will use some heuristic. (default: `nil`).  
Example: `step = 1e-6`.
- tol**            A *number* specifying the tolerance on the variable step. If an update is smaller than `tol`, the command will return the status `"XTOL"`. (default: `0`).  
Example: `tol = 1e-8`.
- get**            A *callable* (`e`) returning the variable value as a *number*, optionally using the matching environment passed as first argument. This attribute is required if the variable is *local* or an *upvalue* to avoid a significant slowdown of the code. (default: `nil`).  
Example: `get := lhcb1.mq_12l4_b1.k1`.

---

<sup>2</sup>The function `mchklost` is provided by the [GPhys](#) module.

**set** A *callable* ( $v, e$ ) updating the variable value with the *number* passed as first argument, optionally using the matching environment passed as second argument. This attribute is required if the variable is *local* or an *upvalue* to avoid a significant slowdown of the code. (default: nil).  
Example: `set = \v,e => lhcb1.mqxa_1l5.k1 = v*e.usrdef.xon end.`

The *variables-attributes* is a set of attributes that specify all variables together, but with a lower precedence than the single variable specification of the same name unless otherwise specified:

**min** Idem *variable-attributes*, but for all variables with no local override.

**max** Idem *variable-attributes*, but for all variables with no local override.

**sign** Idem *variable-attributes*, but for all variables with no local override.

**slope** Idem *variable-attributes*, but for all variables with no local override.

**step** Idem *variable-attributes*, but for all variables with no local override.

**tol** Idem *variable-attributes*, but for all variables with no local override.

**rtol** A *number* specifying the relative tolerance on all variable steps. If an update is smaller than `rtol` relative to its variable value, the command will return the status "XTOL". (default: `eps`).  
Example: `tol = 1e-8.`

**nvar** A *number* specifying the number of variables of the problem. It is useful when the problem is made abstract with functions and it is not possible to deduce this count from single variable definitions, or one needs to override it. (default: nil).  
Example: `nvar = 15.`

**get** A *callable* ( $x, e$ ) updating a *vector* passed as first argument with the values of all variables, optionally using the matching environment passed as second argument. This attribute supersedes all single variable `get` and may be useful when it is better to read all the variables together, or when they are all *locals* or *upvalues*. (default: nil).  
Example: `get = \x,e -> e.usrdef.var:copy(x).`

**set** A *callable* ( $x, e$ ) updating all the variables with the values passed as first argument in a *vector*, optionally using the matching environment passed as second argument. This attribute supersedes all single variable `set` and may be useful when it is better to update all the variables together, or when they are all *locals* or *upvalues*. (default: nil).  
Example: `set = \x,e -> x:copy(e.usrdef.var).`

**nowarn** A *logical* disabling a warning emitted when the definition of `get` and `set` are advised but not defined. It is safe to not define `get` and `set` in such case, but it will significantly slowdown the code. (default: nil).  
Example: `nowarn = true.`

## 5 Constraints

The attributes *equalities* (default: `{}`) and *inequalities* (default: `{}`) define the constraints that the command `match` will try to satisfy while minimizing the objective function. Equalities and inequalities are considered differently when calculating the [penalty function](#).

```

equalities = { constraints-attributes,
 { constraint-attributes },
 ...more equality definitions...
 { constraint-attributes } },
inequalities = { constraints-attributes,
 { constraint-attributes },
 ...more inequality definitions...
 { constraint-attributes } },
weights = { weights-list },

```

The *constraint-attributes* is a set of attributes that specify a single constraint, either an *equality* or an *inequality*:

- expr**            A *callable* (*r*, *e*) returning the constraint value as a *number*, optionally using the result of command passed as first argument, and the matching environment passed as second argument. (default: *nil*)  
Example: `expr = \t -> t.IP8.beta11 - beta_ip8.`
- name**            A *string* specifying the name of the constraint to display when the *info* level is positive. (default: *nil*).  
Example: `name = "betx@IP8".`
- kind**            A *string* specifying the kind to refer to for the weight of the constraint, taken either in the user-defined or in the default *weights-list*. (default: *nil*).  
Example: `kind = "dq1".`
- weight**          A *number* used to override the weight of the constraint. (default: *nil*).  
Example: `weight = 100.`
- tol**             A *number* specifying the tolerance to apply on the constraint when checking for its fulfillment. (default: `1e-8`).  
Example: `tol = 1e-6.`

The *constraints-attributes* is a set of attributes that specify all equalities or inequalities constraints together, but with a lower precedence than the single constraint specification of the same name unless otherwise specified:

- tol**             Idem *constraint-attributes*, but for all constraints with no local override.
- nequ**            A *number* specifying the number of equations (i.e. number of equalities or inequalities) of the problem. It is useful when the problem is made abstract with functions and it is not possible to deduce this count from single constraint definitions, or one needs to override it. (default: *nil*).  
Example: `nequ = 15.`
- exec**            A *callable* (*x*, *c*, *cjac*) updating a *vector* passed as second argument with the values of all constraints, and updating an optional *matrix* passed as third argument with the Jacobian of all constraints (if not *nil*), using the variables values passed in a *vector* as first argument. This attribute supersedes all constraints *expr* and may be useful when it is better to update all the constraints together. (default: *nil*).  
Example: `exec = myinequ`, where (`nvar=2` and `nequ=2`)

```

local function myinequ (x, c, cjac)
 c:fill { 8*x[1]^3 - x[2] ; (1-x[1])^3 - x[2] }
 if cjac then -- fill [2x2] matrix if present
 cjac:fill { 24*x[1]^2, -1 ; -3*(1-x[1])^2, -1 }
 end
end

```

**disp** A *logical* disabling the display of the equalities in the summary if it is explicitly set to false. This is useful for fitting data where equalities are used to compute the residuals. (default: nil).  
Example: `disp = false`.

The *weights-list* is a set of attributes that specify weights for kinds used by constraints. It allows to override the default weights of the supported kinds summarized in Table 14.2, or to extend this list with new kinds and weights. The default weight for any undefined kind is 1.

Example: `weights = { q1=100, q2=100, mykind=3 }`.

**Table 14.2:** List of supported kinds (*string*) and their default weights (*number*).

| Name  | Weight | Name  | Weight | Name  | Weight | Generic name |
|-------|--------|-------|--------|-------|--------|--------------|
| x     | 10     | y     | 10     | t     | 10     |              |
| px    | 100    | py    | 100    | pt    | 100    |              |
| dx    | 10     | dy    | 10     | dt    | 10     | d            |
| dpx   | 100    | dpy   | 100    | dpt   | 100    | dp           |
| ddx   | 10     | ddy   | 10     | ddt   | 10     | dd           |
| ddpx  | 100    | ddpy  | 100    | ddpt  | 100    | ddp          |
| wx    | 1      | wy    | 1      | wz    | 1      | w            |
| phix  | 1      | phiy  | 1      | phiz  | 1      | phi          |
| betx  | 1      | bety  | 1      | betz  | 1      | beta         |
| alfx  | 10     | alfy  | 10     | alfz  | 10     | alfa         |
| mux   | 10     | muy   | 10     | muz   | 10     | mu           |
| beta1 | 1      | beta2 | 1      | beta3 | 1      | beta         |
| alfa1 | 10     | alfa2 | 10     | alfa3 | 10     | alfa         |
| mu1   | 10     | mu2   | 10     | mu3   | 10     | mu           |
| q1    | 10     | q2    | 10     | q3    | 10     | q            |
| dq1   | 1      | dq2   | 1      | dq3   | 1      | dq           |

## 6 Objective

The attribute `objective` (default: `{}`) defines the objective that the command `match` will try to minimize.

```
objective = { objective-attributes },
```

The *objective-attributes* is a set of attributes that specify the objective to fulfill:

**method** A *string* specifying the algorithm to use for solving the problem, see Tables 14.3, 14.4 and 14.5. (default: "LN\_COBYLA" if `objective.exec` is defined, "LD\_JACOBIAN" otherwise).

Example: `method = "LD_LMDIF"`.

- submethod** A *string* specifying the algorithm from NLOpt module to use for solving the problem locally when the method is an augmented algorithm, see Table 14.4 and 14.5. (default: "LN\_COBYLA").  
Example: `method = "AUGLAG", submethod = "LD_SLSQP"`.
- fmin** A *number* corresponding to the minimum to reach during the optimization. For least squares problems, it corresponds to the tolerance on the [penalty function](#). If an iteration find a value smaller than `fmin` and all the constraints are fulfilled, the command will return the status "FMIN". (default: `nil`).  
Example: `fmin = 1e-12`.
- tol** A *number* specifying the tolerance on the objective function step. If an update is smaller than `tol`, the command will return the status "FTOL". (default: `0`).  
Example: `tol = 1e-10`.
- rtol** A *number* specifying the relative tolerance on the objective function step. If an update is smaller than `rtol` relative to its step value, the command will return the status "FTOL". (default: `0`).  
Example: `tol = 1e-8`.
- bstra** A *number* specifying the strategy to select the *best case* of the [objective function](#). (default: `nil`).  
Example: `bstra = 0`.<sup>3</sup>
- broyden** A *logical* allowing the Jacobian approximation by finite difference to update its columns with a *Broyden's rank one* estimates when the step of the corresponding variable is almost collinear with the variables step vector. This option may save some expensive calls to `command`, e.g. `save Twiss` calculations, when it does not degrade the rate of convergence of the selected method. (default: `nil`).  
Example: `broyden = true`.
- reset** A *logical* specifying to the `match` command to restore the initial state of the variables before returning. This is useful to attempt an optimization without changing the state of the variables. Note that if any function amongst `command`, `variables get` and `set`, `constraints expr` or `exec`, or `objective exec` have side effects on the environment, these will be persistent. (default: `nil`).  
Example: `reset = true`.
- exec** A *callable* (`x`, `fgrd`) returning the value of the objective function as a *number*, and updating a *vector* passed as second argument with its gradient, using the variables values passed in a *vector* as first argument. (default: `nil`).  
Example: `exec = myfun`, where (`nvar=2`)
- ```

local function myfun(x, fgrd)
  if fgrd then -- fill [2x1] vector if present
    fgrd:fill { 0, 0.5/sqrt(x[2]) }
  end
  return sqrt(x[2])
end

```
- grad** A *logical* enabling (`true`) or disabling (`false`) the approximation by finite difference of the gradient of the objective function or the Jacobian of the constraints. A `nil` value will

³MAD-X matching corresponds to `bstra=0`.

be converted to `true` if no `exec` function is defined and the selected method requires derivatives (D), otherwise it will be converted to `false`. (default: `nil`).

Example: `grad = false`.

bisec A *number* specifying (LSopt methods only) the maximum number of attempt to minimize an increasing objective function by reducing the variables steps by half, i.e. that is a [line search](#) using $\alpha = 0.5^k$ where $k = 0..bisec$. (default: 3 if `objective.exec` is undefined, 0 otherwise).

Example: `bisec = 9`.

rcond A *number* specifying (LSopt methods only) how to determine the effective rank of the Jacobian while solving the least squares system (see `ssolve` from the Matrix module). This attribute can be updated between iterations, e.g. through `env.objective.rcond`. (default: `eps`).

Example: `rcond = 1e-14`.

jt看 A *number* specifying (LSopt methods only) the tolerance on the norm of the Jacobian rows to reject useless constraints. This attribute can be updated between iterations, e.g. through `env.objective.jtol`. (default: `eps`).

Example: `tol = 1e-14`.

jiter A *number* specifying (LSopt methods only) the maximum allowed attempts to solve the least squares system when variables are rejected, e.g. wrong slope or out-of-bound values. (default: 10).

Example: `jiter = 15`.

jstra A *number* specifying (LSopt methods only) the strategy to use for reducing the variables of the least squares system. (default: 1).

Example: `jstra = 3`.⁴

jstra	Strategy for reducing variables of least squares system.
0	no variables reduction, constraints reduction is still active.
1	reduce system variables for bad slopes and out-of-bound values.
2	idem 1, but bad slopes reinitialize variables to their original state.
3	idem 2, but strategy switches definitely to 0 if <code>jiter</code> is reached.

7 Algorithms

The `match` command supports local and global optimization algorithms through the `method` attribute, as well as combinations of them with the `submethod` attribute (see [objective](#)). The method should be selected according to the kind of problem that will add a prefix to the method name: local (L) or global (G), with (D) or without (N) derivatives, and least squares or nonlinear function minimization. When the method requires the derivatives (D) and no `objective.exec` function is defined or the attribute `grad` is set to `false`, the `match` command will approximate the derivatives, i.e. gradient and Jacobian, by the finite difference method (see [derivatives](#)).

Most global optimization algorithms explore the variables domain with methods belonging to stochastic sampling, deterministic scanning, and splitting strategies, or a mix of them. Hence, all global methods require *boundaries* to define the searching region, which may or may not be internally scaled to a hypercube. Some global methods allow to specify with the `submethod` attribute, the local method to use for searching local minima. If this is not the case, it is wise to refine the global solution with a local method

⁴MAD-X JACOBIAN with `strategy=3` corresponds to `jstra=3`.

afterward, as global methods put more effort on finding global solutions than precise local minima. The global (G) optimization algorithms, with (D) or without (N) derivatives, are listed in Table 14.5.

Most local optimization algorithms with derivatives are variants of the Newton iterative method suitable for finding local minima of nonlinear vector-valued function $\mathbf{f}(\mathbf{x})$, i.e. searching for stationary points. The iteration steps \mathbf{h} are given by the minimization $\mathbf{h} = -\alpha(\nabla^2\mathbf{f})^{-1}\nabla\mathbf{f}$, coming from the local approximation of the function at the point $\mathbf{x} + \mathbf{h}$ by its Taylor series truncated at second order $\mathbf{f}(\mathbf{x} + \mathbf{h}) \approx \mathbf{f}(\mathbf{x}) + \mathbf{h}^T\nabla\mathbf{f}(\mathbf{x}) + \frac{1}{2}\mathbf{h}^T\nabla^2\mathbf{f}(\mathbf{x})\mathbf{h}$, and solved for $\nabla_{\mathbf{h}}\mathbf{f} = 0$. The factor $\alpha > 0$ is part of the line search strategy, which is sometimes replaced or combined with a trusted region strategy like in the Levenberg-Marquardt algorithm. The local (L) optimization algorithms, with (D) or without (N) derivatives, are listed in Table 14.3 for least squares methods and in Table 14.4 for non-linear methods, and can be grouped by family of algorithms:

Newton An iterative method to solve nonlinear systems that uses iteration step given by the minimization $\mathbf{h} = -\alpha(\nabla^2\mathbf{f})^{-1}\nabla\mathbf{f}$.

Newton-Raphson An iterative method to solve nonlinear systems that uses iteration step given by the minimization $\mathbf{h} = -\alpha(\nabla\mathbf{f})^{-1}\mathbf{f}$.

Gradient-Descent An iterative method to solve nonlinear systems that uses iteration step given by $\mathbf{h} = -\alpha\nabla\mathbf{f}$.

Quasi-Newton A variant of the Newton method that uses BFGS approximation of the Hessian $\nabla^2\mathbf{f}$ or its inverse $(\nabla^2\mathbf{f})^{-1}$, based on values from past iterations.

Gauss-Newton A variant of the Newton method for *least-squares* problems that uses iteration step given by the minimization $\mathbf{h} = -\alpha(\nabla\mathbf{f}^T\nabla\mathbf{f})^{-1}(\nabla\mathbf{f}^T\mathbf{f})$, where the Hessian $\nabla^2\mathbf{f}$ is approximated by $\nabla\mathbf{f}^T\nabla\mathbf{f}$ with $\nabla\mathbf{f}$ being the Jacobian of the residuals \mathbf{f} .

Levenberg-Marquardt A hybrid G-N and G-D method for *least-squares* problems that uses iteration step given by the minimization $\mathbf{h} = -\alpha(\nabla\mathbf{f}^T\nabla\mathbf{f} + \mu\mathbf{D})^{-1}(\nabla\mathbf{f}^T\mathbf{f})$, where $\mu > 0$ is the damping term selecting the method G-N (small μ) or G-D (large μ), and $\mathbf{D} = \text{diag}(\nabla\mathbf{f}^T\nabla\mathbf{f})$.

Simplex A linear programming method (simplex method) working without using any derivatives.

Nelder-Mead A nonlinear programming method (downhill simplex method) working without using any derivatives.

Principal-Axis An adaptive coordinate descent method working without using any derivatives, selecting the descent direction from the Principal Component Analysis.

7.1 Stopping criteria

The match command will stop the iteration of the algorithm and return one of the following status if the corresponding criteria, *checked in this order*, is fulfilled (see also Table 14.1):

- STOPPED** Check `env.stop == true`, i.e. termination forced by a user-defined function.
- FMIN** Check $f \leq f_{\min}$ if $c_{\text{fail}} = 0$ or `bstra == 0`, where f is the current value of the objective function, and c_{fail} the number of failed constraints (i.e. feasible point).
- FTOL** Check $|\Delta f| \leq f_{\text{tol}}$ or $|\Delta f| \leq f_{\text{rtol}}|f|$ if $c_{\text{fail}} = 0$, where f and Δf are the current value and step of the objective function, and c_{fail} the number of failed constraints (i.e. feasible point).

XTOL	Check $\max(\Delta \mathbf{x} - \mathbf{x}_{\text{tol}}) \leq 0$ or $\max(\Delta \mathbf{x} - \mathbf{x}_{\text{rtol}} \circ \mathbf{x}) \leq 0$, where \mathbf{x} and $\Delta \mathbf{x}$ are the current values and steps of the variables. Note that these criteria are checked even for non feasible points, i.e. $c_{\text{fail}} > 0$, as the algorithm can be trapped in a local minima that does not satisfy the constraints.
ROUNDOFF	Check $\max(\Delta \mathbf{x} - \varepsilon \mathbf{x}) \leq 0$ if $\mathbf{x}_{\text{rtol}} < \varepsilon$, where \mathbf{x} and $\Delta \mathbf{x}$ are the current values and steps of the variables. The <code>LSopt</code> module returns also this status if the Jacobian is full of zeros, which is <code>jtol</code> dependent during its <code>jstra</code> reductions.
MAXCALL	Check <code>env.ncall</code> \geq <code>maxcall</code> if <code>maxcall</code> > 0 .
MAXTIME	Check <code>env.dtime</code> \geq <code>maxtime</code> if <code>maxtime</code> > 0 .

7.2 Objective function

The objective function is the key point of the `match` command, specially when tolerances are applied to it or to the constraints, or the best case strategy is changed. It is evaluated as follow:

1. Update user's variables with the *vector* \mathbf{x} .
2. Evaluate the *callable* command if defined and pass its value to the constraints.
3. Evaluate the *callable* `objective.exec` if defined and save its value f .
4. Evaluate the *callable* `equalities.exec` if defined, otherwise evaluate all the functions `equalities[]`.`expr` (cmd) and use the result to fill the *vector* $\mathbf{c}^=$.
5. Evaluate the *callable* `inequalities.exec` if defined, otherwise evaluate all the functions `inequalities[]`.`expr` and use the result to fill the *vector* \mathbf{c}^{\leq} .
6. Count the number of invalid constraints $c_{\text{fail}} = \text{card}\{|\mathbf{c}^=| > \mathbf{c}_{\text{tol}}^=\} + \text{card}\{\mathbf{c}^{\leq} > \mathbf{c}_{\text{tol}}^{\leq}\}$.⁵
7. Calculate the *penalty* $p = \|\mathbf{c}\|/\|\mathbf{w}\|$, where $\mathbf{c} = \mathbf{w} \circ \left[\begin{smallmatrix} \mathbf{c}^= \\ \mathbf{c}^{\leq} \end{smallmatrix} \right]$ and \mathbf{w} is the weights *vector* of the constraints. Set $f = p$ if the *callable* `objective.exec` is undefined.⁶
8. Save the current iteration state as the best state depending on the strategy `bstra`. The default `bstra=nil` corresponds to the last strategy.

<code>bstra</code>	Strategy for selecting the best case of the objective function.
0	$f < f_{\text{min}}^{\text{best}}$, no feasible point check.
1	$c_{\text{fail}} \leq c_{\text{fail}}^{\text{best}}$ and $f < f_{\text{min}}^{\text{best}}$, improve both feasible point and objective.
-	$c_{\text{fail}} < c_{\text{fail}}^{\text{best}}$ or $c_{\text{fail}} = c_{\text{fail}}^{\text{best}}$ and $f < f_{\text{min}}^{\text{best}}$, improve feasible point or objective.

7.3 Derivatives

The derivatives are approximated by the finite difference methods when the selected algorithm requires them (D) and the function `objective.exec` is undefined or the attribute `grad=false`. The difficulty of the finite difference methods is to choose the small step h for the difference. The `match` command uses the *forward difference method* with a step $h = 10^{-4} \|\mathbf{h}\|$, where \mathbf{h} is the last `iteration steps`, unless it is overridden by the user with the variable attribute `step`. In order to avoid zero step size, which would be problematic for the calculation of the Jacobian, the choice of h is a bit more subtle:

$$\frac{\partial f_j}{\partial x_i} \approx \frac{f_j(\mathbf{x} + h\mathbf{e}_i) - f_j(\mathbf{x})}{h} ; \quad h = \begin{cases} 10^{-4} \|\mathbf{h}\| & \text{if } \|\mathbf{h}\| \neq 0 \\ 10^{-8} \|\mathbf{x}\| & \text{if } \|\mathbf{h}\| = 0 \text{ and } \|\mathbf{x}\| \neq 0 \\ 10^{-10} & \text{otherwise.} \end{cases}$$

⁵The `LSopt` module sets the values of valid inequalities to zero, i.e. $\mathbf{c}^{\leq} = 0$ if $\mathbf{c}^{\leq} \leq \mathbf{c}_{\text{tol}}^{\leq}$.

⁶The penalty is the norm of the residuals $\|\mathbf{c}\|$, not the usual $\frac{1}{2} \|\mathbf{c}\|^2$, which affects tolerance specification.

Hence the approximation of the Jacobian will need an extra evaluation of the objective function per variable. If this evaluation has a heavy cost, e.g. like a `twiss` command, it is possible to approximate the Jacobian evolution by a Broyden's rank-1 update with the `broyden` attribute:

$$\mathbf{J}_{k+1} = \mathbf{J}_k + \frac{\mathbf{f}(\mathbf{x}_k + \mathbf{h}_k) - \mathbf{f}(\mathbf{x}_k) - \mathbf{J}_k \mathbf{h}_k}{\|\mathbf{h}_k\|^2} \mathbf{h}_k^T$$

The update of the i -th column of the Jacobian by the Broyden approximation makes sense if the angle between \mathbf{h} and \mathbf{e}_i is small, that is when $|\mathbf{h}^T \mathbf{e}_i| \geq \gamma \|\mathbf{h}\|$. The `match` command uses a rather pessimistic choice of $\gamma = 0.8$, which gives good performance. Nevertheless, it is advised to always check if Broyden's update saves evaluations of the objective function for your study.

8 Console output

The verbosity of the output of the `match` command on the console (e.g. terminal) is controlled by the `info` level, where the level `info=0` means a completely silent command as usual. The first verbose level `info=1` displays the *final summary* at the end of the matching, as shown in the Figure 14.2, and the next level `info=2` adds *intermediate summary* for each evaluation of the objective function, as shown in the Figure 14.3. The columns of these tables are self-explanatory, and the sign `>` on the right of the constraints marks those failing.

The bottom line of the *intermediate summary* displays in order:

- the number of evaluation of the objective function so far,
- the elapsed time in second (in square brackets) so far,
- the current objective function value,
- the current objective function step,
- the current number of constraint that failed c_{fail} .

The bottom line of the *final summary* displays the same information but for the best case found, as well as the final status returned by the `match` command. The number in square brackets right after `fbst` is the evaluation number of the best case.

The `LSopt` module adds the sign `#` to mark the *adjusted* variables and the sign `*` to mark the *rejected* variables and constraints on the right of the *intermediate summary* tables to qualify the behavior of the constraints and the variables during the optimization process. If these signs appear in the *final summary* too, it means that they were always adjusted or rejected during the matching, which is useful to tune your study e.g. by removing the useless constraints.

9 Modules

The `match` command can be extended easily with new optimizer either from external libraries or internal module, or both. The interface should be flexible and extensible enough to support new algorithms and new options with a minimal effort.

9.1 LSopt

The `LSopt` (Least Squares optimization) module implements custom variant of the Newton-Raphson and the Levenberg-Marquardt algorithms to solve least squares problems. Both support the options `rcond`, `bisec`, `jtol`, `jiter` and `jstra` described in the section [objective](#), with the same default values. Table 14.3 lists the names of the algorithms for the attribute `method`. These algorithms cannot be used with the attribute `submethod` for the augmented algorithms of the `NLopt` module, which would not make sense as these methods support both equalities and inequalities.

Figure 14.2: Match command summary output (info=1).

Constraints	Type	Kind	Weight	Penalty Value
1 IP8	equality	beta	1	9.41469e-14
2 IP8	equality	beta	1	3.19744e-14
3 IP8	equality	alfa	10	0.00000e+00
4 IP8	equality	alfa	10	1.22125e-14
5 IP8	equality	dx	10	5.91628e-14
6 IP8	equality	dpx	100	1.26076e-13
7 E.DS.R8.B1	equality	beta	1	7.41881e-10
8 E.DS.R8.B1	equality	beta	1	1.00158e-09
9 E.DS.R8.B1	equality	alfa	10	4.40514e-12
10 E.DS.R8.B1	equality	alfa	10	2.23532e-11
11 E.DS.R8.B1	equality	dx	10	7.08333e-12
12 E.DS.R8.B1	equality	dpx	100	2.12877e-13
13 E.DS.R8.B1	equality	mu1	10	2.09610e-12
14 E.DS.R8.B1	equality	mu2	10	1.71063e-12

Variables	Final Value	Init. Value	Lower Limit	Upper Limit
1 kq4.l8b1	-3.35728e-03	-4.31524e-03	-8.56571e-03	0.00000e+00
2 kq5.l8b1	4.93618e-03	5.28621e-03	0.00000e+00	8.56571e-03
3 kq6.l8b1	-5.10313e-03	-5.10286e-03	-8.56571e-03	0.00000e+00
4 kq7.l8b1	8.05555e-03	8.25168e-03	0.00000e+00	8.56571e-03
5 kq8.l8b1	-7.51668e-03	-5.85528e-03	-8.56571e-03	0.00000e+00
6 kq9.l8b1	7.44662e-03	7.07113e-03	0.00000e+00	8.56571e-03
7 kq10.l8b1	-6.73001e-03	-6.39311e-03	-8.56571e-03	0.00000e+00
8 kqt11.l8b1	6.85635e-04	7.07398e-04	0.00000e+00	5.56771e-03
9 kqt12.l8b1	-2.38722e-03	-3.08650e-03	-5.56771e-03	0.00000e+00
10 kqt13.l8b1	5.55969e-03	3.78543e-03	0.00000e+00	5.56771e-03
11 kq4.r8b1	4.23719e-03	4.39728e-03	0.00000e+00	8.56571e-03
12 kq5.r8b1	-5.02348e-03	-4.21383e-03	-8.56571e-03	0.00000e+00
13 kq6.r8b1	4.18341e-03	4.05914e-03	0.00000e+00	8.56571e-03
14 kq7.r8b1	-5.48774e-03	-6.65981e-03	-8.56571e-03	0.00000e+00
15 kq8.r8b1	5.88978e-03	6.92571e-03	0.00000e+00	8.56571e-03
16 kq9.r8b1	-3.95756e-03	-7.46154e-03	-8.56571e-03	0.00000e+00
17 kq10.r8b1	7.18012e-03	7.55573e-03	0.00000e+00	8.56571e-03
18 kqt11.r8b1	-3.99902e-03	-4.78966e-03	-5.56771e-03	0.00000e+00
19 kqt12.r8b1	-1.95221e-05	-1.74210e-03	-5.56771e-03	0.00000e+00
20 kqt13.r8b1	-2.04425e-03	-3.61438e-03	-5.56771e-03	0.00000e+00

ncall=381 [4.1s], fbst[381]=8.80207e-12, fstp=-3.13047e-08, status=FMIN.

9.2 NLopt

The NLopt (Non-Linear optimization) module provides a simple interface to the algorithms implemented in the embedded NLopt library. Tables 14.4 and 14.5 list the names of the local and global algorithms

Figure 14.3: Match command intermediate output (info=2).

Constraints	Type	Kind	Weight	Penalty Value
1 IP8	equality	beta	1	3.10118e+00 >
2 IP8	equality	beta	1	1.85265e+00 >
3 IP8	equality	alfa	10	9.77591e-01 >
4 IP8	equality	alfa	10	8.71014e-01 >
5 IP8	equality	dx	10	4.37803e-02 >
6 IP8	equality	dpx	100	4.59590e-03 >
7 E.DS.R8.B1	equality	beta	1	9.32093e+01 >
8 E.DS.R8.B1	equality	beta	1	7.60213e+01 >
9 E.DS.R8.B1	equality	alfa	10	2.98722e+00 >
10 E.DS.R8.B1	equality	alfa	10	1.04758e+00 >
11 E.DS.R8.B1	equality	dx	10	7.37813e-02 >
12 E.DS.R8.B1	equality	dpx	100	6.67388e-03 >
13 E.DS.R8.B1	equality	mu1	10	7.91579e-02 >
14 E.DS.R8.B1	equality	mu2	10	6.61916e-02 >

Variables	Curr. Value	Curr. Step	Lower Limit	Upper Limit
1 kq4.l8b1	-3.36997e-03	-4.81424e-04	-8.56571e-03	0.00000e+00 #
2 kq5.l8b1	4.44028e-03	5.87400e-04	0.00000e+00	8.56571e-03
3 kq6.l8b1	-4.60121e-03	-6.57316e-04	-8.56571e-03	0.00000e+00 #
4 kq7.l8b1	7.42273e-03	7.88826e-04	0.00000e+00	8.56571e-03
5 kq8.l8b1	-7.39347e-03	0.00000e+00	-8.56571e-03	0.00000e+00 *
6 kq9.l8b1	7.09770e-03	2.58912e-04	0.00000e+00	8.56571e-03
7 kq10.l8b1	-5.96101e-03	-8.51573e-04	-8.56571e-03	0.00000e+00 #
8 kqt11.l8b1	6.15659e-04	8.79512e-05	0.00000e+00	5.56771e-03 #
9 kqt12.l8b1	-2.66538e-03	0.00000e+00	-5.56771e-03	0.00000e+00 *
10 kqt13.l8b1	4.68776e-03	0.00000e+00	0.00000e+00	5.56771e-03 *
11 kq4.r8b1	4.67515e-03	-5.55795e-04	0.00000e+00	8.56571e-03 #
12 kq5.r8b1	-4.71987e-03	5.49407e-04	-8.56571e-03	0.00000e+00 #
13 kq6.r8b1	4.68747e-03	-5.54035e-04	0.00000e+00	8.56571e-03 #
14 kq7.r8b1	-5.35315e-03	4.58938e-04	-8.56571e-03	0.00000e+00 #
15 kq8.r8b1	5.77068e-03	0.00000e+00	0.00000e+00	8.56571e-03 *
16 kq9.r8b1	-4.97761e-03	-7.11087e-04	-8.56571e-03	0.00000e+00 #
17 kq10.r8b1	6.90543e-03	4.33052e-04	0.00000e+00	8.56571e-03
18 kqt11.r8b1	-4.16758e-03	-5.95369e-04	-5.56771e-03	0.00000e+00 #
19 kqt12.r8b1	-1.57183e-03	0.00000e+00	-5.56771e-03	0.00000e+00 *
20 kqt13.r8b1	-2.57565e-03	0.00000e+00	-5.56771e-03	0.00000e+00 *

ncall=211 [2.3s], fval=8.67502e-01, fstp=-2.79653e+00, ccnt=14.

respectively for the attribute method. The methods that do not support equalities (column Equ) or inequalities (column Iqu) can still be used with constraints by specifying them as the submethod of the AUGmented LAGrangian method. For details about these algorithms, please refer to the [Algorithms](#) section of its [online documentation](#).

Table 14.3: List of supported least squares methods (LSopt).

method	Equ	Iqu	Description
LD_JACOBIAN	y	y	Modified Newton-Raphson algorithm.
LD_LMDIF	y	y	Modified Levenberg-Marquardt algorithm.

Table 14.4: List of supported non-linear local methods (NLOpt).

method	Equ	Iqu	Description
<i>Local optimizers without derivative (LN_)</i>			
LN_BOBYQA	n	n	Bound-constrained Optimization BY Quadratic Approximations algorithm.
LN_COBYLA	y	y	Bound Constrained Optimization BY Linear Approximations algorithm.
LN_NELDERMEAD	n	n	Original Nelder-Mead algorithm.
LN_NEWUOA	n	n	Older and less efficient LN_BOBYQA.
LN_NEWUOA_BOUND	n	n	Older and less efficient LN_BOBYQA with bound constraints.
LN_PRAXIS	n	n	PRincipal-AXIS algorithm.
LN_SBPLX	n	n	Subplex algorithm, variant of Nelder-Mead.
<i>Local optimizers with derivative (LD_)</i>			
LD_CCSAQ	n	y	Conservative Convex Separable Approximation with Quadratic penalty.
LD_LBFGS	n	n	BFGS algorithm with low memory footprint.
LD_LBFGS_NOCEDAL	n	n	Variant from J. Nocedal of LD_LBFGS.
LD_MMA	n	y	Method of Moving Asymptotes algorithm.
LD_SLSQP	y	y	Sequential Least-Squares Quadratic Programming algorithm.
LD_TNEWTON	n	n	Inexact Truncated Newton algorithm.
LD_TNEWTON_PRECOND	n	n	Idem LD_TNEWTON with preconditioning.
LD_TNEWTON_PRECOND- _RESTART	n	n	Idem LD_TNEWTON with preconditioning and steepest-descent restarting.
LD_TNEWTON_RESTART	n	n	Idem LD_TNEWTON with steepest-descent restarting.
LD_VAR1	n	n	Shifted limited-memory VARIable-metric rank-1 algorithm.
LD_VAR2	n	n	Shifted limited-memory VARIable-metric rank-2 algorithm.

10 Examples

10.1 Matching tunes and chromaticity

The following example below shows how to match the betatron tunes of the LHC beam 1 to $q_1 = 64.295$ and $q_2 = 59.301$ using the quadrupoles strengths $kqtf$ and $kqtd$, followed by the matching of the chromaticities to $dq_1 = 15$ and $dq_2 = 15$ using the main sextupole strengths ksf and ksd .

```

local lhcb1 in MADX
local twiss, match in MAD

local status, fmin, ncall = match {
  command := twiss { sequence=lhcb1, cofind=true,
                    method=4, observe=1 },
  variables = { rtol=1e-6, -- 1 ppm

```

Table 14.5: List of supported non-linear global methods (NLopt).

method	Equ	Iqu	Description
<i>Global optimizers without derivative (GN_)</i>			
GN_CRS2_LM	n	n	Variant of the Controlled Random Search algorithm with Local Mutation (mixed stochastic and genetic method).
GN_DIRECT	n	n	DIViding RECTangles algorithm (deterministic method).
GN_DIRECT_L	n	n	Idem GN_DIRECT with locally biased optimization.
GN_DIRECT_L_RAND	n	n	Idem GN_DIRECT_L with some randomization in the selection of the dimension to reduce next.
GN_DIRECT*_NOSCAL	n	n	Variants of above GN_DIRECT* without scaling the problem to a unit hypercube to preserve dimension weights.
GN_ESCH	n	n	Modified Evolutionary algorithm (genetic method).
GN_ISRES	y	y	Improved Stochastic Ranking Evolution Strategy algorithm (mixed genetic and variational method).
GN_MLSL	n	n	Multi-Level Single-Linkage algorithm (stochastic method).
GN_MLSL_LDS	n	n	Idem GN_MLSL with low-discrepancy scan sequence.
<i>Global optimizers with derivative (GD_)</i>			
GD_MLSL	n	n	Multi-Level Single-Linkage algorithm (stochastic method).
GD_MLSL_LDS	n	n	Idem GN_MLSL with low-discrepancy scan sequence.
GD_STOGO	n	n	Branch-and-bound algorithm (deterministic method).
GD_STOGO_RAND	n	n	Variant of GD_STOGO (deterministic and stochastic method).
<i>Augmented methods for other NLOpt optimizers</i>			
AUGLAG	y	y	Augmented Lagrangian algorithm, combines objective function and nonlinear constraints into a single "penalty" function.
AUGLAG_EQ	y	n	Idem AUGLAG but handles only equality constraints and pass inequality constraints to submethod.
G_MLSL	n	n	MLSL with user-specified local algorithm using submethod.
G_MLSL_LDS	n	n	Idem G_MLSL with low-discrepancy scan sequence.

```

        { var='MADX.kqtf_b1' },
        { var='MADX.kqtd_b1' }},
    equalities = {{ expr=\t -> t.q1-64.295, name='q1' },
                  { expr=\t -> t.q2-59.301, name='q2' }},
    objective = { fmin=1e-10, broyden=true },
    maxcall=100, info=2
}
local status, fmin, ncall = match {
  command := twiss { sequence=lhcb1, cofind=true, chrom=true,
                    method=4, observe=1 },
  variables = { rtol=1e-6, -- 1 ppm
                { var='MADX.ksf_b1' },
                { var='MADX.ksd_b1' }},
  equalities = {{ expr=\t -> t.dq1-15, name='dq1' },
                { expr=\t -> t.dq2-15, name='dq2' }},
  objective = { fmin=1e-8, broyden=true },
  maxcall=100, info=2
}

```


10.2 Matching interaction point

The following example hereafter shows how to squeeze the beam 1 of the LHC to $\beta^* = \text{beta_ip8} * 0.6^2$ at the IP8 while enforcing the required constraints at the interaction point and the final dispersion suppressor (i.e. at makers "IP8" and "E.DS.R8.B1") in two iterations, using the 20 quadrupoles strengths from kq4 to kqt13 on left and right sides of the IP. The boundary conditions are specified by the beta0 blocks bir8b1 for the initial conditions and eir8b1 for the final conditions. The final summary and an instance of the intermediate summary of this match example are shown in the Figures 14.2 and 14.3.

```

local SS, ES = "S.DS.L8.B1", "E.DS.R8.B1"
lhcb1.range = SS.."/"..ES
for n=1,2 do
  beta_ip8 = beta_ip8*0.6
  local status, fmin, ncall = match {
    command := twiss { sequence=lhcb1, X0=bir8b1, method=4, observe=1 },
    variables = { sign=true, rtol=1e-8, -- 20 variables
      { var='MADX.kq4_l8b1', name='kq4.l8b1', min=-lim2, max=lim2 },
      { var='MADX.kq5_l8b1', name='kq5.l8b1', min=-lim2, max=lim2 },
      { var='MADX.kq6_l8b1', name='kq6.l8b1', min=-lim2, max=lim2 },
      { var='MADX.kq7_l8b1', name='kq7.l8b1', min=-lim2, max=lim2 },
      { var='MADX.kq8_l8b1', name='kq8.l8b1', min=-lim2, max=lim2 },
      { var='MADX.kq9_l8b1', name='kq9.l8b1', min=-lim2, max=lim2 },
      { var='MADX.kq10_l8b1', name='kq10.l8b1', min=-lim2, max=lim2 },
      { var='MADX.kqtl11_l8b1', name='kqtl11.l8b1', min=-lim3, max=lim3 },
      { var='MADX.kqt12_l8b1', name='kqt12.l8b1', min=-lim3, max=lim3 },
      { var='MADX.kqt13_l8b1', name='kqt13.l8b1', min=-lim3, max=lim3 },
      { var='MADX.kq4_r8b1', name='kq4.r8b1', min=-lim2, max=lim2 },
      { var='MADX.kq5_r8b1', name='kq5.r8b1', min=-lim2, max=lim2 },
      { var='MADX.kq6_r8b1', name='kq6.r8b1', min=-lim2, max=lim2 },
      { var='MADX.kq7_r8b1', name='kq7.r8b1', min=-lim2, max=lim2 },
      { var='MADX.kq8_r8b1', name='kq8.r8b1', min=-lim2, max=lim2 },
      { var='MADX.kq9_r8b1', name='kq9.r8b1', min=-lim2, max=lim2 },
      { var='MADX.kq10_r8b1', name='kq10.r8b1', min=-lim2, max=lim2 },
      { var='MADX.kqtl11_r8b1', name='kqtl11.r8b1', min=-lim3, max=lim3 },
      { var='MADX.kqt12_r8b1', name='kqt12.r8b1', min=-lim3, max=lim3 },
      { var='MADX.kqt13_r8b1', name='kqt13.r8b1', min=-lim3, max=lim3 },
    },
    equalities = { -- 14 equalities
      { expr='\t -> t.IP8.beta11-beta_ip8, kind='beta', name='IP8' },
      { expr='\t -> t.IP8.beta22-beta_ip8, kind='beta', name='IP8' },
      { expr='\t -> t.IP8.alfa11, kind='alfa', name='IP8' },
      { expr='\t -> t.IP8.alfa22, kind='alfa', name='IP8' },
      { expr='\t -> t.IP8.dx, kind='dx', name='IP8' },
      { expr='\t -> t.IP8.dpx, kind='dpx', name='IP8' },
      { expr='\t -> t[ES].beta11-eir8b1.beta11, kind='beta', name=ES },
      { expr='\t -> t[ES].beta22-eir8b1.beta22, kind='beta', name=ES },
      { expr='\t -> t[ES].alfa11-eir8b1.alfa11, kind='alfa', name=ES },
      { expr='\t -> t[ES].alfa22-eir8b1.alfa22, kind='alfa', name=ES },
      { expr='\t -> t[ES].dx-eir8b1.dx, kind='dx', name=ES },
      { expr='\t -> t[ES].dpx-eir8b1.dpx, kind='dpx', name=ES },
      { expr='\t -> t[ES].mul-muxip8, kind='mul', name=ES },
    },
  }
end

```

```

    { expr=\t -> t[ES].mu2-muyip8, kind='mu2', name=ES },
  },
  objective = { fmin=1e-10, broyden=true },
  maxcall=1000, info=2
}
MADX.n, MADX.tar = n, fmin
end

```

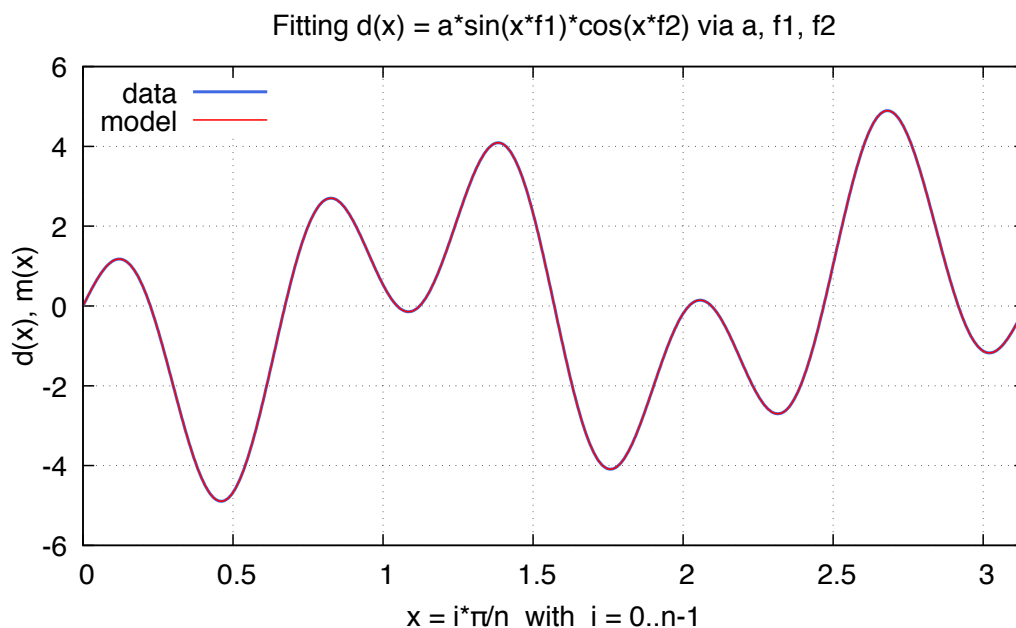
10.3 Fitting data

The following example shows how to fit data with a non-linear model using the least squares methods. The "measurements" are generated by the data function:

$$d(x) = a \sin(x f_1) \cos(x f_2), \quad \text{with } a = 5, f_1 = 3, f_2 = 7, \text{ and } x \in [0, \pi).$$

The least squares minimization is performed by the small code below starting from the arbitrary values $a = 1$, $f_1 = 1$, and $f_2 = 1$. The 'LD_JACOBIAN' methods finds the values $a = 5 \pm 10^{-10}$, $f_1 = 3 \pm 10^{-11}$, and $f_2 = 7 \pm 10^{-11}$ in 2574 iterations and 0.1 s. The 'LD_LMDIF' method finds similar values in 2539 iterations. The data and the model are plotted in the Figure 14.4.

Figure 14.4: Fitting data using the Jacobian or Levenberg-Marquardt methods.



```

local n, k, a, f1, f2 = 1000, pi/1000, 5, 3, 7
local d = vector(n):seq():map \i -> a*sin(i*k*f1)*cos(i*k*f2) -- data
if noise then d=d:map \x -> x+randtn(noise) end -- add noise if any
local m, p = vector(n), { a=1, f1=1, f2=1 } -- model parameters
local status, fmin, ncall = match {
  command := m:seq():map \i -> p.a*sin(i*k*p.f1)*cos(i*k*p.f2),
  variables = { { var='p.a' },
                { var='p.f1' },
                { var='p.f2' }, min=1, max=10 },

```

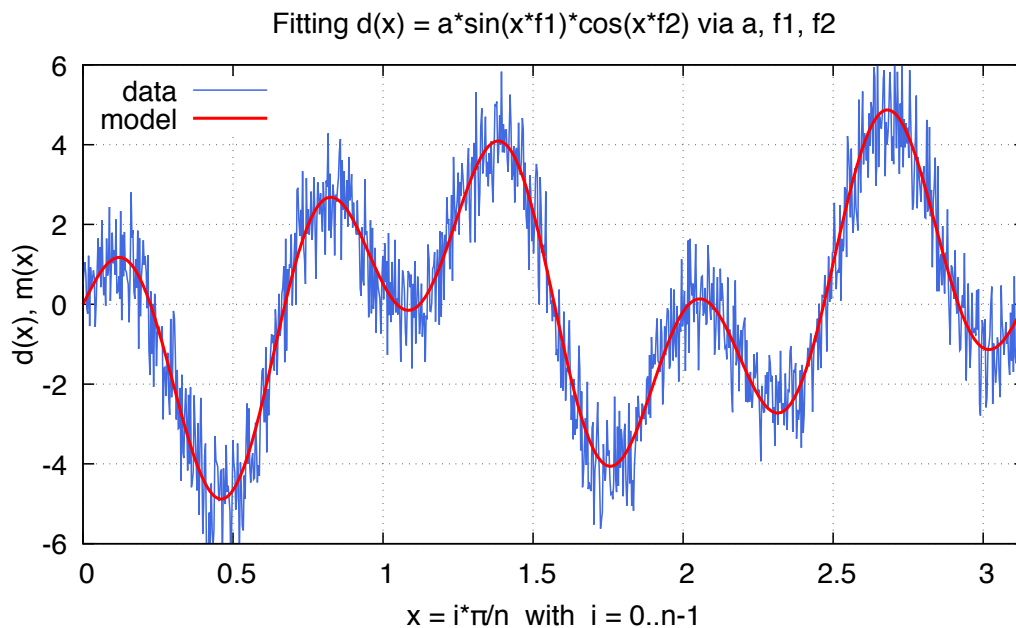
```

equalities = { { expr=\m -> ((d-m):norm()) } },
objective   = { fmin=1e-9, bisec=noise and 5 },
maxcall=3000, info=1
}

```

The same least squares minimization can be achieved on noisy data by adding a gaussian RNG truncated at 2σ to the data generator, i.e. `noise=2`, and by increasing the attribute `bisec=5`. Of course, the penalty tolerance `fmin` must be moved to variables tolerance `tol` or `rtol`. The 'LD_JACOBIAN' methods finds the values $a = 4.98470$, $f_1 = 3.00369$, and $f_2 = 6.99932$ in 704 iterations (404 for 'LD_LMDIF'). The data and the model are plotted in the Figure 14.5.

Figure 14.5: Fitting data with noise using Jacobian or Levenberg-Marquardt methods.



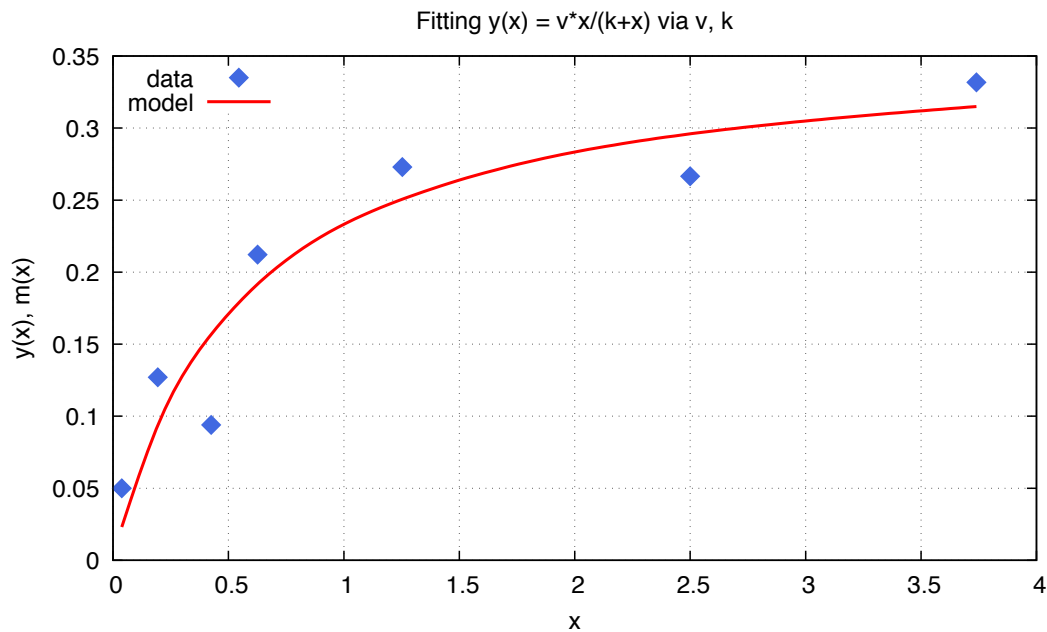
10.4 Fitting data with derivatives

The following example shows how to fit data with a non-linear model and its derivatives using the least squares methods. The least squares minimization is performed by the small code below starting from the arbitrary values $v = 0.9$ and $k = 0.2$. The 'LD_JACOBIAN' methods finds the values $v = 0.362 \pm 10^{-3}$ and $k = 0.556 \pm 10^{-3}$ in 6 iterations. The 'LD_LMDIF' method finds similar values in 6 iterations too. The data (points) and the model (curve) are plotted in the Figure 14.6, where the latter has been smoothed using cubic splines.

```

local x = vector{0.038, 0.194, 0.425, 0.626, 1.253, 2.500, 3.740}
local y = vector{0.050, 0.127, 0.094, 0.2122, 0.2729, 0.2665, 0.3317}
local p = { v=0.9, k=0.2 }
local n = #x
local function eqfun (_, r, jac)
  local v, k in p
  for i=1,n do
    r[i] = y[i] - v*x[i]/(k+x[i])
    jac[2*i-1] = -x[i]/(k+x[i])
    jac[2*i] = v*x[i]/(k+x[i])^2

```

Figure 14.6: Fitting data with derivatives using the Jacobian or Levenberg-Marquardt methods.

```

end
end
local status, fmin, ncall = match {
  variables = { tol=5e-3, min=0.1, max=2,
               { var='p.v' },
               { var='p.k' } },
  equalities = { nequ=n, exec=eqfun, disp=false },
  maxcall=20
}

```

10.5 Minimizing function

The following example⁷ hereafter shows how to find the minimum of the function:

$$\min_{\mathbf{x} \in \mathbb{R}^2} \sqrt{x_2}, \quad \text{subject to the constraints} \quad \begin{cases} x_2 \geq 0, \\ x_2 \geq (a_1 x_1 + b_1)^3, \\ x_2 \geq (a_2 x_1 + b_2)^3, \end{cases}$$

for the parameters $a_1 = 2, b_1 = 0, a_2 = -1$ and $b_2 = 1$. The minimum of the function is $f_{\min} = \sqrt{\frac{8}{27}}$ at the point $\mathbf{x} = (\frac{1}{3}, \frac{8}{27})$, and found by the method LD_MMA in 11 evaluations for a relative tolerance of 10^{-4} on the variables, starting at the arbitrary point $\mathbf{x}_0 = (1.234, 5.678)$.

```

local function testFuncFn (x, grd)
  if grd then x:fill{ 0, 0.5/sqrt(x[2]) } end
  return sqrt(x[2])
end
local function testFuncLe (x, r, jac)
  if jac then jac:fill{ 24*x[1]^2, -1, -3*(1-x[1])^2, -1 } end

```

⁷This example is taken from the NLOpt [documentation](#).

```
r:fill{ 8*x[1]^3-x[2], (1-x[1])^3-x[2] }
end
local x = vector{1.234, 5.678} -- start point
local status, fmin, ncall = match {
  variables      = { rtol=1e-4,
                    { var='x[1]', min=-inf },
                    { var='x[2]', min=0 } },
  inequalities = { exec=testFuncLe, nequ=2, tol=1e-8 },
  objective     = { exec=testFuncFn, method='LD_MMA' },
  maxcall=100, info=2
}
```

This example can also be solved with least squares methods, where the LD_JACOBIAN method finds the minimum in 8 iterations with a precision of $\pm 10^{-16}$, and the LD_LMDIF method finds the minimum in 10 iterations with a precision of $\pm 10^{-11}$.

Chapter 15. Correct

The correct command (i.e. orbit correction) provides a simple interface to compute the orbit steering correction and setup the kickers of the sequences from the analysis of their track and twiss mtables.

1 Command synopsis

Figure 15.1: Synopsis of the correct command with default setup.

```
m1st = correct {
  sequence=nil,      -- sequence(s) (required)
  range=nil,        -- sequence(s) range(s) (or sequence.range)
  title=nil,        -- title of mtable (default seq.name)
  model=nil,        -- mtable(s) with twiss functions (required)
  orbit=nil,        -- mtable(s) with measured orbit(s), or use model
  target=nil,       -- mtable(s) with target orbit(s), or zero orbit
  kind='ring',      -- 'line' or 'ring'
  plane='xy',       -- 'x', 'y' or 'xy'
  method='micado',  -- 'LSQ', 'SVD' or 'MICADO'
  ncor=0,           -- number of correctors to consider by method, 0=all
  tol=1e-5,        -- rms tolerance on the orbit
  units=1,         -- units in [m] of the orbit
  corcnd=false,    -- precondition of correctors using 'svdcnd' or 'pcacnd'
  corcut=0,        -- value to threshold singular values in precondition
  cortol=0,        -- value to threshold correctors in svdcnd
  corset=true,     -- update correctors correction strengths
  monon=false,     -- fraction (0<?<=1) of randomly available monitors
  moncut=false,    -- cut monitors above moncut sigmas
  monerr=false,    -- 1:use mrex and mrey alignment errors of monitors
                  -- 2:use msec and msec scaling errors of monitors
  info=nil,        -- information level (output on terminal)
  debug=nil,       -- debug information level (output on terminal)
}
```

The correct command format is summarized in Figure 15.1, including the default setup of the attributes. The correct command supports the following attributes:

- sequence** The *sequence* (or a list of *sequence*) to analyze. (no default, required).
Example: sequence = lhcb1.
- range** A *range* (or a list of *range*) specifying the span of the sequence to analyze. If no range is provided, the command looks for a range attached to the sequence, i.e. the attribute seq.range. (default: nil).
Example: range = "S.DS.L8.B1/E.DS.R8.B1".
- title** A *string* specifying the title of the *mtable*. If no title is provided, the command looks for the name of the sequence, i.e. the attribute seq.name. (default: nil).
Example: title = "Correct orbit around IP5".

model	A <i>mtable</i> (or a list of <i>mtable</i>) providing twiss-like information, e.g. elements, orbits and optical functions, of the corresponding sequences. (no default, required). Example: <code>model = twmtbl</code> .
orbit	A <i>mtable</i> (or a list of <i>mtable</i>) providing track-like information, e.g. elements and measured orbits, of the corresponding sequences. If this attribute is <code>nil</code> , the model orbit is used. (default: <code>nil</code>). Example: <code>orbit = tkmtbl</code> .
target	A <i>mtable</i> (or a list of <i>mtable</i>) providing track-like information, e.g. elements and target orbits, of the corresponding sequences. If this attribute is <code>nil</code> , the design orbit is used. (default: <code>nil</code>). Example: <code>target = tgmtbl</code> .
kind	A <i>string</i> specifying the kind of correction to apply among <code>line</code> or <code>ring</code> . The kind <code>line</code> takes care of the causality between monitors, correctors and sequences directions, while the kind <code>ring</code> considers the system as periodic. (default: <code>'ring'</code>). Example: <code>kind = 'line'</code> .
plane	A <i>string</i> specifying the plane to correct among <code>x</code> , <code>y</code> and <code>xy</code> . (default: <code>'xy'</code>). Example: <code>plane = 'x'</code> .
method	A <i>string</i> specifying the method to use for correcting the orbit among <code>LSQ</code> , <code>SVD</code> or <code>micado</code> . These methods correspond to the solver used from the Matrix module to find the orbit correction, namely <code>solve</code> , <code>ssolve</code> or <code>nsolve</code> . (default: <code>'micado'</code>). Example: <code>method = 'svd'</code> .
ncor	A <i>number</i> specifying the number of correctors to consider with the method <code>micado</code> , zero meaning all available correctors. (default: <code>0</code>). Example: <code>ncor = 4</code> .
tol	A <i>number</i> specifying the rms tolerance on the residuals for the orbit correction. (default: <code>1e-5</code>). Example: <code>tol = 1e-6</code> .
unit	A <i>number</i> specifying the unit of the orbit and target coordinates. (default: <code>1 [m]</code>). Example: <code>units = 1e-3 [m]</code> , i.e. <code>[mm]</code> .
corcnd	A <i>logical</i> or a <i>string</i> specifying the method to use among <code>svdcnd</code> and <code>pcacnd</code> from the Matrix module for the preconditioning of the system. A true value corresponds to <code>svdcnd</code> . (default: <code>false</code>). Example: <code>corcnd = 'pcacnd'</code> .
corcut	A <i>number</i> specifying the thresholds for the singular values to pass to the <code>svdcnd</code> and <code>pcacnd</code> method for the preconditioning of the system. (default: <code>0</code>). Example: <code>cortol = 1e-6</code> .
cortol	A <i>number</i> specifying the thresholds for the correctors to pass to the <code>svdcnd</code> method for the preconditioning of the system. (default: <code>0</code>). Example: <code>cortol = 1e-8</code> .
corset	A <i>logical</i> specifying to update the correctors strengths for the corrected orbit. (default: <code>true</code>). Example: <code>corset = false</code> .

monon	A <i>number</i> specifying a fraction of available monitors selected from a uniform RNG. (default: false). Example: <code>monon = 0.8</code> , keep 80% of the monitors.
moncut	A <i>number</i> specifying a threshold in number of sigma to cut monitor considered as outliers. (default: false). Example: <code>moncut = 2</code> , cut monitors above 2σ .
monerr	A <i>number</i> in 0..3 specifying the type of monitor reading errors to consider: 1 use scaling errors <code>msex</code> and <code>msey</code> , 2 use alignment errors <code>mrex</code> , <code>mrey</code> and <code>dpsi</code> , 3 use both. (default: false). Example: <code>monerr = 3</code> .
info	A <i>number</i> specifying the information level to control the verbosity of the output on the console. (default: nil). Example: <code>info = 2</code> .
debug	A <i>number</i> specifying the debug level to perform extra assertions and to control the verbosity of the output on the console. (default: nil). Example: <code>debug = 2</code> .

The correct command returns the following object:

m1st	A <i>mtable</i> (or a list of <i>mtable</i>) corresponding to the TFS table of the correct command. It is a list when multiple sequences are corrected together.
-------------	---

2 Correct mtable

The correct command returns a *mtable* where the information described hereafter is the default list of fields written to the TFS files.¹

The header of the *mtable* contains the fields in the default order:

name	The name of the command that created the <i>mtable</i> , e.g. "correct".
type	The type of the <i>mtable</i> , i.e. "correct".
title	The value of the command attribute <code>title</code> .
origin	The origin of the application that created the <i>mtable</i> , e.g. "MAD 1.0.0 OSX 64".
date	The date of the creation of the <i>mtable</i> , e.g. "27/05/20".
time	The time of the creation of the <i>mtable</i> , e.g. "19:18:36".
refcol	The reference <i>column</i> for the <i>mtable</i> dictionary, e.g. "name".
range	The value of the command attribute <code>range</code> . ²
__seq	The <i>sequence</i> from the command attribute <code>sequence</code> . ³

The core of the *mtable* contains the columns in the default order:

¹The output of *mtable* in TFS files can be fully customized by the user.

²This field is not saved in the TFS table by default.

³Fields and columns starting with two underscores are protected data and never saved to TFS files.

name	The name of the element.
kind	The kind of the element.
s	The s -position at the end of the element slice.
l	The length from the start of the element to the end of the element slice.
x_old	The local coordinate x at the s -position before correction.
y_old	The local coordinate y at the s -position before correction.
x	The predicted local coordinate x at the s -position after correction.
y	The predicted local coordinate y at the s -position after correction.
rx	The predicted local residual r_x at the s -position after correction.
ry	The predicted local residual r_y at the s -position after correction.
hkick_old	The local horizontal kick at the s -position before correction.
vkick_old	The local vertical kick at the s -position before correction.
hkick	The predicted local horizontal kick at the s -position after correction.
vkick	The predicted local vertical kick at the s -position after correction.
shared	A <i>logical</i> indicating if the element is shared with another sequence.
idx	The index of the element in the sequence.

Note that `correct` does not take into account the particles and damaps `ids` present in the (augmented) track `mtable`, hence the provided tables should contain single particle or damap information.

3 Examples

TODO

Chapter 16. Emit

This command is not yet implemented in MAD. It will probably be implemented as a layer on top of the Twiss and Match commands.

Chapter 17. Plot

The `plot` command provides a simple interface to the [Gnuplot](#) application. The Gnuplot release 5.2 or higher must be installed and visible in the user PATH by MAD to be able to run this command.

1 Command synopsis

x

x

Figure 17.1: Synopsis of the `plot` command with default setup.

```
cmd = plot {
  sid      = 1,    -- stream id 1 <= n <= 25 (Gnuplot instances)
  data     = nil, -- { x=tbl.x, y=vec } (precedence over table)
  table    = nil, -- mtable
  tablerange = nil, -- mtable range (default table.range)
  sequence = nil, -- seq | { seq1, seq2, ... } | "keep"
  range    = nil, -- sequence range (default sequence.range)
  name     = nil, -- (default table.title)
  date     = nil, -- (default table.date)
  time     = nil, -- (default table.time)
  output   = nil, -- "filename" -> pdf | number -> wid
  scrdump  = nil, -- "filename"
  survey-attributes
  windows-attributes
  layout-attributes
  labels-attributes
  axis-attributes
  ranges-attributes
  data-attributes
  plots-attributes
  custom-attributes
  info=nil, -- information level (output on terminal)
  debug=nil, -- debug information level (output on terminal)
}
```

The `plot` command format is summarized in [Figure 17.1](#), including the default setup of the attributes. The `plot` command supports the following attributes:

- info** A *number* specifying the information level to control the verbosity of the output on the console. (default: `nil`).
Example: `info = 2`.
- debug** A *number* specifying the debug level to perform extra assertions and to control the verbosity of the output on the console. (default: `nil`).
Example: `debug = 2`.

The `plot` command returns itself.

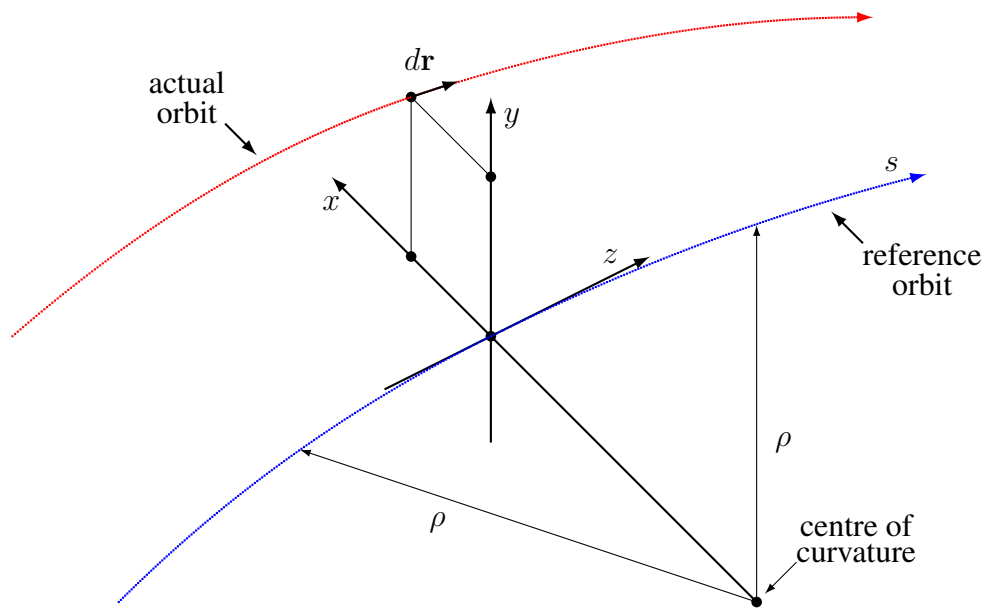
Part III

Physics

Chapter 18. Introduction

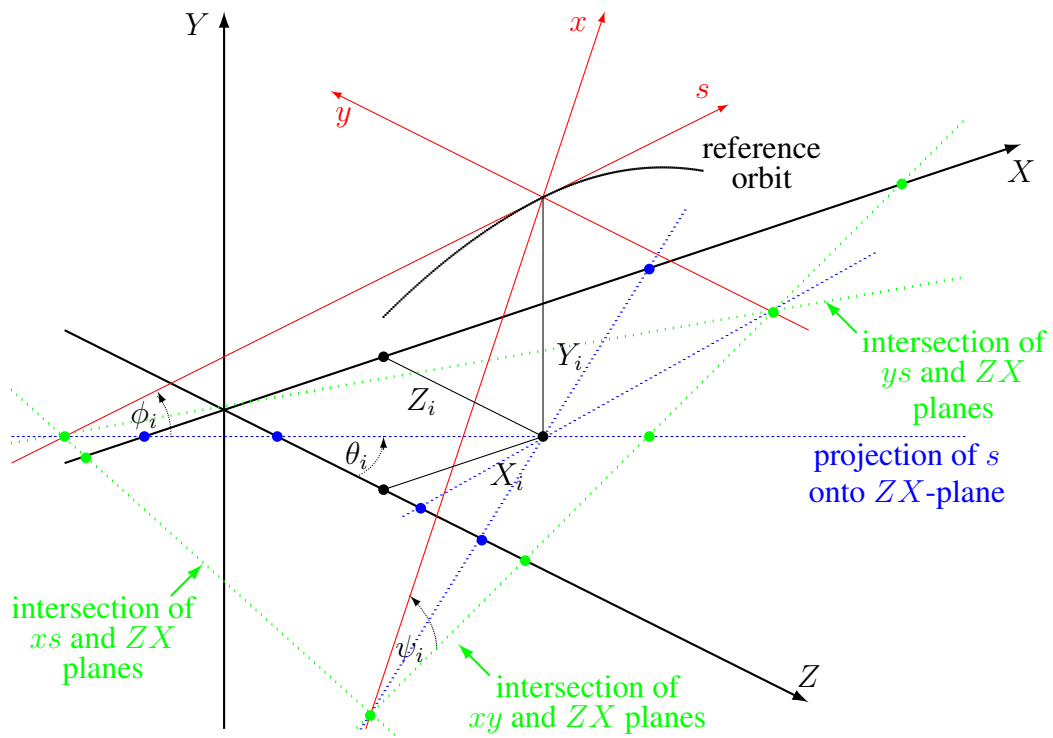
1 Local reference system

Figure 18.1: Local Reference System



2 Global reference system

Figure 18.2: 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.



Chapter 19. Geometric Maps

Chapter 20. Dynamic Maps

Chapter 21. Integrators

Chapter 22. Orbit

1 Closed Orbit

Chapter 23. Optics

Chapter 24. Normal Forms

Chapter 25. Misalignments

Chapter 26. Aperture

Chapter 27. Radiation

Part IV

Modules

Chapter 28. Introduction

Chapter 29. Types

Chapter 30. Constants

Chapter 31. Generic Utilities

Chapter 32. Generic Math

Chapter 33. Range

Chapter 34. Complex

Chapter 35. Matrix

Chapter 36. GTPSA

Chapter 37. DA Map

Chapter 38. Generic Physics

Chapter 39. External modules

Part V

Programming

Chapter 40. Introduction

Chapter 41. MAD environment

Chapter 42. Tests

1 Adding Tests

Chapter 43. Elements

1 Adding Elements

Chapter 44. Commands

1 Adding Commands

Chapter 45. Modules

1 Adding Modules

2 Embedding Modules

Chapter 46. Using C FFI

Part VI
Appendix

Chapter 47. GitHub Repository

Chapter 48. Contributors

Chapter 49. Bibliography

Chapter 50. Index

Index

`--add`, 47, 75
`--beam`, 41
`--call`, 37
`--copy`, 37, 62, 76
`--cycle`, 58, 71
`--dat`, 58, 71
`--elem`, 47
`--env`, 33
`--flg`, 33
`--id`, 33
`--index`, 33, 37, 62, 75
`--init`, 36, 41, 62, 76
`--ipairs`, 37
`--len`, 37, 47, 62, 75
`--map`, 88, 95, 111
`--metatable`, 37
`--mtbl`, 76
`--mul`, 47
`--newindex`, 37, 41, 62, 75
`--obj`, 37
`--pairs`, 37
`--parent`, 37
`--same`, 36
`--seq`, 71, 87, 94, 110, 136
`--sequ`, 62
`--tostring`, 37, 47
`--unm`, 47
`--var`, 33
`command`, 113
`equalities`, 114
`inequalities`, 114
`weights`, 114
`objective`, 114
`variables`, 113
boolean, 24
callable, 26
cdata, 25
function, 25
indexable, 26
iterable, 26
lengthable, 26
logical, 25
mappable, 26
nil, 24
number, 24
reference, 25
string, 24
table, 24
thread, 25
value, 25
NYI, 11
TBD, 11
TBR, 11
Todo, 11
`addcol`, 72
`addrow`, 73
`alfa11`, 110
`alfa12`, 112
`alfa13`, 112
`alfa21`, 112
`alfa22`, 111
`alfa23`, 112
`alfa31`, 112
`alfa32`, 112
`alfa33`, 111
`alfap`, 109
`align`, 59
`angle`, 45, 53, 87
`aperture`, 46
`apertype`, 46
`aphot`, 39
`arch`, 14
`at`, 63
`bbox`, 55
`beam`, 39, 58
`bend`, 54
`beta`, 39
`beta11`, 111
`beta12`, 112
`beta13`, 112
`beta2`, 39
`beta21`, 112
`beta22`, 111
`beta23`, 112
`beta31`, 112
`beta32`, 112
`beta33`, 111
`betgam`, 39
`betgam2`, 39
`brho`, 39
`bsearch`, 35
`charge`, 39

- check_mtbl, 75
- check_sequ, 62
- chkick, cvkick, 51
- chrom, 109
- circle, 55
- class, 64
- clear, 73
- clear_all, 34
- clear_array, 34
- clear_flags, 35
- clear_variables, 34
- close_env, 36
- clrrow, 73
- cofind
 - ataper, 101
 - atdebug, 101
 - atentry, 101
 - atexit, 101
 - atsave, 101
 - atslice, 101
 - beam, 98
 - codiff, 102
 - coiter, 102
 - comethod, 102
 - cotol, 102
 - debug, 102
 - deltap, 98
 - dir, 98
 - fringe, 100
 - implicit, 100
 - info, 102
 - mapdef, 100
 - method, 100
 - mflow, 102
 - mflw, 102
 - misalign, 100
 - model, 100
 - mtbl, 102
 - nslice, 99
 - nstep, 98
 - nturn, 98
 - 00, 98
 - observe, 101
 - ptcmodel, 100
 - radiate, 100
 - range, 98
 - s0, 98
 - save, 100
 - savemap, 101
 - savesel, 101
 - sequence, 98
 - title, 101
 - totalpath, 100
 - usrdef, 102
 - X0, 98
 - X1, 102
- cofind, 98
- colname, 71
- colnames, 71
- column, 70
- comb, 54
- combqs, 54
- copy, 33, 61, 74
- copy_variables, 34
- correct
 - corcnd, 135
 - corcut, 135
 - corset, 135
 - cortol, 135
 - debug, 136
 - info, 136
 - kind, 135
 - method, 135
 - mlst, 136
 - model, 135
 - moncut, 136
 - monerr, 136
 - monon, 136
 - ncor, 135
 - orbit, 135
 - plane, 135
 - range, 134
 - sequence, 134
 - target, 135
 - title, 134
 - tol, 135
 - unit, 135
- correct, 134
- coupling, 109
- cycle, 61, 74

- damap, 52
- date, 70, 87, 94, 109, 136
- ddpx, 111
- ddpy, 111
- ddx, 111
- ddy, 111
- deltap, 94, 109
- deselect, 46, 60, 73
- dir, 52, 58

- direction, 87, 94, 109
- disabled, 53
- dksl, 46, 50
- dmu1, 111
- dmu2, 111
- dnrj, 53
- dphi, 56
- dpsl, 51
- dpsi, 56
- dpx, 111
- dpy, 111
- dq1, 111
- dq2, 111
- dq3, 111
- drift, 43
- ds, 59
- dtheta, 56
- dtheta, dphi, dpsi, 53
- dump_env, 36
- dumpobj, 36
- dumpseq, 62
- dx, 111
- dx, dy, ds, 53, 56
- dy, 111

- e1, e2, 46
- eidx, 88, 95, 110, 137
- elem, 58
- element, 43
- ellipse, 55
- emrad, 39
- energy, 39
- et, 39
- etap, 110
- etn, 40
- ex, 39
- ex, ey, 52
- exl, eyl, 52
- exn, 40
- extrn, 43
- ey, 39
- eyn, 40

- f1, f2, 46
- filter, 60, 74
- fint, 46
- fintx, 46
- flag, 64
- foreach, 60, 73
- freq, 51
- fringe, 46, 48, 49

- fringemax, 46
- from, 63

- gama11, 111
- gama12, 112
- gama13, 112
- gama21, 112
- gama22, 111
- gama23, 112
- gama31, 112
- gama32, 112
- gama33, 111
- gamma, 39
- gammatr, 110
- Gauss-Newton, 122
- get, 72
- get_flags, 35
- get_variables, 33
- get_varkeys, 33
- getcol, 72
- getrow, 72
- Gradient-Descent, 122

- h1, h2, 46
- harmon, 51
- header, 70
- hgap, 46
- hkick, 50, 137
- hkick_old, 137

- id, 95, 110
- implicit, 53, 87, 94, 109
- index, 59, 71
- index_of, 59, 71
- index_sat, 47
- inscol, 72
- insert, 35, 74
- insert_sat, 47
- insrow, 73
- install, 60
- is_class, 33
- is_disabled, 47
- is_final, 33
- is_implicit, 47
- is_instanceOf, 33
- is_observed, 47
- is_open_env, 36
- is_readonly, 33
- is_selected, 46
- is_view, 62, 74
- iter, 59, 73

- JIT compiler, 17
- k0, 48
- k0, k0s, 49
- k0s, 48, 49
- k1, k1s, 48, 49
- k2, k2s, 48, 49
- k3, k3s, 50
- k4, k4s, 50
- k5, k5s, 50
- kick, 51
- kill_ent_fringe, 46
- kill_exi_fringe, 46
- kind, 54, 77, 87, 95, 110, 137
- knl, ksl, 46, 50
- ks, ksi, 50
- l, 45, 58, 87, 95, 110, 137
- lag, 51
- length, 109
- length_of, 59, 72
- Levenberg-Marquardt , 122
- load_env, 36
- lost, 94, 109
- lrad, 45
- lsearch, 35
- Lua, 17
- LuaJIT, 17
- m, 14
- make_dict, 75
- maper, 54
- mass, 39
- match
 - command, 115
 - constraint
 - disp, 119
 - exec, 118
 - expr, 118
 - kind, 118
 - name, 118
 - nequ, 118
 - tol, 118
 - weight, 118
 - debug, 114
 - environment
 - command, 115
 - debug, 115
 - dtime, 114
 - equalities, 115
 - inequalities, 115
 - info, 115
 - ncall, 114
 - stop, 114
 - usrdef, 115
 - variables, 115
 - weights, 115
 - equalities, 117
 - fmin, 114
 - inequalities, 117
 - info, 114
 - maxcall, 114
 - maxtime, 114
 - ncall, 114
 - objective
 - bisec, 121
 - broyden, 120
 - bstra, 120
 - exec, 120
 - fmin, 120
 - grad, 120
 - jstra, 121
 - jiter, 121
 - jtol, 121
 - method, 119
 - rcond, 121
 - reset, 120
 - rtol, 120
 - submethod, 120
 - tol, 120
 - objective, 119
 - status, 114
 - usrdef, 114
 - variables
 - get, 116, 117
 - max, 116
 - min, 116
 - name, 116
 - nowarn, 117
 - nvar, 117
 - rtol, 117
 - set, 117
 - sign, 116
 - slope, 116
 - step, 116
 - tol, 116
 - var, 116
 - variables, 116
- match, 113
- max, 117
- method, 45, 50

- min, 117
- minlen, 58
- misalign, 46, 61, 87, 94, 109
- model, 45
- move, 35, 61
- mredx, mredy, 51
- mresx, mresy, 51
- mtable, 70
- mu1, 111
- mu2, 111
- mu3, 111
- mult, 54

- n, 14
- n_bessel, 51
- name, 32, 87, 94, 95, 109, 110, 136, 137
- name_of, 59, 71
- nbunch, 40
- ncol, 71
- Nelder-Mead, 122
- new_particle, 40
- Newton, 122
- Newton-Raphson, 122
- ngen, 71
- none, 53, 54
- novector, 70
- npart, 40
- nrow, 71
- nslice, 45, 52
- nturn, 52

- object, 28
 - __init(**self**), 30
 - class, 29
 - creation, 28
 - deferred expressions, 30
 - dynamic lookup, 28
 - incomplete, 29
 - object model, 28
 - prototypical object, 28
 - single inheritance, 28
- observe, 87, 94, 109
- observed, 53
- octagon, 55
- open_env, 35
- origin, 70, 87, 94, 109, 136
- owner, 58, 70

- parent, 32
- particle, 39
- patch, 43

- pattern, 64, 76
- pc, 39
- pc2, 39
- phi, 88
- phix, 111
- phiy, 112
- playout, 53
- plot
 - debug, 139
 - info, 139
- plot, 139
- pnl, psl, 51
- polygon, 55
- Principal-Axis, 122
- print, 75
- psi, 88
- pt, 95, 110
- publish, 61
- px, 95, 110
- py, 95, 110

- q1, 109
- q2, 109
- q3, 109
- qsad, 54
- Quasi-Newton, 122

- racetrack, 55
- range, 52, 87, 94, 110, 136
- range_of, 59, 71
- raw_get, 36
- raw_len, 36
- raw_set, 36
- read, 74
- rectangle, 55
- rectcircle, 55
- rectellipse, 55
- refcol, 70, 87, 94, 109, 136
- refer, 58
- reflect, 61
- refpos, 45, 63
- remcol, 72
- remove, 35, 61, 74
- remove_sat, 47
- remrow, 73
- rencol, 72
- replace, 61
- replace_sat, 47
- reserve, 70
- restore_flags, 62
- restore_sel, 75

- rfcav, 54
- rx, 137
- ry, 137

- s, 87, 95, 110, 137
- same, 33
- save_flags, 62
- save_sel, 75
- select, 46, 60, 64, 73, 77
- selected, 53, 76
- sequence, 52, 58
- set, 72
- set_final, 33
- set_flags, 35
- set_metamethods, 35
- set_methods, 35
- set_parent, 34
- set_readonly, 33, 62, 74
- set_variables, 34, 40
- setcol, 72
- setrow, 72
- share, 61
- shared, 64, 137
- showdb, 40
- sige, 40
- sign, 117
- sigt, 40
- Simplex, 122
- siter, 60
- slc, 88, 95, 110
- slope, 117
- sort, 35, 74
- specl, 43
- spin, 39
- spos, 58
- square, 55
- status, 95, 110
- step, 117
- subelem, 64
- survey
 - A0, 85
 - atdebug, 86
 - atentry, 86
 - atexit, 86
 - atsave, 86
 - atslice, 86
 - debug, 86
 - dir, 85
 - implicit, 85
 - info, 86
 - mflow, 86
 - eidx, 87
 - mflw, 87
 - misalign, 85
 - mtbl, 87
 - nslice, 85
 - nstep, 85
 - nturn, 85
 - observe, 85
 - range, 84
 - s0, 85
 - save, 85
 - savemap, 86
 - savesel, 85
 - sequence, 84
 - title, 85
 - usrdef, 86
 - X0, 85
- survey, 84
- swprow, 73
- synch1, 110
- synch_1, 110
- synch_2, 110
- synch_3, 110
- synch_4, 110
- synch_5, 110

- t, 95, 110
- table, 64
- tdir, 88, 95, 110
- test_flags, 35
- theta, 88
- thick, 43
- thin, 43
- tilt, 45, 54, 87
- time, 70, 87, 94, 109, 136
- title, 70, 87, 94, 109, 136
- tol, 117, 118
- totalpath, 51
- track
 - ataper, 93
 - atdebug, 93
 - atentry, 93
 - atexit, 93
 - atsave, 93
 - atslice, 93
 - beam, 90
 - debug, 94
 - deltap, 90
 - dir, 90

- fringe, 92
- implicit, 92
- info, 93
- mapdef, 91
- method, 92
- mflow, 94
- eidx, 94
- mflw, 94
- misalign, 92
- model, 92
- mtbl, 94
- nslice, 91
- nstep, 90
- nturn, 90
- 00, 90
- observe, 93
- ptcmodel, 92
- radiate, 92
- range, 90
- s0, 90
- save, 92
- savemap, 93
- savesel, 93
- sequence, 90
- title, 92
- totalpath, 92
- usrdef, 94
- X0, 90
- track, 90
- true_rband, 49
- turn, 88, 95, 110
- twiss
 - ataper, 107
 - atdebug, 108
 - atentry, 107
 - atexit, 107
 - atsave, 107
 - atslice, 107
 - beam, 104
 - chrom, 104
 - codiff, 108
 - cofind, 108
 - coiter, 108
 - comethod, 108
 - cotol, 108
 - coupling, 104
 - debug, 108
 - deltap, 104
 - dir, 104
 - fringe, 106
 - implicit, 106
 - info, 108
 - mapdef, 106
 - method, 106
 - mflow, 108
 - eidx, 109
 - mflw, 109
 - misalign, 106
 - model, 106
 - mtbl, 109
 - nslice, 106
 - nstep, 106
 - nturn, 106
 - 00, 104
 - observe, 107
 - ptcmodel, 106
 - radiate, 106
 - range, 104
 - s0, 104
 - save, 107
 - savemap, 107
 - savesel, 107
 - sequence, 104
 - title, 107
 - totalpath, 107
 - usrdef, 108
 - X0, 104
 - X1, 108
- twiss, 104
- type, 70, 87, 94, 109, 136
- unique, 61
- update, 52
- upos, 59
- userdata, 25
- v, 14
- var_get, 36
- var_val, 36
- vkick, 50, 137
- vkick_old, 137
- volt, 51
- wrap_variables, 34
- write, 74
- wx, 111
- wy, 111
- x, 87, 95, 110, 137
- x_old, 137
- xoff, yoff, 54

y, [88](#), [95](#), [110](#), [137](#)
y_old, [137](#)

z, [88](#)