# MAD-X PROGRESS AND FUTURE PLANS

L. Deniau, CERN, Geneva, Switzerland

*Abstract*

The design efforts for the High Luminosity upgrade of the Large Hadron Collider (HL-LHC) will require significant extensions of the MAD-X code widely used for designing and simulating particles accelerators. These changes are framed into a global redesign of the MAD-X architecture meant to consolidate its structure, increase its robustness and flexibility, and improve its performance. Some examples of recent extensions to MAD-X like the RF-Multipole element will be presented. Improvement for models and algorithms selection providing better consistency of the results and a wider range of use will be discussed. The computation efficiency will also be addressed to profit better of modern technologies. In this paper, we will describe the last improvements and the future plans of the project.

## INTRODUCTION

The Methodical Accelerator Design (MAD) project has a long history, aiming to be at the forefront of computational physics in the field of particle accelerator design and simulation. The MAD scripting language is de facto the standard to describe particle accelerators, simulate beam dynamics, and optimize beam optics.

MAD-X is the successor of MAD-8 and was first released in June 2002 [1]. It offers most of the MAD-8 functionalities, with some additions, corrections, and extensions [2]. The most important of these extensions is the Polymorphic Tracking Code (PTC) of E. Forest [3].

A decade after its first release, MAD-X is still the main tool used to design and simulate accelerators at CERN. But its original design was mainly focusing on the urgent needs for the LHC, and a large part of the code was inherited from old software written in the 80's. The framework of the LHC upgrade studies is a good opportunity to reorganize and upgrade the overall core of MAD-X to support recent hardware (64 bit, multicores) and new needs. In parallel, the project must continue to incorporate new functionalities in the legacy code, like the two recent optical elements added for modeling thin RF-Multipoles and thin non-linear lenses with elliptical magnetic potential.

The long-term evolution of MAD-X is an essential aspect of the project, because many users around the world consider the application as one of the most flexible and accurate for optics design, as mentioned in comparisons of optics codes [4, 5]. Moreover, MAD-X with PTC is often taken as the reference for benchmarking other codes, and seen by the community of particle accelerator physicists as a key tool that is poised to evolve.

In this paper we expose four different aspects of the project, namely the project improvements and the feature extensions performed during the past year, and the physics and application improvements that are planned for the next couple of years. Each of these aspects addresses different concerns of the project, which are of equal importance from our point of view for the future of MAD-X.

## PROJECT IMPROVEMENT

### *Motivations*

The MAD-X project falls into the category of middle-size complex projects. The size of the source code ($\approx$ 165K SLOC[1]) and the number of features provided is not very large but most of the features rely on very complex knowledge difficult to implement and support. In this kind of project, *simplicity and discipline* should have been the rule of thumb during the development process, because the complexity comes inevitably from the implementation of the provided features. Presumably due to priority issues, these rules were not strictly followed during the first decade of development of MAD-X and the drift resulting from the added complexity has led to the untidy feeling perceived by the users. The usual software metrics based on the number of SLOC under-evaluates the complexity of the application. As a consequence, a new improvement process has been set out (Fig. 1), starting from the outside layers of the project to emphasize the restoration of some cross-platform invariants (e.g. build and test system), and to simplify the development process (e.g. code organization) before important new developments is launched.

### *Global Redesign*

The need for a global redesign of MAD-X became obvious with time, as the amount of resources required to implement new features became exponential or equivalently, the time to completion with constant resources became logarithmic. The observable behavior of the application was not always matching the users expectations, and the feedback from the active MAD community has been collected during the past decade and recorded into a project tracker.

After some attempts to improve locally the implementation, it became clear that the process would not converge to a stable solution because some undecidable and coupled bugs were found.

**Undecidable bugs** occur for example, when memory must be managed in the absence of proper reference handling or garbage collection: freeing the faulty object crashes the application while not freeing the object provokes significant memory leaks that will crash the application later. This latter kind of bug can be painful for ap-

[1] Source Line Of Code

ISBN 978-3-95450-116-8

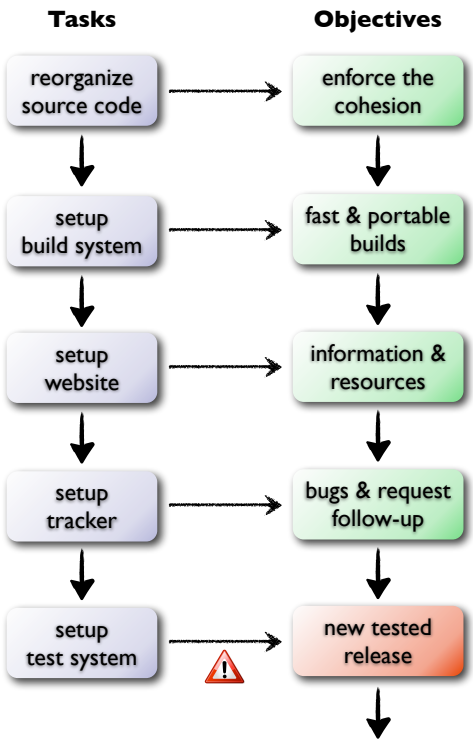


Figure 1: Sequence of tasks and corresponding objectives required before starting new MAD-X developments. The danger icon mark the difficulty to ensure that tests validate *enough* the application.

plications that run MAD-X as a subprocess, like the Java-API JMAD [6] used by the Optic and Knobs Management application, a component of the LHC online model project [7].

**Coupled bugs** (i.e. bugs with a life cycle) are more difficult to identify because they reappear after few debugging cycles, sometimes months later, where the final state makes the situation worst than the initial state. Some coupled bugs were identified to be a consequence of the abuse of global variables in the code. Hopefully, tools like Subversion and Trac allow keeping history and rolling back the modifications.

In the light of these attempts, it was concluded that only a significant redesign of the core could cure the situation.

### *Improvement Strategy*

When it is realized that a large portion of the code needs to be redesigned, the option often envisaged is to start a new development from scratch and profit from the lessons learnt from the previous project. This approach offers more opportunities for strategic changes, like adopting better supported programming languages and tools, but usually requires significant time investment.

In the case of MAD-X, the option retained was to improve the project on the existing ground, making the task more difficult because it has to fit with the legacy design and code. This strategy was motivated by past experience to minimize the risk to end up with inadequate project on the mid-term (e.g. MAD9) and assuring a continuous maintainability of the code. The application had to remain operational at every step of the evolution, a very important point for satisfying the MAD community. The goal was also to avoid the problems encountered during the first attempt resulting from the coupled bugs, and to make the process convergent toward a stable solution. The list and the details of the 24 major tasks achieved during these past months can be found on the project roadmap "phase one" [8].

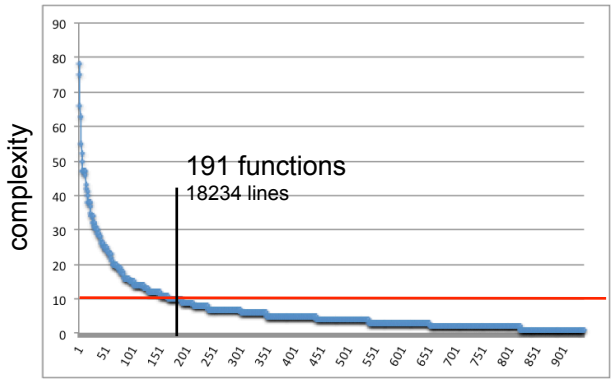


Figure 2: Cyclomatic complexity of functions written in C. The horizontal red line indicates the complexity threshold of 10 where it is recommended to split the function into smaller ones.

### *Code Complexity*

MAD-X has hundreds of large functions involving few hundred of SLOC each. The *cyclomatic complexity*[2] [9] was considered as an appropriate software metric (i.e. measuring control flow branches) for this *structured programming*[3] style to identify the critical functions. The Fig. 2 shows the cumulated number of functions above given cyclomatic complexity for the C part of the code, which represent about 20% of the application. The Structured Testing Methodology of the National Institute of Standards and Technology recommends to split functions with *complexity above 10* into smaller units §2.5 [9]. About 20% of all functions in C, C++ and Fortran are concerned (770 over 3908). *Above a complexity of 20*, the function becomes a serious candidate for bugs because it has reached the testability limit, as too many input variables are needed to cover all the branches of the function. It is also more difficult for the developers to grasp the control flow and to identify the kind of configurations that could result in unexpected or invalid states. About 8% of the functions (298 over 3908) are concerned in MAD-X.

[2] `http://en.wikipedia.org/wiki/Cyclomatic_complexity`
[3] By opposition to *object oriented programming*.



ISBN 978-3-95450-116-8

### *Code Reorganization*

The core of MAD-X was developed focusing on the demands for the design of the LHC and the optimization of its optics. Its architecture appears to be monolithic, weakening the cohesion and strengthening the coupling in the source code. This *single-file* approach prevented modular architecture exposing only well-defined interfaces; all functions and most variables ended in the global namespace, leading to a large state machine with unexpected side effects.

One of the first major task to reorganize the source code was to split the tens of thousands C lines of the core into 62 files following the *Separation of Concerns* principle [10], and to close the *visibility*[4] of the variables and functions as much as possible. The reorganization of the core helped to detect inconsistent function signatures and to correct a couple of stack frame corruptions. This effort was not enough to get rid of the structural problems, namely dangling pointers, reentrancy and memory leaks, which will be addressed by the future developments.

### *Website and Tracker*

The MAD-X website [8] has been completely redesigned to provide the community an easy access to the information and materials for download: documentations, examples, accelerator optics and new releases. The online manual is undergoing important modifications during the summer 2012. The new website has been visited more than 25000 times during the past six months, testifying the activity of the community.

The Trac web application [11] was setup and fed with all the reported bugs and requests to improve the follow-up of the pending tickets and backup the solutions. An invaluable tool that allowed to detect (non-)convergence of bug corrections, classify the issues, and steer the priorities toward the users requests. The latter will be mentioned using Trac ticket numbers all along this paper in order to highlight the orientation on user requests and projects needs.

### *Portability and Builds*

Another important aspect of the project was the ability to deliver portably and synchronously the MAD-X releases for all the main platforms — Linux, MacOS X and Windows — almost automatically, and save the resources previously allocated to this task. It was important to rely upon single methodology and technology for building and testing the application, because some identified bugs were resulting from mixing incompatible compilers settings.

For this purpose, a new portable and efficient build system has been developed, which handles all the three platforms aforementioned in 32 bit and 64 bit[5] architectures, and allows to build applications and libraries mixing code written in C, C++, Fortran 77 and Fortran 95. The new build system supports 12 compilers and provides extensible, portable, and easy-to-configure makefiles. As a demonstration of its flexibility, it took only a couple of days to extend the build system to build and distribute MAD-X and PTC as standalone libraries. The former was a request from advanced MAD-X users [11] ticket #156, and the latter was part of the collaboration with the PTC-ORBIT activities at CERN [12].

```
[ Jacobian testsuite ]
 + test-jacobian               (0.00 s) -  1/ 1 : PASSED
 + test-jacobian-2             (0.00 s) -  1/ 1 : PASSED
 + test-jacobian-knobs         (0.00 s) -  2/ 2 : PASSED
[ RF multipole testsuite ]
 + test-rfmultipole            (0.00 s) -  9/ 9 : PASSED
 + test-rfmultipole-2          (0.00 s) -  2/ 2 : PASSED
 + test-rfmultipole-3          (0.00 s) -  2/ 2 : PASSED
 + test-rfmultipole-4          (0.00 s) -  2/ 2 : PASSED
[ PTC Twiss testsuite ]
 + test-ptc-twiss              (0.00 s) -  4/ 4 : PASSED
```

Figure 3: Example showing the output of few test suites after a run of the MAD-X test system. About a hundred of tests will be setup by the end of 2012.

### *Test System*

Before relaunching large-scale development, it was essential to equip MAD-X with an efficient and robust test system. On the long term, it will give the confidence that improvements are incremental, and it will avoid unexpected regressions or undesirable backward incompatibilities.

The new test system was implemented as an extension of the new build system, taking advantage of its portability and flexibility. In the long term, it fully automatizes hundreds of tests grouped into test suites (see Fig. 3), which could be run as often as needed to check the correct behavior of the MAD-X components; a task that was performed manually and was taking days beforehand. Another purpose is to check the stability of the numerical algorithms across combinations of platforms and builds settings.

At the time of writing, the new test system already revealed a dozen of bugs never detected before, like the asynchronous outputs between C and Fortran when executed on Windows in batch mode and built with GNU compilers (tickets #159, #160), the truncation of outputs on Windows when built with Intel compilers (ticket #161) or the implicit drifts computation during beam-lines construction that varies between platforms (ticket #155).

The test process is schematized in Fig. 4. The principle is to run a single MAD-X job for each test, and to compare all the output and data files generated (purple boxes) with their reference files (yellow boxes). This black box testing[6] process relies heavily on the `numdiff` tool, a new program developed for running test suites that need to compare unformatted files with plain numerical content, and where text

[4] `http://en.wikipedia.org/wiki/Scope_(computer_science)`

[5] The source code of MAD-X is not yet 64 bit ready, therefore this achievement must be seen as a first step toward 64 bit compliance.

[6] By opposition to white box or glass box testing methods.

ISBN 978-3-95450-116-8

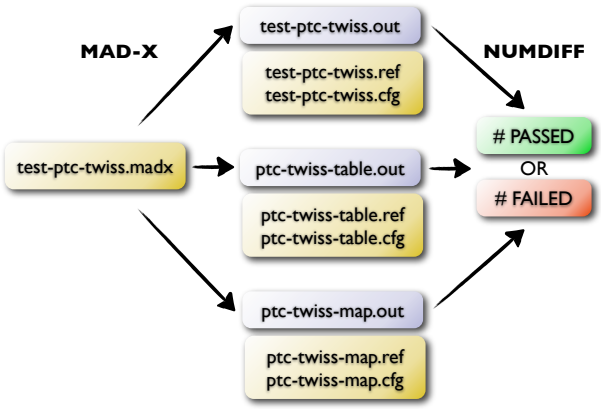


Figure 4: Schematic description of the process used to test MAD-X components. This PTC Twiss test runs a single `madx` job that generates three outputs. The yellow boxes are reference files provided to `numdiff` by the test system and backup on SVN.

```
# Test config for the Jacobian knobs
# file test-jacobian-knobs.cfg

# rows     cols     constraints
1-7        *        skip          # head banner
149-$      *        skip          # tail banner

# first matching
37-38      1-2      rel=1e-12
39         2        abs=1e-21     # from job
41         1        rel=1e-12

# second matching
109-110    1-2      rel=1e-12
111        2        abs=1e-21     # from job
113        1        rel=1e-12
```

Figure 5: Example of `numdiff` configuration file used for the test of the matching command using the Jacobian algorithm with knobs.

and numbers can be arbitrarily mixed [13].

This `numdiff` tool is able to deal efficiently with user-defined constraints applicable to numerical comparisons, and understand complex specifications with both, numerical precision and physical accuracy. The configuration files use a simple format similar to `crontab` files to specify the constraints, i.e. Fig. 5. Most of the time, table-like files (e.g. Twiss and Survey tables) can be compared directly using the default behavior of `numdiff`, which always treats input numbers as floating point values independently of their string representation. This makes the tests easy to configure and portable, and avoid spurious alarms to be reported by tests run on different platforms or with different build settings.

### *Future Plans*

The project improvement is planned to finish by the end of 2012, once the following list of items will be achieved:

- Setup more tests
- Cleanup the examples
- Cleanup the documentation
- Produce "pro" release 5.01.00

Development releases of MAD-X are actually untested releases that include most recent extensions (see next section) and bug corrections. After the completion of these items, the baseline will be to check the development releases against all the registered tests, and the production releases against all the registered tests augmented with the CERN studies. This will put the MAD project into a position where new developments will be possible and safe.

## RECENT EXTENSIONS

In this section, we present recent extensions that have been added to MAD-X. They have been implemented into the legacy code, in parallel to the project improvements aforementioned, and following the requirements of CERN projects like HL-LHC and the concerns registered on [11].

### *RF-multipole [14] (Tickets #104, #114)*

**Motivation** The electromagnetic field of accelerating structures and crab-cavities can exhibit undesired transverse field components due to asymmetries in the azimuthal direction of the element geometry, for instance in the input and output power couplers. Tracking simulations are performed to evaluate the impact of such transverse RF-kicks on the beam dynamics [15].

**Model** The RF-field is decomposed into pulsed normal and skew components similarly to the multipolar field decomposition used to model the magnetic elements [16]:

$$C_n = B_n + i\,A_n. \tag{1}$$

The RF-multipolar coefficients oscillate at the same angular frequency $\omega_{\mathrm{RF}}$ as the generating electromagnetic field, and vary with the relative longitudinal position $z = -c\Delta t$ of the particle experiencing the kick to the synchronous particle. Hence, the formalism of Eq. 1 can be written as:

$$\tilde{C}_n(z) = \tilde{B}_n(z) + i\,\tilde{A}_n(z), \tag{2}$$

with

$$\begin{aligned} \tilde{B}_n(z) &= \mathrm{Re}\left[B_n\, e^{i(\vartheta_n - k_{\mathrm{RF}} z)}\right], \\ \tilde{A}_n(z) &= \mathrm{Re}\left[A_n\, e^{i(\varphi_n - k_{\mathrm{RF}} z)}\right], \end{aligned} \tag{3}$$

where $k_{\mathrm{RF}}$ is the RF wave number ($k_{\mathrm{RF}} z = -\omega_{\mathrm{RF}}\Delta t$), and $\vartheta_n$ and $\varphi_n$ are the $n$-th normal and skew phases. Hence, to characterize a RF-Multipole element, it is necessary to specify the RF voltage $V_{\mathrm{RF}}$, frequency $f_{\mathrm{RF}}$ and



ISBN 978-3-95450-116-8

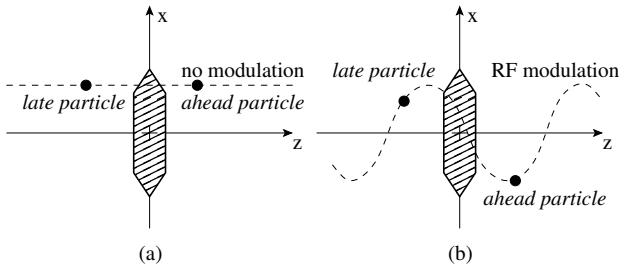


Figure 6: In an ordinary multipole (a), each particle experiences a kick that depends only on its transverse coordinates; in a RF-Multipole (b), the strength of the kick is also a function of the relative longitudinal coordinate $z$.

phase $\vartheta_{\rm RF}$, plus the amplitude and the phase of each normal and skew harmonic:

$$\text{RF-Multipole:} \begin{cases} V_{\rm RF} & \textit{RF voltage} \\ f_{\rm RF}, \vartheta_{\rm RF} & \textit{RF freq. \& phase}, \\ C_n = B_n + i\,A_n & 2\times n \textit{ amplitudes}, \\ \phi_n = \vartheta_n + i\,\varphi_n & 2\times n \textit{ phases}. \end{cases}$$

Within MAD-X, the convention is to provide the kicks expressed in terms of the integrated multipole strength $K_n L$ instead of the field multipoles:

$$\begin{aligned} B_n + iA_n &= \frac{1}{n!}\left[K_{N,n}L + iK_{S,n}L\right] \\ &= \frac{1}{n!}\left[K_{N,L,n} + iK_{S,L,n}\right] \end{aligned} \tag{4}$$

**Physics** The Hamiltonian of an ultra-relativistic particle traversing a thin RF-Multipole is similar to that of an ordinary thin multipole with RF modulation due to the rotating harmonics, plus a zero-order term for the longitudinal acceleration [17]:

$$\begin{aligned} H = &-\frac{1}{k_{\rm RF}}\frac{qV_{\rm RF}}{p_s c}\cos(\vartheta_{\rm RF} - k_{\rm RF} z) \\ &+ \sum_{n=0}^{N}\frac{1}{(n+1)!}\mathrm{Re}\Big[\Big(K_{N,L,n}\cos(\vartheta_n - k_{\rm RF} z) \\ &+ iK_{S,L,n}\cos(\varphi_n - k_{\rm RF} z)\Big)(x+iy)^{n+1}\Big] \end{aligned} \tag{5}$$

From this expression one can derive the kick experienced by a particle at coordinates $(x, y, z)$ with respect to the reference particle (Fig. 6):

$$\begin{aligned} \Delta p_x = &-\sum_{n=0}^{N}\frac{1}{n!}\,\mathrm{Re}\Big[\Big(K_{N,L,n}\cos(\vartheta_n - k_{\rm RF} z) \\ &+ iK_{S,L,n}\cos(\varphi_n - k_{\rm RF} z)\Big)(x+iy)^n\Big] \\ \Delta p_y = &\sum_{n=0}^{N}\frac{1}{n!}\,\mathrm{Im}\Big[\Big(K_{N,L,n}\cos(\vartheta_n - k_{\rm RF} z) \\ &+ iK_{S,L,n}\cos(\varphi_n - k_{\rm RF} z)\Big)(x+iy)^n\Big] \\ \Delta p_z = &\frac{qV_{\rm RF}}{p_s c}\sin(\vartheta_{\rm RF} - k_{\rm RF} z) \\ &-k_{\rm RF}\sum_{n=0}^{N}\frac{1}{(n+1)!}\,\mathrm{Re}\Big[\Big(K_{N,L,n}\sin(\vartheta_n - k_{\rm RF} z) \\ &+ iK_{S,L,n}\sin(\varphi_n - k_{\rm RF} z)\Big)(x+iy)^{n+1}\Big] \end{aligned} \tag{6}$$

**Implementation** A new thin element `RFMULTIPOLE` has been added to the Twiss and Track modules of MAD-X. This command accepts all the attributes of the existing command `MULTIPOLE` (i.e. `L, LRAD, TILT, KNL, KSL`), augmented with the quantities inherited from an RF element (i.e. `RMF_XXX` parameters), plus the normal and skew phases arrays `PNL` and `PSL` specific to this element. The syntax of the command to create a RF-Multipole is [8]:

```
RFMULTIPOLE,
 L=real, LRAD=real, TILT=real,
 RFM_VOLT=real, RFM_LAG=real,
 RFM_HARMON=integer, RFM_FREQ=real,
 KNL = { knl0, knl1, ... }, ! Normal amplitudes
 KSL = { ksl0, ksl1, ... }, ! Skew   amplitudes
 PNL = { pnl0, pnl1, ... }, ! Normal phases [2pi]
 PSL = { psl0, psl1, ... }; ! Skew   phases [2pi]
```

**Export to SixTrack** For the tracking needs of the LHC, the SixTrack code is routinely used [18]. The data loaded for simulations are provided by the MAD-X interface, which has been extended to export the RF-Multipoles. SixTrack currently accepts only quadrupoles, sextupoles, and octupoles, the dipoles being exported as a crab-cavity element. Therefore, a RF-Multipole in MAD-X is converted into and up to three equivalent thin elements in SixTrack and higher order terms are discarded. Simulations of RF-Multipoles associated to crab-cavities will be started soon for LHC upgrade studies.

### *Intrabeam Scattering (Ticket #103)*

The main improvement to the IBS implementation in MAD-X was to add the missing terms $6\beta_x^2\phi_x^2/(H_x\epsilon_x)$ and $6\beta_x^2\beta_y\phi_x^2/(H_x\epsilon_x\epsilon_y)$ in the expressions for $a_x$ and $b_x$ [19]. There were also some other minor issues; the dispersion in the MAD-X Twiss table is defined as $\Delta x/(\beta\Delta\delta)$ with $\beta = v/c$, which differs from the standard convention for $\beta$ smaller than 1. We also now distinguish the r.m.s. relative energy spread $\Delta E_{\rm rms}/E$ from the r.m.s. relative momentum spread, $\delta_{\rm rms}$ according to $\sigma_\delta \equiv \delta_{\rm rms} = (\Delta E_{\rm rms}/E)/\beta^2$.

ISBN 978-3-95450-116-8

### *Nonlinear Lenses (Ticket #105)*

The new NLLENS element models a thin nonlinear lens with the magnetic potential of 'Elliptic' type as specified in [20]. The lens is used to create fully integrable 2D nonlinear accelerator lattice with very large nonlinear tune spread/shift. The NLLENS element is recognized by the thin tracking module of MAD-X and the quadrupole term of the potential is included in the transport map computation and effects the calculation of tunes and Twiss functions.

### *PTC Beam-beam Element (Ticket #167)*

The beam-beam element of PTC was recently connected to the equivalent MAD-X element to study single-particle source of beam losses during the squeeze of the LHC beams. The connection and the physics of the beam-beam element of PTC are undergoing validation and the scripts will be used to setup the test suite.

## PHYSICS IMPROVEMENT

### *Motivation*

Particle accelerator optics codes [4, 5], including MAD-X, are routinely used by physicists to perform the following tasks:

1. **definition**: define or modify machine parameters using the MAD language.
2. **tacking**: track particles or maps to find periodic, quasi-periodic or constrained solutions, i.e. one-turn map and closed orbit[7].
3. **analysis**: compute optics functions for the one-turn map, use normal forms for high-order terms.
4. **optimization**: optimize the design with user-defined constraints, e.g. interaction regions matching.
5. **validation**: perform single-particle tracking campaign to validate the design, e.g. check the dynamic aperture.

where each step frequently includes iterations over the preceding steps.

Step 1 is mainly related to the scripting language and its interpreter, as well as the management of the data structures representing the machine layout; this important aspect will be discussed in the next section. Steps 2 and 4 are the heart of MAD-X and may include extra matching procedures to find fixed points of transfer maps between boundary conditions or periodic conditions (i.e. closed orbit). Step 3 is the difficult part for the physicists, because the obtained results depend on the choice and the parameters of the integrators selected in step 2; this is where PTC excels and offers many options. Finally, step 5 is the validation of long-term model behavior through massive tracking, like for example dynamic aperture studies. Actually MAD-X is not really efficient for this kind of task and export the lattice attributes to SixTrack, which is better suited for massive tracking.

The main problem encountered by MAD-X users with the physics concerns the discrepancy between the models of the machine elements used by the different integrators. To quote L. Nadolski [5], *"the amazing discrepancy between codes has its origin mainly in the integrator scheme"*. In practice, MAD-X contains many tracking codes and gathers somehow most of the problems of consistency.

### *Modeling the Lattice*

**Track** The Track module is a single-particle 6D drift-kick-drift symplectic integrator, which accepts only thin elements sliced by the *Makethin* module. The fixed point (i.e. closed orbit) is computed by a 6D Newton approximation. Track has symplectic integrators for drift, multipole, cavity, and few other special elements like solenoid and orbit corrector, as well as recently added elements. The splitting method of Makethin uses equidistant slices for all thick elements (e.g. dipoles, quadrupoles, sextupoles, octupoles, etc.) if the number of slices requested is greater than 4, otherwise it uses the TEAPOT placements and coefficients [21]. This method is based on the matching of the sine and cosine trajectories at their respective focal points between the thick and the thin representations of the same quadrupole, minimizing the error for the first order approximation. This gives the positions and the strength fraction of each slice relative to the thick quadrupole center and integrated strength. This geometrical method builds in practice a $2^{\text{nd}}$ order symplectic integrator as noted in [22], which is then applied indifferently to all kind of thick elements.

**Twiss** The Twiss module is an *almost*-symplectic 4D matrix-kick-matrix integrator used to track the linear terms of the transfer maps in 6D, and performs optical functions analysis from the Jacobian matrix $R$ ($1^{\text{st}}$ order) and the tensor $T$ ($2^{\text{nd}}$ order). Twiss uses a simplectification procedure described in [23] to restore the symplecticity of $R$ after each element if the deviation becomes significant. It must be noted that (Six)Track and Twiss use different element models and methods to find the reference orbit, and that Twiss uses the lattice *as-defined*, treating thick elements directly. This means that to make the two different machine representations and models behavior equivalent between thin and thick, the users have to match some parameters (e.g. beta functions) between the two lattice models using knobs (e.g. quadrupoles strengths). Once the convergence is achieved, the (Six)Track simulations can be run on the *almost* equivalent model from Twiss. This manual procedure is somehow painful and time consuming during complex process where the matching must be performed iteratively; as for example during the squeeze that occurs simultaneously at all the interaction regions, in particular IR1 and IR5, before the two beams enter into collision in

[7] In the following of this paper, we denote for convenience the *one-turn map* as the composition of the transfer maps between two boundaries and the *closed orbit* as the fixed point of the one-turn map.



ISBN 978-3-95450-116-8

the LHC.

**PTC** The PTC library embedded into MAD-X provides symplectic integrators for the naive 2nd order method, the Ruth-Neri-Yoshida 4th order method and the Yoshida 6th order method [24]. It supports the drift-kick-drift, matrix-kick-matrix and delta-matrix-kick-matrix models. The latter has been introduced for speed and compatibility with the SixTrack[8] application. PTC handles exact as well as expanded version, i.e. for the square root, of the Hamiltonians. The number of integration steps (slices) NST of the splitting model is specified by the user. But if the RESPLIT flag is provided, NST is scaled by the integrated focusing strength $KL$ divided by the parameter THIN for quadrupoles, or XBEND for dipoles. The order of the integrator is selected by the two values of the parameter LIM according to the rules:

- $\text{LIM}_1 \geq KL/\text{THIN}$ : 2nd order
- $\text{LIM}_2 \geq KL/\text{THIN} > \text{LIM}_1$ : 4th order
- $KL/\text{THIN} > \text{LIM}_2$ : 6th order

This splitting scheme makes the final layout quite different from the layout created by the Makethin module or treated by the Twiss module. Hence, the aforementioned discrepancy between the integrator models becomes prominent because the communication between MAD-X and PTC in only one-way; results computed by PTC cannot be reused directly by MAD-X for the time being.

The study [25] makes a detailed comparison of the transfer maps computed by PTC and by Twiss for a single dipole magnet (SBEND and RBEND) under various conditions and inputs. The main discrepancies observed are relative to the variations in $\Delta p/p$ for off-momentum beams with $p_x(0) \neq 0$, and to the 6th canonical variable $p_t$, which is always zero in Twiss (Twiss is 4D). Hence, PTC and MAD-X outputs do not match even when tracking through a single element.

## *Symplectic Integrators*

The Fig. 7 schematizes the process used to derive symplectic integrators, i.e. geometric integrators that preserve symplectic structures, from the Lorentz forces induced by magnets and RF-cavities in charged particles accelerator. The "danger" icons mark boxes resulting from integration steps where approximations are usually needed to solve the differential equations of the previous level. In this paper we are mainly interested by the second integration step involving the construction of symplectic integrators that solve the motion equations. When the Hamiltonian is not integrable directly (right branch in Fig. 7), one can either (1) find an approximate solution to the exact Hamiltonian, (2) find an exact solution to an approximate Hamiltonian, or (3) model the potentials and energies such that the obtained Hamiltonian becomes integrable (i.e. upward arrow) as in the RF-Multipole example.

[8] SixTrack is a matrix-kick-matrix code, this mode is also an extension.

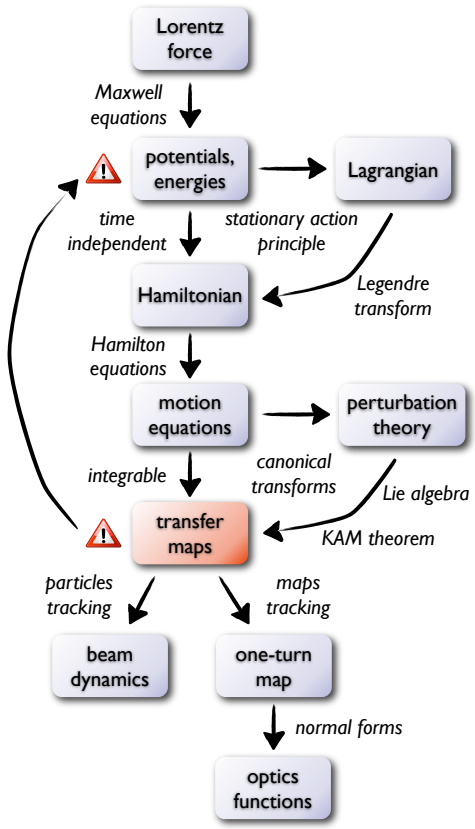


Figure 7: Schematic process to construct symplectic integrators (red box) and their application to particles beam optics. The danger icons mark integration steps where approximations may occur to solve the differential equations.

**Implicit integrators** The former approach (1) is used to construct implicit integrators (numerical integrators) developed to solve large class of motion equations, typically the class derived from time-independent Hamiltonian that expressed the total energy of the system as the sum of the potential and the kinetic energies.

These integrators do not preserve the phase space symplectic structure of the canonical variables $(\mathbf{q}, \mathbf{p})$; the $2n$-form $d\mathbf{p} \wedge d\mathbf{q} \equiv \{,\}$, i.e. the Poisson brackets, does not preserve the volumes [26] p. 204-207. In other words, they do not preserve the first integral invariant (i.e. incompressibility) of the Hamiltonian flow $\{, H\}$ (Liouville's theorem), at the heart of the stationary action principle. Such implicit integrators let the error of the total energy of the system grow secularly, i.e. diverging spirals appear on the Poincaré sections of the phase space. The map of such integrator does not describe the motion in an Hamiltonian system.

However, the long-term behavior can be kept under control by using backward error analysis and averaging transformations to make the system time-reversible (i.e. integrable) and restore the *near-preservation* of the symplecticity [27] (ch. XII). Nevertheless, these methods are not recommended in [22] by E. Forest, the author of PTC.

ISBN 978-3-95450-116-8

**Explicit integrators** The second approach (2) is used to construct symplectic integrators (i.e. transfer maps) that give exact solutions for a *perturbed* Hamiltonian admittedly close to the original Hamiltonian, with the guarantee that there is no secular trend in the error of the total energy caused by truncation error. This latter statement does not prevent symplectic integrator to be inaccurate, and yet very stable and precise. Their practical interest lies in the fact that many physical systems are perturbations of integrable systems.

A non-integrable Hamiltonian $H$ can be seen, under regular conditions specified by the Kolmogorov-Arnold-Moser theorem, as the perturbed version of and integrable systems $\tilde{H} = H - \delta H$. Various methods and formulae are provided by the perturbation theory to find the appropriate generating functions and canonical transformations that formulate the Hamiltonian on a different configuration space and exhibit properties — stabilization, separation, cancellation, invariance, etc. — useful to simplify and solve the problem. In particular, the Baker-Campbell-Hausdorff formula is very useful to split, eliminate (the perturbative terms to some order), and combine non-commutative maps while preserving the symplectic structure of the system, as in the construction of high-order symplectic integrators [24].

There is no fundamental difference between slicing an element and splitting its Hamiltonian into integrable parts. In both cases, the goal is to build the maps of a symplectic integrator that solve the motion equations in that element. With the exception that geometric interpretation of the resulting high-order maps may leads to negative drifts, somehow unphysical.

### *Tracking Maps*

In practice, the order of a symplectic integrator is given by the order of the error of its maps in term of the independent variable between the exact and the perturbed Hamiltonians. To model the transfer map of a single element or a fraction of this element, i.e. a slice, $2^{\text{nd}}$, $4^{\text{th}}$ or $6^{\text{th}}$ order symplectic integrators are generally accurate enough for *particles* tracking. But to model a full one-turn map $M = M_n \circ \cdots \circ M_2 \circ M_1$, one needs higher orders.

The solution is to track *truncated power series* (i.e. Taylor maps) that approximate the complete composition up to some defined higher orders. The truncation breaks the symplecticity of the one-turn map, but it can be computationally restored using the same methods used to construct symplectic integrators. The error of the deviation will be anyway of higher order than the order of the truncated power series [22].

The PTC library embedded into MAD-X is able to track any order of maps. The order is fixed by the users and depends on the non-linearities supposedly present in the lattice, the off-axis off-momentum initial conditions of the simulations, and the order of the analysis that follows, i.e. bottom right branch in Fig. 7. Because the time of the computation grows exponentially with the order of the maps, some tradeoff must be found to avoid lengthy tracking time, e.g. days or weeks for large lattice and $10^{\text{th}}$ order.

As a comparison, Track is a 0-order map symplectic tracking code (particles orbits) and Twiss is a $2^{\text{nd}}$-order map non-symplectic tracking code (orbit, $R$ and $T$), both being used with some success at CERN. Hence, symplectified high order maps are adapted to model accurately one-turn maps for further analysis and to some extend perform short-time tracking, while symplectic 0-order maps are adapted for long-time tracking. Both kinds of maps are handled indifferently by PTC on the same lattice model, making PTC the only valid and unified approach for complex lattices. Unfortunately, PTC is slow comparing to (Six)Track and Twiss, even when processing with same maps order, because of its very flexible splitting model as shown in [28]. As for most components of the application, some room exists for improvement.

### *Matching Methods*

Figure 8 shows the general principle of the matching module available in MAD-X. The optimization algorithm searches for a global minimum of the figure of merit $F$. If such a minimum is reached within the tolerances, the iterative process considers the problem as being solved and stops. Otherwise it updates its new position in the problem state space $C$ (i.e. user variables), and starts a new iteration. This optimization process follows a trajectory into the state space, which depends on the properties of the figure of merit $F$ and the nature of the problem to solve; different algorithms follow different trajectories. The figure of merit is a cost function part of the optimization algorithm, which combines the user-defined penalty functions to be minimized and the constraints on the solution domain, both defining the nature of the problem:

- local — global
- linear — quadratic — non-linear
- continuous — discrete
- smooth — irregular
- unconstrained — constrained
- constrained by equality — inequality

For a complete description of the various algorithms available for each combination of these criteria see [29]. In our case, we give a particular interest to the subclass of global, non-linear, continuous, smooth and constrained problems with both equalities and inequalities. This subclass is relevant to systems described by the Hamiltonians modeling lattice elements and to the usage made of the MATCH command. The algorithms actually available in MAD-X do not fall into this subcategory as we will see.

**Simplex** The Simplex minimization is an *active set method* (i.e. the algorithm class) well suited for linear programming (i.e. problem class), that performs well to solve linear problems with inequality constraints. Its strength lies



ISBN 978-3-95450-116-8

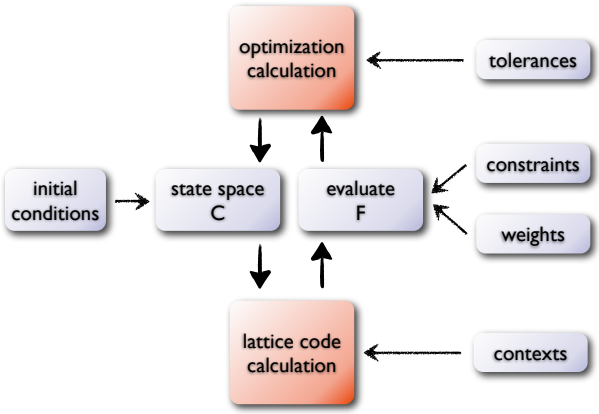


Figure 8: Optimization principle, the red boxes are the key components of the *matching* optimization process.

in its capacity to detect active constraints at the solution. In practice, it does not work well on the class of problems of interest to MAD-X users, because the features that make it so attractive for linear programming vanish in the presence of non-linearities.

**Gradient** The LMDIF and MIGRAD minimization are based on gradient descent method using different functions of merit. These are the simplest and fastest methods available in MAD-X, but they require more evaluations than other methods and get easily trapped by local minima, with no user-defined criteria to avoid the problem.

**Newton** The Jacobian minimization [30] uses the Newton algorithm and was added to deal with over-determined or under-determined problems that other methods were not addressing, using respectively QR or LQ decompositions. But this algorithm as well as the two previous ones are not supposed to deal with constraints and special tricks have been setup to bound the search in the state space and eliminate (freeze) non-active constraints. If the procedure fails, the user has to put extra weights to control the algorithms when it approaches the boundaries.

The implementation of this command was also the occasion to provide a more flexible way to define the constraints and act on the knobs with the help of the VARY command and some MAD macros. This is an important methodological aspect of the way the MATCH command might be used; some users like commands on which they get fine control. This approach works in practice because users' knowledge of beam dynamics generally let them start with good guesses and use the algorithm for fine-tuning of the expected results.

**New methods** On the long term, we propose to add two new methods better suited to manage large-scale non-linear problems (lattice knobs!) with large set of constraints all together. They are based on the *Augmented Lagrangian methods* to handle equality and non-equality constraints in a uniform well-defined framework. With these forefront methods in hands, MAD-X users should be able to work on complex large-scale problems efficiently.

The *active-set sequential quadratic programming* (SQP) method corresponds to the general purpose of the MATCH command where the problem is smooth and well defined, and where the inequality constraints specify soft boundaries of the solution domain, that is penalty functions can be evaluated beyond during the optimization steps. This algorithm should not trouble the users understanding since it is in the line of the Gradient and Newton methods, except that it works on the Hessian of the augmented Lagrangian quadratic form. The *active set* qualifier means that it possesses the same ability as the Simplex to select effectively the active constraints, a combinatorial problem, meaning that even large set of constraints can be handled efficiently.

The *interior point with log-barrier* method is the complementary approach of the active-set SQP essential to handle problems where the inequality constraints define strong barriers on the solution domain, i.e. user-defined penalty functions should not be evaluated beyond.

These two methods require evaluating the local Jacobian and Hessian matrices of the merit function accurately. The existing methods in MAD-X use inaccurate numerical differences to estimate the local derivatives. We propose to implement or reuse an existing *automatic differentiation* module for this purpose.

The computational power required by these new algorithms should remain equivalent to the already implemented ones, because the extra computations required by the second order derivatives should be compensated by the quadratic convergence of the algorithms. The development of this new component could be also the occasion to improve the computations performed by the differential algebra maps during high-order tracking.

### *Thick Elements* *(Ticket #113)*

This ticket requests to add thick *dipole* and *quadrupole* to the tracking module of MAD-X. This achievement would avoid extra work currently needed to generate a thin lens version of the LHC lattice and rematch the optics during squeeze simulations. While dipoles and quadrupoles can be defined as thick elements in the lattice sequence, they actually have to be sliced to be accepted by the Track module. The request boils down to re-introduce thick elements into the Track module as in MAD8, making the code a mixed drift-kick-drift and matrix-kick-matrix code. Some care will have to be taken with guards rejecting thick elements in few places.

The architecture of Track and Twiss modules requires that transfer matrices have to be entirely recomputed each time an element is processed, including all the costly transcendental functions, which slow down significantly the simulation as observed from simulation with SixTrack [31]. In order to avoid this degradation, another flag must be added to the Makethin module to select precisely the

ISBN 978-3-95450-116-8

dipoles and the quadrupoles that must remain thick.

### *Teapot Slicing (Ticket #154)*

This ticket proposes to apply iteratively the TEAPOT splitting scheme aforementioned until the number $N$ of user-defined slices for the element is reached. This request should be technically easy to address, even in the legacy code and should be implemented soon. Some care will have to be taken with the memory management of the elements. For example, this extension could split $N$ linearly leading to one of the following compositional schemes according to the remainder $\{0, 1, 2, 3\}$ of $n = \left[\frac{N}{4}\right]$ respectively:

$$T_4^{(n)}, \quad T_3 \circ T_4^{(n-1)} \circ T_2, \quad T_3 \circ T_4^{(n-1)} \circ T_3, \quad T_4^{(n)} \circ T_3 \quad (7)$$

where $T_k$ with $k = \{2, 3, 4\}$ refers to the already implemented TEAPOT splitting schemes.

As an alternative[9], it could possible to extend the TEAPOT splitting schemes beyond 4 slices to keep the accuracy of the underlying 2nd order symplectic integrator. For examples, first estimates on $T_8$ show that it should improve the error by approximately a factor 2.2 with respect to a $T_4^2$ scheme, in the line of $T_4$ that improves $T_2^2$ approximately by a factor 2.5. With the Yoshida methods [24] and 8 steps, it is possible to construct a 6th order symplectic integrator but containing negative drifts that MAD-X is not able to handle for the time being. An issue that might be useful to solve if we like to extend the range of application of MAD-X to small or special accelerator lattices.

### *Symplecticity Checks (Ticket #172)*

This ticket proposes to add symplecticity checks into MAD-X. In fact, these checks are already present in Twiss for the matrix $R$ and could be improved to include the tensor $T$. The same checks could be added to Track to evaluate the precision and the stability of the calculations in debug mode, because Track is already symplectic.

### *Lattice Transformation*

This ticket proposes to implement a lattice reflector to get rid of the problem of consistency between the beam direction flag `bv`, the beam 2, and the beam 4 inside MAD-X in order to simplify the LHC simulations. The details of the transformation are given in [32] and the implementation cost in the legacy code will be carefully studied to evaluate if it is not better to wait after the redesign of the core, which should be able to process complex 3D lattice topologies in line with PTC.

### *Other Plans*

**Benchmarks** MAD-X users observed unexpected differences or inconsistencies in their results from time to time. While the origins are sometime the misunderstanding of correct usage, often due to the abstruseness or the incompleteness of the documentation, some of these feedbacks are valid complaints. Therefore, further studies like in [25] should be performed on other single-element models as well as on small reference lattices in order to evaluate the impact of parameters changes. Special considerations should be addressed for:

- small $p_0c$,
- large $\Delta p/p$,
- large aperture $A$ versus integrated field strength $KL$,
- transverse offsets off-axis $(x_0, y_0)$ and off-momentum $(p_{x0}, p_{y0})$ vs small length $L$ and small radius $\rho$.

These points should quantify the validity of the approximations made in the models, like the small angle approximation, the error in series truncation and the radius of convergence of the series. An application like MAD-X should work for any value of the particle kinetic energy $p_0c$, i.e. $0 < \beta \leq 1$, and any combination of $(x_0, p_{x0}, y_0, p_{y0})$, $A$, $L$, $KL$, and proper ratio of transverse and longitudinal momentum.

This benchmark might be very useful to quantify the discrepancies between MAD-X and PTC when modeling small or special accelerators (e.g. Fixed-Field Alternating Gradient) studied at CERN where approximations are mostly noticeable [33]. Moreover, the conclusions could be used to draw some recommendations for the users and improve the application in that direction. On the long-term, the best strategy would be to merge these three components into a well-tested single one, most probably following the more flexible and accurate approach of PTC. Then the improvements would mainly consist to speed-up the PTC approach.

**Manuals and guides** The actual online documentation of MAD-X [8] has grown untidily with time, and the initial structure inherited from the MAD8 user's manual has not been preserved. The current table of content does not reflect the application logic, and users often prefer searching the information in the subject index first. The look and feel of the user's manual is under improvement but the restructuring of the document and the correction of the inconsistency is a long-term objective. In parallel, a new physics guide will be started on the model of the MAD8 physics guide, and extended to take into account the evolution of MAD-X. In particular, the Hamiltonians used in Track, Twiss and PTC for each element should be collected and carefully described with their respective assumptions and limitations. This task will go in concert with the benchmarks aforementioned to identify the approximation done by the models and the recommendations that should follow.

## APPLICATION IMPROVEMENT

### *Motivation*

In the section on project improvement, we have mentioned the untidy feeling perceived by the MAD-X users.

[9] Not part of the request ticket #154.



ISBN 978-3-95450-116-8

Table 1: Classification of MAD-X users complaints that turned out to be problematic for their studies, 77% of the tickets are related to the core of the application, 23% to the physics.

| Groups | # | Tickets no |
|---|---|---|
| interpreter | 14 | 56, 57, 64, 67, 73, 89, 109, 110, 124, 125, 132, 136, 147, 165, 171 |
| table, select | 10 | 48, 81, 85, 99, 122, 123, 130, 139, 141, 148 |
| sequence, use | 7 | 61, 74, 80, 89, 93, 120, 126 |
| plots | 12 | 42, 69, 71, 76, 85, 86, 102, 115, 116, 131, 150, 164 |
| memory leaks | 7 | 1, 3, 4, 92, 111, 151, 153 |
| MAD-X physics | 13 | 68, 77, 79, 83, 88, 106, 112, 117, 127, 135, 137, 138, 149 |
| PTC physics | 2 | 75, 118 |

Table 1 classifies the meaningful complaints of the users to spot the problematic parts of the application. The complaints relative to the core of the application (non-physics groups) represent 77% of the total, making this part of the application a priority for improvement, while the physics part (two last groups) represents only 23% of the total. In the following, we will address few of these points and make proposals to improve directly or indirectly the situation.

### *Data Management*

**Memory leaks** Memory leaks appear in many occasions in MAD-X; each time a table, macro, file inclusion (i.e. call file), command, class or element are *created*, some memory leaks happen. The three latter points are equivalent in practice because classes and elements are represented as commands internally[10]. The problem becomes rapidly critical for the jobs that intermix macros and loops to create long sequences.

**Data sharing** The design of the application was intended to run "one-shot" jobs, where allocated memory is never reclaimed except by the operating system when the process quits. The later introduction of macros has changed the big picture, and real cases leaking up to 7 MB/s have been observed (ticket #111). These leaks cannot be cured in the present design because there is no ownership policy, that is the number of references sharing the same object is unknown, making these bugs undecidable.

**Data lookup** The situation is complex because all searches are based on plain strings comparison, a notoriously slow process which is also required to retrieve elements and beam parameters at each tracking step in Track and Twiss. To improve the performance, the core cashes a lot of information through pointers and indexes into large data structures and global variables (i.e. side effects) that must be manually updated upon changes. Another strategy adopted was to enforce constraints when defining new elements or data structures, like constant offsets for specific fields within structures, or hardcoded indexes to access properties stored as key-value pairs (ticket #57). These two optimizations are very error prone and make the management of data structures intrusive and very complicated.

[10] An interesting design choice that will be preserved.

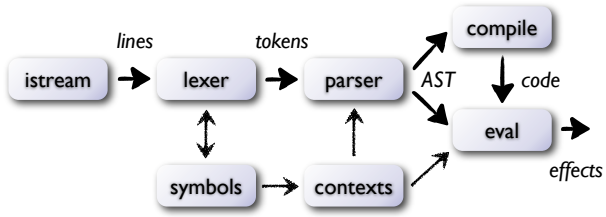


Figure 9: Components of the future interpreter, split into the analyzer (left to AST) and the evaluator (right to AST).

**Side effects** In many cases, the only available solution to recover from undetermined state of global variables and side-effects between modules interaction is to destroy and to recompute everything; a well known user trick used to reload sequence or call USE and TWISS commands for arbitrary reasons, hoping to restore a proper lattice (ticket #80).

### *Interpreter Design*

A deep analysis of the situation highlighted that most design problems were coming from the development of the interpreter itself. The interpreter is not following a well-defined syntax and grammar, and the implementation grew from timely requests with little cohesion, introducing inconsistent constructions in the language.

**No parser** The analyzer part of the interpreter is string-based, with no lexer, no parser and no abstract syntax tree (AST), which are essential components of any interpreter, i.e. Fig. 9. As a consequence, the interpreter is spread over the entire code and processes only strings. A strategy that conditioned all further developments down to the physics computations. The slow performance of plain strings comparison generated plenty design decisions that did not scaled with the size of the core, with a strong impact on the complexity, performance and support of the application.

**Ambiguous grammar** There are many examples, i.e. Tab. 1, which create grammar issues that remain undetected by the analyzer. The commutative positional arguments of the `TABLE` command are at the origin of some tricks in the code and painful bugs for users. The `column` argument in `SELECT` that collects an undelimited list of names and therefore needs to know the names of the formal parameters to detect the end of the list. The unquoted

ISBN 978-3-95450-116-8

strings converted to sequence of "tokens" but still interpreted as lowercase strings, making the scripts platform dependent (e.g. file names). The extra comma in a TWISS command that cancels silently its invocation, leaving the previous Twiss table in place as if it was called, i.e. thanks to global variables and side effects. The use of unrecognized operator (e.g. ** instead of ^) that makes expressions returning zero silently. The extra space before (or after) some operators (e.g. ->) that breaks silently the expression evaluation leaving unchanged the value in assignment. The missing statement delimiter ; silently ignored, leading to concatenation of statements and strange behaviors experienced by most users.

The list is long but each of these points (and more) were reported by users, after having wasted hours if not days to figure out the problem[11], with a real cost for their projects.

**Interpreter redesign** We propose to redesign the interpreter, which is the cornerstone of the creation and management of most data structures, following the standard structure of interpreters schematically shown in the Fig. 9. The components will be developed step by step and introduced into the legacy code for portability check, testing and debugging purpose. The parser will be a recursive descent parser for its simplicity and flexibility and because utterly performances are not needed at this level. Moreover, it will allow parsing context dependent grammar required for backward compatibility with the MAD language that we do not want to leave out. The code will be activated once the interpreter is ready for use, and all modules adapted to its standard interface. For further details about parsing technics and interpreter implementation, see [34, 35].

The language syntax will keep almost full backward compatibility because many users are familiar with it, and like its compactness and flexibility; except for the cases aforementioned that make the grammar ambiguous or context dependent.

### *Application Logic*

We mentioned that the design and the implementation of the interpreter has driven all MAD-X development during a decade. For the same reason, the development of a new well designed interpreter, i.e. Fig. 9, will improve all aspects of the application by introducing few concepts missing in MAD-X. We describe hereafter four concepts that we plan to introduce in the code and the implications for the application logic as shown in Fig. 10.

**Ownership policy** We propose to introduce intrusive manual reference counting with ownership semantic in all objects, i.e. data structures. This semantic has been defined and implemented in the Objective-C language and used with success by the software industry for two decades [36]. This will simplify the memory management drastically and should remove the undecidability of all pending memory leaks. This policy will be an integral part of the new interpreter, because all user objects declared in the scripts must be built by the parser and held by the nodes of the AST. Hence the new interpreter will be the first claimant for the ownership policy; variables, elements, sequences and tables being the next.

**Runtime polymorphism** A direct consequence of building an AST, is that runtime polymorphism must be supported. Indeed, the type of the nodes in an AST are known only at runtime and need polymorphic interfaces with late binding to be used or evaluated. This feature is already present in C++, easy to implement in C, and tedious to simulate in Fortran 95.

**Contexts and scopes** Contexts and scopes are important concepts of programming languages. A scope defines the validity and visibility of variables and identifiers, and is delimited by brackets pairs. Variables are bound to values within the scope where they are created and are destroyed on exit. A context includes the current scope and all enclosing scopes. Only variables that are part of the current context are visible, i.e. can be used. To create local variables bound to the current scope, we propose to introduce the keyword `var` and extend the semantic of `const` with the usual semantics borrowed from most programming languages.

**Notification policy** The last concept that we propose to introduce is the notification policy, which consists to notify registered objects about specific changes or actions through well-defined polymorphic interfaces. This will help to update the values of variables, expressions, elements, sequences, etc. consistently and only on-need. It will also help to implicitly manage the requirements of some computations, i.e. data and modules interdependencies.

**Polymorphic variables** With the introduction of runtime polymorphism, variables will be able to hold and deal with any kind of objects through properly defined polymorphic interfaces. Currently MAD-X is limited to deal with boolean values, integer numbers, floating point numbers, strings and expressions thereof. The evaluator (right in Fig. 9) will become polymorphic too, with the ability to evaluate any kind of objects and expressions.

**Polymorphic containers** By extension to variables, arrays and associative arrays will inherit for free of the same features and will be able to hold and deal with any kind of objects or group of objects. By composition, a variable will be able to store an array or table containing various kinds of elements, or an entire lattice definitions. This improvement will allow to get rid of all the global variables and most side effects present in MAD-X. Tables, sequences, files, commands, etc. will be stored in system-defined or user-defined variables, and these variables will

[11] Many thanks to all MAD-X users for their fruitful feedbacks.



ISBN 978-3-95450-116-8

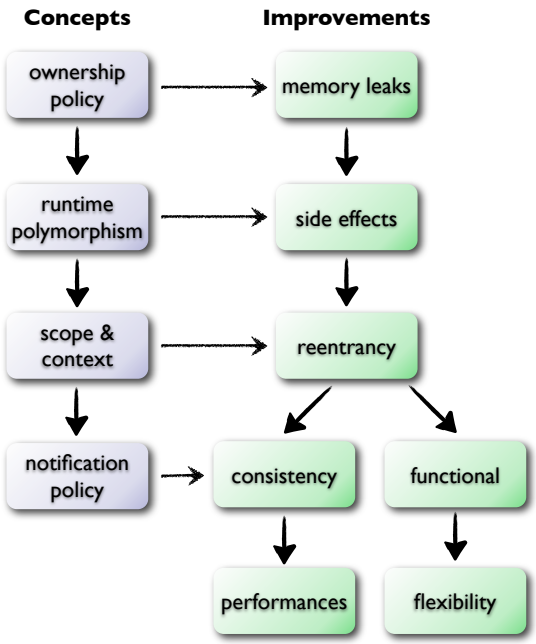


Figure 10: Schematic impact of concepts on the application logic and performance.

be passed as arguments to commands. This will restore the functional programming capability missing in the scripts.

The purpose of the simple `twiss` example hereafter is to show how hierarchical objects can be built and stored into variables and passed to the command, highlighting the future uniform data management:

```
file       = file="optics";
pattern    = pattern="^m.*";
column     = column={name,s,betx,bety};
sector_sel = sectormap=pattern;
twiss_sel  = select={pattern,column};
twiss (file, twiss_sel, sector_sel);
```

The second equal sign in the first five lines does not perform an assignment, but returns a key-value pair object, which in turn is assigned to a variable. The brackets are used to group expressions into a polymorphic array. The parenthesis of the `twiss` invocation means "evaluate in the current context augmented by these variables", i.e. the arguments. This syntax is equivalent to the first comma or space following the command name currently supported, but with a less ambiguous notation. Without going into the details, this approach standardizes the way variables, elements, sequences, commands, etc. are handled internally, with a deep simplification of the MAD-X core.

**Types extensions** Since the evaluator will handle polymorphic expressions everywhere[12], the language can be extended very easily to new types useful to the users. Needs have been identified for complex numbers, vectors, matrices and functions, and more may be added later painlessly.

[12]Currently expressions are only allowed in limited places.

But for the sack of simplicity, the scripting language should be kept as simple as possible, with a syntax yet far enough expressive for all needs of MAD-X users.

**Sequences** The following example is coming from a user request and shows an application of functional syntax to embedding sequences into other sequence:

```
seq := sequence (L=len) {
...
subseq1: seq1, at=at1;
subseq2: seq2, at=at2;
...
}
seq1 := sequence (L=len1) { ... }
seq2 := sequence (L=len2) { ... }
```

For backward compatibility of scripts, all current notations will be supported, but the new notation reflects again the uniform treatment of the information by the new interpreter.

The possibility to define subsequences of lattices stored in variables with notification of changes will allow the users to change the lattice definitions on-the-fly through the knobs of the optimization process, and avoid to rebuild the entire beam-line with costly macros at each iteration of the matching, i.e. Fig. 8 bottom red box.

**Matching reentrancy** A very positive consequence of the support of functional programming style is that it will allow reentrancy of commands. The reentrancy of the MATCH command is a strong user request to optimize large complex optics, e.g. interaction regions of the LHC, that need to be split into few steps to converge. This splitting method called *state space partitioning*, makes the optimization algorithms more efficient and stable.

## *Data Visualisation*

Data visualization is an important topic for MAD-X users. As for any numerical intensive application, visualizing data is important to quickly check the validity of the results. But this is also a very time consuming programming task that cannot be addressed directly within the MAD-X project due to the limited resource. Hence, the proposed strategy for the PLOT command is to delegate the task to third party portable application, like Gnuplot [37]. The interface already exists but is actually underexploited, that is only used for 2D phase-space tracking plots (i.e. trackfile), while other kinds of plots relies on old unmaintained codes written in Fortran 77.

With the notification policy aforementioned, it will be possible to update the plots on-the-fly upon changes, e.g. Twiss table content during matching. If important limitations not compatible with MAD-X objectives are encountered, the possibility to use a portable library will be studied in the last resort, but the amount of work to support such strategy is significantly larger.

ISBN 978-3-95450-116-8

### *Speed Performance*

**Tracking** Software speed is a strong point for the community of users, but the complexity of MAD-X forbids serious optimization. As a consequence many parts of the code waste CPU cycles. The old programming style based on cascades of `if-then-else` (up to few tens per function) breaks frequently the instruction prefetching of pipelines, which is a performance killer on modern CPU. With the new core design, the transfer maps will be stored in the beam-lines nodes (i.e. *fibers* in PTC) with the element attributes. Hence, Twiss and Track should experience a significant speed-up by a factor of about 4 and 4×#cpu-core respectively. As for reentrancy, the absence of global variables and side effects is mandatory to go with parallel computation on multi-core CPUs. The replacement of all identifier strings by symbols within the lexer, i.e. Fig. 9, should also have a significant speed-up on modules that relies on lattice parameters.

**Interpreter** One of the main drawbacks of introducing runtime polymorphism in high performance computing is the substantial loss of performance. Few programming techniques have been tested in C and C++ to overcome the limitations induced by runtime polymorphism. First attempts to test the performance of the main loop of the evaluator, i.e. Fig. 9, have shown that the choice of the technologies (not discussed in this paper) allows to implement an extremely fast polymorphic evaluator about $\times 8 - \times 10$ faster than the mainstream interpreters, e.g. Python. For example the simple loop `while(1) x=x+1;` is evaluated 1.67 $10^8$ times by a single core of an Intel i5 2.3 GHz CPU, where `x` is dynamically typed as an integer and all operations are performed through polymorphic interfaces. Hence this test is a direct measure of the performance of the evaluator without cheating with built-in calls to C libraries. If these performances are confirmed on the large scale, this evaluator will offer tremendous opportunities to the MAD-X users and should deeply change the way elements and lattices might be parameterized and optimized.

**Math kernel** Another important aspect of the future performance of MAD-X is the efficiency the matrix-matrix and polynomial-polynomial multiplications. The former is involved in many places, from geometrical transformations to orbit and linear-term tracking (i.e. $R$ matrix and $T$ tensor). The latter is involved in truncated power series algebra (i.e. high-order maps tracking) and differential algebra. With the new design, it will be possible to build highly specialized modules dealing with specific tasks and tuned for specific platforms and technologies; with a particular interest for the SSE2 and AVX Intel technologies present on all platforms supported by MAD-X. A first attempt as been performed to improve the matrix-matrix multiplication using SSE2 Intrinsics and exploit the topology of the data structures to vectorize the operations.

The Fig. 11 shows timing comparison in seconds between three implementations evaluating $10^8$ the matrix expression $M = M_1 M_2 + M_3 M_4$, for square matrix with dynamic sizes $N = 1..9$. The results for static sizes (i.e. known at compile time) gives another $\times 10$ improvement versus dynamics sizes (i.e. known at runtime) for all implementations, leaving unchanged the performance ratio and the outcome of the study.

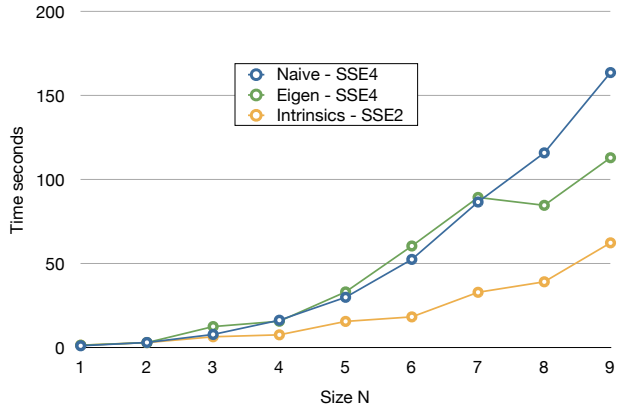


Figure 11: Timings in seconds to compute $10^8$ times the matrix expression $M = M_1 M_2 + M_3 M_4$ for small dynamic matrix sizes $N$. For static matrix sizes $N$, Eigen and MAD-X equals with a $\times 10$ speed-up versus dynamic sizes.

The *naive* curve shows what C/C++ and Fortran compilers can achieve on the naive handwritten algorithm, with maximum optimizations enabled including SSE vectorization. This is the current performance status of MAD-X. The C++ Eigen library [38], one of the fastest available library with Armadillo [39], uses advanced programming technics like meta-template expressions to optimize the computations of expressions and avoid temporaries. The main goal of these C++ libraries is to provide Matlab-like syntax to programmers using operators overloading, while performing as well as plain handwritten C and Fortran code as shown by the *Eigen* curve. Finally the *Intrinsics* curve shows that handwritten code exploiting data structures topology for vectorization — an optimization beyond the capability of compilers — can perform at least $\times 2$ better than other approaches. This kind of optimization requires moderate efforts because only few functions need to be optimized, with a significant performance impact on matrix and polynomial operations at the heart of many computations in MAD-X. Estimating the performance improvement for complex computations like Track, Twiss or Match is premature and further investigations are needed.

## CONCLUSIONS AND OUTLOOK

MAD-X is undergoing its first renovation process since years, a process that is meant to improve the code quality and flexibility and to restore the communication within the MAD-X community. Its maintenance and portability have been simplified and strengthened, and a bug tracking system has been setup. Some efforts are also devoted to



ISBN 978-3-95450-116-8

implement new elements in the legacy code, like the RF-Multipole necessary to evaluate the detrimental impact of the crab-cavities on the luminosity in the LHC upgrade scenarios. By the end of 2012, MAD-X should be ready to sustain new large-scale developments for the benefit of its users.

In parallel to the achievements of the past year, a structured analysis of the situation and the collected feedback from the community allowed to draw new orientations for the future development. In the present paper, we have made few proposals to deeply improve the application logic and the physics results and extend MAD-X to a wider range of uses; taking into account the needs of the future accelerator projects at CERN and the limited resource available to achieve the work over the next period of 2-3 years.

## ACKNOWLEDGMENTS

We would like to thank F. Schmidt for his invaluable contribution to the project as the MAD-X custodian over the decade 2002–2011, R. De Maria for the many fruitful discussions on MAD-X, A. Latina for the implementation of the RF-Multipole, P. Skowronski for his support on PTC, and M. Giovannozzi and O. Bruning for useful feedbacks on this paper.

## REFERENCES

[1] H. Grote and F. Schmidt, *"MAD-X : An Upgrade from MAD8"*, PAC'03, Portland, USA, May 2003.

[2] F. Schmidt, *"MAD-X a worthy successor for MAD8?"*, Nucl. Instrum. Methods Phys. Res., A 558, (1) 2006.

[3] E. Forest, F. Schmidt and E. Mcintosh, *"Introduction to the Polymorphic Tracking Code"*, KEK Report, 2002.

[4] W. Decking, *"Single Particle Beam Dynamics Codes*, Care HHH Workshop, CERN, November 2004.

[5] D. Einfeld, *"Optic Codes"*, OMCM Workshop, CERN, June 2011.

[6] K. Fuchsberger, X. Buffat, Y.I. Levinsen and G.J. Muller *"Status of JMAD, The Java-API for MAD-X"*, IPAC'11, San Sebastian, Spain, September 2011.

[7] G.J. Muller, *et al.*, *"Toolchain for Online Modeling of the LHC"*, ICALEPCS'11, Grenoble, 2011.

[8] MAD-X website `http://cern.ch/madx`

[9] A.H. Watson Thomas and J. McCabe, *Structured Testing: A Testing Methodology Using the Cyclomatic Complexity Metric*, NIST Special Publication 500-235, 1996.

[10] J. van Gurp, and J. Bosch, *Design, Implementation and Evolution of Object-Oriented Frameworks: Concepts & Guidelines*, Software Practice & Experience, John Wiley & Sons, March 2001.

[11] MAD-X Trac `https://svnweb.cern.ch/trac/madx`

[12] E. Forest, *et al.*, *"Synopsis of the PTC and ORBIT Integration"*, KEK Report, 2008.

[13] L. Deniau, *"MAD-X meeting"*, presentation available on [8], CERN, June 2012.

[14] L. Deniau, and A. Latina, *"Evoluation of MAD-X in the framework of LHC upgrade studies"*, IPAC'12, New Orleans, May 2012 (MOPPC074).

[15] A. Grudiev, *et al.*, *"Study of Multipolar RF Kicks from the Main Deflecting Mode in Compact Crab Cavities for LHC"*, IPAC'12, New Orleans, May 2012 (TUPPR027).

[16] S. Russenschuck, *"Field Computation for Accelerator Magnets"*, Wiley-VCH Verlag, Weinheim, 2010.

[17] R. De Maria, *"Symplectic integrator for a relativistic particle in a RF cavity"*, BE/ABP–LCU meeting, `http://cern.ch/ab-dep-abp/LCU/LCU_meetings/Minutes.html`, April 2011.

[18] F. Schmidt, *"SixTrack version 1.2"*, CERN-SL-94-56, 1994.

[19] F. Antoniou and F. Zimmermann, *"Revision of IBS with Non-Ultrarelativistic Corrections and Vertical Dispersion for MAD-X"*, Report CERN-ATS-2012-066, March 2012.

[20] A. Valishev *et al.*, *"Ring for Test of Nonlinear Integrable Optics"*, IPAC'12, New Orleans, May 2012 (WEP070).

[21] R. Talman, *"Representation of Thick Quadrupoles by Thin Lenses"*, technical report SSC-N-33, August 1985.

[22] E. Forest, *"Geometric Integration for Particle Accelerators"*, J. Phys. A: Math. Gen. **39** (2006) 5321-5377.

[23] L.M. Healy, *"Lie Algebraic Methods for Treating Lattice Parameter Errors in Particle Accelerators"*, PhD thesis, University of Maryland, 1986.

[24] H. Yoshida, *"Construction of higher order symplectic integrators"*, Physics Letters A, vol. 150, November 1990.

[25] O. Berrig, *"Comparison of transfer maps of PTC and MAD-X for the dipoles magnets: SBENDS and RBENDS"*, `http://cern.ch/cern-accelerators-optics`, CERN, 2008.

[26] V.I. Arnold, *"Mathematical Methods of Classical Mechanics"*, Springer, 2nd, 2006.

[27] E. Hairer, C. Lubich, and G. Wanner, *"Geometric Numerical Integration: Structure Preserving Algorithms for Ordinary Differential Equations"*, Springer, 2nd edition, 2006.

[28] C. Hernalsteens, *"Modeling the PS for PTC simulations"*, LIS meeting, `http://cern.ch/ab-dep-abp/LIS/Minutes/Minutes.html`, CERN, March 2012.

[29] J. Nocedal, and S.J. Wright, *"Numerical Optimization"*, Springer, 2nd edition, 2006.

[30] R. De Maria, *et al.*, *"Advances in Matching with MAD-X"*, ICAP'06, Chamonix, 2006 (WEPPP14).

[31] H. Burkhardt, M. Giovannozzi, and T. Risselada, *"Tracking LHC Models with Thick Lens Quadrupoles: Results and Comparisons with the Standard Thin Lens Tracking"*, IPAC'12, New Orleans, May 2012 (TUPPC079).

[32] S. Fartoukh, *"Proposals for changes in the MAD-X code and in the LHC sequence"*, unpublished report, CERN, December 2008.

[33] E. Keil, *"Emma in MAD-X and Comparison with Other Programs"*, CERN ATS Note 2010-044, 2010.

[34] D. Grune, and C.J.H. Jacobs, *"Parsing Technics, a Pratical Guide"*, Springer, 2008.

ISBN 978-3-95450-116-8

[35] A.V. Aho, M.S. Lam, R. Sethi, and J.D. Ullman, *"Compilers: Principles, Techniques, and Tools"*, Prentice Hall, 2nd edition, 2006.

[36] *"Advanced Memory Management Programming Guide"*, Apple Inc., 2011.

[37] P.K. Janert, *"Gnuplot in Action, Understanding Data with Graphs"*, Manning Publications, 2009.

[38] G. Guennebaud, B. Jacob, *et al.*, *"The Eigen 3 C++ Library"*, `http://eigen.tuxfamily.org`, 2010.

[39] C. Sanderson, *et al.*, *"Armadillo: An Open Source C++ Linear Algebra Library for Fast Prototyping and Computationally Intensive Experiments."*, Technical Report, NICTA, 2010.



ISBN 978-3-95450-116-8