MAD-NG experience and outlook for LHC and HL-LHC

R. De Maria

Thanks to L. Deniau for supporting material and quick feedback

LHC and HL-LHC optics use cases

1. Read existing sequence files from layout DB or optics repository and running existing MAD-X scripts.
2. Compute orbit and Twiss parameters (and derivatives), Survey of an LHC or HL-LHC sequence.
3. Optics matching (compute boundary conditions, objectives and optimize variables).
4. Optics transitions (compute smooth optics matching solutions).
5. Read aperture information and aperture calculation (n1).
6. Introduce field imperfections from wise tables, misalignment tables.
7. Prepare lattice for long term SixTrack simulations.
8. Short term tracking for tune footprint, injection oscillations,…
9. Compute non-linear quantities (detuning terms, HDT, with parameters) and derived quantities (emittance, IBS).
10. Integration in Python workflows.

MAD-X experience on LHC, HL-LHC

Entire optics model written in MAD-X language and large set of MAD-X script and tools.

- Reasonably fast for all native computations.
- Speed improvements in Twiss and aperture would reduce the time needed for optics design and aperture checks.
- Complex scripts are difficult to write and maintain, many workarounds needed. Complex sequence manipulation are not possible within the language and/or severely slow (e.g. Beam 4 sequence generator, automatic slice misalignments based on mech_sep).
- Cpymad (Python wrapper to MAD-X) is today essential to mitigate the shortcoming of the scripting environment increasing robustness and simplifying scripting. Performance might be limiting in certain contexts (e.g. manipulating entire sliced sequence).
- MADX-PTC integration provide a simple path to access the normal form analysis package, still flexibility and speed are limiting (e.g. it is faster to get 3rd- order chromaticity by finite difference rather than using PTC normal form analysis).
- Modelling overlapping and/or misaligned beam elements and field requires and external tools (being improved).

MAD-NG structure from a user perspective

(Based on 0.9.1 and 0.9.3 experience)

MAD-NG can be discussed at several levels (or layers):

1. Modified and enhanced LuaJIT platform with new syntax, GTPSA, vector, matrix objects with linear algebra and FFT, object model(s).
2. Survey and Tracking maps.
3. Linear optics and derived quantities calculations.
4. Sequence and beam-line manager, helper commands to orchestrate calculations (twiss, matching, matching, survey).
5. MAD-X compatibility layer.

1. LuaJIT platform for the LHC use case

LuaJIT offers a simple and fast language with excellent interface with compiled C codes. MAD-NG extensions made more suitable for numerical codes.

Obviously deviates completely from MAD-X language:

- needs complete write of the existing script and tools,
- therefore any transition needs to be smooth and accounted in man-power cost.

Interoperability with Python is today essential to leverage use the large associated scientific ecosystem and interact with LHC control systems.

- MAD-NG is missing this feature. It is clear a blocking point for MAD-NG adoption.
- Lupa library provides Python interoperability for standard LuaJIT, but not clear if it can be a starting point for MAD-NG and how much needs to be extended.

LuaJIT generates optimized assembly code for different architecture. The development and support network is small compared mainstream languages. The future of LuaJIT and MAD-NG has few single points of failure (Industry support, LuaJIT lead developer, MAD-NG lead developer).

- Risk of potential issues in the long term can be mitigated by writing in parallel to the code, a physics manual that would allow (among other advantages) to reimplement the same functionality if an alternative framework would emerge.

2. Survey and Tracking maps.

MAD-NG uses the PTC(time=true with optional total time) formalism and implements a subset set of maps (the most accurate).

A subset of PTC maps has been implemented (the most accurate maps):

- The choice excludes the maps that gives correct linear optics in one integration step, which introduces sides effects that impacts some optics design workflow:
    - inserting a marker in an element alters the linear optics (precise matching cannot be enforced);
    - the convergence on linear optics depends on the integration steps.
- Less common maps not implemented (wire, e-lens,…).

One MAD-NG map (non-linear kick with curvature) modified from PTC is not symplectic. Fix in progress.

This suggests:

- MAD-NG maps needs to be completed.
- Maps need to be tested thoroughly (benchmark not sufficient) by proving that they are all symplectic in their validity range. Tests have been added.
- Alternative integration schemes to be proven they converge to the same solutions.

3. Linear optics and derived quantities calculations.

- MAD-NG can compute arbitrary truncated maps with parameters, however it implements only linear normalization library (like MAD-X, contrary to MADX-PTC).
- The methods implemented are sufficient for linear optics calculations and matching.
- The truncated map are not useful until a normalization routine is available.

It is important to implement the parametric normal form analysis to fully exploit the capability of the GTPSA.

4. Sequence and beam-line manager, helper commands to orchestrate calculations, matching.

- The sequence and beam-line element manager are more flexible than MAD-X.
- Twiss, track, survey commands are very flexible. I expect fine tuning of defaults after broader usage. Speed is not far from MAD-X, with potential improvements.
- Aperture module (IBS…) not available, it is a blocking point currently.
- It is possible to insert kicks and markers inside elements and displace elements arbitrarily.
- It is not possible natively to describe inner element using absolute positions and overlapping elements (additional user defined structures will be needed).
- It is not possible to defined overlapping thick elements (additional user defined structures will be needed).

To exceed the modelling capabilities of MAD-X one would need to allow:

- defining inner and overlapping elements such as: BRAN inside TAXN, wire inside collimators, beam-beam kicks inside quadrupoles, experimental solenoid overlapping with quadrupoles, …

5. MAD-X compatibility layer

MAD-NG needs to convert MAD-X sequences and optics data into special data structure. MAD-NG does not execute MAD-X scripts on these data structures.

- A massive quantity of MAD-X code cannot then be reused.
- Effectively until MAD-NG does not offer substantially improved capabilities, the scripts will not be ported.
- Initially MAD-NG can be used as post processor after saving an optics model build by MAD-X.

Given the extensibility of MAD-NG:

- it can be envisaged to wrap MAD-X capabilities, slowly replacing the MAD-X scripting language. This will allow a smoother transition from MAD-X, but will not offer substantial benefit compared to cpymad;
- it is essential that MAD-NG offers substantial improvements with respect to MAD-NG to solve the MAD-X compatibility issues by a slow and user driven migration of the existing tools.

Conclusion

Today MAD-X, PTC, CPYMAD allows to do everything is needed for LHC, HL-LHC, except:

- automatic splitting to insert thin element in thick element (being discussed);
- flexible selection of orders in parametric normal form;
- scripting numerical performance and robustness (MAD-X crashes still too often, cpymad mitigates this).

MAD-NG leverages an unconventional platform to allow:

- more robust, concise, faster scripts;
- simplified and more accessible usage of PTC numerical methods for non-linear optics.

MAD-NG has to the potential of fulfilling the LHC, HL-LHC use-case provided:

- verification and extension of maps and Hamiltonian splitting options (benchmarks with existing codes not sufficient);
- implementation of fully parametric nonlinear normal forms;
- integration in Python workflows (not easy although lupa package successful for LuaJIT);
- seamless interoperability with existing MAD-X scripts (FFI approach like cpymad);
- community effort to implement a transition from MAD-X.

- The risk of the LuaJIT platform can mitigated by:
    - providing an extensive physis documentation (such as MAD-8 physics guide) that allows to migrate numerical methods to alternative platforms.

Back-up

Example

```
local beam, damap, matrix in MAD
local fnil in MAD.gfunc
local curex_kick in MAD.dynmap
local pbm = beam {
          particle = 'proton',
          energy = 10 }

local map = damap {xy=2}
map.x:set0(1)
map.y:set0(3)

local mflw = {
  map,
  el = 0,   eh = 10,
  tdir = 1,   nmul = 3,
  knl = {0,0,3},   ksl = {0,0,0},
  beam = pbm,
  npar = 1,
  atdebug = fnil,   mdump = fnil,
}
```

```
curex_kick (nil, mflw, 1, true)
map.px:print('PX')

local mat = map:get1()
mat:print("JAC")

local w, vl, vr, info = mat:eigen()
w:print("eigenval")

local mat6 = matrix(6)
local err, _ = mat:symperr(mat6)
print("symperr=", err)

OUTPUT
symperr=     254.55844122716
symperrmat[6x6] =
 0  0 -180  0  0  0
 0  0  0  0  0  0
 180  0  0  0  0  0
 0  0  0  0  0  0
 0  0  0  0  0  0
 0  0  0  0  0  0
```

```
function M.curex_kick (elm, m, lw, no_k0l)
  m.atdebug(elm, m, lw, 'curex_kick:0', no_k0l)

  local el, eh, tdir, nmul, knl, ksl, beam in m
  local bdir = tdir*beam.charge
  local dby  = no_k0l == true and knl[1] or
          eh*el*bdir-knl[1]

  for i=1,m.npar do
    local x, px, y, py, beam in m[i]
    local  bdir = beam and
           tdir*beam.charge or bdir
    local   dby = beam and not no_k0l and
           eh*el*bdir-knl[1] or dby
    local bx,by = bxby(nmul, knl, ksl, x, y)
    local     r = 1+eh*x

    m[i].px = px - (lw*bdir)*(by-dby)*r
    m[i].py = py + (lw*bdir)* bx     *r
  end

  m.atdebug(elm, m, lw, 'curex_kick:1', no_k0l)
end
```