Machine Learning for Non Linear Corrections in the LHC

Alejandro Börjesson Carazo

21/08/2023

Summary

1. **Introduction**
    1. Theoretical Background
    2. Methods

2. **Results**
    1. MADX-PTC vs MADNG RDT tracking
    2. Example of simple bayesian optimization for the LHC

3. **Conclusion and future improvements**
4. **Backup slides**

Introduction

Exploring possible ways ML can help RDT Correction

Nonlinear corrections in IRs currently use a lot of different methods, explore ways to correct all RDTs at once with ML

- Bayesian Optimization
- Supervised learning

ML Is computationally intensive, needing hundreds of thousands of data points, NL simulations usually are too slow

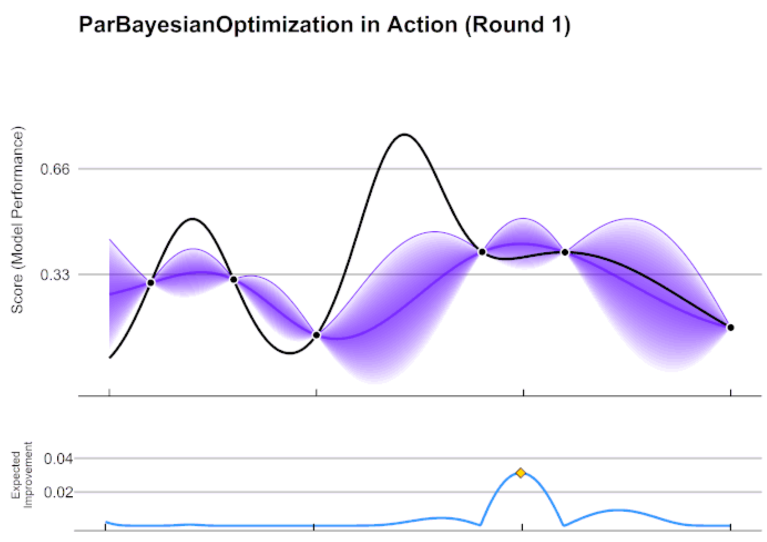
Using MADNG to test possible uses of faster computation

- Model creation, error generation
- Tracking RDTs

Theoretical Background: Bayesian Optimization

Bayesian optimization is a sequential design strategy for global optimization of black-box functions with as few iterations as possible

- For cheap functions to compute, a grid search approach might be possible
- Bayesian optimization uses Bayes theorem to iteratively update our knowledge of the **objective function**
- Approximation of the **ground truth** function with a **surrogate function**, usually Gaussian process
- Finding next sample point with an **acquisition function** for example expected improvement function, this function **quantifies how good each point is as a guess**
- Exploration Exploitation hyperparameters help us tune the algorithm!

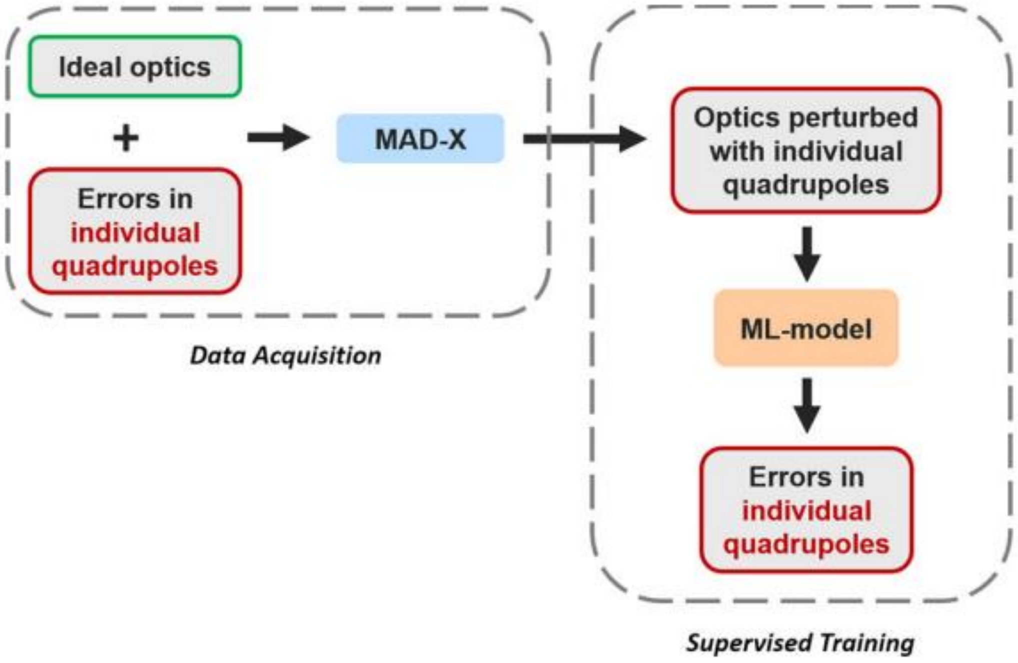


[1] *Fig 1. Bayesian optimization*

Theoretical Background: Supervised Learning

Supervised learning is a type of ML that uses labelled data to improve

- As shown in Elenas paper and in my previous presentation linear errors can be predicted using ML, and linear models perform best
- The idea is to apply the same method to predict non linear errors using non linear optic parameters, RDTs
- There are multiple models that allow for non linear modelling such as **neural networks**
- **In progress**



[2] Fig 2. Data pipeline for linear errors

Methods: Simulation requirements for ML

In order to create a useful ML model, good data is the most important requirement

- Meaningful observables, in this case **RDTs**
- High volumes of data! => **Fast simulations**
- Realistic and unbiased

Tracking + OMC analysis: Too slow

MADX-PTC: Too slow

MADNG: Fast!

Methods: MADX-PTC Error Simulation

Generate random errors according to WISE table distribution variances

- EFCOMP function to assign the errors in MADX

$$\Delta K_i = \mathtt{DKNR(i)} \cdot (k_{ref}) \cdot R^{n-j} \cdot \frac{j!}{n!}$$

- Assign relative errors **with respect to the quadrupolar strength** of the triplet magnets
- So far only testing with normal sextupolar errors, planning on simulating octupolar order errors and skew errors too
- Typo in MADX manual $j = i$

```
EFCOMP, RADIUS=0.017, ORDER=1, DKNR={{0,0,b3,0}};
```

Methods: MADNG Error Simulation

Assigning the same errors in MADNG 0.9.7-pre

- Elements have a dknl and dksl attribute
- Applying the same formula as in MADX documentation and multiplying by element length the same KNL errors are obtained
- PYMADNG Was also used since it provides an easy to use python API
- Using the “trkrdt” method in MADNG, much faster!

```
! Making absolute errors
local k_ref = element.k1
local k2l_err = 2*k2_err*k_ref*element.l/0.017

element.dknl={0, 0, k2l_err, 0}
```

Results: MADNG vs PTC for RDT tracking

Calculating RDTs for all points in the LHC is needed:

- MADNG execution time: **20.00 [s]**
- MADX-PTC execution time: 1630.12 [s] = **27.17 [min]**

MADNG Opens new possibilities to use computationally expensive methods such as ML for non linear optics

For testing purposes a cost function is defined as the RMS of the deviation from nominal $\Delta|f_{3000}|$ for all LHC

- $\Delta\left|f_{3000}^{MADNG}\right| = \dfrac{\left|f_{3000}^{MADNG}\right|_{Err} - \left|f_{3000}^{MADNG}\right|_{Nom}}{\left|f_{3000}^{MADNG}\right|_{Nom}}$

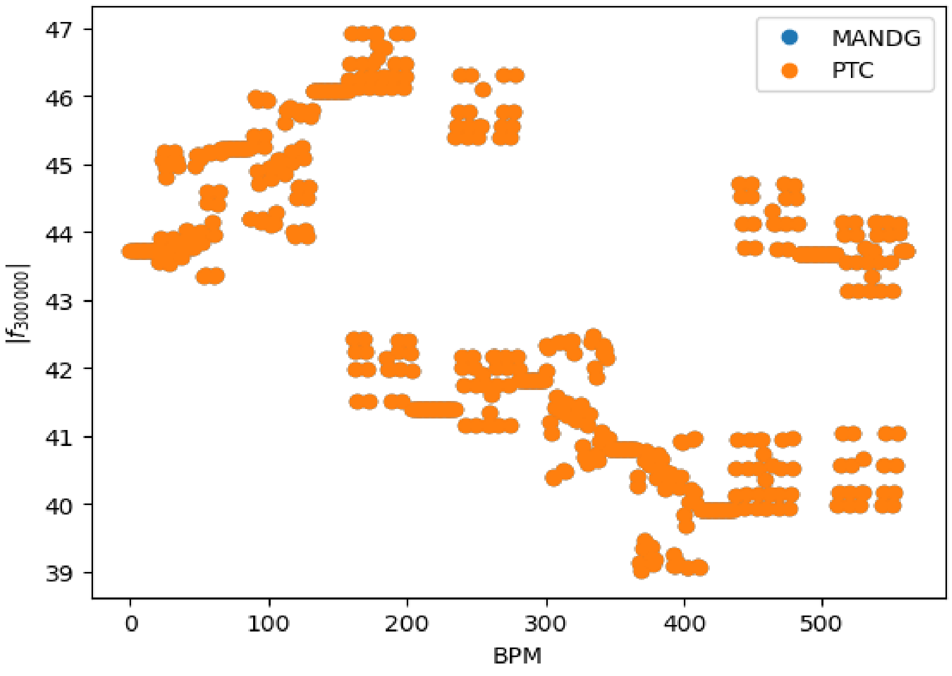


Fig 3. Nominal RDTs MADNG vs PTC

Results: MADNG vs PTC for RDT tracking

	RMS $\Delta\left\|f_{3000}^{MADNG}\right\|$	RMS $\Delta\left\|f_{3000}^{PTC}\right\|$	RMS $\left\|f_{3000}^{MADNG}\right\|$	RMS $\left\|f_{3000}^{PTC}\right\|$
Nominal	0	0	42.87036738	42.87037055
$b_3 = 10^{-3}$ in MQXA.3L2	1.4170 %	1.4197 %	43.46449129	43.46564095
WISE Errors in triplets	0.469297 %	0.469374 %	43.07155673	43.07159292

Tab 1. Example cost function comparison

Results: Simple Bayesian Optimization for sextupolar errors in the LHC

Trivial unrealistic example

Trying to correct MQXA.3L2 triplet using MCSX.3L2

Planning on optimizing multiple RDTs at once

Convergence in 6 iterations

RMS $\Delta\left|f_{3000}^{MADNG}\right|$=1.417 %

RMS $\Delta\left|f_{3000}^{MADNG}\right|$=0.147% kcsx3_l2= -0.00979

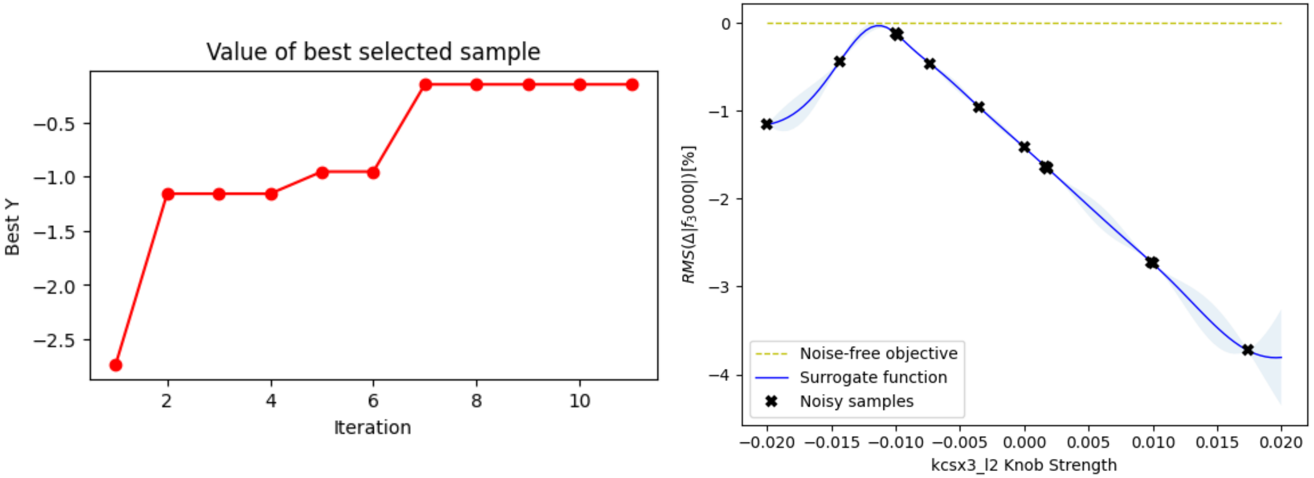


Fig 4. Bayesian optimization results for ten iterations

Conclusions and Future Improvements

- **Experience as a new user in MADNG vs MADX:**
  - Since MADNG is still in development it has a steep learning curve as of today
  - However since MADNG is based in LuaJIT this allows for more flexibility than MADX
  - Using MADX-PTC a ML application for RDT optimization is not feasible, MADNG proves to be a very powerful tool, and much faster
- **ML For RDT optimization:**
  - Further testing must be done to see how useful Bayesian optimization might be, more error types, more correctors…
  - After finally setting up the error generation section the ML part can be explored more in depth, using BO or more classical ML
- **ML For tune signal denoising (autoencoders):**
  - In progress..

References

- [1] *Fig 1. Bayesian optimization.* ***By AnotherSamWilson - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=84842869***
- [2] *Fig 1. Data pipeline. "Supervised learning-based reconstruction of magnet errors in circular accelerators" by E. Fol, 2021, https://doi.org/10.1140/epjp/s13360-021-01348-5*

Backup slides: Bayesian Optimization

The Bayesian optimization procedure is as follows. For t=1,2,… repeat:

1. Find the next sampling point x_t by optimizing the acquisition function over the Gaussian Process $x_t = argmax_x \, u(x|D_{1:t-1})$
2. Obtain a possibly noisy sample $y_t = f(x_t) + \epsilon$ from the objective function f
3. Add the sample to previous samples $D_{1:t} = D_{1:t-1}, (x_t, y_t)$ and update the surrogate function

$$\mathrm{EI}(\mathbf{x}) = \begin{cases} (\mu(\mathbf{x}) - f(\mathbf{x}^+) - \xi)\Phi(Z) + \sigma(\mathbf{x})\phi(Z) & \text{if } \sigma(\mathbf{x}) > 0 \\ 0 & \text{if } \sigma(\mathbf{x}) = 0 \end{cases}$$

$$Z = \begin{cases} \frac{\mu(\mathbf{x}) - f(\mathbf{x}^+) - \xi}{\sigma(\mathbf{x})} & \text{if } \sigma(\mathbf{x}) > 0 \\ 0 & \text{if } \sigma(\mathbf{x}) = 0 \end{cases}$$

$x^+: Best\ point \quad \phi: PDF \quad \Phi: CDF \quad \xi: Exploration\ Explotation\ hyperparameter$

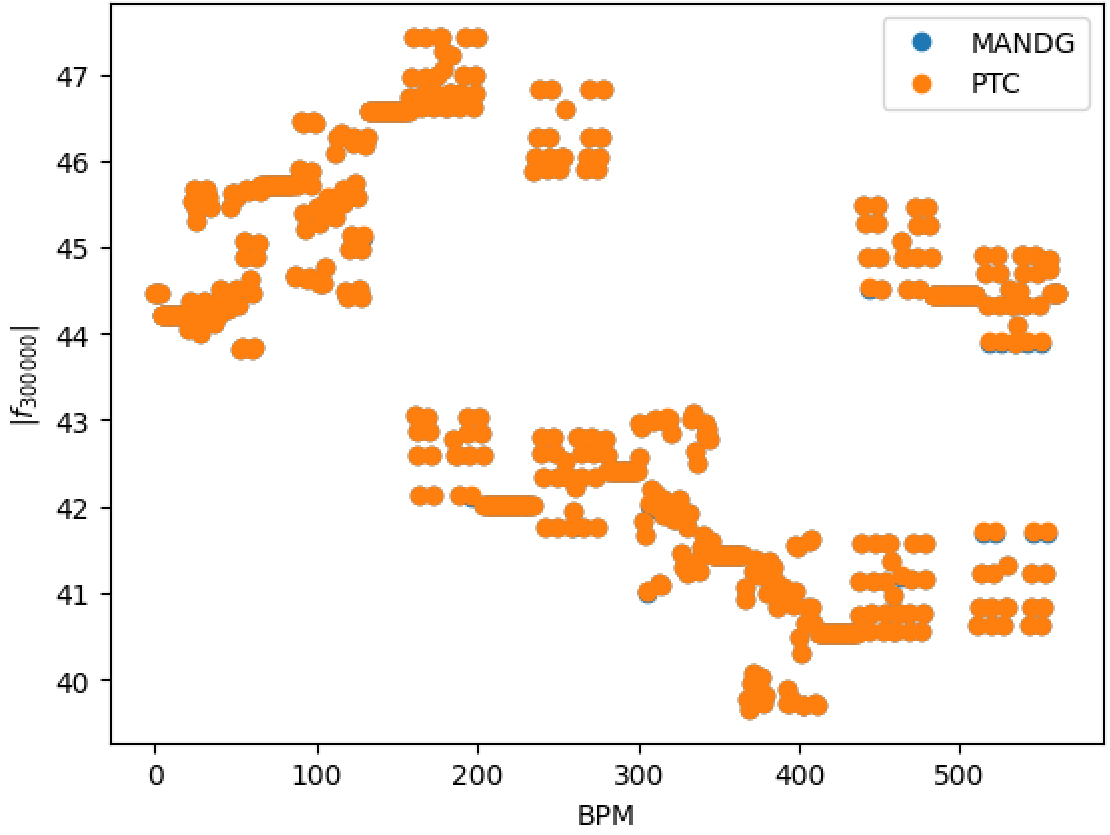
Backup slides

```
elseif mth == "trkrdt" then  -- 2nd method
  io.write("** Tracking ", obs_points[1], "\n") !obs_points[1] must be a marker
  lhc:cycle(obs_points[1])

  local _, mflw = track{sequence=lhc, method=4, save=false, X0=X0}
  local nf1 = normal(mflw[1]):analyse()

  io.write("** Tracking RDT\n")
  local X1 = nf1.a:real():set0(nf1.x0) ; X1.status = 'Aset'
  local mtbl, mflw = track{sequence=lhc, method=4, savemap=true, X0=X1,
                          range=obs_points[1].."/"..obs_points[#obs_points]}
  for i,obs_point in ipairs(obs_points) do
    nf[i] = nf1:analyse('gnf', mtbl[obs_point].__map)
    !nf[i].a:write("A_"..ip.."_trkrdt")
  end
```

```
ptc_create_universe;
  ptc_create_layout, model=2, method=4, nst=1, exact=true, time=true;
  ptc_setswitch, madprint=true;
  ptc_twiss, normal=true, trackrdts=true, no=4, icase=56;
ptc_end;
```

- trkrdt Method in MADNG 0.9.7-pre is a much faster method than cycling

Backup slides



CERN