

MAPCLASS: a code for optimization

Edu Marin¹
emarinla@cern.ch

¹CERN, Geneva, Switzerland

August 17th, 2017
ABP-CWG meeting #19

Acknowledgements: **Rogelio Tomás**

Outline

- 1 **Code Description**
- 2 **Code Implementation**
 - Python Version
 - C++ Version
- 3 **Example**

Description

- MapClass (2006) & MapClass2 (2012) are two codes [1] written in Python and C++
- Used to optimize the non-linear aberrations in beam lines using 1st and 2rd order moments at a given location as a figure of merit

$$x_f = \sum_{jklmnp} X_{jklmnp} x_0^j p_{x_0}^k y_0^l p_{y_0}^m z^n \delta_0^p \quad (1)$$

$$\langle x_f^2 \rangle = \int x_f^2 \rho_f dv_f \quad (2)$$

we can express $\langle x_f^2 \rangle$ as a function of (x_i) through the map coefficients (X_{jklmnp}) (assuming symplecticity: $\rho_i dv_i = \rho_f dv_f$)

Description II

- Gaussian distributions are assumed in all coordinates (though a uniform distribution in dp/p is also accepted)
- Centered bunch is assumed to speed up the calculation of the integral
 - Initial $\alpha_{x,y}$ need to be 0 (it may require a matching section)
- MapClass takes the X_{jklmnp} coefficients as input from MADX-PTC (*fort. 18*) to produce the start-to-end \mathcal{M}
- MapClass2 can produce the transfer map up by using:
 - *fort. 18* (as MapClass)
 - *twiss* file
- Order-by-order analysis of the map \mathcal{M} provides a thorough and insightful understanding of the high order aberrations at any given location
- Map is obtained at the user-defined order (q) thus only contributions from X_{jklmnp} such that

$$j + k + l + m + n + p \leq q \quad (3)$$

MapClass2

MapClass2 can also generate transfer map by loading a Twiss file, generated by MADX-PTC, by using the module pytpsa[2]

- Twiss file should contain at least *NAME*, *KEYWORD*, *S*, *L*, *BETX*, *BETY*, *ALFX*, *ALFY*, *MUX*, *MUY*, *DX*, *DPX*, *DY*, *DPY*, *ANGLE*, *K1L*, *K2L*, *K3L*, *K4L* at the end of each element¹ to properly generate either matrix transport or map
- Transport matrices are available for drifts, bends and quads
- Maps for thin and thick (6-order symplectic integrator[3]) multipoles are also available
- MatrixToMap function is available to construct the global map of a beamline

¹in MADX, set *REFER=EXIT*

MapClass Features

Arbitrary calculation order:

- $q = 1$ linear contributions (linear Twiss)
- $q = 2$ includes chromatic effects and sextupolar fields
- $q = 3$ includes octupolar fields
- ...
- Correct calculation occurs when $q \rightarrow \infty$ (it is the user responsibility to identify the minimum q order that gives a **satisfactory approximation**)
- Collective effects, acceleration/deceleration and synchrotron radiation are not included when using the *fort.18* to build up the map
- MapClass2 can take into account synchrotron radiation (bends and quads) when building the map from twiss command
- Suitable to take into account magnet field errors

Impact of MapClass on Accelerator Community

Applications of Mapclass:

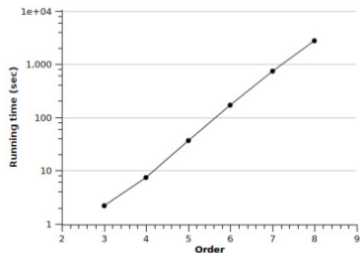
- Design of BDS (Collimation & FFS) of linear colliders
 - CLIC-BDS 380 GeV, 500 GeV and 3 TeV
 - ILC-BDS 250 GeV and 500 GeV
 - ATF2-FFS 1.3 GeV (FFS prototype of CLIC and ILC)
- Optimization of FFS of circular machines
 - CEPC IR design
 - LHC low β^*
 - IR design
 - Dynamic aperture
 - Collimation System
 - Crab cavity

Code Implementation

- Originally the programming language is Python
- Object oriented through classes:
 - class: `map2` gives access to: *offset*, *sigma*, *generatelist*, *comp* and *correlations*
 - class: `twiss2` gives access to: *getBeta*, *getDisp*, *getPhase*, *getNatChrom*, *getChrom*, *Oide*, *getH* and *sigmaBends*
- System requirements:
 - *Python2.6* (or higher)
 - Windows, Linux or MacOS X
- Prerequisites:
 - MADX-PTC
 - Numpy package
 - *argparse* library
 - *OrderedDict* class
 - *unittest2* package (included in the git project)

Performance

- Computational time for constructing the map at different orders

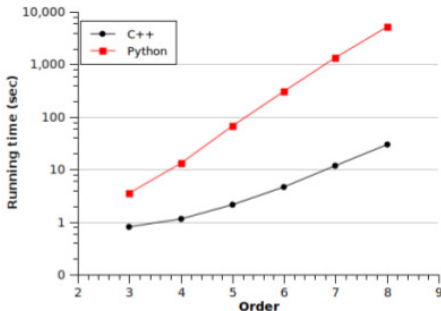


- Size of Polynomials

Var \ Order	3	4	5	6	7	8	9	10
x	31	69	135	245	415	670	1035	1546
px	31	69	135	245	415	670	1035	1546
y	24	56	116	216	376	616	966	1456
py	24	56	116	216	376	616	966	1456

MapClass C++

- In 2013 a C++ version was made available to reduce the required computational time to construct the map



Parallelization

Issues:

- Parallelization should respect the order in which the elements appear in the beamline (map composition is not commutative)
- Load imbalance (FFS is not homogeneous)

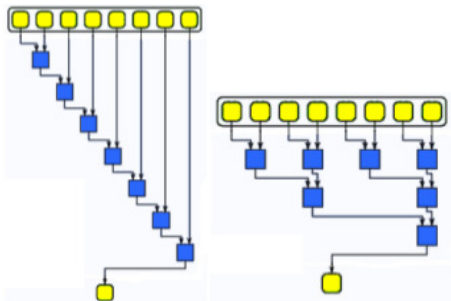
Pros:

- C++ version opened up the possibility to implement faster computational architectures (GPU)
- Developed a new multi-variable polynomial multiplication algorithm [4] (x548 compared to VLI polynomial multiplication)

Parallelization II

Adopted Solution:

- Split map construction in 2 phases :
 - Phase-1 (small maps): using multi-cores processors
 - Phase-2 (large maps): Composition of global map by GPU (new multiplication algorithm)
- Obtained speed up **x2 for 12-order map**



Installation

Extremely easy installation

Steps:

- The code and the required libraries can be found in the public repository:

`https://github.com/pylhc/MapClass2.git`

- In the MapClass2 folder, the following 2 commands have to be run:

```
git submodule init
git submodule update
```

Contributors

- Rogelio Tomás³
- Edu Marin
- Javier Barranco
- David Martinez
- Alice Rosam
- Hector Garcia Morales
- Andrea Popescu
- Riccardo de Maria

³contact person

MapClass Example

```
##### EXAMPLE SCRIPT #####
# Varies position of all sextupoles in the beamline in 0.1m steps and
# recalculates sigmas each time. If sigma has been reduced, prints the
# position change of the sextupole. For each sextupole the sigmas are
# compared with those of the original twiss object.

# Constants
order = 2

betx = 64.99988501
bety = 17.99971417
gamma = 3e6
ex = 68e-8
ey = 2e-8
sigmaFFS = [sqrt(ex*betx/gamma), sqrt(ex/betx/gamma),
            sqrt(ey*bety/gamma), sqrt(ey/bety/gamma), 0.01, 0.002]

# Initialising
t0 = twiss2('assets/ffs.twiss')
print "* Stripping the line."
t0 = t0.stripLine()
print "* Merging the elements."
t0 = t0.mergeElems()
print "* Calculating the map."
m0 = Map2(t0, order=order)
print "* Calculating original sigma x ",
sOrig = sqrt(m0.sigma('x', sigmaFFS).real)
print sOrig

# Loop through the elements in the stripped and merged beamline
print "* Starting main loop"
for i in range(1, len(t0.elems)-1):
    # Define current, previous and subsequent elements and continue if
    # i is a sextupole surrounded by drifts
    e = t0.elems[i]
    prev = t0.elems[i-1]
    nxt = t0.elems[i+1]
    if e.KEYWORD == "SEXTUPOLE" and prev.KEYWORD == "DRIFT" and nxt.KEYWORD == "DRIFT":
        print ""
        print "!!! Sextupole found with element No. %d. Looking for optimisations" % i,
        # Define lower and upper bounds as lengths of surrounding drifts
        lower = prev.L * -1
        upper = nxt.L
        # For each sextupole the original sigma is used for comparison
        # dPosIdeal is the change in position resulting in the
        # smallest sigma found for the current sextupole
```

MapClass Example

```

s0 = sOrig
dPosIdeal = 0
# Vary position of sextupole from lower bound to upper bound
# in increments of 0.1m. Recalculates twiss, map and sigmas
# having altered the element's position
while lower < upper:
    t = t0.alterElem(i, dPos=lower)
    if t is None: break # Stops if alterElem finds an issue
    m = Map2(t, order=order, nbProc=nbProc)
    s = sqrt(m.sigma('x', sigmaFFS).real)
    # If the new sigma is smaller than the previous one, make
    # them equal to compare with the next position alteration
    # and store the position change.
    if s < s0:
        s0 = s
        dPosIdeal = lower
    lower += 0.1

print ""
if dPosIdeal != 0:
    print "Original sigma x: ", sOrig
    print "Optimised sigma x: ", s0
    print "Sigma x reduced by: ", sOrig - s0
    print "This reduction occurred at a position change of: ", dPosIdeal
else:
    print "This sextupole could not be optimised"

```


[1]

<https://twiki.cern.ch/twiki/bin/view/ABPComputing/MapClass>

[2] <https://github.com/rdemaria/pytpsa>

[3] E. Forest, KEK Preprint 2005-107 (2006), page 55,
equation 124

[4] Diana Andreea Popescu, Rogelio Tomas Garcia,
Multivariate Polynomial Multiplication on GPU, Procedia
Computer Science, Volume 80, 2016, Pages 154-165

BACK UP

Map (3-Order)

- Drift Space

```
x = 25.9219*d^2*px + 25.9219*px + 1*x - 25.9219*d*px
px = 1*px
y = 25.9219*py + 1*y - 25.9219*d*py + 25.9219*d^2*py
py = 1*py
d = 1*d
s = 1*s
```

- Sector Bending Magnet

```
x = 1.714029999998822*d^2*px + 1.160223910812919e-06*d + 1.714029999999476*px
- 1.160218336849645e-06*d^2 + 1.160218336849469e-06*d^3 + 9.1637368204972e-13*d*x
+ 0.9999999999990836*x - 1.714029999998953*d*px - 9.163736820495803e-13*d^2*x
px = - 9.163736820495803e-13*d^2*px + 1.353789999999586e-06*d + 0.9999999999990836*px
- 1.069262127325333e-12*x + 9.1637368204972e-13*d*px
y = 1.71403*py + 1*y - 1.71403*d*py + 1.71403*d^2*py
py = 1*py
d = 1*d
s = 1*s
```

Map (3-Order)

- Quadrupole Magnet

```

x = 0.8000672712127237*d^2*px + 0.8049129014811202*px + 0.008996189534693885*d*x
+ 0.9909902567778666*x - 0.8024889921989531*d*px - 0.008982644002452351*d^2*x
px = - 0.008982644002452351*d^2*px + 0.9909902567778666*px - 6.711199256587141e-05*d*x
- 0.02228602739293874*x + 0.008996189534693885*d*px + 6.705140411820271e-05*d^2*x
y = 0.8097694807572918*py - 0.009050469578631945*d*y + 1.009036883238841*y
- 0.8122021545172273*d*py + 0.8109836236924255*d^2*py + 0.004511640278543325*d^2*y
py = 1.009036883238841*py - 6.735465905968741e-05*d*y + 0.02242049394029508*y
- 0.009050469578631945*d*py + 0.004511640278543325*d^2*py - 3.373807438134674e-05*d^2*y
d = 1*d
s = 1*s

```

- Sextupole Magnet (only x)

```

x = - 0.064221880564161*d*py^2 + 0.00356582805005316*px*py*y + 0.2101865852525504*py*y
- 0.021407293521387*px^2 + 0.07482424085628742*px*x^2 - 0.7738904600899998*d*y^2
- 0.4203731705051008*d*py*y + 0.7738904600899998*d*x^2 + 0.407395999999999*d^2*px
+ 0.7738904600899998*y^2 - 0.407395999999999*d*px - 0.7738904600899998*x^2
- 0.2101865852525504*px*x + 0.021407293521387*py^2 + 0.064221880564161*d*px^2
+ 0.0005385657050757909*px*py^2 + 0.006153420086374919*py^2*x
+ 0.0005385657050757909*px^3 + 0.1857085299651543*x^3
+ 0.009719248136428079*px^2*x + 0.1857085299651543*x*y^2
- 0.01835750008585933*px*y^2 + 0.407395999999999*px + 1*x
+ 0.09318174094214675*py*x*y + 0.4203731705051008*d*px*x

```