Introduction to MADX for Beamlines

Adam Watts

EBD bi-weekly meeting

21 March 2019

Overview

From http://mad.web.cern.ch/mad/:

MAD-X is a project with a long history, aiming to be at the forefront of computational physics in the field of particle accelerator design and simulation. Its scripting language is de facto the standard to describe particle accelerators, simulate beam dynamics and optimize beam optics at CERN.

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

MAD-X is released for the Linux, Mac OS X and Windows platforms for 32 bit (on demand) and 64 bit architectures (see releases). The source code is written in C, C++, Fortan77 and Fortran90.

Overview

- MADX is widely used and supported in the accelerator community (and USPAS).
- It is very fast, easily-distributed, and heavily scrutinized (i.e. CERN and LHC operation).
- Program is run on a user-created configuration file that describes the beamline and any calculates requested, a.k.a. the “deck”.

Linux/Unix command line:

```
madx  numi.madx > out.txt
```

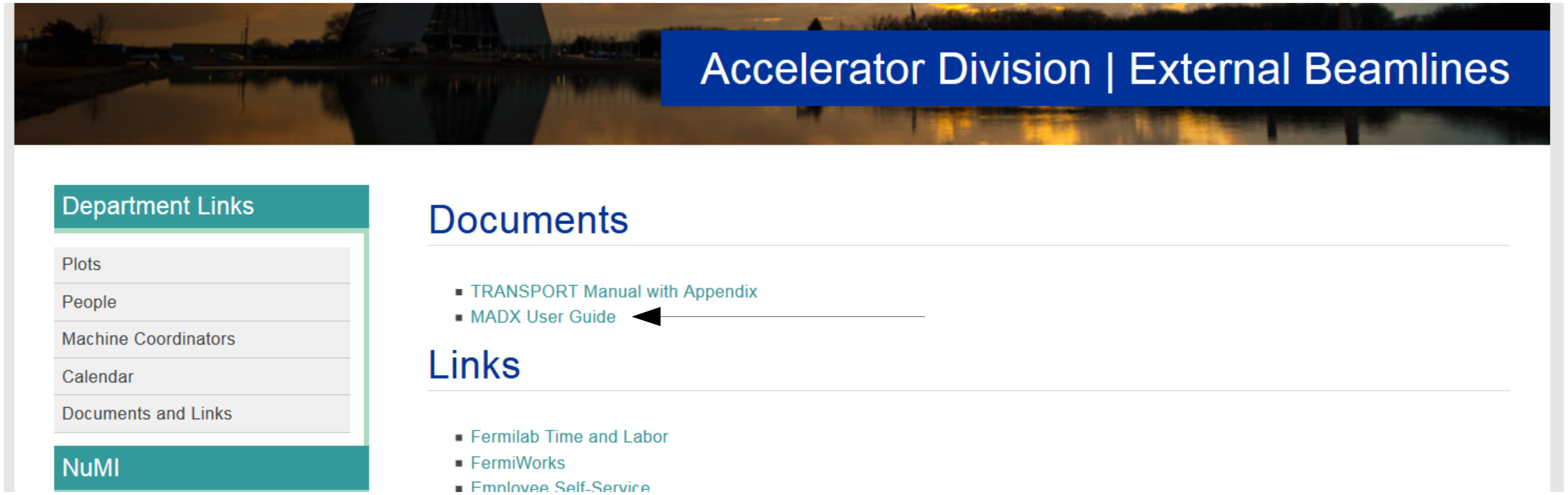
Windows command line:

```
madx < numi.madx > out.txt
```

Fermilab

MADX User Guide

- Copy available on the EBD website under “Documents and Links”: http://extbeams.fnal.gov/documents.html
- Fairly extensive in terms of usage details, but not so much on calculation methodology or examples.


Accelerator Division | External Beamlines

Department Links

- Plots
- People
- Machine Coordinators
- Calendar
- Documents and Links

NuMI

Documents

- TRANSPORT Manual with Appendix
- MADX User Guide

Links

- Fermilab Time and Labor
- FermiWorks
- Employee Self-Service

Example Input File

```
qk1 := 0.250076;
ld = 5.0;
lq = 1.0;

D: DRIFT,L=ld;
QF: QUADRUPOLE,L=0.5*lq,K1:=qk1;
QD: QUADRUPOLE,L=0.5*lq,K1:=-qk1;

FODO: LINE=(QF,D,QD,QD,D,QF);

beam,particle=proton,energy=120.938;
use,period=FODO;

twiss, save, file=twiss.out,
BETX=15.0, BETY=5.0;

plot ,haxis=s, vaxis=betx, bety,
file=beta_plot;
```

This example input file describes a single FODO cell, starting at the center of the F quadrupole and ending at the center of the next F quadrupole.

Example Input File

```
qk1 := 0.250076;
ld = 5.0;
lq = 1.0;

D: DRIFT,L=ld;
QF: QUADRUPOLE,L=0.5*lq,K1:=qk1;
QD: QUADRUPOLE,L=0.5*lq,K1:=-qk1;

FODO: LINE=(QF,D,QD,QD,D,QF);

beam,particle=proton,energy=120.938;
use,period=FODO;

twiss, save, file=twiss.out,
BETX=15.0, BETY=5.0;

plot ,haxis=s, vaxis=betx, bety,
file=beta_plot;
```

Define some variables for quadrupole strength and drift lengths.

Deferred assignment, “:=”, allows for MADX to change these variables later if necessary.

Regular assignment, “=”, means the value cannot change after initalization.

All MADX input commands must end with a semicolon!

Example Input File

```
qk1 := 0.250076;
ld = 5.0;
lq = 1.0;

D: DRIFT,L=ld;
QF: QUADRUPOLE,L=0.5*lq,K1:=qk1;
QD: QUADRUPOLE,L=0.5*lq,K1:=-qk1;

FODO: LINE=(QF,D,QD,QD,D,QF);

beam,particle=proton,energy=120.938;
use,period=FODO;

twiss, save, file=twiss.out,
BETX=15.0, BETY=5.0;

plot ,haxis=s, vaxis=betx, bety,
file=beta_plot;
```

Define beamline elements.

Use deferred assignment for quadrupole strength so it can be tuned later.

Beamline element quantities can be hard-coded, variables, or computed values.

Example Input File

```
qk1 := 0.250076;
ld = 5.0;
lq = 1.0;

D: DRIFT,L=ld;
QF: QUADRUPOLE,L=0.5*lq,K1:=qk1;
QD: QUADRUPOLE,L=0.5*lq,K1:=-qk1;

FODO: LINE=(QF,D,QD,QD,D,QF);

beam,particle=proton,energy=120.938;
use,period=FODO;

twiss, save, file=twiss.out,
BETX=15.0, BETY=5.0;

plot ,haxis=s, vaxis=betx, bety,
file=beta_plot;
```

$\longleftarrow$ Build the beamline as a sequence of elements.

Sequences of sequences are also possible, but discouraged for operational simulations (hard to read).

Example Input File

```
qk1 := 0.250076;
ld = 5.0;
lq = 1.0;

D: DRIFT,L=ld;
QF: QUADRUPOLE,L=0.5*lq,K1:=qk1;
QD: QUADRUPOLE,L=0.5*lq,K1:=-qk1;

FODO: LINE=(QF,D,QD,QD,D,QF);

beam,particle=proton,energy=120.938;
use,period=FODO;

twiss, save, file=twiss.out,
BETX=15.0, BETY=5.0;

plot ,haxis=s, vaxis=betx, bety,
file=beta_plot;
```

Define the beam.

Energy is total per-particle energy in GeV.

Example Input File

```
qk1 := 0.250076;
ld = 5.0;
lq = 1.0;

D: DRIFT,L=ld;
QF: QUADRUPOLE,L=0.5*lq,K1:=qk1;
QD: QUADRUPOLE,L=0.5*lq,K1:=-qk1;

FODO: LINE=(QF,D,QD,QD,D,QF);

beam,particle=proton,energy=120.938;
use,period=FODO;

twiss, save, file=twiss.out,
BETX=15.0, BETY=5.0;

plot ,haxis=s, vaxis=betx, bety,
file=beta_plot;
```

$\longleftarrow$ Select the beamline we created for use in calculations.

Example Input File

```
qk1 := 0.250076;
ld = 5.0;
lq = 1.0;

D: DRIFT,L=ld;
QF: QUADRUPOLE,L=0.5*lq,K1:=qk1;
QD: QUADRUPOLE,L=0.5*lq,K1:=-qk1;

FODO: LINE=(QF,D,QD,QD,D,QF);

beam,particle=proton,energy=120.938;
use,period=FODO;

twiss, save, file=twiss.out,
BETX=15.0, BETY=5.0;

plot ,haxis=s, vaxis=betx, bety,
file=beta_plot;
```

$\longleftarrow$ Run the “Twiss” module on the beamline we defined, given initial Beta functions (called “beamline mode”).

Any parameters not specified default to zero.

Example Input File

```
qk1 := 0.250076;
ld = 5.0;
lq = 1.0;

D: DRIFT,L=ld;
QF: QUADRUPOLE,L=0.5*lq,K1:=qk1;
QD: QUADRUPOLE,L=0.5*lq,K1:=-qk1;

FODO: LINE=(QF,D,QD,QD,D,QF);

beam,particle=proton,energy=120.938;
use,period=FODO;

twiss, save, file="twiss.out",
BETX=15.0, BETY=5.0;

plot, haxis=s, vaxis=betx, bety,
file=beta_plot;
```

$\longleftarrow$ Plot the beta functions along the beamline.

Output file (“out.txt”)

Input file is repeated above. Warnings or errors will (sometimes) be printed after the offending line.

```
enter Twiss module
++++++ info: Zero value of SIGT replaced by 1.
++++++ info: Zero value of SIGE replaced by 1/1000.
open line - error with deltap:   0.000000E+00
end values:   0.000000E+00  0.000000E+00  0.000000E+00  0.000000E+00  0.000000E+00  0.000000E+00
++++++ table: summm
          length          orbit5            alfa         gammatr
              12              -0               0               0
              q1             dq1         betxmax           dxmax
    0.2500002035               0     20.33685241               0
           dxrms          xcomax          xcorms              q2
               0               0               0    0.2500003265
             dq2         betymax           dymax           dyrms
               0     22.23767957               0               0
          ycomax          ycorms          deltap         synch_1
               0               0               0               0
         synch_2         synch_3         synch_4         synch_5
               0               0               0               0

  Number of warnings: 0

  ++++++++++++++++++++++++++++++++++++++++++++
  + MAD-X 5.01.00 (32 bit) finished normally +
  ++++++++++++++++++++++++++++++++++++++++++++
```

Important, but isn't a guarantee that the deck ran properly.

Effectively shows “compile” errors, not logical errors.

Fermilab

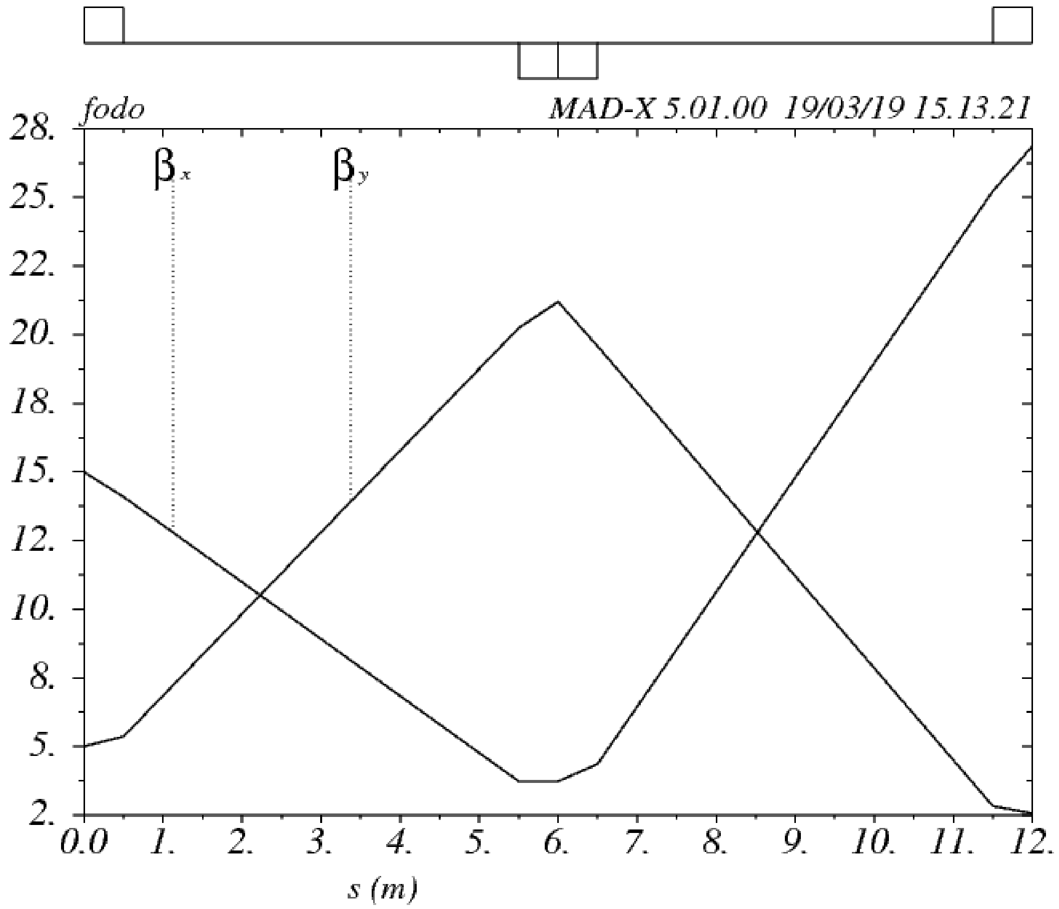
Twiss output (“twiss.out”)

This is just a sample of information contained in twiss.out. Other information includes average trajectory offset if kicks are present, 6x6 beamline matrix at each element, etc.

```
* NAME              KEYWORD                            S          BETX              ALFX               MUX              BETY              ALFY               MUY
$ %s                %s                               %le           %le               %le               %le               %le               %le               %le
 "FODO$START"       "MARKER"                           0   19.81848928                 0                 0       5.674619595                 0                 0
 "QF"               "QUADRUPOLE"                     0.5   18.61741767       2.351873467    0.004100232083       6.081827976     -0.8313186165     0.01370431647
 "D"                "DRIFT"                          5.5    3.86911094      0.5977878786       0.104363217       21.34642188      -2.221600164     0.08600674203
 "QD"               "QUADRUPOLE"                       6   3.582617698    -0.01291002165      0.1260273929       22.23767957      0.4763875707     0.08962155185
 "QD"               "QUADRUPOLE"                     6.5   3.896020682     -0.6269042524      0.1476161582       20.43286418       3.057704298     0.09331630172
 "D"                "DRIFT"                         11.5   19.10372816      -2.414637244      0.2460044372       2.518700327      0.5251284721      0.2160518571
 "QF"               "QUADRUPOLE"                      12   20.33685241  3.344315192e-008      0.2500002035       2.261474988 -1.234072405e-006      0.2500003265
 "FODO$END"         "MARKER"                          12   20.33685241  3.344315192e-008      0.2500002035       2.261474988 -1.234072405e-006      0.2500003265
```

Plot Output ("beta_plot.ps")

Plot is in "post-script" file format, similar to PDF. Beamline schematic is at the top, with focusing quad as a box on top of the line and de-focusing underneath the line.



Twiss Module Ring Mode

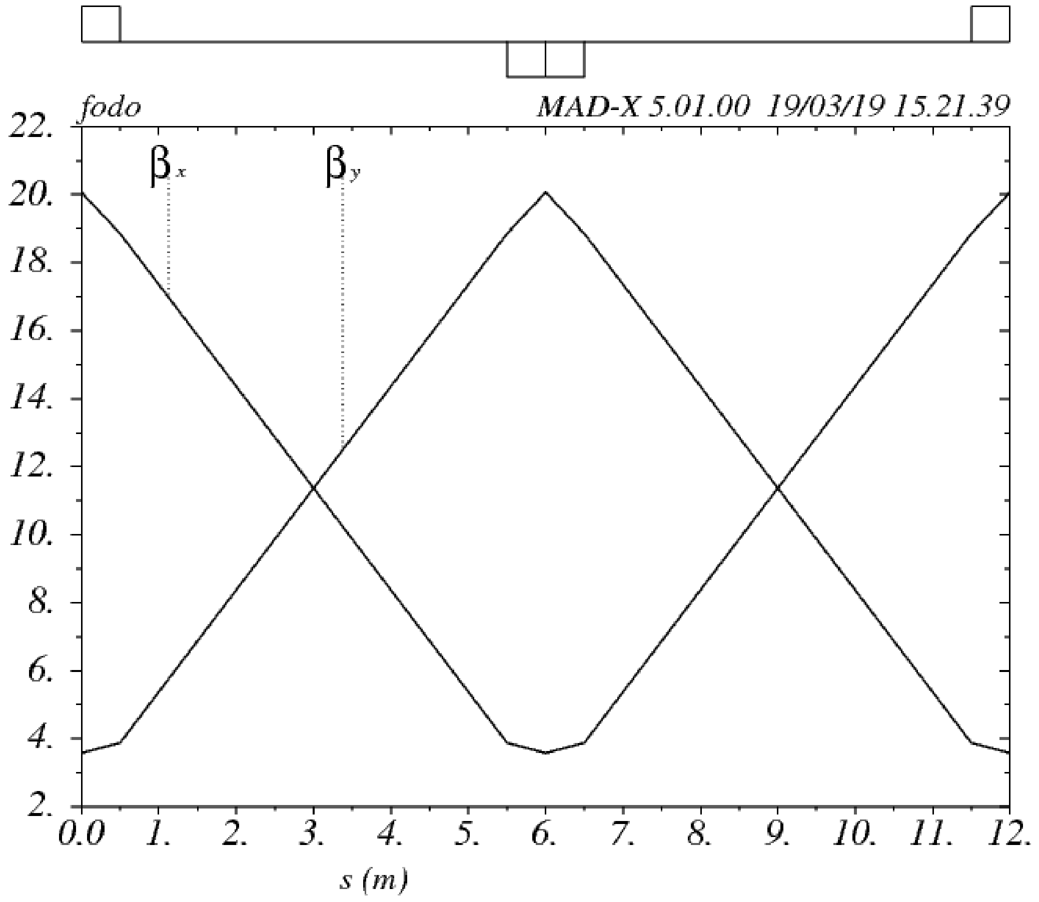
```
qk1 := 0.250076;
ld = 5.0;
lq = 1.0;

D: DRIFT,L=ld;
QF: QUADRUPOLE,L=0.5*lq,K1:=qk1;
QD: QUADRUPOLE,L=0.5*lq,K1:=-qk1;

FODO: LINE=(QF,D,QD,QD,D,QF);

beam,particle=proton,energy=120.938;
use,period=FODO;

twiss, save, file=twiss.out;

plot, haxis=s, vaxis=betx, bety,
file=beta_plot;
```

Since our example beamline is symmetric, let's have MADX find the matched solution, i.e. the input Courant-Snyder parameters that start and end at the same values when propagating through our beamline.

We call the Twiss module without any initial conditions, signifying we want to compute the matched solution. This is equivalent to a ring, where Courant-Snyder parameters return to the same values after every revolution in the ideal matched condition; hence this is called “ring mode”.

Plot Output ("beta_plot.ps") of Matched Solution

If a matched solution is possible, the Twiss module will find it and run those initial conditions through the beamline. If not (for example, if the beamline is not symmetric), then the Twiss module will fail.



The Match Module

```
qk1 := 0.250076;
ld = 5.0;
lq = 1.0;

D: DRIFT,L=ld;
QF: QUADRUPOLE,L=0.5*lq,K1:=qk1;
QD: QUADRUPOLE,L=0.5*lq,K1:=-qk1;

FODO: LINE=(QF,D,QD,QD,D,QF);

beam,particle=proton,energy=120.938;
use,period=FODO;

match, sequence=FODO,
BETX=19.81848928, BETY=5.674619595;
vary, name=qk1, step=.00001;
constraint, sequence=FODO, range=#e, mux=0.4 muy=0.4;
LMDIF, calls=10000, tolerance=1E-10;
endmatch;

twiss, save, file="twiss.out",
BETX=15.0, BETY=5.0;

plot, haxis=s, vaxis=betx, bety, file=beta_plot;
```

The Match module of MADX allows for optimization of beamline parameters, either standard or user-defined.

For our example FODO cell, we will provide initial beta functions and ask MADX to vary the quadrupole strengths until the phase advance in both planes is 90 degrees over the cell.

The Match Module

```
qk1 := 0.250076;
ld = 5.0;
lq = 1.0;

D: DRIFT,L=ld;
QF: QUADRUPOLE,L=0.5*lq,K1:=qk1;
QD: QUADRUPOLE,L=0.5*lq,K1:=-qk1;

FODO: LINE=(QF,D,QD,QD,D,QF);

beam,particle=proton,energy=120.938;
use,period=FODO;

match, sequence=FODO,
BETX=19.81848928, BETY=5.674619595;
vary, name=qk1, step=.00001;
constraint, sequence=FODO, range=#e, mux=0.4 muy=0.4;
LMDIF, calls=10000, tolerance=1E-10;
endmatch;

twiss, save, file="twiss.out",
BETX=15.0, BETY=5.0;

plot, haxis=s, vaxis=betx, bety, file=beta_plot;
```

Call the Match module, tell it which beamline to use, and give it our initial conditions (matched solution from Twiss ring mode).

The Match Module

```
qk1 := 0.250076;
ld = 5.0;
lq = 1.0;

D: DRIFT,L=ld;
QF: QUADRUPOLE,L=0.5*lq,K1:=qk1;
QD: QUADRUPOLE,L=0.5*lq,K1:=-qk1;

FODO: LINE=(QF,D,QD,QD,D,QF);

beam,particle=proton,energy=120.938;
use,period=FODO;

match, sequence=FODO,
BETX=19.81848928, BETY=5.674619595;
vary, name=qk1, step=.00001;
constraint, sequence=FODO, range=#e, mux=0.4,muy=0.4;
LMDIF, calls=10000, tolerance=1E-10;
endmatch;

twiss, save, file="twiss.out",
BETX=15.0, BETY=5.0;

plot, haxis=s, vaxis=betx, bety, file=beta_plot;
```

Tell the Match module which beamline parameters to vary and suggest a step amount (or omit for default).

We're only changing one variable for this example, but more is possible (see later example).

Variable must have been defined with deferred assignment! (":=", not "=")

The Match Module

```
qk1 := 0.250076;
ld = 5.0;
lq = 1.0;

D: DRIFT,L=ld;
QF: QUADRUPOLE,L=0.5*lq,K1:=qk1;
QD: QUADRUPOLE,L=0.5*lq,K1:=-qk1;

FODO: LINE=(QF,D,QD,QD,D,QF);

beam,particle=proton,energy=120.938;
use,period=FODO;

match, sequence=FODO,
BETX=19.81848928, BETY=5.674619595;
vary, name=qk1, step=.00001;
constraint, sequence=FODO, range=#e, mux=0.4, muy=0.4;
LMDIF, calls=10000, tolerance=1E-10;
endmatch;

twiss, save, file="twiss.out",
BETX=15.0, BETY=5.0;

plot, haxis=s, vaxis=betx, bety, file=beta_plot;
```

$\longleftarrow$ Define a goal for the optimizer to achieve. In this case, we want the betatron phase at the end (“#e”) of the line to be 90 degrees (i.e. 0.25 in units of 2-pi).

The Match Module

```
qk1 := 0.250076;
ld = 5.0;
lq = 1.0;

D: DRIFT,L=ld;
QF: QUADRUPOLE,L=0.5*lq,K1:=qk1;
QD: QUADRUPOLE,L=0.5*lq,K1:=-qk1;

FODO: LINE=(QF,D,QD,QD,D,QF);

beam,particle=proton,energy=120.938;
use,period=FODO;

match, sequence=FODO,
BETX=19.81848928, BETY=5.674619595;
vary, name=qk1, step=.00001;
constraint, sequence=FODO, range=#e, mux=0.4, muy=0.4;
LMDIF, calls=10000, tolerance=1E-10;
endmatch;

twiss, save, file="twiss.out",
BETX=19.81848928, BETY=5.674619595;

plot, haxis=s, vaxis=betx, bety, mux, muy, file=beta_plot;
```

Choose an optimization algorithm and initialize with parameters (or omit for default).

Close out the Match module.

The Match Module

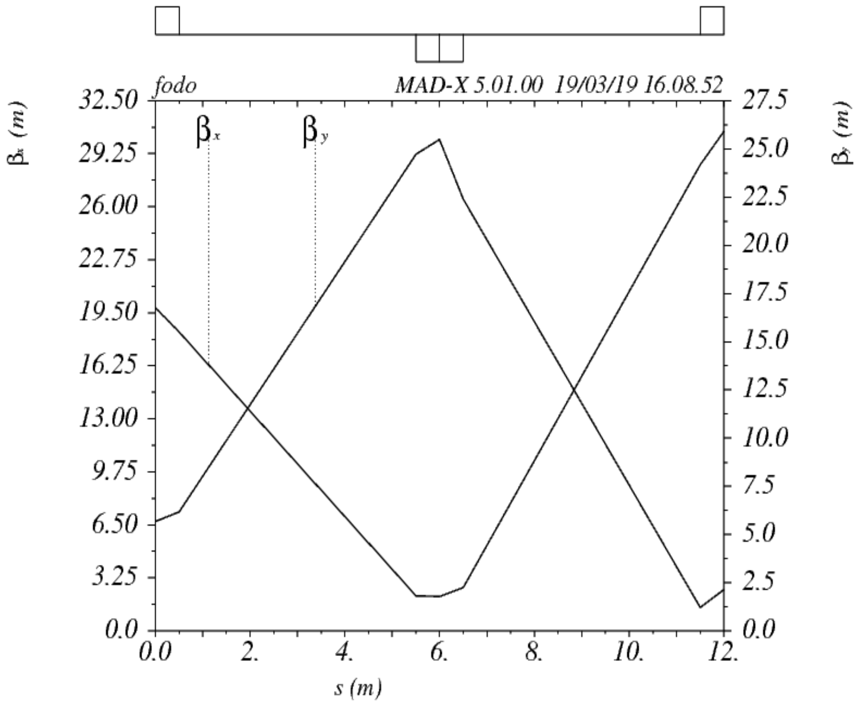
```
qk1 := 0.250076;
ld = 5.0;
lq = 1.0;

D: DRIFT,L=ld;
QF: QUADRUPOLE,L=0.5*lq,K1:=qk1;
QD: QUADRUPOLE,L=0.5*lq,K1:=-qk1;

FODO: LINE=(QF,D,QD,QD,D,QF);

beam,particle=proton,energy=120.938;
use,period=FODO;

match, sequence=FODO,
BETX=19.81848928, BETY=5.674619595;
vary, name=qk1, step=.00001;
constraint, sequence=FODO, range=#e, mux=0.4, muy=0.4;
LMDIF, calls=10000, tolerance=1E-10;
endmatch;

twiss, save, file="twiss.out",
BETX=19.81848928, BETY=5.674619595;

plot, haxis=s, vaxis=betx, bety file=beta_plot;
plot, haxis=s, vaxis=mux, muy, file=phase_plot;
```

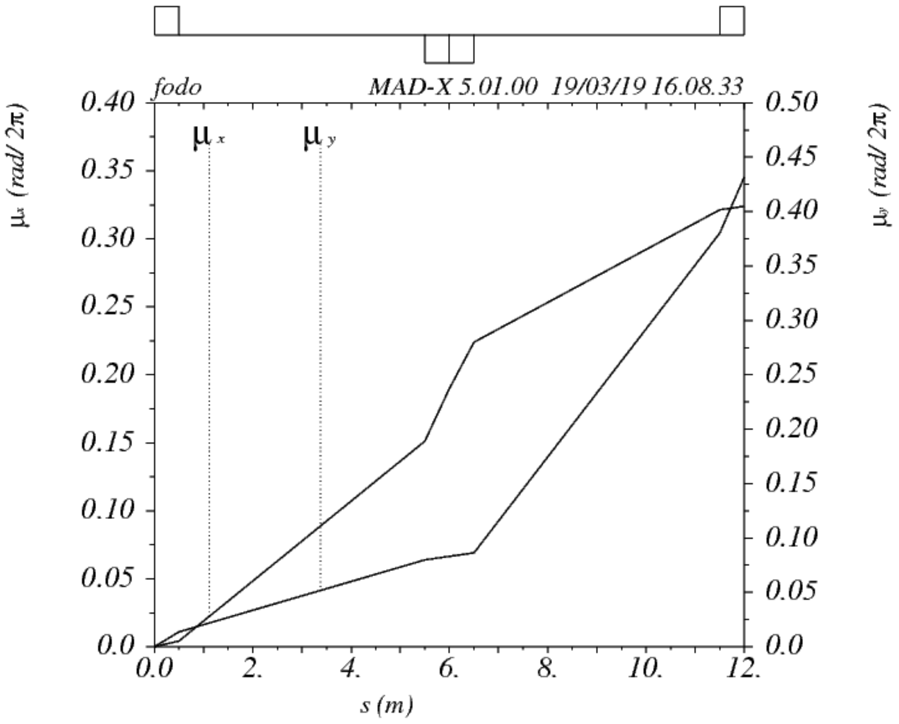
Run the Twiss module using the same initial parameters as the Match module (or we'll get very confused at the result).

Plot the betas and phases to see the result. MADX automatically uses the quad strength found by the Match module.

Results of Match

Looking at the plots, it looks like the optimizer got close to what we wanted. Let's look at the output file to see how close and how hard it tried.





Results of Match ("out.txt")

```
number of variables:    1
user given constraints: 2
total constraints:      2

START LMDIF:

Initial Penalty Function =   0.44999841E+01


call:       3   Penalty function =   0.91121290E+00
call:       5   Penalty function =   0.67929649E+00
call:       7   Penalty function =   0.67895022E+00
call:       9   Penalty function =   0.67894244E+00
call:      11   Penalty function =   0.67894222E+00
call:      13   Penalty function =   0.67894221E+00
call:      15   Penalty function =   0.67894221E+00
call:      17   Penalty function =   0.67894221E+00
call:      19   Penalty function =   0.67894221E+00
call:      21   Penalty function =   0.67894221E+00
 ++++++++++ LMDIF ended: converged without success
call:      21   Penalty function =   0.67894221E+00
endmatch;


MATCH SUMMARY

Node_Name                Constraint   Type  Target Value       Final Value        Penalty
---------------------------------------------------------------------------------------------------
fodo$end:1             mux        4     4.00000000E-01    3.23960408E-01    5.78201961E-01
fodo$end:1             muy        4     4.00000000E-01    4.31739604E-01    1.00740249E-01


Final Penalty Function =  6.78942210e-001



Variable                 Final Value  Initial Value Lower Limit  Upper Limit
--------------------------------------------------------------------------------
qk1                    3.12104e-001 2.50076e-001 -1.00000e+020 1.00000e+020

END MATCH SUMMARY
```

The optimizer failed after several iterations because the penalty function was no longer changing. It's possible that one of the more advanced algorithms may fare better.

This table shows how close the optimizer got to the desired constraints.

This table summarizes the before and after values of the parameters that the optimizer varied.

Fermilab

Bigger Match Example: Beam to SY Dump

```
CALL, FILE="circuit.cir";
CALL, FILE="p1_lam52.lat";
CALL, FILE="p2.lat";
CALL, FILE="p3.lat";
CALL, FILE="xhall.lat";
CALL, FILE="EncB.lat";
CALL, FILE="EncCdump.lat";

SYline:line=(p1line,p2line,p3line,xhline,Bline,Cline);
use,sequence=SYline;

match, sequence=SYline,
= Slow-spill effective ellipse from John's simulation
BETX = 131.876421237, ALFX = -5.77039864992,
BETY = 29.3132039862, ALFY = 2.21653246787,
DX = -0.0428991, DPX = -0.0035253,
DY = 0.0, DPY = 0.0;
vary, name=K1P1Q1,    step=.0001;
vary, name=K1P1Q2,    step=.0001;
vary, name=K1P1CELL,  step=.0001;
vary, name=K1P1Q10,   step=.0001;
vary, name=K1P1Q11,   step=.0001;
vary, name=K1P1Q12,   step=.0001;
vary, name=K1P1Q13,   step=.0001;
vary, name=K1P1Q14,   step=.0001;
vary, name=K1QF11A,   step=.0001;
vary, name=K1QF11B,   step=.0001;
vary, name=QP2,       step=1;
vary, name=QP3,       step=1;
vary, name=Q80,       step=0.1, lower=-50, upper=50;
vary, name=Q90,       step=0.1, lower=-50, upper=50;
vary, name=Q100,      step=0.1, lower=-50, upper=50;
vary, name=Q101,      step=0.1, lower=-50, upper=50;
vary, name=Q102,      step=0.1, lower=-50, upper=50;
constraint,  sequence=SYline, range=Q701/D3IN, betx<100., bety<100., DX<6., DY<6., DX>-6., DY>-6.;
constraint,  sequence=SYline, range=P2/XHALL, betx<200., bety<200., DX<8., DY<8., DX>-8., DY>-8.;
constraint, sequence=SYLINE, range=XHALL/SYDUMP, betx<600., bety<600.;
constraint, sequence=SYline, range=SYDUMP, betx=100., bety=100., DX<6., DY<6., DX>-6., DY>-6.;
LMDIF, calls=10000, tolerance=1E-08;
endmatch;

use,sequence=SYline;
twiss, RMATRIX=true, save, file="twiss.out",
!= Slow-spill effective ellipse from John's simulation
BETX = 131.876421237, ALFX = -5.77039864992,
BETY = 29.3132039862, ALFY = 2.21653246787,
DX = -0.0428991, DPX = -0.0035253,
DY = 0.0, DPY = 0.0;
```

This real-world example shows how to build a large beamline out of multiple lattice files (i.e. sublines) by calling separate files.

Comment lines start with a “!”

Fermilab

Bigger Match Example: Beam to SY Dump

```
CALL, FILE="circuit.cir";
CALL, FILE="p1_lam52.lat";
CALL, FILE="p2.lat";
CALL, FILE="p3.lat";
CALL, FILE="xhall.lat";
CALL, FILE="EncB.lat";
CALL, FILE="EncCdump.lat";

SYline:line=(p1line,p2line,p3line,xhline,Bline,Cline);
use,sequence=SYline;

match, sequence=SYline,
= Slow-spill effective ellipse from John's simulation
BETX = 131.876421237, ALFX = -5.77039864992,
BETY = 29.3132039862, ALFY = 2.21653246787,
DX = -0.0428991, DPX = -0.0035253,
DY = 0.0, DPY = 0.0;
vary, name=K1P1Q1,    step=.0001;
vary, name=K1P1Q2,    step=.0001;
vary, name=K1P1CELL,  step=.0001;
vary, name=K1P1Q10,   step=.0001;
vary, name=K1P1Q11,   step=.0001;
vary, name=K1P1Q12,   step=.0001;
vary, name=K1P1Q13,   step=.0001;
vary, name=K1P1Q14,   step=.0001;
vary, name=K1QF11A,   step=.0001;
vary, name=K1QF11B,   step=.0001;
vary, name=QP2,       step=1;
vary, name=QP3,       step=1;
vary, name=Q80,       step=0.1, lower=-50, upper=50;
vary, name=Q90,       step=0.1, lower=-50, upper=50;
vary, name=Q100,      step=0.1, lower=-50, upper=50;
vary, name=Q101,      step=0.1, lower=-50, upper=50;
vary, name=Q102,      step=0.1, lower=-50, upper=50;
constraint,  sequence=SYline, range=Q701/D3IN, betx<100., bety<100., DX<6., DY<6., DX>-6., DY>-6.;
constraint,  sequence=SYline, range=P2/XHALL, betx<200., bety<200., DX<8., DY<8., DX>-8., DY>-8.;
constraint, sequence=SYLINE, range=XHALL/SYDUMP, betx<600., bety<600.;
constraint, sequence=SYline, range=SYDUMP, betx=100., bety=100., DX<6., DY<6., DX>-6., DY>-6.;
LMDIF, calls=10000, tolerance=1E-08;
endmatch;

use,sequence=SYline;
twiss, RMATRIX=true, save, file="twiss.out",
!= Slow-spill effective ellipse from John's simulation
BETX = 131.876421237, ALFX = -5.77039864992,
BETY = 29.3132039862, ALFY = 2.21653246787,
DX = -0.0428991, DPX = -0.0035253,
DY = 0.0, DPY = 0.0;
```

Call individual beamline lattice files, as well as a "circuit" file that sets the values of all magnet strengths.

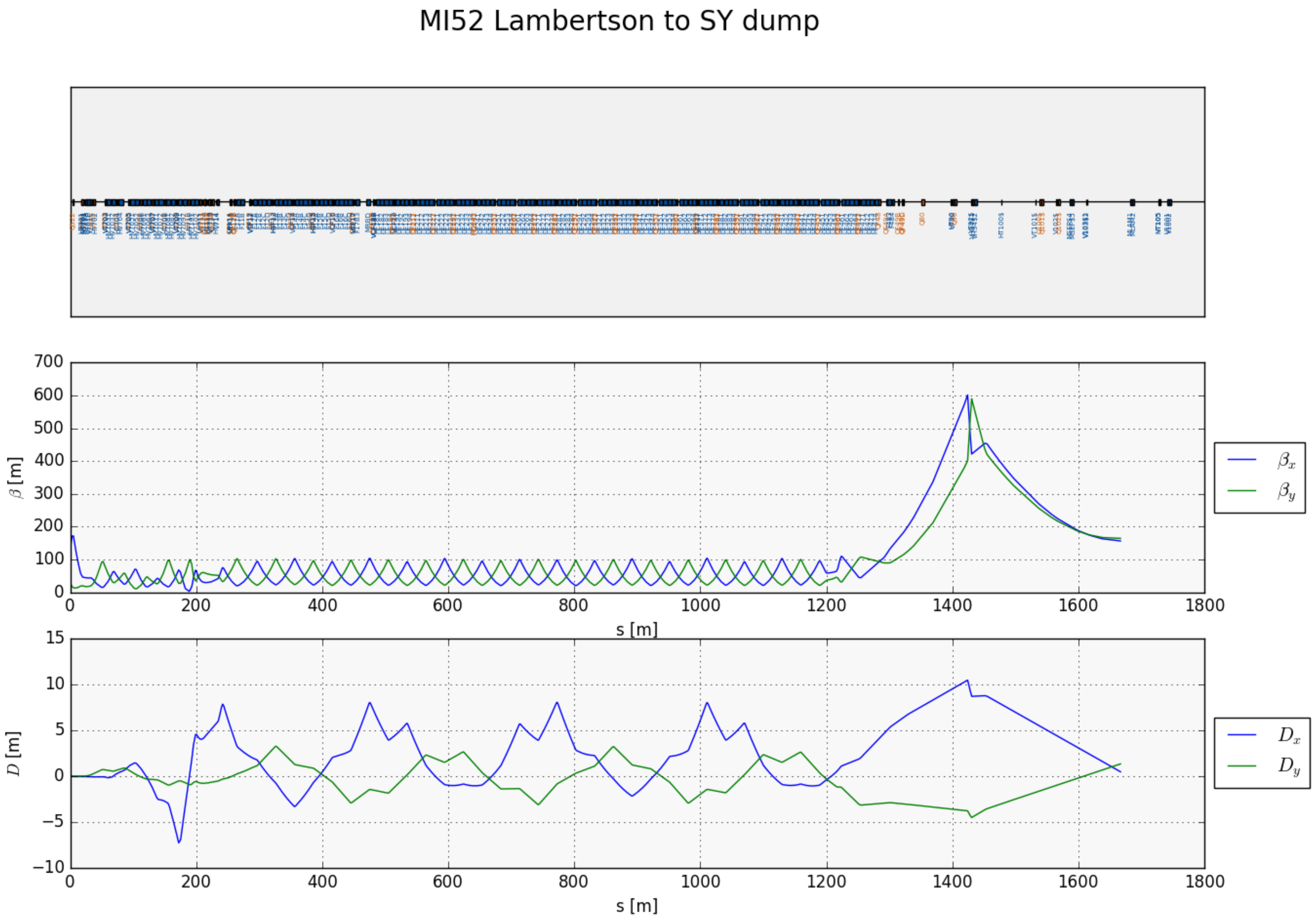
Build a beamline out of individual sub-lines defined in each lattice file.

Tune every quadrupole in the beamline, given an initial ellipse from John's resonant extraction simulation.

- Try to achieve different goals in each section.
- For P1, keep dispersion and betas low.
- For P2/P3, relax beta and dispersion constraints a bit.
- After P3, relax beta constraint even more.
- Finally, try to focus the beam down on the dump.

Fermilab

Results of Match (with Python plotting script)



Next class?

What does everyone want to see for more advanced topics? Some ideas:

- Running everything from our new department machine ebdgpvm01.fnal.gov (including interface to CVS repository)
- Particle tracking
- Generating survey information (i.e. “floor coordinates”)