

Guide and Examples for AT 2.0

October 12, 2015

Abstract

We guide the reader through the basic operations of AT 2.0, starting with lattice creation and manipulation, then with tracking and optics and beam size computation.

1 Introduction

AT is a toolbox of functions in Matlab for charged particle beam simulation. It was created by Andrei Terebilo in the late 1990's. The original papers [1, 2] still serve as a good introduction to AT. The AT described in those papers is AT1.3, the latest version produced by Terebilo. The next version of AT is considered AT2.0. Here we provide examples showing some of the changes from AT1.3, but also serving as an introduction for someone just starting AT.

1.1 Coordinates

The 6-d phase space coordinates used in AT are as follows

$$\vec{Z} = \begin{pmatrix} x \\ \frac{p_x}{P_0} \\ y \\ \frac{p_y}{P_0} \\ \delta \\ (-)ct \end{pmatrix} \quad (1)$$

The momenta are defined as

$$p_x = x' P_z \quad p_y = y' P_z \quad (2)$$

with $P_z = P_0(1 + \delta)$. P_0 is the reference momentum.

2 Creation of Elements and Lattices

A lattice in AT is a Matlab cell array containing the lattice elements (can be a row or a column). These elements may be created using element creation functions. These functions output element structures. For example, a quadrupole may be created with the function **atquadrupole**

```
>> QF=atquadrupole('QF',0.5,1.2,'QuadMPoleFringePass');
```

Observing the resulting structure, we see

```
QF =
```

```

      FamName: 'QF'
      PassMethod: 'QuadMPoleFringePass'
      Length: 0.5000
      Class: 'Quadrupole'
      K: 1.2000
      PolynomB: [0 1.2000]
      MaxOrder: 1
      PolynomA: [0 0]
      NumIntSteps: 10
```

We note that the family name of this quadrupole is 'QF' and the pass method is QuadMPoleFringePass. The fields following are parameters necessary to be able to pass an electron through this quadrupole (i.e., the set of arguments required by the pass method). We now create some other elements needed in a FODO lattice:

```

>> Dr=atdrift('Dr',0.5);
>> HalfDr=atdrift('Dr',0.25);
>> QD = atquadrupole('QD',0.5,-1.2,'QuadMPoleFringePass');
>> Bend=atsbend('Bend',1,2*pi/40,'PassMethod','BndMPoleSymplectic4Pass','Energy',3e9);
```

In addition to **atquadrupole** that we already saw, we have created a drift (region with no magnetic field), using **atdrift** and a sector dipole, or bend magnet using **atsbend**.

A cell of a FODO lattice may now be constructed as follows

```
>> FODOcell=[{HalfDr};{Bend};{Dr};{QF};{Dr};{Bend};{Dr};{QD};
{HalfDr}];
```

This cell is only 1/20 of a FODO lattice. The entire lattice may be created by repeating this cell 20 times as follows

```
>> FODO=repmat(FODOcell,20,1);
```

We have now created a valid AT lattice, using drifts, dipoles, and quadrupoles. We will later add some sextupoles to this lattice, and also an RF cavity, but one could track particles through this lattice, as is.

For completeness, we list all of the AT element creation functions: **atapture**, **atbaseelem**, **atchromMatElem**, **atcorrector**, **atdrift**, **atdampMatElem** (?), **atdrift**, **atidtable**, **atM66**, **atmarker**, **atmonitor**, **atmultipole**, **atoctupole**, **atquadrupole**, **atQuantDiff**, **atrbend**, **atrfcavity**, **atringparam**, **atsbend**, **atsextupole**, **atsolenoid**, **atthinmultipole**, **atwiggler**.

3 Lattice Querying and Manipulation

There are many parameters in a storage ring lattice. We need tools to view these parameters and to change them. In this section, we describe three important functions for working with AT lattices: **findcells**, **getcellstruct**, and **setcellstruct**. **findcells** allows one to find the indices of elements in the lattice searching for various criteria. The second two allow extraction and setting of element parameters.

The use of **findcells** in [1] used 'FamName' to retrieve element indices. For example, for our FODO lattice, we could extract the indices of all of the QF's as follows

```
>> QFIndices = findcells(FODO,'FamName','QF')
```

which results in

```
QFIndices =
```

```
Columns 1 through 11
```

```
4    13    22    31    40    49    58    67    76    85    94
```

```
Columns 12 through 20
```

```
103   112   121   130   139   148   157   166   175
```

where we see the indices of the 20 QF's.

A new feature added to AT 2.0 is element classes. These are *Drift*, *Bend*, *Quadrupole*, *Sextupole*, *Octupole*, *Multipole*, *ThinMultipole*, *Wiggler*, *KickMap*, *RFCavity*, *QuantDiff*, *Monitor*, *Corrector*, *Solenoid*, *Matrix66*, *RingParam*. This allows us to get the indices for all of the quadrupoles with the command

```
>> QuadIndices = findcells(FODO,'Class','Quadrupole')
```

with the result

```
QuadIndices =
```

```
Columns 1 through 12
```

```
4     8    13    17    22    26    31    35    40    44    49    53
```

```
Columns 13 through 24
```

```
58    62    67    71    76    80    85    89    94    98   103   107
```

```
Columns 25 through 36
```

```
112   116   121   125   130   134   139   143   148   152   157   161
```

Columns 37 through 40

```
166    170    175    179
```

showing the indices of all the quadrupoles in the FODO lattice.

The reader can look further into the help for **findcells** by typing

```
>> help findcells
```

to see other ways this function can be used, including the use of matching regular expressions.

The **getcellstruct** command may be used to extract the fields with the elements. For example the quadrupole strengths may be retrieved using the command

```
>> Kvals=getcellstruct(FODO,'PolynomB',QuadIndices,2);
```

The **setcellstruct** command may be used to set the values of parameters. As a simple example, we may add some random errors to all the quadrupoles strengths:

```
>> Kvalserr=Kvals+0.2*(rand(length(Kvals),1)-0.5);  
>> FODOerr=setcellstruct(FODO,'PolynomB',QuadIndices,Kvalserr,2);
```

Note that the output of **setcellstruct** is another lattice structure, which here we call 'FODOerr' which has the quadrupole strengths 'Kvalserr'. Now we have both 'FODO' and 'FODOerr' in the Matlab workspace and we can examine them or do calculations for comparison.

Suppose we would like to plot the quadrupole strengths along the lattice. We first need to find the longitudinal, s position of each quadrupole. This may be done with the function **findspos**:

```
>> quadspos = findspos(FODO,QuadIndices);
```

The quadrupole strengths of 'FODO' and 'FODOerr' may now be plotted with

```
>> plot(quadspos,Kvals,'*r',quadspos,Kvalsb,'*b');
```

The plot is shown in Figure 1.

4 Tracking

Once a lattice is defined, electrons may be tracked through it. **ringpass** is the function that does the tracking. An example of its use is as follows:

```
>> nturns=200;  
>> Z01=[.001;0;0;0;0;0];  
>> Z02=[.002;0;0;0;0;0];  
>> Z03=[.003;0;0;0;0;0];
```

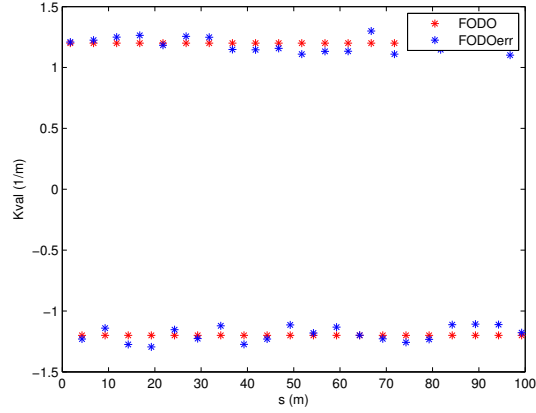


Figure 1: Plot of quadrupole strengths for FODO lattice with and without errors.

```
>> Z1=ringpass(FODO,Z01,nturns);
>> Z2=ringpass(FODO,Z02,nturns);
>> Z3=ringpass(FODO,Z03,nturns);
>> plot(Z1(1,:),Z1(2,:),'.r',Z2(1,:),Z2(2,:),'.b',Z3(1,:),Z3(2,:),'.k')
```

The plot is shown in Figure 2. In this example, we started with one initial

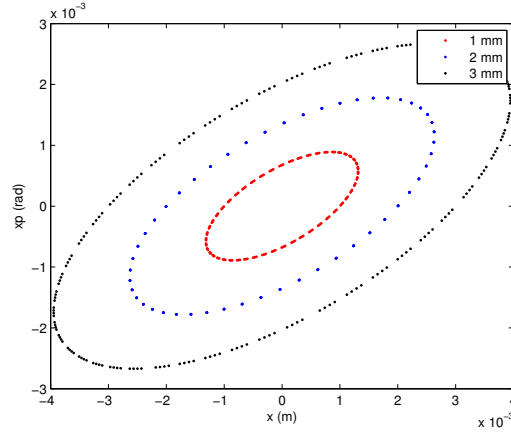


Figure 2: Tracking results in FODO lattice.

condition, and all subsequent turns are returned by ringpass. We may also start with multiple initial conditions:

```
>> Z0=[.001;0;0;0;0;0]*(1:3);
>> Z200=ringpass(FODO,Z0,nturns);
```

Examining the variable Z200, we find

```
>> whos Z200
      Name      Size      Bytes  Class      Attributes

      Z200      6x600      28800  double
```

The structure of this matrix is that the first three columns are the results of tracking for one turn the three initial conditions. The following columns are likewise grouped in threes with the subsequent results of tracking further turns. We may retrieve the same results as we had previously by extracting every third column vector as follows:

```
>> ZZ1=Z200(:,1:3:3*nturns);
>> ZZ2=Z200(:,2:3:3*nturns);
>> ZZ3=Z200(:,3:3:3*nturns);
```

Now the same plot as in Figure 2 results from the plot command

```
>> plot(ZZ1(1,:),ZZ1(2,:),'.r',ZZ2(1,:),ZZ2(2,:),'.b',ZZ3(1,:),ZZ3(2,:),'.k')
```

Another way to process the output of ringpass with multiple initial particles uses the **reshape** command

```
>> ZZ200=reshape(Z200,6,3,nturns);
>> ZZZ1=ZZ200(:,1,:);
>> ZZZ2=ZZ200(:,2,:);
>> ZZZ3=ZZ200(:,3,:);
```

This can be a convenient approach to process the results when many initial conditions are tracked.

5 Computation of beam parameters

Now that particles can be tracked through the lattice, we can use the tracking to understand different properties of the lattice. First, we would like to understand the linear properties such as Twiss paramers, tunes, chromaticities, etc. These can all be calculated with the function `atlinopt`.

```
>> [param,t,c]=atlinopt(FODO,0,1);
```

Note first the three inputs. The first is the FODO lattice we have created. The second argument of 0 says we want to compute the linear optics on energy with $\delta = 0$. The third argument is a list of indices for where we want to compute the different linear optics parameters. The 1 says that we will just compute it at the beginning of the lattice.

Next, there are three outputs, the first is the optics parameters, evaluated at the positions requested. The `t` gives the horizontal and vertical tunes and the `c` gives the chromaticity. Examining first the tunes and chromaticities, we find

```
>> t,c

t =

    0.2867    0.6990

c =

   -1.8248   -1.7404
```

which tells us the tunes are $\nu_x = 0.2867$ and $\nu_y = 0.6990$ and the chromaticities are $\xi_x = -1.8248$, $\xi_y = -1.7404$.

How did AT calculate these quantities? Without digging into the details of `atlinopt`, you could still figure it out, just based on the ability to track with the ringpass function. In fact, AT computes the one turn transfer matrix by tracking several initial conditions and interpolating. The one turn transfer matrix (here we focus on 4x4) is computed with the function `findm44` contained within `atlinopt`. Calling this on the FODO lattice, we find

```
>> findm44(FODO,0)

ans =

    0.3140    1.1838         0         0
   -0.4988    1.3040         0         0
         0         0   -2.2637   -6.2810
         0         0    0.7523    1.6457
```

The 0 as the second argument tells us to compute with $\delta = 0$. We note that the ring is uncoupled, and computing the eigenvalues of submatrices, we derive the tunes reported in `atlinopt` above.

Computing the tunes with varying initial δ allows the computation of the chromaticity.

Now, suppose we would like to change the tunes in our FODO lattice. We know that we should change the quadrupole strengths, but we may not know exactly what values to use.

Here we reach the question of tuning. How do we set the parameters for these quadrupoles in order to correct the tunes? In principle we have the tools that we need. We can set the values of the quadrupoles using the function `setcellstruct` and then recompute the chromaticity with `atlinopt`. But we still don't know what values to actually give the quadrupoles. One could compute the value, or instead use an optimization routine to vary the values until the correct output tunes are achieved. This is the approach followed with the function `atfittune`.

This allows you to vary quadrupole strengths until the desired tune values are reached. It is used as follows

```
>> FODO2=atfittune(FODO,[0.15,0.75], 'QF', 'QD');
```

which would set the tunes to $\nu_x = 0.15$ and $\nu_y = 0.75$ using the quadrupoles QF and QD. When we actually try this, and compute the linear optics, we find the result

```
>> [param2,t2,c2]=atlinopt(FOD02,0,1); t2
>> t2 =

    0.1472    0.7503
```

We see that the requested values have almost been reached, but not exactly. A second call to **atfittune** gives

```
>> FOD02=atfittune(FOD02,[0.15,0.75], 'QF', 'QD');
>> [param2,t2,c2]=atlinopt(FOD02,0,1); t2
>> t2 =

    0.1500    0.7500
```

Giving satisfactory results for the tunes.

Now, in case you have some experience with storage ring dynamics, you will know that these negative chromaticity values will lead to instability and thus our FODO lattice, as is, is not acceptable. To fix this problem, we add sextupoles to our lattice. We define a focusing and defocussing sextupoles (0.1 meter long) as follows:

```
SF=atsextupole('SF',0.1,1);
SD=atsextupole('SD',0.1,-1);
```

Now we want to add these to the lattice at locations where they will be effective. We will put them in the middle of the 0.5 meter drift sections: SF before the QF and SD before the QD. Thus, we need a new drift of 0.2 meters long:

```
p2Dr=atdrift('Dr',0.2);
```

And now we can define our lattice cell with sextupoles as follows

```
FOD0cellSext=[{HalfDr};{Bend};{p2Dr};{SF};{p2Dr};...
{QF};{Dr};{Bend};{p2Dr};{SD};{p2Dr};{QD};{HalfDr}];
FOD0Sext= repmat(cell,20,1);
```

Now that we have added the sextupoles, we need to know how to set them. Back to the chromaticity, we now fix the chromaticities, using the function **atfitchrom**. This function works analogously to **atfittune** except the sextupoles are varied instead of the quadrupoles to fit the chromaticity. Applying this function (twice to get a more precise result):

```
FOD0Sext=atfitchrom(FOD0Sext,[0.5 0.5], 'SF', 'SD');
FOD0Sext=atfitchrom(FOD0Sext,[0.5 0.5], 'SF', 'SD');
```

After changing the tunes and fixing the chromaticities, we find


```
>> [paramS,tS,cS]=atlinopt(FODOSeXT,0,1);tS,cS
tS =
```

```
0.1500    0.7500
```

```
cS =
```

```
0.4996    0.4993
```

You may have noticed that we bypassed the initial output of **atlinopt**, the parameters. This contains linear optics parameters that vary around the ring. These are the Twiss parameters, dispersions, phase advance, and coupling parameters. To compute these around the ring, we need to give the indices for the reference points. To compute them at all lattice elements, we call

```
>> [paramAll,t2,c2]=atlinopt(FODOSeXT,0,1:length(FODOSeXT)+1);
```

Examining 'paramAll', we find

```
>> paramAll
paramAll =
```

```
1x261 struct array with fields:
```

```
ElemIndex
SPos
ClosedOrbit
Dispersion
M44
gamma
C
A
B
beta
alpha
mu
```

'ElemIndex' is the set of indices where the optics parameters are defined. 'SPos' is the set of s positions. 'ClosedOrbit' is the x, x', y, y' of the closed orbit at each element location. 'Dispersion' gives the horizontal and vertical dispersion and dispersion derivative. 'M44' is the local 4×4 transfer matrix. 'gamma', 'C', 'A', and 'B' are coupling parameters[4]. 'beta' gives the horizontal and vertical beta functions. 'alpha' gives the Twiss parameters $\alpha_{x,y}$ and 'mu' gives the phase advances (times 2π).

Let us use these results to plot the beta functions around the ring.

```
>> beta=cat(1,paramAll.beta);
```

```

>> betax=beta(:,1);
>> betay=beta(:,2);
>> spos=cat(1,paramAll.SPos);
>> figure
>> plot(spos,betax,'-r',spos,betay,'-b');
>> xlabel('s (m)');
>> ylabel('beta (m)');
>> legend('betax','betay');

```

The result is shown in Figure 3.

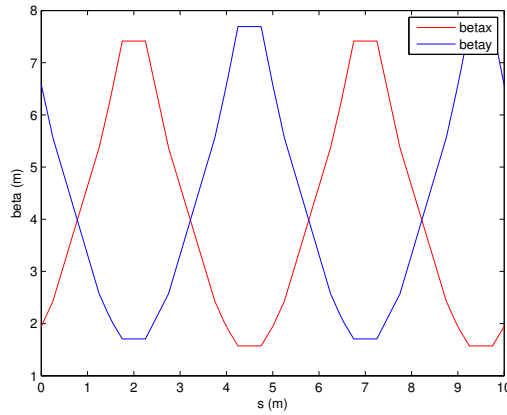


Figure 3: Plot of FODO lattice beta functions.

We may also plot the lattice parameters using a dedicated function called **atplot** with the command

```
atplot([FODOcellSext;FODOcellSext]);
```

The result is shown in Figure 4. Note that the magnets are displayed below the function, giving a convenient visualization. Also note that the lattice functions are smoother than those we see in Figure 3. They have been computed at more positions, but slicing the magnets in the atplot function.

5.1 Beam sizes

The parameters computed thus far use only the tracking through the lattice, with no radiation effects. In reality, for electrons, we know that there are radiation effects which cause a damping and diffusion and determine equilibrium emittances and beam sizes. This is computed in AT using the Ohmi envelope formalism. A simple interface to this code is provided via the function **atx** which gives the linear optics parameters we have just computed, and also the beam sizes.

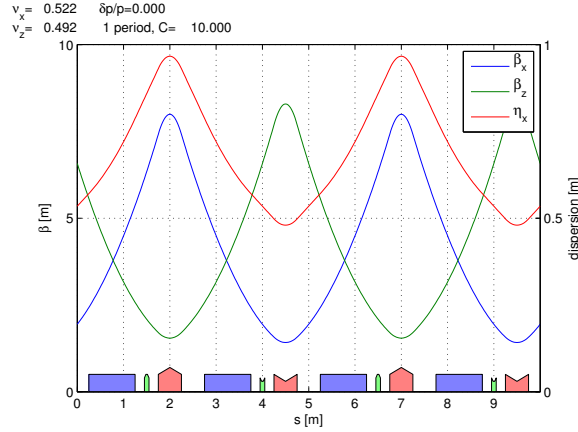


Figure 4: Plot result of **atplot** on the FODO lattice showing beta functions and dispersion, smoothed via element slicing and also including magnet representation.

In order to use **atx**, we first need to make sure the beam is stable longitudinally as well, requiring us to add an RF cavity to our FODO lattice. This is done with the command

```
>> RFC=atrfcavity('RFCav');
```

and the cavity is added to the lattice

```
FODOSextRF=[{RFC};FODOSext];
```

Now, we need to set the values of the RF cavity. This can be done with the function **atsetcavity** as follows

```
FODOSextRF=atsetcavity(FODOSextRF,5e5,0,100);
```

which says that the RF cavity has a voltage of 5MV and harmonic number of 100.

We may now call the function **atx** as follows

```
>> [BEAMDATA,PARAMS]=atx(FODOSextRF);
```

with the results

```
l1: 100.0000
alpha: 0.0419
fractunes: [0.2199 0.9178]
fulltunes: [5.2199 4.9178]
nuh: 5.2199
nuv: 4.9178
chromaticity: [0.5000 0.5083]
```

```
dampingtime: [0.0536 0.0478 0.0228]
  espread: 3.3112e-04
  blength: 0.0121
modemittance: [3.3655e-08 7.3019e-37 3.9995e-06]
  energy: 1.0000e+09
  fs: 5.4812e+04
  eloss: 1.3901e+04
synchrophase: 0.0278
momcompact: 0.0419
```

We see that our FODO lattice has an emittance of 33.66 nm, an energy spread of 3.3×10^{-4} and a bunch length of 12.1 mm.

References

- [1] A. Terebilo *Accelerator Toolbox for Matlab*, SLAC-PUB 8732 (May 2001)
- [2] A. Terebilo, *Accelerator Modeling with Matlab Accelerator Toolbox*, PAC 2001 Proceedings
- [3] B. Nash et. al. *New Functionality for Beam Dynamics in Accelerator Toolbox*, IPAC 2015
- [4] D. Sagan, D. Rubin, *Linear Analysis of Coupled Lattices*, Phys. Rev. Special Topics - Accelerators and Beams, Vol 2,(1999)