Training for Accelerator Toolbox – L. R. Carver, S. Liuzzo

Some useful basics:

To see required variables for any function (matlab or AT), simply type:

```
>> help function
```

Or

```
>> doc function
```

Element definition:

atdrift: creates drift element
atsbend: creates dipole element
atquadrupole: creates quadrupole element
atsextupole: creates sextupole element
atmarker: creates a marker element
atrfcavity: creates a cavity element

Pass Methods:

Each element that is created needs to have defined a *passmethod.* i.e. you need to tell AT what this element is and how to treat it. The main pass methods you will encounter in the beginning are

DriftPass: treat this element as a drift length, the element will of course require a defined length.
IdentityPass: do absolutely nothing, these elements normally have length 0 (e.g. markers)
StrMPoleSymplectic4Pass: Straight multipole with given polynomial coefficients (A = skew coefficients, B = normal coefficients, dipole→quadrupole→etc, 1/m→1/m^2→etc).
BndMPoleSymplectic4Pass: Multipoles defined as above but on a curved trajectory.
CavityPass: treat this element as an RF cavity.

Modifying a parameter of an existing element

You define a quadrupole e.g.:

```
>> qf = atquadrupole('QF', 1, 2, 'PassMethod', 'StrMPoleSymplectic4Pass');
```

To change the length of the quadrupole, you can do so by

```
>> qf.Length = 10;
>> qf.PolynomB = [0 20];
```

<u>Constructing a lattice:</u>

You have your individual elements and you want to put them into a cell array that AT will be able to interpret as lattice. If you have 3 elements, “QF”, “QD”, ”DL” and you want to construct a lattice, you can do this by creating a cell array:

```
>> lattice = {QF ; DL ; QD;} ;
```

If you wanted to repeat this lattice, you can do so as follows:

```
>> lattice2 = [lattice ; lattice];  %Note the square brackets instead of the curly brackets
```

If you wanted to repeat a matrix or cell array *n* times, you can use the MatLab function ***repmat***.

```
>> lattice2 = repmat(lattice, n, 1);
```

To reverse an array:

```
>>revLattice = lattice(end:-1:1) %go from end to 1 in steps of -1.
```

Use reverse with care as some magnets can not be tracked backwards due to different entrance and exit fields.

<u>Plotting a lattice:</u>

There is a predefined function which produces beautiful plots of your lattice. It is called ***atplot***. When using atplot you might notice that it is called beta_z instead of beta_y. It is by name only.

In **atplot** you can also give initial twiss parameters to see how it changes the optics. Typically, if you only want to change one parameter (for example you want to force the beta_x at the entrance to the cell), you should use **atlinopt** to obtain all the initial twiss parameters and then modify the one you want to change. Alternatively, you can do it completely by hand, you just have to give the full set of parameters in the same structure that is provided by atlinopt.

<u>Calculating the twiss parameters of a lattice</u>

Here: ***atlinopt*** (at linear optics) is the function to use. Best summarised as the following command:

```
>> [twiss_output, ~, ~] = atlinopt(lattice, 0, REFPTS); %The two tildes mean that you are asking
```

atlinopt to calculate the additional parameters (dispersion). However the results are then added into twiss_output so the extra returns from atlinopt are not needed. Try it without the brackets and tildes and compare the difference.

REFPTS is an important concept that is used for many different functions in AT. It is either a Boolean array (1s and 0s of the same length of the lattice), or it is an array of indices. It is basically saying, please compute the linear optics only at the positions that I specify.

There are very handy ways of getting the indexes of many elements within a lattice so you can easily automate repetitive changes. For example, if you have a full lattice of many elements and you want to get the indexes of all the quadrupoles in your lattice you can use ***atgetcells*** like so:

```
>> inds = atgetcells(lattice, ‘Class’, ’Quadrupole’); %returns all quadrupoles
```

Or you can use the MatLab wildcard character:

```
>> inds = atgetcells(lattice, ‘FamName’, ‘QF\w*’); %returns all elements that start with QF
```

Similarly, if you wanted to extract a parameter from each element you can use ***atgetfieldvalues***

>> lengths = atgetfieldvalues(lattice, ‘Length’) %can fail if you have elements with length 0, alternatively there is a function called ***findspos***

The two functions can be used together, for example, to get the strength of all the quadrupoles in your lattice:

>> ks = atgetfieldvalues(lattice, atgetcells(lattice, ‘Class’, ’Quadrupole’), ‘PolynomB’, {1,2}); %the curly brackets indicate which elements of the output to take. Try it without

Super important feature (bug) in AT that needs to be considered

If you want to calculate the betatronic phase advance (mu) through your lattice, you **MUST** include all of the intermediate elements, otherwise the phase advance that is calculated is wrong. For example

```
>> twiss_output_1 = atlinopt(lattice,0, [length(lattice)]);
```
and
```
>> twiss_output_2 = atlinopt(lattice, 0, 1:length(lattice));
```

Give very different results for the phase advance over a longer lattice. Only the latter is correct! This is not so important for FODO lattices, but is very important when you start to have longer lattices with more elements and cells.

So to summarise ***atlinopt***, the best way of using it is as follows:

```
>> [output, ~, ~] = atlinopt(lattice, 0, 1:length(lattice))
```

To easily extract one parameter along the length of the lattice you can use ***arrayfun*** as follows:
```
>> beta_x = arrayfun(@(a).a.beta(1), output);
```

<u>Performing matching in AT:</u>

Main functions:

atVariableBuilder: define an array of all of the things in the lattice you want to vary in the match

```
>> var1 = atvariablebuilder(lattice, ‘QF1’, {‘PolynomB’, {1, 2}}); %if multiple elements called QF1 then ALL
    are varied in the matching. It is treated as one variable.
>> var2 = …;
>> var = [var1 var2];
```

atlinconstraint: define an array of all of the constraints you want to force for your match

```
>> c1 = atlinconstraint(elementIndex, {{‘Dispersion’,{1}}}, [minVal], [maxVal], [1/weight]);
>> c2 = …;
>> cons = [c1 c2];
```

atmatch: the main function that does the matching

```
>> matchedLattice = atmatch(lattice, var, cons, 1e-15, 1000, 3, @fminsearch, twissin); %atmatch(lattice,
    variables, constraints, tolerance, calls, verbosity, algorithm, twissin (can be excluded if periodicity is
    desired)).
```

For some more information on matching, see this document written by Simone (note it uses some quite advanced functionality but is solving some difficult problems):

http://atcollab.sourceforge.net/atdocs/manuals/user/ATMATCH.pdf

<u>Some other useful functions:</u>

atx: uses the Ohmi envelope to compute the natural emittance of the lattice (needs energy definition and RF cavities).
ringpara: Compute some lattice parameters based on analytical formula.
ringpass: Can be used to perform tracking of an array of particles over *n* turns. If aperture elements are included, then particles are automatically excluded and you can even extract the turn and element on which the particle was lost. However it cannot give you particle positions on any intermediate elements, only at the end of each turn.
linepass: Can be used to track an array of particles through one transfer line, i.e. from the start to the end of a given lattice. It can be set with REFPTS to provide particle positions at intermediate elements.
atgeometry: Will give the coordinates for the physical geometry of the ring (horizontal plane only). Useful when considering the size and shape of an accelerator with real world constraints.
atdynap: used to compute the dynamic aperture.
atsetcavity: used to set the parameters of the rf cavity (voltage, energy, harmonic number).
atbeam: create matched beam distributions into a given lattice
atsigma: create the sigma matrix, which can be given to atbeam to create a beam matched into some given lattice parameters
atfittune: varies two quadrupole families in order to match the tunes of the lattice to what you want
atfitchrom: varies two sextupole families in order to match the chromaticity of the lattice to what you want
atradon: changes relevant passmethods to switch on RF cavities and synchrotron radiation effects.
atsbreak: breaks the lattice at a given s location and inserts a marker with name 'xyzt'. Similarly one can use ***atinsertelems*** to directly input a given element (but watch out for lengths).

<u>Some old documentation with more examples:</u>

http://atcollab.sourceforge.net/docs.html
which contains two useful documents

1. http://atcollab.sourceforge.net/atdocs/manuals/user/ATMATCH.pdf
2. http://atcollab.sourceforge.net/atdocs/manuals/user/AT2Primer.pdf

They contain much of the same material as shown here but have a few other functions and examples.
You can also look at the documentation on atcollab github page:
https://github.com/atcollab/atdoc
which also contains lots of papers written that give some other examples of AT usage.