ESRF | The European Synchrotron

PyAT Introduction

AT Workshop - 03/10/23

Lee Carver

INFORMATION

PyAT Website:
https://atcollab.github.io/at/p/index.html

Detailed instructions and plenty of interesting information can be found there.

- **This PyAT tutorial will be made using jupyter-notebooks.**
  - Python scripts with all the functions will be made available.
- **Please consider this tutorial as informal and a chance to engage and have your questions answered about PyAT.**
- **We have 3 hours available, but I have ~1h30 of material.**
  - We can decide as a group how to spend the remaining time.
- **You do not have to engage if you don't want to - there are no obligations**
  - If you do want to follow along and run the examples during the session, please try to have PyAT and jupyter installed and running on your computer.
  - Installation instructions are in the following slides.

INSTALLATION FOR BEGINNERS

- You must be running python3.
- If you are administrator of your computer:

    pip install accelerator-toolbox

- Done! Pip handles all dependencies.

INSTALLATION FOR DEVELOPERS

- **Needed (see website for full list of requirements and dependencies):**
  - python
    - numpy , scipy, setuptools, matplotlib (optional)
  - git
  - Microsoft Build Tools (windows only): https://visualstudio.microsoft.com/visual-cpp-build-tools/
    - Download installer, click ‘modify’ and install ‘Desktop development with C++’, then add to path.
  - mpi4py (for parallelised collective effects)

INSTALLATION ON LINUX

- **If you have no intention of doing any development:**
  - pip install accelerator-toolbox

- **If you would like to do some development:**
  - git clone https://github.com/atcollab/at
  - cd at
  - pip install -e .

- **If you want to install with mpi4py for collective effects simulations**
  - pip install mpi4py
  - git clone https://github.com/atcollab/at
  - cd at
  - pip install - -config-settings mpi=1 ".[mpi]"

- **Note: there is no space between the dashes.**

- **If you do not have root privileges, follow the instructions "Installation (All Platforms)" on the website in order to create a virtual environment.**

INSTALLATION (ALL PLATFORMS)

- python3 -m venv venv_name

  source venv_name/bin/activate (or venv_name\Scripts\activate on Windows)

  pip install --upgrade pip

  git clone https://github.com/atcollab/at

  cd at

  pip install -e .

OVERVIEW

- **The core of accelerator toolbox is the PassMethod.**
  - A PassMethod takes a set of input particle coordinates and modifies them.
  - All PassMethods written in C (although python PassMethods are now possible).

- **Python and MatLab AT are actually both environments that are wrappers for the PassMethods.**
  - Both Python and Matlab share the same PassMethods.

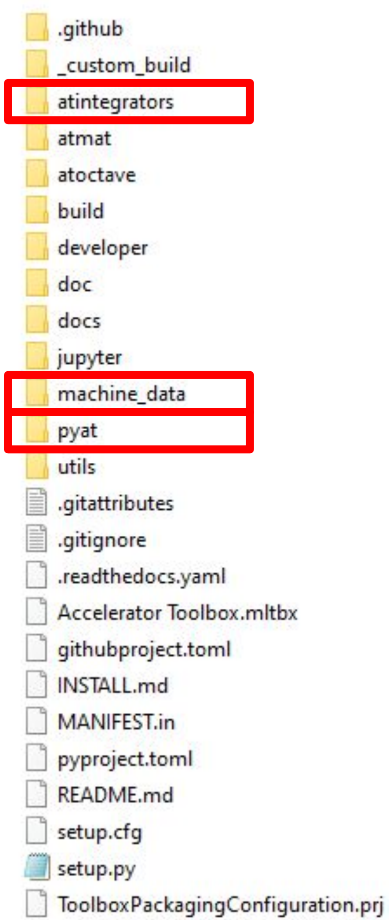
- **There is a different PassMethod for each machine element: (for example, DriftPass, CavityPass).**

- **Each PassMethod is initialised with a needed set of parameters (e.g. for a Quadrupole it needs a length and a strength, for a drift it needs only a length). Then you can pass to it an array of particles of shape (6,N), and it will perform the necessary transformation.**

- **All of the AT computations are done with tracking. (Unless specifically mentioned).**

- **When you install PyAT, the PassMethods are compiled and stored in 'at/build/dist/at/integrators'. The source of each PassMethod is in 'at/atintegrators'**

INSTALLATION



- **atintegrators contains the source files of the PassMethods.**

machine_data contains a few example lattices that can be imported.

- **pyat contains all of the functions for PyAT.**

A SIMPLE PASSMETHOD EXAMPLE

- Lets look at DriftPass.c to get a feeling for how it works.
- In conclusion, 6d array of particles is squashed into (x0,xp0,y0,yp0,delta0,ct0, x1,xp1,...), and this is given to PassMethod as r_in.
- Arithmetic pointers are used to loop through each particle. (line 30 and 31). Each particle is iterated in the function ATdrift6.

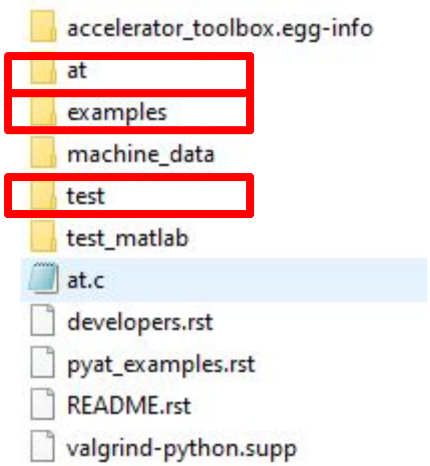
atintegrators/DriftPass.c

```c
15
16 void DriftPass(double *r_in, double le,
17            const double *T1, const double *T2,
18            const double *R1, const double *R2,
19            double *RApertures, double *EApertures,
20            int num_particles)
21 /* le - physical length
22    r_in - 6-by-N matrix of initial conditions reshaped into
23    1-d array of 6*N elements
24 */
25 {
26   double *r6;
27   int c;
28
29   #pragma omp parallel for if (num_particles > OMP_PARTICLE_THRESHOLD*10) default(shared) shared(r_in,num_particles) private(c,r6)
30   for (c = 0; c<num_particles; c++) { /*Loop over particles  */
31     r6 = r_in+c*6;
32     if(!atIsNaN(r6[0])) {
33       /*  misalignment at entrance  */
34       if (T1) ATaddvv(r6, T1);
35       if (R1) ATmultmv(r6, R1);
36       /* Check physical apertures at the entrance of the magnet */
37       if (RApertures) checkiflostRectangularAp(r6,RApertures);
38       if (EApertures) checkiflostEllipticalAp(r6,EApertures);
39       ATdrift6(r6, le);
40       /* Check physical apertures at the exit of the magnet */
41       if (RApertures) checkiflostRectangularAp(r6,RApertures);
42       if (EApertures) checkiflostEllipticalAp(r6,EApertures);
43       /* Misalignment at exit */
44       if (R2) ATmultmv(r6, R2);
45       if (T2) ATaddvv(r6, T2);
46     }
47   }
48 }
49
```

atintegrators/atlalib.c

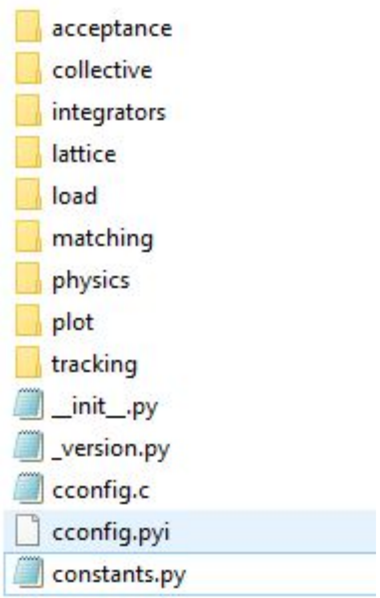
```c
static void ATdrift6(double* r, double L)
/*   Input parameter L is the physical length
     1/(1+delta) normalization is done internally
*/
{    double p_norm = 1/(1+r[4]);
     double NormL  = L*p_norm;
     r[0]+= NormL*r[1];
     r[2]+= NormL*r[3];
     r[5]+= NormL*p_norm*(r[1]*r[1]+r[3]*r[3])/2;
}
```

AT/PYAT



- at contains all of the functions and classes for PyAT.
- examples contains a few examples, (only CollectiveEffects for now).
- test is run after each commit to ensure compatibility.

AT/PYAT/AT



- This now contains all of the python functions for all of the different features of PyAT.
- We will see how to use PyAT, and the help functions will be pointing back to files found in this directory.