NSLS-II DEDICATED PYTHON TOOLS FOR SIMULATION AND ANALYSIS*

J. Choi†, BNL, Upton, NY 11973, USA

Abstract

Python is a high-level interpreted programming language. Despite its slow benchmarks, because of its fast coding cycle and dynamic property, the users are increasing fast in all areas. Also, because it does not need special care for the memory management, both professional and non-professional programmers can easily make bug-free code just by concentrating on logics. Furthermore, fast increasing libraries are making the language more and more useful. With these advantages, we developed python tools which simulate and analyze the particle accelerator with some parts being dedicated to NSLS-II operation.

INTRODUCTION

Since we chose Python [1] as the programming language of NSLS-II high level application [2], many high level applications for machine operation and study were developed with Python in the collaboration between control and physics group. One of the list was developing Python interface for the TRACY-2 [3] which was adopted as the simulation code for the NSLS-II virtual machine [4]. By wrapping TRACY-2 with SIP [5], a convenient python library was developed named PyTracy [6] and it was proved very useful calling TRACY-2 functions from python code.

However, as we develope more and more high level softwares using PyTracy, the strict limitations begin to be showed clearly. That is, we wanted full implementation of objective property of the language for the lattice elements but the interface points only the whole element and could not directly access a property of an element nor modify it individually. Therefore, parallel to the wrapping of TRACY-2, we made pure Python code which makes every element in the lattice as Python object and the lattice is a Python list.

This code is named as N2Track because some parts are dedicated only to NSLS-II storage ring. For example, it has information about EPICS PVs for magnet power supplies and conversion table between hardware and physics units. Thereby, it can construct live lattice by accessing the control system.

MODULES

As usual, N2Track consists of various modules depending on functionalities. The modules having functions directly called by users are as follows.

Globvalrec Using this module, one can be set or get the global parameters. The examples are energy and flags which decide whether the radiation and RF will be included in the tracking. Also the resulting parameters are kept whenever calculated, such as tunes, chromaticities, one-turn map, twiss values at origin, emittance, radiation loss and others.

latticeTools Most NSLS-II specific functions are in this module. Using this module, one can generate specific NSLS-II lattice and find indices for specific elements. Also, it has functions giving the family properties even based on magnet design model.

tracyIn and elegantIn Actually, because official lattice is distributed in the ELEGANT [7] input format, the NSLS-II lattice making function is using the function in the elegantIn module. Using these modules, one can read both the TRACY-2 and ELEGANT format. However, only part of ELEGANT element types and attributes are recognized.

Family Family information of the elements.

Element Using this module, one can access every property in the element to get its value or modify it. By doing modification after the lattice is read, N2Track provides much more flexibility than conventional codes.

util This module has interface functions for physical calculation. The typical ones are calculating closed orbit and twiss parameters. Then save all results in the appropriate places.

miaTools This is an auxiliary module helping analyzing the turn-by-turn data using model indecent analysis (MIA) [8].

archiveTools This is dedicated module to NSLS-II which are helping analyzing the data from the NSLS-II archive system.

BASIC USES

In this section, some basic uses common to all simulation codes, such as reading input file, calculate twiss values, are shown.

Construct NSLS-II lattice and generate the element list. Using tracyIn or elegantIn module, the lattice files can be directly read in. However, N2Track provides higher level input method for NSLS-II storage ring where the lattice type, whether it is a bare lattice or DWs are closed, is accepted as an argument. Once the lattice is constructed, it returns Globvalrec object and Python lists for families and elements.

* Work supported by DOE contract No: DE-SC0012704
† jchoi@bnl.gov

```
Globval,Fams,Lattice=latticeTools.genNSLS2Lattice()
# default:  Bare lattice
```

The above snippet is equivalent to the following.

```
# create globvalrec object
Globval = globvalrec.globvalrec()
# create Family and Element list
Fams,Lattice=
    elegantIn.constructBL(Globval, "nsls2sr.lte")
# import the unit conversion information
util.setConversion(Globval,Lattice)
# if several elements with same kind are found in one
place, combine them into one element
Fam,Lattice = util.simplify(Fam,Lattice)
# import the magnet design model information
util.registerModel(Lattice)
```

Get the closed orbit at BPM positions and plot them.

```
import matplotlib.pyplot as plt
# find the BPM positions as indices in the lattice
mons=latticeTools.getBPMindex(Lattice)
# calculate the closed orbit by iteration
util.getCOD(Globval, Lattice)
# BPM positions
s_bpm=[Lattice[m].S for m in mons]
# horizontal closed orbit
cod_x = [Lattice[m].BeamPos[0,0] for m in mons]
# vertical closed orbit cod_y =
[Lattice[m].BeamPos[0,2] for m in mons]
plt.plot(s_bpm, cod_x) plt.plot(s_bpm, cod_y)
plt.show()
```

Calculate the Twiss parameters and print the tunes.

```
util.getTwiss(Globval, Lattice)
print(Globval.TotalTune)
```

Read the Twiss parameters from the lattice elements and save them as Pyhon lists.

```
betax = [ele.Beta[0,0] for ele in Lattice]
betay = [ele.Beta[0,1] for ele in Lattice]
alphax = [ele.Alpha[0,0] for ele in Lattice]
alphay = [ele.Alpha[0,1] for ele in Lattice]
nux = [ele.Nu[0] for ele in Lattice]
nuy = [ele.Nu[1] for ele in Lattice]
```

UNIQUE FEATURES

As a Python application, we can think two major advantages. First, because each lattice element is a Python object and the lattice is a Python list, we can modify them dynamically. The other advantage is that we can use Pyhon libraries, such as NumPy [9] and SciPy [10], seamlessly at any place.

In this section, some examples are shown which, when using the conventional application, could be quite complicated or sometimes cannot be implemented.

Obtain the interpolated Twiss.

Twiss values are calculated at each element, using the interpolation methods provided by SciPy, we can plot the Twiss values as smooth functions. This is possible because the derivatives are also in Twiss parameters too and this tool is included in the latticeTool module.

```
# Twiss is already calculated using util.getTwiss
bx,by,ex = latticeTools.interpolatedTwiss(Lattice)
# slice the ring into 1000 for the good smoothness
xs = np.linspace(0,globval.C,1000)
plt.plot(xs,bx(xs))
plt.plot(xs,by(xs))
# Eta is too small in the same scale
plt.plot(xs,10*ex(xs))
plt.show()
```

Add a new element into the existing lattice.

Create the pinger element and insert it to the given position of the lattice.

```
# define a pinger family with length of 30 cm
ping_family =\
   Family.mpole("PINGER",len(famobjs),Family.PING,0.3)
# give a name for a new pinger element
ping_family.name = 'HPNG1C22A'
# place the pinger at 176.718m
ping_element = \
   Element.createOne(576.8718,**(props(ping_family)))
Family.append(ping_family)
# sort the lattice so that the new element is properly
# positioned
Lattice= \
   sorted(Lattice, key=lambda element:  element.S)
```

Find an element and modify its propety.

Find a quadrupole with the given name and increase the focusing strength.

```
# find a quadrupole with the name
quad=[ele for ele in Lattice if ele.Name=="QH1G2C28A"]
# increase the quadrupole strength by 10^-4
Lattice[quad[0]].setDBpar(2,1.e-4)
```

Track a particle and record the beam positions at BPMs.

When a particle is tracked, all the 6d coordinate values are saved at the beginning and end of each element with the name of BeamPos.

```
# find the BPM positions as indices in the lattice
```

```
mons = latticeTools.getBPMindex(Lattice)
# initialize the buffers
allx = numpy.empty((NTURN, len(mons)),'d')
ally = numpy.empty((NTURN, len(mons)),'d')
# initialize the particle coordinate with x0=y0=1mm
x = numpy.array([0.001,0,0.001,0,0,0],'d')
for n in range(NTURN):
   util.vecOnePass(Globval, Lattice, x)
   allx[n,:]=np.array(\
             [Lattice[m].BeamPos[0,0] for m in mons])
   ally[n,:]=np.array(\
             [Lattice[m].BeamPos[0,2] for m in mons])
```

Calculate the transfer matrix by tracking the particle. The transfer matrix can be obtained by tracking using the linear automatic differentiation [11].

```
# initialize the numpy array for the transfer matrix
mat = numpy.zeros((6,7),'d'); mat[:,1:]  =
numpy.eye(6,dtype='d')
# track the particle with class function mPass
for n in range(i0,i1):
     Lattice[n].mPass(Globval, mat)
# get transfer matrix from element i0 to element i1-1
transf_mat=mat[:,1:]
```

Generate the live lattice based on the magnet currents. As an example of NSLS-II dedicated functionalities, the live lattice is constructed from the magnet currents.

```
# read the magnet currents from control system or files
# and make a Python dictionary named pv_val_ whose keys
# are PVs and values are corresponding currents
util.readDict(Lattice,pv_val_Dict)
# apply the set-point currents to the quadrupoles
# and sextupoles using unit-conversions
util.loadCurrent(Globval,Lattice,source='SP',target='QS')
```

SUMMARY

We showed many conveniences and flexibilities of N2Track as a Python application. Many other unique functionalities other than those introduced are also available and new convenient functions also can be easily added. With all these advantages, however, the slow performance is a serious drawback especially when we need track particles many turns. Because few efforts were made for the performance improvement yet, there is a room to improve it by, for example, python vectorization. However, slow performance of Python is very well proved and the only known solutions are using C-compiled libraries and/or parallel programming.

Because adopting C-compiled libraries means giving up pure python and many flexibilities together, we are looking for an alternative solution and found the relatively new language called julia [12] has all the conveniences and desired performance. We also found that running julia in parallel environment is very native. Therefore, after some benchmark tests, if the performance of julia is satisfied, moving the julia will be seriously considered.

REFERENCES

[1] Python, https://www.python.org/

[2] L. Yang, J. Choi, Y. Hidaka, G. Shen and G. Wang, "Development Progress of NSLS-II Accelerator Physics High Level Applications," Proceedings of IPAC2012, New Orleans, Louisiana, USA, p. 4005, 2012

[3] J. Bengtsson, "TRACY-2 User's Manual," SLS Internal Document, 1997; M. Böge, "Update on TRACY-2 Documentation," SLS Internal Note, SLS-TME-TA-1999-0002, 1999

[4] G. Shen, L. Yang, Y. Li, "Virtual Accelerator at NSLS-II Project," Proceedings of ICALEPCS2013, San Francisco, CA, USA, p. 890, 2013

[5] SIP, https://pypi.python.org/pypi/SIP

[6] L. Yang, J. Choi, Y. Hidaka, Y. Li, G. Shen, and G. Wang, "The Design of NSLS-II High-level Physics and Applications," Proceedings of ICALEPCS2013, San Francisco, CA, USA, p. 890, 2013

[7] M. Borland, "elegant: A Flexible SDDS-compliant Code for Accelerator Simulation," APS LS-287, September 2000

[8] C. Wang, V. Sajaev, and C. Y. Yao "Phase advance and β function measurements using model-independent analysis," *Phys. Rev. ST Accel. Beams*, vol. 6, p. 104001, Oct. 2003.

[9] NumPy, http://www.numpy.org/

[10] SciPy, https://www.scipy.org/

[11] M. Berz "Differential Algebraic Description of Beam Dynamics to Very High Orders," *Part. Accel.*, vol. 24, pp. 109-24, 1989

[12] julia, http://julialang.org/