A PYTHON TRACKING CODE AND GUI FOR CONTROL ROOM OPERATIONS

M. T. Heron[#], J. Rowland, Diamond Light Source Ltd, Oxfordshire, U.K.

Abstract

Considerable use has been made in recent years of accelerator physics modelling and on-line tools under MATLAB. This work has demonstrated the benefits of operating in a rich integrated environment and given good portability across projects and operating systems. As a possible alternative to MATLAB, Diamond has been evaluating options based on Python. Python together with the Numpy libraries and Qt Graphics provides an environment which offers a lot of the functionality of MATLAB. This paper presents these developments, which include a tracking code, a symplectic integrator, and code to calculate the Twiss functions and response matrix, together with a GUI interface.

INTRODUCTION

Diamond, a third generation 3GeV synchrotron light source, commenced operation in January 2007 [1]. The storage ring is based on a 24-cell double bend achromatic lattice of 561m circumference. The photon output is optimised for high brightness from undulators and high flux from multi-pole wigglers. The current operational state includes twenty photon beamlines, with a further twelve beamlines in design or construction.

From initial commissioning, Diamond has used the accelerator physics application MATLAB Middle Layer (MML) [2] as part of on-line physics studies. Whilst MML has been used very successfully at Diamond and at a number of other light source projects, it requires MATLAB to operate. This brings with it the need for a MATLAB licence for each MATLAB environment, or the overhead of building and deploying MATLAB compiled applications. Historically, up to and including version R14, we also experienced poor stability of MATLAB on a Linux platform.

This work explores the practicalities of realising similar functionality to MML in an open source, cross platform environment based on the scripting language Python [3].

PYTHON AS AN ALTERNATIVE TO MATLAB

Developing scientific applications in Python has been aided by work on a number of support packages. These components, when brought together, provide a rich development environment. They are:

- Numpy [4]: A fast, compact, multi-dimensional array toolbox. Whilst Python has heterogeneous Lists of objects, Numpy extends these to homogeneous Arrays and Matrices which support element-wise and linear algebra mathematical operations.
- Matplotlib [5]: A plotting library for Python and Numpy which provides an object-oriented API allowing plots to be embedded into applications, and also provides a procedural interface based on a state machine designed to resemble closely that of MATLAB plotting.
- PyQt [6]: A Python binding for the Qt cross platform GUI environment.

Within Diamond, Python together with one or more of the above components has been used for the development of a number of self-contained control system applications [7]. These components were also used in the development of the particle tracking code Serpentine [8] and as the basis of Spyder [9], a Python development environment providing MATLAB-like features.

REQUIREMENTS

The requirements for the application are:

- MML-like functionality: Built-in physics applications, scripting, and a command line interface to work with data interactively.
- Object-orientated design: Application design should be object-orientated, but should retain the ability to operate with data and to have common objects and data between in-built applications, scripts and the command line interface.
- Control System interface: A control system interface that abstracts on-line data from the underlying control system, and is independent of any specific control system toolkit.
- On-line model: Connection to an on-line model.
- Ease of deployment: Minimal dependencies and support for cross-platform deployment (Linux and Windows).
- Based on Python: Utilising Python and the related components identified above.
- Built-in plotting: Support for common visualization of the data, together with flexibility in user-generated plots.
- Ease of physics application development: provided by the use of clean interfaces between components.

THE MAPP APPLICATION

The application, called MAPP, consists of a user interface, together with interfaces to multiple data sources including the control system and a model server.

The user interface data flow, Fig. 1, is based on two objects, the Name Object and Get Put Object. The former defines the information for each family of devices including the number of members, conversions to and

[#] mark.heron@diamond.ac.uk

from engineering units, units, spatial information, and mapping to control system parameters, e.g. Quad family. For each family this object is an extension of a Base Name Object class. The Get Put Object defines the methods to get data from and put data to the model server, the control system and files, or to get data randomised and normalised to unity or as zeros. For the Get Put Object a default data source and destination are defined – the machine, the model server, files etc – to or from which Get and Put methods are directed. Alternatively, the source or destination can be specified by calling a source- or destination-specific method, e.g. GetModel. All calls are subject to specified conversion to or from engineering or physics units.

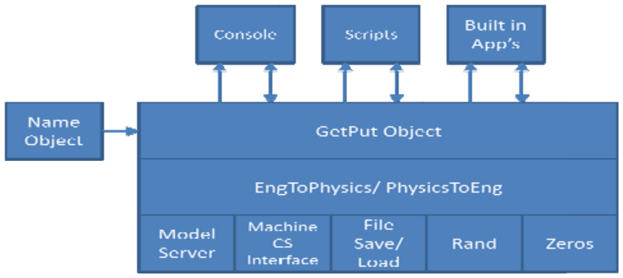

Figure 1: Data flow.

The user interface, Fig 2, is a Qt 4 application providing pre-defined plotting of data against longitudinal position, a group of labels and text boxes for displaying parameters, a group of buttons for invoking in-built applications, two sets of radio buttons for selecting data source and destination, and a command window with a Python interpreter. The plotting was originally realised in Matplotlib, but this proved too slow for real-time updates due to anti-aliased software rendering into a buffer followed by image transfer (blitting), and was re-implemented using the Python QWT [10] module which gives acceptable real-time performance.

The control system interface is realised using the Cothread [7] interface to EPICS Channel Access. The interface to the model server provides methods to get and put parameters to or from the model server by family name, cell and instance. For performance reasons it also provides the option to get and put all values for a family in one operation.

Initialisation creates Name Objects, e.g. HBPMNo for all specified families of devices, and then creates Get Put Objects, e.g. HBPMGetPut from each Name Object, and Numpy Arrays e.g. HBPM, for the data values from each Get Put Object. All actions are then event-driven, based on running a script, on running an in-built application or on commands entered in the command window, and act through these objects and data. A plot object is created inheriting configuration from Get Put Object and thereby defining the plot data size, the range, the units, etc. for plotting methods which operate on Arrays of data. The command window includes a Python interpreter, running in the Qt thread, which provides access to the global variable space of the application and provides code completion, including the application-specific objects.

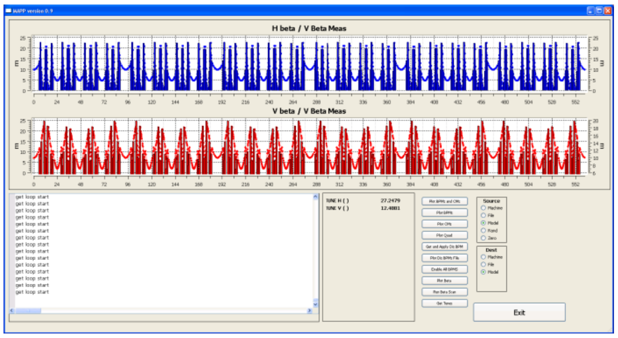

Figure 2: User Interface.

Example Applications

An example function to get the current horizontal and vertical beam positions from the default source and plot them is shown in Fig. 3. The explicit copy of the Array of BPM values to the global Arrays of BPM values makes the latest values available to any in-built application, the command window or a script.

```
def plotBPM():

    HBPM = GHBPMGetPut.get()
    VBPM = GVBPMGetPut.get()
    p = Plotting(GHBPMGetPut,None,
        GVBPMGetPut, None)
    p.plotHandV(HBPM,None,VBPM,None)
    GHBPM[:] = HBPM
    GVBPM[:] = VBPM
```

Figure 3: Script to plot horizontal and vertical BPMs.

TRACKING CODE

The Python tracking code in the model server is similar to the MATLAB and C tracking code AT [11] in MML. It was originally developed as an educational and prototyping tool and so supports a variety of integrators and magnet models in Python and optionally in C++:

- An exact Hamiltonian for rectangular magnets in small machines
- Explicit and implicit 4th order symplectic integrators for separable or non-separable Hamiltonians
- 2nd order symplectic dipole fringe fields
- A Newton method closed orbit solver
- Automatic differentiation for Taylor map extraction

The code has been tested against AT, Tracy-3 [12] and MADX-PTC [13] and gives identical results to within numerical precision, provided that corresponding co-ordinate systems, magnet body Hamiltonian and fringe field models are chosen.

The Python tracking code contains symbolic matrices for the 4-D linear case and symplectic integration for all 6-D elements. Transport matrices are recovered by the finite difference method (as used in AT) or by automatic differentiation (used in MADX-PTC and Tracy-3). In the case of automatic differentiation the recovered matrices are identical to the analytic solutions, to within numerical precision. The subset used in the model server is pure Python and Numpy, and uses the 4-D matrix solutions of the expanded paraxial Hamiltonian (see Eq. 1) [14] (transverse co-ordinates in scaled momentum, longitudinal co-ordinates in relative momentum deviation), and the linear part of the SLAC-75 [15] dipole fringe field, taking into account field integral and pole gap. This is adequate for linear optics in an easy-to-understand code as a proof of concept. If required, non-linear elements and solvers are available as drop-in replacements.

$$H = \frac{p_x^2+p_y^2}{2(1+\delta)} - \frac{x\delta}{\rho} + \frac{x^2}{2\rho^2} + V \qquad (1)$$

The uncoupled Twiss functions are calculated from the one-turn matrix and propagated through each element, and the analytic orbit response matrix is then calculated from the Twiss functions.

Lattice Description

The lattice is read in Tracy-3 format by a recursive descent parser, with the Tracy-3 lattice file generated from an AT deck, which is the master lattice format at Diamond. The AT deck is not read directly, as it is stored in the MATLAB language and so would require a complete MATLAB parser. The simplified domain-specific lattice languages, such as MAD (of which Tracy-3 is a variant), are therefore more suitable for interchange.

Model Server

The model server is a Python XMLRPC [16] server running as a web service on port 8080. XMLRPC provides a simple transport mechanism as it supports the common cross-language types of integer, string, floating point, heterogeneous list (called array) and dictionary (structure) and provides clients and servers in many languages. Whilst the serialization of messages can be up to 20 times larger than their native storage, this is not a significant limitation, and for large data sets this can be improved by using the base 64 to encode large arrays as binary blobs, making the overhead only 30%. An application of this would be to support the return of turn-by-turn tracking data from all BPMs.

On start-up the model server reads a lattice file from disk and calculates positions, initial Twiss functions and the response matrix. It then accepts magnet set-point changes and returns closed orbits, as measured by the BPMs, Twiss functions at all lattice points, and tunes. Subsequent changes are processed using a command to run the simulator.

STATUS

To date the application, model server and interface to the Diamond control system are working. The built-in applications support plotting of data for the families and some elementary physics functionality, orbit correction, and beta function measurement. The command line interface and scripting is working, but the management of global name space needs improving, and all the configuration of interface parameters is hard-coded for the Diamond storage ring. At present all the application components run in the Qt application thread, so any errant script or command can lock the application up. The Qt user interface needs developing to include resizing, menu functions, etc.

CONCLUSION

The work to date has validated the initial assumptions that Python together with the support packages make a rich environment suitable for developing accelerator physics tools. However, Diamond's needs have meant that this work has had low priority and so the application as yet is no more than a proof of principle. Considerable work would be required to develop a usable tool.

Furthermore the value of this environment needs to be considered, in the context of the acceptability of a Python-based application in comparison to a MATLAB-based one for the intended users, operations staff and accelerator physicists.

REFERENCES

[1] R.P. Walker, "Commissioning and Status of the Diamond Storage Ring", APAC 2007, Indore, India

[2] G. Portmann, et al, "An Accelerator Control Middle Layer Using MATLAB", PAC 2005, Knoxville, Tennessee

[3] http://www.python.org/

[4] http://numpy.scipy.org/

[5] http://matplotlib.sourceforge.net/

[6] http://www.riverbankcomputing.co.uk/software/pyqt/intro

[7] M. G. Abbott, et al, "Diverse uses of Python at Diamond", PCAPAC 2008, Ljubljana, Slovenia

[8] S. Molloy, S. Boogert, "Serpentine : A New Code For Particle Tracking", IPAC 2011, Kyoto, Japan

[9] http://code.google.com/p/spyderlib/

[10] http://pyqwt.sourceforge.net/

[11] A. Terebilo, "Accelerator Modelling with MATLAB Accelerator Toolbox", PAC 2001, Chicago, Illinois

[12] J. Bengtsson, Tracy-3, Private communication

[13] http://mad.web.cern.ch/mad/

[14] J. Bengtsson, "TRACY-2 User's Manual", SLS Internal Document, February 1997

[15] K. L. Brown, "A First and Second Order Matrix Theory for the Design of Beam Transport Systems and Charged Particle Spectrometers", SLAC 75, 1982

[16] http://www.xmlrpc.com/