other PACs numbers?

`eqtools`: Modular, Extensible, Open-Source, Cross-Machine Python Tools for Working with Magnetic Equilibria

M. A. Chilenski,* I. C. Faust, and J. R. Walk

MIT Plasma Science and Fusion Center

(Dated: January 30, 2015)

As experimental plasma physics research transitions to be predominantly conducted across institutions and machines, it becomes increasingly important that portable tools be developed to enable data from diverse sources to be analyzed in a consistent manner. This paper presents `eqtools`, a modular, extensible, open-source toolkit implemented in the Python programming language for handling magnetic equilibria and associated data from tokamaks. `eqtools` provides a single interface for working with magnetic equilibrium data, both for handling derived quantities and mapping between coordinate systems, extensible to function with data from different experiments, data formats, and magnetic reconstruction codes, replacing the diverse, non-portable solutions currently in use. Moreover, implementing these tools in Python removes a substantial barrier to code migration and new development in the open-source Python programming language, which presents a number of advantages. In this paper, we introduce the design of the `eqtools` package and detail the workflow for usage and expansion to additional devices. The implementation of a novel three-dimensional spline solution (in two spatial dimensions and in time) is also detailed. Finally, benchmarking for accuracy and speed against existing tools are detailed. Wider deployment of these tools will enable efficient sharing of data and software between institutions and machines as well as self-consistent analysis of the shared data.

PACS numbers: 52.55.Fa

* markchil@mit.edu

I. INTRODUCTION

The basic computational tasks associated with magnetic equilibrium reconstructions – namely, the handling of derived quantities (*e.g.,* calculated plasma current, safety-factor profiles) and the mapping between real-space and flux coordinate systems for experimental data – are universal among tokamak experiments. Despite this commonality, experiments typically utilize in-house solutions developed for the particulars of that experiment's data storage and usage. This ad-hoc development of base-level functionality inhibits the mobility of higher-level codes to other devices (as the code may require substantial modification to address the particulars of the new data storage design). Moreover, such development is often quite static, such that the implementation is difficult to extend to new data formats (for example, to handle both a primary MDSplus-based data storage system [1] and the *eqdsk* storage files produced directly by the EFIT reconstruction code [2]), necessitating parallel workflows for functionally identical tasks depending on the data source. Best design practices call for the vagaries of data source and storage implementation to be placed in the back end, presenting a consistent, straightforward interface common between machines and code implementations to the research scientist.

The new `eqtools` package provides a modular, extensible, cross-machine toolkit developed in the Python programming language for handling magnetic equilibrium data. The `eqtools` package provides a consistent and straightforward interface to the researcher for both coordinate mapping routines (historically handled in separate standalone routines) and access to derived quantities (often handled with manual hooks into data storage). Moreover, `eqtools` is constructed with a modular, object-oriented design, such that the package is easily extensible to handle data from different experiments and reconstruction codes, giving the researcher a single unified interface for data from any machine or code. The implementation of reconstructed-equilibrium handling in the Python language removes a substantial barrier to the adoption of Python as a day-to-day analysis language for tokamak research, which offers numerous advantages in ease of use, computational speed, user/developer base, and free and open-source implementations compared to current common working languages for fusion research. The `eqtools` package is open-source and licensed under the GNU General Public License [3], and distributed via GitHub [4].

This paper details the design and implementation of `eqtools`, particularly the paradigm

for extension to new machines (section II), describes the basic algorithms used to convert between coordinate systems (section III), describes the implementation of a trivariate spline method for improved interpolation in the time dimension (section IV), and presents run-time and accuracy benchmarks against the current IDL implementation at Alcator C-Mod (section V).

II. PACKAGE DESIGN AND USE

The `eqtools` package is designed to present a consistent, human-readable interface to the user for both data handling of derived quantities and mapping routines between coordinate systems, all contained within a single persistent object (herein referenced in code snippets as `eq`). For example, consider how the poloidal flux on-axis would be accessed using the current system on Alcator C-Mod using the MDSobjects [5] Python interface:

```
import MDSplus
t = MDSplus.Tree('analysis', shotnum)
n = t.getNode('efit.results.a_eqdsk:simagx')
psi0 = n.data()
```

And compare to the approach to access the same data using `eqtools`:

```
import eqtools
eq = eqtools.CModEFITTree(shotnum)
psi0 = eq.getFluxAxis()
```

Notice that the `eqtools` approach requires fewer lines and that the cryptic six letter EFIT tag name "simagx" is replaced with the more descriptive "getFluxAxis." Similarly, mapping routines are handled internally in the `eq` object – mapping from the machine RZ grid to normalized poloidal flux, for example, is simply `eq.rz2psinorm(R,Z,t,...)`. One notable consequence of this is that intermediate spline calculations for the mapping routines are stored in the persistent object – compared to typical standalone mapping routines (which must start from scratch for each function call) this allows substantial speed gains for subsequent mapping calculations with the same equilibrium object, as is illustrated in section V.

The `eqtools` package is structured such that the user-facing methods (for data access and coordinate mapping) are consistent for different machines or reconstruction codes, and

moreover adding support for new machines or codes requires minimal new code. The package inheritance structure is depicted in figure 1. The base level of the inheritance structure (shown in blue in figure 1) provides the abstract class `Equilibrium` – this defines the skeleton structure for the data system (ensuring that child classes use a consistent format) and defines the mapping routines, which are inherited and used by each child class. The methods therein are sufficiently general to be used by any grid-based equilibrium reconstruction.

An intermediate inheritance level (shown in yellow in figure 1) provides more specific methodology (while still maintaining cross-machine generality) – for example, the `EFITTree` class provides methods for EFIT reconstructions stored in MDSplus tree structures (as is used on C-Mod, NSTX, and DIII-D). This minimizes the need for repeated code in machine-specific versions of common code implementations. The user-facing inheritance level (shown in green in figure 1) finalizes the specific details of implementation for a given reconstruction code, data-storage methodology, and machine implementation (*e.g.,* `CModEFITTree`, `NSTXEFITTree`, `D3DEFITTree`). These user-facing implementations typically require relatively little code (on the order of 100 lines in the case of `CModEFITTree` and `NSTXEFITTree`), and extension modules to the code can be quickly developed in most cases.

cites?

In addition to the already-developed modules inheriting `EFITTree`, the code currently contains the `EqdskReader` module, which directly interfaces with the *eqdsk* text files (specifically, the "g-file" and "a-file" containers, which store equilibrium and scalar derived quantities, respectively) generated by EFIT. This allows a single unified interface for both the tree-based and portable text-file data storage common to US experiments. Due to the unique structure of the code necessary to read text-file data (which is done directly in `EqdskReader`), `EqdskReader` directly inherits mapping routines and skeletal structure from `Equilibrium`, without an intermediate stage; however, as new text-file-based storage methods are implemented in `eqtools` sufficient commonality may be found to necessitate the creation of a intermediate abstraction level for generalized text-file storage systems in subsequent versions of the `eqtools` package.

cite?

expand?

other countries?

In addition to the `EqdskReader` module, which handles equilibrium and derived-quantity data from g- and a-file outputs from EFIT, the `eqtools` package provides a `PFileReader` object to handle plasma-profile data from the "p-files" associated with EFIT. While this does not require access to the reconstructed equilibrium (and thus is separate from the `Equilibrium` inheritance structure) profile data is commonly paired with the reconstructed

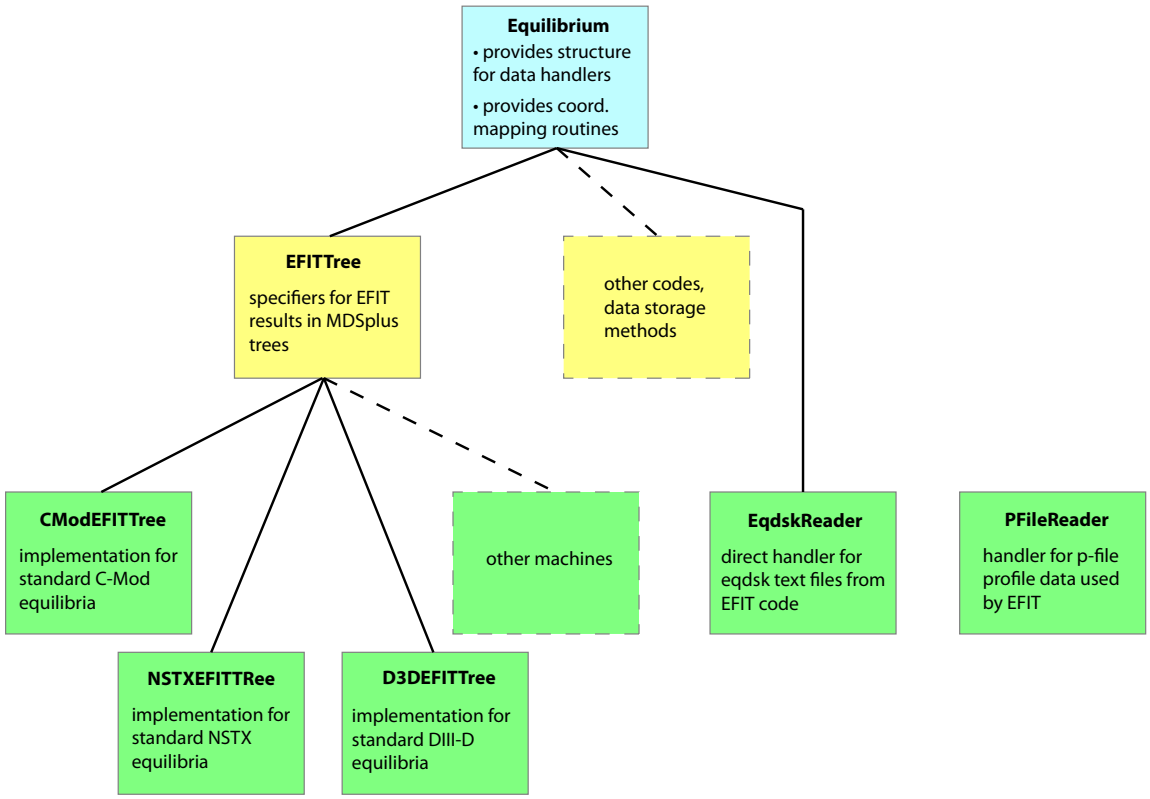

FIG. 1: Inheritance schematic for the `eqtools` package. The base abstract class (blue) provides a skeleton structure for the derived-data handlers, as well as providing the complete set of coordinate-mapping routines. Intermediate abstract classes (yellow) prescribe the handling for data storage systems and codes – at present the relatively-ubiquitous EFIT reconstruction stored in MDSplus tree structures is provided. User-facing classes (green) handle the details of machine-specific implementations. Dashed lines denote classes that have not yet been implemented, but which can be introduced into the package in a straightforward manner due to its modular construction. The user interface is consistent, as it is provided by the parent classes – code migration between machines requires only changing which child class is called for the reconstruction. The `EqdskReader` class, which directly handles *eqdsk* text files from EFIT, inherits directly from the base class as it is sufficiently unique to not warrant an intermediate abstract class. add filewriter?

equilibrium – as such, the `eqtools` package provides a built-in methodology to handle the additional data. The `eqtools` package also contains a package-level method, `filewriter` to produce g-files from classes in the `Equilibrium` inheritance tree, allowing easy generation of portable datasets from the main data-storage method.

check naming!

reword?

III. DETAILS OF THE COORDINATE MAPPING ROUTINES

One of the strengths of `eqtools` is its support for conversions between the majority of coordinate systems in common use: `eqtools` supports transformations between a wide variety of coordinates including real space (R, Z) coordinates, mapped outboard midplane major radius R_{mid}, normalized minor radius r/a, unnormalized poloidal flux ψ, normalized poloidal flux ψ_{n}, normalized toroidal flux ϕ_{n}, normalized flux surface volume V_{n} and the square roots of these quantities.

John, not sure what more breakdown you want here. Please clarify your comments.

The basic transformation is to map a given point (R, Z) at a given time t to the poloidal flux ψ at that location as computed with the magnetic reconstruction code. The default implementation uses a nearest-neighbor interpolation in time: the code first retrieves the flux reconstruction at the time closest to t, then a bivariate interpolating spline [6, 7] is used to map from (R, Z) to ψ. The bivariate spline coefficients from each time are stored in memory as they are computed in order to speed up subsequent calculations at that time slice. A more advanced method using a tricubic interpolating spline to interpolate smoothly in space *and* time is described in section IV.

Once the coordinate has been mapped to ψ, subsequent calculations are simpler. Normalized poloidal flux is defined as

$$\psi_n = \frac{\psi - \psi_0}{\psi_a - \psi_0}, \tag{1}$$

where ψ_0 is the poloidal flux at the magnetic axis and ψ_a is the poloidal flux at the boundary. To ensure self-consistency, the default behavior is to use nearest-neighbor interpolation to get the values of ψ_0 and ψ_a at the desired time. When a tricubic spline is used to map from (R, Z) to ψ a cubic spline is used to interpolate these values in time.

Normalized toroidal flux is defined in terms of normalized poloidal flux as

$$\phi(\psi) = \int_{\psi_0}^{\psi} q(\psi')\,\mathrm{d}\psi' \tag{2}$$

$$\phi_\mathrm{n} = \frac{\phi}{\phi_a}, \tag{3}$$

where $q = \mathrm{d}\phi/\mathrm{d}\psi$ is the safety factor profile (typically computed when EFIT is run) and $\phi_a = \int_{\psi_0}^{\psi_a} q(\psi')\,\mathrm{d}\psi'$ is the toroidal flux at the last closed flux surface. The integral in (2) is numerically evaluated using the trapezoid rule.

The normalized flux surface volume is defined as

$$V_\mathrm{n}(\psi) = \frac{V(\psi)}{V_a}, \tag{4}$$

where V_ψ is the volume enclosed by the (closed) flux surface with flux ψ and V_a is the volume enclosed by the last closed flux surface. In the case of Alcator C-Mod, the flux surface volume $V(\psi)$ is computed automatically when EFIT is run, but it would be straightforward to override the `getFluxVol` method of the `Equilibrium` class to compute this from the $\psi(R, Z)$ grid when this quantity is not already available in the tree. Mapping from ψ_n to R_mid is accomplished by forming a dense radial grid of R points that go from the magnetic axis to the edge of the grid the flux is reconstructed on, finding the vertical location of the magnetic axis at the desired time(s) (called Z_0), then converting the resulting (R, Z_0) points to ψ_n with the routines described above. This one-to-one mapping between R_mid and ψ_n is then interpolated to give the desired conversion from ψ_n to R_mid.

By default, r/a is defined in terms of R_mid as

$$r/a = \frac{R_\mathrm{mid} - R_0}{R_a - R_0}, \tag{5}$$

where R_0 is the major radius of the magnetic axis and R_a is the outboard midplane major radius of the last closed flux surface. Since other definitions of r/a are preferred when the Shafranov shift is high, the specific definition of r/a can be changed simply by overriding the methods `_rmid2roa` and `_roa2rmid` in the `Equilibrium` class. Variants on these basic routines are then used to map between other pairs of coordinates.

IV. TRI-SPLINE IMPLEMENTATION

Time-dependent interpolation poses a difficult problem with respect to the instantaneous nature of most equilibrium reconstructions. Usually coordinates are often determined via

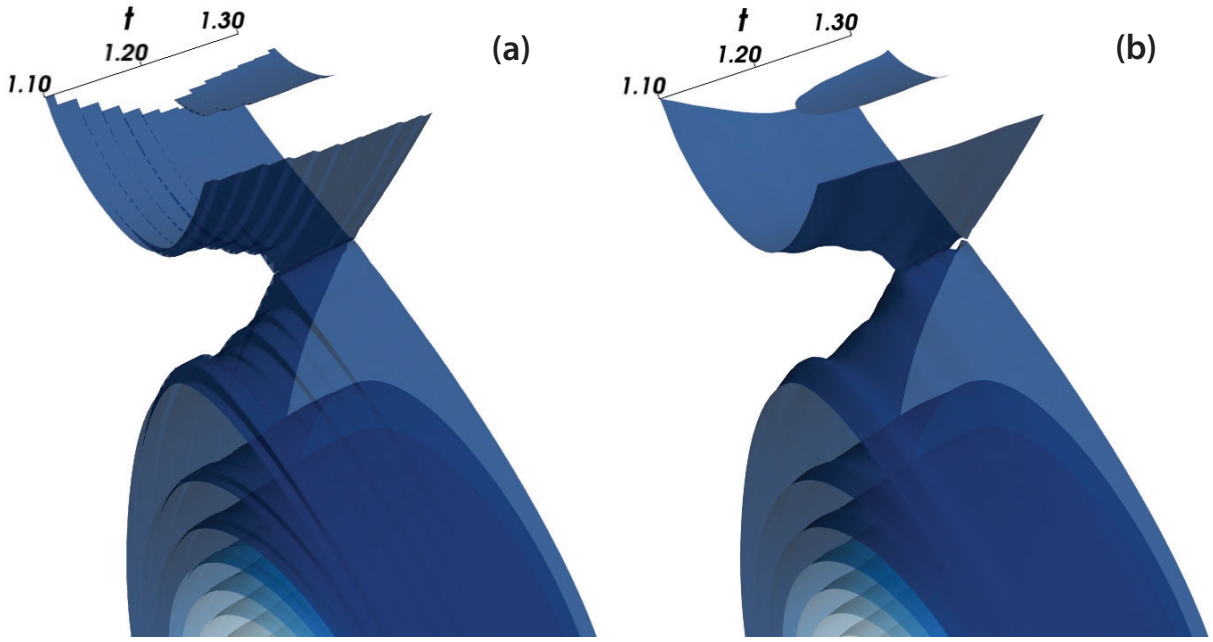

FIG. 2

interpolation using the closest equilibrium in time. When the sampling rate of equilibria is lower than the desired coordinate sampling rate, this can induce aliasing behaviors. The aliasing-induced discontinuities pose an issue for long timescale behaviors of coordinates. When the set of equilibria achieves the Nyquist frequency for variations of interest, an additional interpolation in time can accurately remove aliasing errors induced by this time-base mismatch. The additional temporal dimension requires the use of higher dimensional interpolators, for which a fast tricubic interpolation scheme was developed.

The additional time dimension and subsequent higher-dimensionality increases the evaluation time of the mapping routines. A tricubic interpolating spline [8] is used to evaluate the three-dimensional flux grid. The time dimension is assumed to be like a rectilinear spatial dimension which allows for the use of developed optimized matrix methods for extracting the necessary spline coefficients. The higher dimensionality of the problem scales the necessary computational time due to the increased necessary information ($\times 64$ increase in matrix computation). Other non-three-dimensional parameters also require further computation due to the inherent complication of increasing the number of dimensions. Use of persistence of the coefficients allows for similar computational times to previous equilibrium mappings when subject to calculations in the same 'voxel', the overall computation is slower when

compared to lower-dimensional mapping.

Significant time savings can be recovered for tricubic interpolation computation when the data grid is regular, which is often the case for sets of magnetic equilibria. Recovering the spline coefficients requires the calculation of a number of derivatives in each dimension, which for a regular grid can be achieved with finite difference matrix $\boldsymbol{B}$. The new *a priori* inclusion of $\boldsymbol{B}$ in the spline calculation matrix $\boldsymbol{A_{inv}}$ removes the necessary derivative generation and reduces total computation by half (denoted $\boldsymbol{A^*_{inv}}$). This calculation requires a collection of the nearest $4 \times 4 \times 4$ grid of points to the coordinates of interest $\boldsymbol{x}$ to extract spline coefficients $\boldsymbol{y}$.

$$\boldsymbol{y} = \boldsymbol{A_{inv}}(\boldsymbol{Bx}) = \boldsymbol{A^*_{inv}}\boldsymbol{x} \tag{6}$$

Where,

$$\boldsymbol{A^*_{inv}} = \boldsymbol{A_{inv}}\boldsymbol{B} \tag{7}$$

The code for the tricubic spline interpolation was written in C and was integrated into Python using F2PY [9] package. Other programming languages and codes with sufficient C APIs can access the optimized trispline code for use outside of magnetic reconstructions, and is well contained in accepting 3 dimensional data for analysis.

V. BENCHMARKING

`eqtools` has been benchmarked against the existing, thoroughly-tested IDL routines presently in use for handling coordinate mapping at Alcator C-Mod. Figure 3 shows the discrepancy between IDL and `eqtools` for the conversion of (R, Z) to ψ. The differences are small (of order 10^{-4}), and are consistent with the fact that Python defaults to double precision whereas IDL defaults to single precision. Timing tests were conducted by converting a 66×66 element (R, Z) grid (double the density of the grid EFIT provides the flux on) into each of the coordinates supported. This conversion was performed at each of the 90 time slices output by EFIT and nearest-neighbor temporal interpolation was used. The conversion was performed twice with the same `Equilibrium` object in `eqtools` in order to assess the time savings from storing the spline coefficients. The test was repeated 100 times, and the mean execution time to convert the (R, Z) grid at each time slice is given in Table I. `eqtools` is faster than the IDL-based routines by as much as a factor of ten even on the

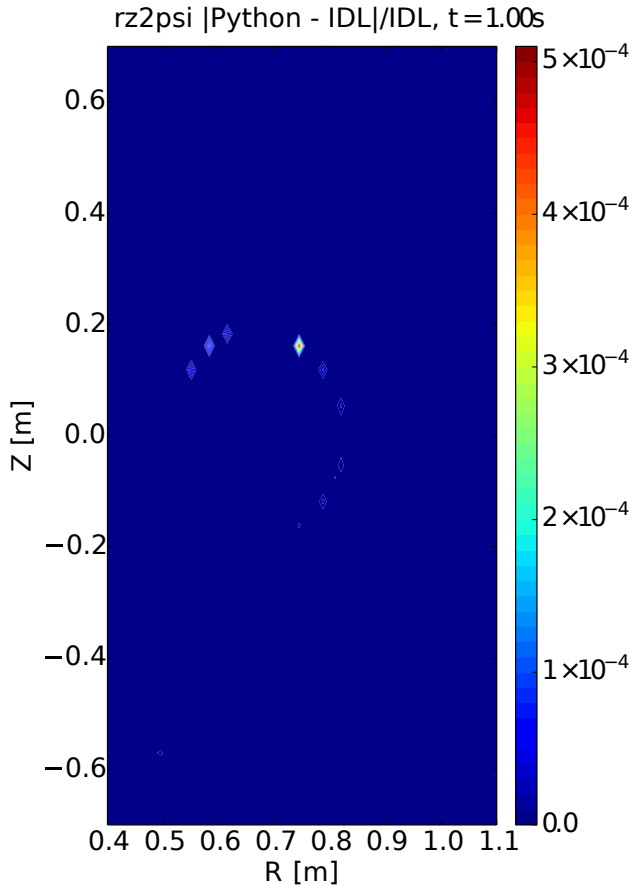

FIG. 3: Relative difference between calculations of a mapping between the RZ grid and poloidal flux for the `eqtools` and the current IDL implementation of the mapping routine for an example Alcator C-Mod reconstructed equilibrium. Relative differences of order 10^{-4} are typical, and are consistent with the different default precisions of IDL versus Python.

TABLE I: Mean execution times in milliseconds

Conversion	IDL	`eqtools`	
		first run	second run
$(R, Z) \rightarrow \psi$	11.7	2.5	1.9
$(R, Z) \rightarrow \psi_{\mathrm{n}}$	21.8	2.0	2.0
$(R, Z) \rightarrow \phi_{\mathrm{n}}$	30.2	3.1	2.3
$(R, Z) \rightarrow V_{\mathrm{n}}$	28.0	3.0	2.3
$(R, Z) \rightarrow R_{\mathrm{mid}}$	36.1	12.1	2.4

first run. This improvement is from a combination of the switch to Python and the fact that `eqtools` stores the equilibrium data in a persistent object instead of having to fetch it from the MDSplus tree for each call. Subsequent runs with the same `Equilibrium` object are faster, particularly for the conversion to R_{mid}, which has considerable overhead on the first call.

VI. SUMMARY

The `eqtools` package provides a modular, extensible framework for handling the basic tasks associated with magnetic equilibrium reconstructions for tokamaks – namely, accessing derived quantities and mapping between real-space and flux coordinate systems for experimental data. This package, developed in the open-source Python programming language and freely available from Github [4], presents a consistent, user-friendly interface independent of data source or storage method, replacing machine- and storage-specific methods which are often nonintuitive to use and which often use a static code structure that can be difficult to extend for experimental data from multiple machines, storage methods, or reconstruction codes. Moreover, the `eqtools` package is designed with a modular, object-oriented construction, such that the package is easily extended to include new experiments, reconstruction codes, and data storage methods (in the current distribution version, EFIT reconstructions [2] in MDSplus-based data storage from NSTX, C-Mod, and DIII-D are implemented, along with a class to read the portable *eqdsk* datafile).

The package includes a complete set of mapping routines between real-space machine coordinates (that is, the RZ grid defined for the reconstruction), midplane-mapped real-space coordinates (major radius and normalized minor radius), and flux-space coordinate systems (normalized poloidal and toroidal flux and normalized flux surface volume), as well as their square roots – in short, the coordinate systems customarily used in a broad variety of analysis applications are supported within a single unified user interface. In addition to the standard bivariate analysis mapping the R and Z coordinate with nearest-neighbor interpolation in time (as is used in the mapping routines presently used on C-Mod), `eqtools` allows for a trivariate spline providing smooth interpolation along the time axis as well. This allows the user to optionally trade computational time for a substantially more accurate treatment of the time variation in cases where experimental data are sampled at times

significantly different from the reconstruction timebase.

The mapping routines in `eqtools` have been thoroughly benchmarked against the existing IDL implementations used at Alcator C-Mod. Mapping results from `eqtools` are consistent with the previous IDL results to within numerical error (arising from double-precision calculations in Python compared to the default single precision used in IDL). Initial runs of the `eqtools` mapping routines are a factor of three to ten faster than the previous IDL implementation. Moreover, as the persistent `Equilibrium` object stores intermediate calculations in the mapping routines, subsequent calculations with the same shot show additional speed improvements compared to the IDL implementation.

The development of `eqtools` clears a substantial hurdle to the adoption of Python as a standard data analysis language for tokamak research, the use of which offers numerous advantages in terms of development base and support, computational speed, ease of adoption, and cost over other languages in common use at present. Additionally, `eqtools` allows for substantially more straightforward implementation of cross-machine analysis tools, a significant benefit in light of increasing emphasis on modeling and cross-machine collaboration in fusion research.

ACKNOWLEDGMENTS

The authors would like to acknowledge their thanks for the work S. Wolfe performed in writing the original IDL coordinate mapping routines for Alcator C-Mod.

This material is based upon work conducted using the Alcator C-Mod tokamak, a DOE Office of Science user facility. This material is based upon work supported by the U.S. Department of Energy, Office of Science, Office of Fusion Energy Sciences under Award Number DE-FC02-99ER54512. This material is based upon work supported in part by the U.S. Department of Energy Office of Science Graduate Research Fellowship Program (DOE SCGF), made possible in part by the American Recovery and Reinvestment Act of 2009, administered by ORISE-ORAU under contract number DE-AC05-06OR23100.

---

[1] J. A. Stillerman, T. W. Fredian, K. A. Klare, and G. Manduchi, Review of Scientific Instruments **68**, 939 (1997).

[2] L. L. Lao, H. St. John, R. D. Stambaugh, A. G. Kellman, and W. Pfeiffer, Nuclear Fusion **25**, 1611 (1985).

[3] "GNU general public license, version 3," `http://www.gnu.org/licenses/gpl.html` (2007).

[4] M. A. Chilenski, I. C. Faust, and J. R. Walk, "eqtools: Python tools for magnetic equilibria in tokamak plasmas (github distribution)," `https://github.com/PSFCPlasmaTools/eqtools` (2014); "eqtools: tools for interacting with magnetic equilibria (online documentation)," `eqtools.readthedocs.org/en/latest/` (2014).

[5] T. Fredian, J. Stillerman, and G. Manduchi, Fusion Engineering and Design **85**, 568 (2010).

[6] P. Dierckx, *Curve and Surface Fitting with Splines* (Oxford University Press, 1993).

[7] E. Jones, T. Oliphant, P. Peterson, *et al.*, "SciPy: Open source scientific tools for Python," (2001–).

[8] F. Lekien and J. Marsden, International Journal for Numerical Methods in Engineering **63**, 455 (2005).

[9] P. Peterson, International Journal of Computational Science and Engineering **4**, 296 (2009).