# `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: November 17, 2015)

As plasma physics research for fusion energy transitions to an increasing emphasis on cross-machine collaboration and numerical simulation, 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, while the open-source Python programming language offers a number of advantages as a scripting language for research purposes, the lack of basic tokamak-specific functionality has impeded the adoption of the language for regular use. Implementing equilibrium-mapping tools in Python removes a substantial barrier to new development in and porting legacy code into Python. 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, verification and 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, 07.05.Kf, 52.70.Ds

* 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 profile) and the mapping between real-space and flux coordinate systems for experimental data – are universal among tokamak experiments. Despite this commonality, experiments typically utilize locally-developed 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 (*e.g.,* 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 PyPI [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 novel trivariate spline method for improved interpolation in the time dimension (section IV), presents runtime and accuracy benchmarks against the current IDL [5] implementation at Alcator C-Mod (section V), and shows examples of the code in use (section VI).

## 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, accessing the calculated value for the poloidal flux on-axis, $\psi_0$, is simply `eq.getFluxAxis()` – this replaces arrangements that may involve cumbersome manual hooks into the data system and interaction with opaque internal EFIT variable names. Similarly, mapping routines are handled internally in the `eq` object – mapping from $(R, Z)$ coordinates 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. Moreover, adding support for new machines or codes requires minimal new code development. 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 prototypical structure for the data (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 support for specific codes and data systems (while still maintaining cross-machine generality) – for example, the `EFITTree` class provides methods for EFIT reconstructions stored in MDSplus tree

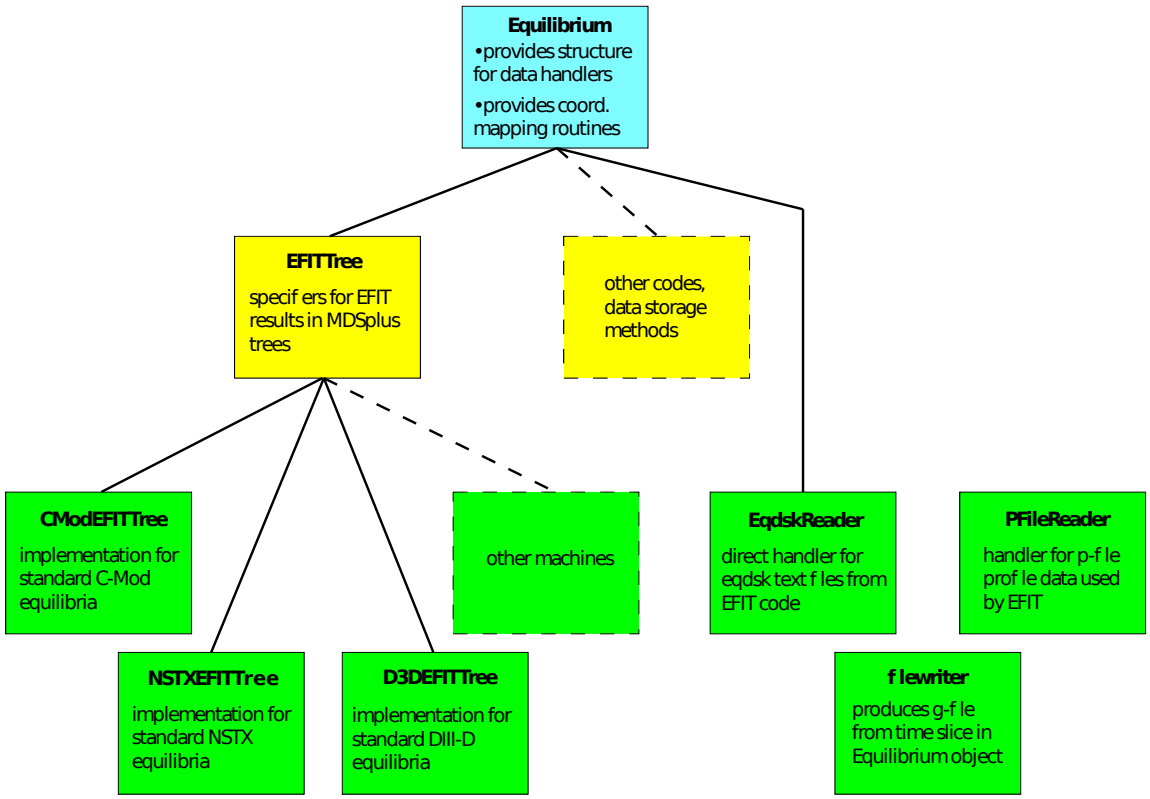



FIG. 1: Package structure for the `eqtools` package. The base abstract class (blue) provides a prototypical structure for the derived-data handlers, as well as providing the complete set of coordinate-mapping routines. Intermediate abstract classes (yellow) prescribe the handling for data storage systems 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. Outside the `Equilibrium` inheritance structure, the package also provides the `PFileReader` class to handle the "p-file" plasma profile data associated with EFIT, as well as the `filewriter` module to produce portable g-files from `Equilibrium` objects.

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.

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 MDSplus-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 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.

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 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 (for example, MDSplus tree) data-storage method.

## III. DETAILS OF THE COORDINATE MAPPING ROUTINES

`eqtools` supports transformations between a wide variety of coordinates in common use including real-space $(R, Z)$ coordinates, mapped outboard midplane major radius $R_{\text{mid}}$,

normalized minor radius $r/a$, unnormalized poloidal flux $\psi$, normalized poloidal flux $\psi_{\mathrm{n}}$, normalized toroidal flux $\phi_{\mathrm{n}}$, normalized flux surface volume $V_{\mathrm{n}}$ and the square roots of these quantities.

The most important transformation maps a given point $(R, Z)$ at a given time $t$ to the poloidal flux $\psi$ 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 $\psi$. 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 $\psi$, subsequent calculations are simpler. Normalized poloidal flux is defined as

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

where $\psi_0$ is the poloidal flux at the magnetic axis and $\psi_a$ 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 $\psi_0$ and $\psi_a$ at the desired time. When a tricubic spline is used to map from $(R, Z)$ to $\psi$ a cubic spline is used to interpolate $\psi_0$ and $\psi_a$ in time.

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

$$\begin{aligned} \phi(\psi) &= \int_{\psi_0}^{\psi} q(\psi')\,\mathrm{d}\psi' \\ \phi_{\mathrm{n}} &= \frac{\phi}{\phi_a}, \end{aligned} \tag{2}$$

where $q(\psi) = \mathrm{d}\phi/\mathrm{d}\psi$ 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

$$V_{\mathrm{n}}(\psi) = \frac{V(\psi)}{V_a}, \tag{3}$$

where $V(\psi)$ is the volume enclosed by the (closed) flux surface with flux $\psi$ 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 $\psi_{\mathrm{n}}$ to $R_{\mathrm{mid}}$ 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 $\psi_{\mathrm{n}}$ with the routines described above. This one-to-one mapping between $R_{\mathrm{mid}}$ and $\psi_{\mathrm{n}}$ is then interpolated to give the desired conversion from $\psi_{\mathrm{n}}$ to $R_{\mathrm{mid}}$.

By default, $r/a$ is defined in terms of $R_{\mathrm{mid}}$ as

$$r/a = \frac{R_{\mathrm{mid}} - R_0}{R_a - R_0}, \tag{4}$$

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

Typically, coordinates are determined via nearest-neighbor interpolation to the reconstruction timebase. When the timebase of the equilibrium reconstruction is coarser than and/or offset from the desired timebase this can induce aliasing and discontinuities (as can be seen in figure 2). These limit the nearest-neighbor approach when considering fast timescale behaviors with comparatively slow reconstructions. When reconstruction achieves the Nyquist frequency for variations of interest, an additional interpolation in time can accurately remove aliasing errors induced by this timebase mismatch. The additional temporal dimension requires the use of higher-dimensional interpolators, for which a fast tricubic interpolation scheme has been developed.

A tricubic interpolating spline [8] is used to evaluate the three-dimensional flux grid. The additional time dimension and subsequent higher dimensionality increases the evaluation time of the mapping routines. The $(R, Z, t)$ points are assumed to form a tensor product grid which allows for the use of finite element matrix methods for extracting the derivatives

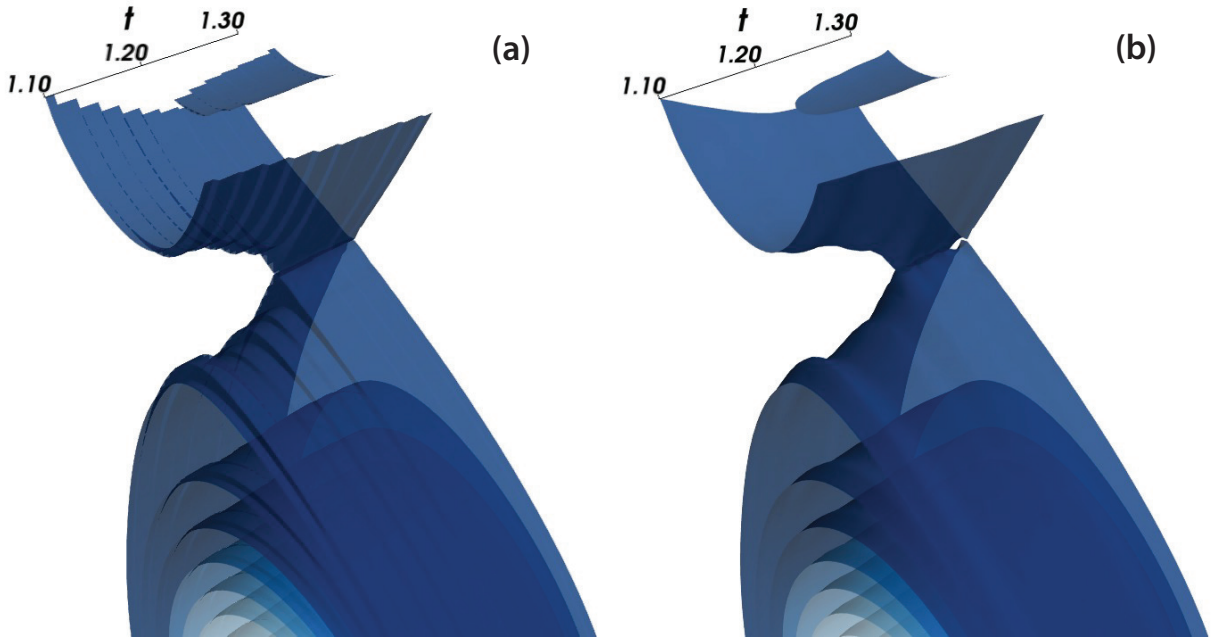



FIG. 2: Flux surface reconstruction in time of an Alcator C-Mod discharge using nearest-neighbor time interpolation (a) and a tricubic spline interpolation (b). Aliasing behavior is removed by the use of a spline in the time-dimension.

used to compute the necessary spline coefficients. The higher dimensionality of the problem scales the necessary computational time due to the increased necessary information (64-fold increase in matrix computation). Other non-three-dimensional parameters also require further computation due to the inherent complication of increasing the number of dimensions. Persistence of the coefficients allows for similar computational times to previous equilibrium mappings when subject to calculations in the same voxel, but the overall computation is still slower when compared to lower-dimensional mapping.

The code for the tricubic spline interpolation was written in C and was integrated into Python using the F2PY package [9]. Two separate methods are used to calculate and evaluate the spline, dependent on the nature of the grid. In cases which the grid is regular, the novel inclusion of a finite-difference matrix into the spline coefficient matrix reduces computational time (see Appendix A). In other cases, the derivatives in the necessary directions and orders are calculated before being used in the spline coefficient matrix multiplication. The C code is compartmentalized and can accept arbitrary three dimensional data, so other programming languages and codes with sufficient C APIs can access the optimized trispline

code for general use.

## V. VERIFICATION AND BENCHMARKING

`eqtools` has been verified against the existing, thoroughly-tested IDL routines presently in use for handling coordinate mapping at Alcator C-Mod. Figure 3 shows the discrepancy between the IDL routines and `eqtools` for the conversion of $(R, Z)$ to $\psi$. The differences are small (of order $10^{-7}$ Wb/rad, compared to a signal of order $\pm 0.5$ Wb/rad), 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 $66 \times 66$ 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 180 time slices (double the temporal resolution output by EFIT) and nearest-neighbor temporal interpolation was used. The test was run both with all points being processed at once (denoted "all" in Table I) and with the routine being called inside a loop over all time points (denoted "loop" in Table I). 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 IDL routines (with the exception of the conversion to $R_{\mathrm{mid}}$) support reuse of the spline coefficients from a single time slice, so a second run of the IDL code with the stored spline coefficients was performed when looping over time slices. The test was repeated 100 times, and the mean execution time to convert the $(R, Z)$ grid for all time points is given in Table I.

In all cases `eqtools` is faster than the IDL routines. Furthermore, the test results highlight the additional flexibility `eqtools` provides users. For the most efficient case where all of the points are passed at once, `eqtools` is only slightly faster for the simple conversions of $(R, Z)$ to $\psi$ and $\psi_{\mathrm{n}}$. But, for the more complicated conversions to $\phi_{\mathrm{n}}$, $V_{\mathrm{n}}$ and $R_{\mathrm{mid}}$, `eqtools` is anywhere from 1.5 to 1.9 times faster than the IDL routines. Furthermore, the caching of intermediate results in `eqtools` accelerates the second call to a routine by as much as a factor of 5 (for the conversion to $R_{\mathrm{mid}}$). Where `eqtools` stands in stark contrast to the IDL routines is the case where the conversions are evaluated in a loop. While this case is not something which would be used directly as both codes support the evaluation of multiple time points at once, it is indicative of the performance to be expected should a user write a script which needs to evaluate each time point on a different grid, or otherwise needs

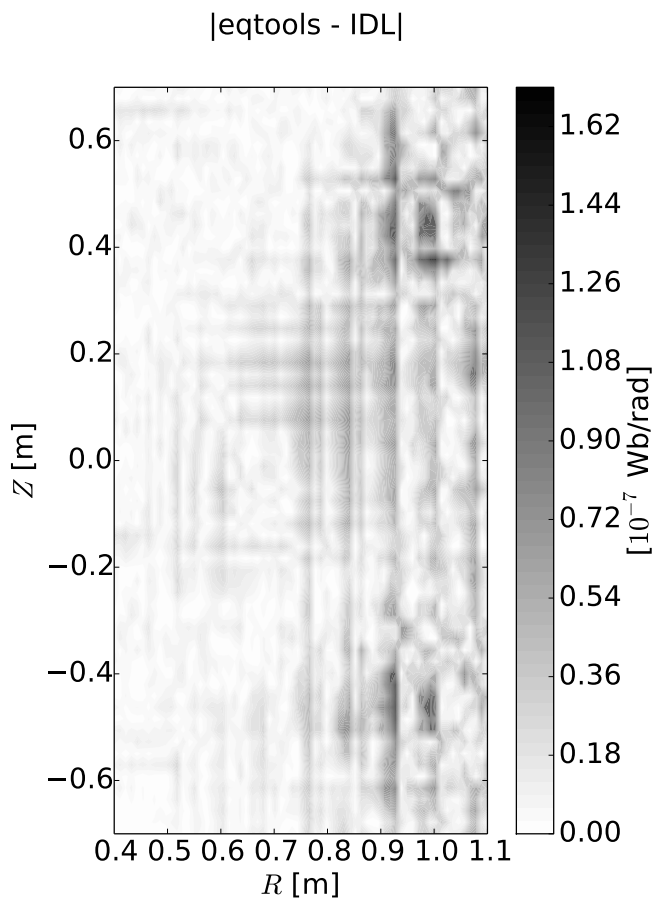



FIG. 3: Absolute difference between calculations of a mapping between the $(R, Z)$ grid and poloidal flux $\psi$ for `eqtools` and the current IDL implementation of the mapping routine for an example Alcator C-Mod reconstructed equilibrium. Relative differences of order $10^{-7}$ Wb/rad are typical (the value of $\psi$ is in the range $\pm 0.5$ Wb/rad), and are consistent with the different default precisions of IDL versus Python.

the flexibility to split an operation up into steps applied to each time point. For this case, `eqtools` is anywhere from 5 to 20 times faster than the IDL routines. This is believed to be a result of the large overhead associated with function calls in the IDL language. These test results highlight the improved performance and extra programming flexibility provided by `eqtools`.

TABLE I: Time to convert all 180 time slices in milliseconds. "All" refers to passing all points to the routine at once, whereas "loop" refers to calling the routine in a loop over time points. "First" refers to the first call to the routine when data are collected from the server and spline coefficients are computed, "second" refers to the second call once the data and spline coefficients have been cached.

| Conversion | IDL | | | | `eqtools` | | | |
|---|---|---|---|---|---|---|---|---|
| | all | | loop | | all | | loop | |
| | first | second | first | second | first | second | first | second |
| $(R, Z) \to \psi$ | 149 | — | 3147 | 3052 | 134 | 116 | 179 | 161 |
| $(R, Z) \to \psi_{\mathrm{n}}$ | 163 | — | 4024 | 3880 | 137 | 119 | 184 | 165 |
| $(R, Z) \to \phi_{\mathrm{n}}$ | 473 | — | 5307 | 5196 | 254 | 165 | 303 | 215 |
| $(R, Z) \to V_{\mathrm{n}}$ | 468 | — | 5326 | 5220 | 253 | 164 | 302 | 215 |
| $(R, Z) \to R_{\mathrm{mid}}$ | 1400 | — | 5328 | — | 942 | 187 | 993 | 238 |

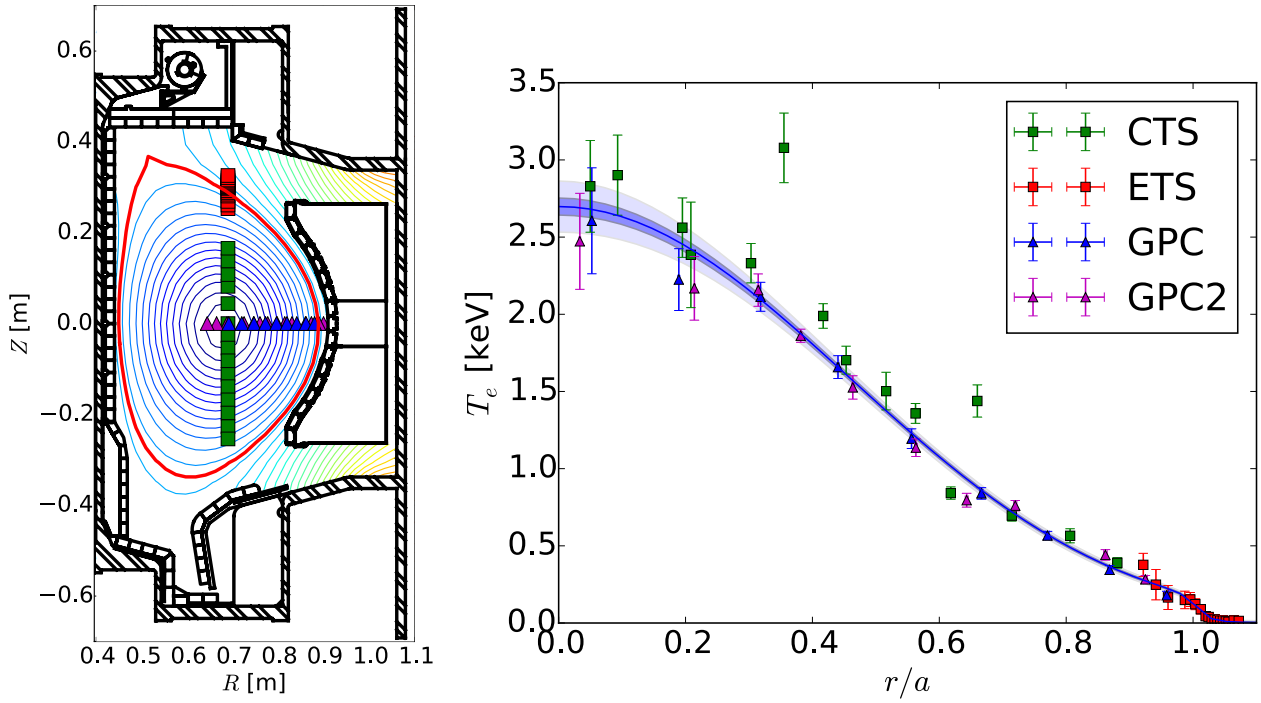
## VI. EXAMPLE APPLICATIONS

### A. TRIPPy GOES HERE!!!

Add TRIPPy section!

### B. Profile fitting

Combining data from multiple diagnostics into a consistent, smooth profile is a fundamental task in experimental plasma physics. `profiletools` [10] is a Python package which aims to simplify this task by presenting a uniform interface to load, time-average, combine and fit data from different diagnostics. `eqtools` is used internally by `profiletools` both to map diagnostics to a consistent grid and to output the smooth fit in the desired coordinates. In line with the `eqtools` philosophy of shielding the user from obtuse calls to the underlying data system (be it MDSplus or some other system), for each diagnostic supported, `profiletools` provides a function to fetch the data (and the accompanying error bars). These quantities are returned in a `BivariatePlasmaProfile` object, where "bivariate" refers to the fact that the data are, in general, a function of space and time. The coordinates the measurement




(a) equilibrium

(b) data and smoothed profile

FIG. 4: Equilibrium with diagnostic locations (a) and individual measurements with smoothed electron temperature profile (b). The data includes two Thomson scattering systems (ETS, CTS) which measure the local temperature along a vertical chord and two electron cyclotron emission systems (GPC, GPC2) which measure the temperature at the midplane. The symbols are the same between the two figures. The fit and its 1- and 3-sigma error envelopes were computed using Gaussian process regression with the maximum a posteriori estimate for the hyperparameters.

locations are stored in can be converted using the `convert_abscissa(...)` method. Multiple `BivariatePlasmaProfile` instances from different diagnostics or even different shots can be combined as necessary using the `add_profile` method. The combined data set can then be used to predict a smoothed profile using Gaussian process regression [11] through integration with the `gptools` Python package [12]. An example of this sensor fusion task is shown in Figure 4. The use of `eqtools` to provide consistent coordinate transformations and a consistent interface for combining data from multiple diagnostics simplifies the task of building automated data analysis workflows.

## VII. 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 and PyPI [4], presents a consistent, user-friendly interface independent of data source or storage method. `eqtools` replaces machine- and storage-specific methods which are often nonintuitive to use and which 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 (*i.e.,* the $(R, Z)$ 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 presently used in the mapping routines at C-Mod), `eqtools` includes a novel trivariate spline implementation to provide 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 imple-

mentation. 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.

## Appendix A: Generation of tricubic spline coefficents for a regular grid

Significant time savings can be recovered for tricubic interpolation computation when the data grid ($f(x,y,z)$) 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 $4\times4\times4$ grid of values $\boldsymbol{y}$ to the coordinates of interest $x,y,z$ to

extract spline coefficients $\boldsymbol{x}$.

$$\boldsymbol{x} = \boldsymbol{A_{inv}}(\boldsymbol{B}\boldsymbol{y}) = \boldsymbol{A^*_{inv}}\boldsymbol{y} \tag{A1}$$

Where

$$\boldsymbol{A^*_{inv}} = \boldsymbol{A_{inv}}\boldsymbol{B} \tag{A2}$$

$\boldsymbol{y}$ comprises the values of the grid necessary for calculating the derivatives at the corners of the voxel $f$, and must be made into a from which is appropriate for $A_{inv}$. $A_{inv}$ expects the values of $f$ at each voxel corners then $\frac{\partial f}{\partial x}$, $\frac{\partial f}{\partial y}$, $\frac{\partial f}{\partial z}$, $\frac{\partial^2 f}{\partial x \partial y}$, $\frac{\partial^2 f}{\partial y \partial z}$, $\frac{\partial^2 f}{\partial x \partial z}$, $\frac{\partial^3 f}{\partial x \partial y \partial z}$ in the same manner. $\boldsymbol{B}$ converts the points into these derivatives, through first order finite difference matricies for $\frac{\partial f}{\partial x}$, $\frac{\partial f}{\partial y}$, $\frac{\partial f}{\partial z}$, second order finite difference matricies for $\frac{\partial^2 f}{\partial x \partial y}$, $\frac{\partial^2 f}{\partial y \partial z}$, $\frac{\partial^2 f}{\partial x \partial z}$, and a third-order for $\frac{\partial^3 f}{\partial x \partial y \partial z}$. The form of $\boldsymbol{B}$ is dependent on the form of $\boldsymbol{y}$ for the formation of the finite differences.

---


[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 package index (PyPI distribution)," `https://pypi.python.org/pypi/eqtools` (2015); "eqtools: Python tools for magnetic equilibria in tokamak plasmas (developer page)," `https://github.com/PSFCPlasmaTools/eqtools` (2015); "eqtools: tools for interacting with magnetic equilibria (online documentation)," `eqtools.readthedocs.org/en/latest/` (2015).

[5] IDL is a registered trademark of Exelis Visual Information Solutions.

[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).

[10] M. A. Chilenski, "profiletools: Classes for working with profile data of arbitrary dimension," `https://github.com/markchil/profiletools` (2015); "profiletools: Classes for working with profile data of arbitrary dimension (online documentation)," `http://profiletools.readthedocs.org/en/latest/` (2015).

[11] M. A. Chilenski, M. Greenwald, Y. Marzouk, N. T. Howard, A. E. White, J. E. Rice, and J. R. Walk, Nuclear Fusion **55**, 023012 (2015).

[12] M. A. Chilenski, "gptools: Gaussian processes with arbitrary derivative constraints and predictions," `https://github.com/markchil/gptools` (2015); "gptools: Gaussian processes with arbitrary derivative constraints and predictions," `http://gptools.readthedocs.org/en/latest/` (2015).