




pyEQL: A Python interface for water chemistry

Ryan Kingsbury ¹

¹ Department of Civil and Environmental Engineering and the Andlinger Center for Energy and the Environment, Princeton University, USA

DOI: [10.21105/joss.06295](https://doi.org/10.21105/joss.06295)

Software

- [Review](#) 
- [Repository](#) 
- [Archive](#) 

Editor: [Lucy Whalley](#)  

Reviewers:

- [@orionarcher](#)
- [@JacksonBurns](#)
- [@yuxuanzhuang](#)

Submitted: 08 November 2023

Published: 25 March 2024

License

Authors of papers retain copyright and release the work under a Creative Commons Attribution 4.0 International License ([CC BY 4.0](#)).

Summary

The properties and behavior of aqueous solutions – that is, water containing dissolved minerals and other solutes – are vital to understanding natural systems and to developing new technologies for water purification, wastewater treatment, and sustainable industrial processes ([Stumm & Morgan, 1999](#)). pyEQL provides object representations for aqueous solutions, creating a stable, intuitive, and easy to learn interface for calculating properties of solutions and dissolved solutes. Its purpose is to save researchers time by making a variety of different models accessible through a single interface and by aggregating hundreds of properties and model parameters into a built-in database.

Statement of need

Accurately predicting the thermodynamic and transport properties of complex electrolyte solutions containing many solutes, especially at moderate to high salt concentrations commonly encountered in water desalination and resource recovery applications, remains a major scientific challenge ([Rowland & May, 2019](#)). This challenge is compounded by the fact that the best available models, such as the Pitzer model ([May et al., 2011](#)), are difficult to implement on an as-needed basis and require looking up many parameters. Researchers and practitioners in fields such as water treatment and desalination, electrochemistry, or environmental engineering need accurate information about electrolyte solutions to perform their work, but are typically not specialists in solution chemistry or electrolyte thermodynamics.

Available software such as PHREEQC ([Charlton & Parkhurst, 2011](#)), GeoChemist's Workbench ([Aqueous Solutions, 2023](#)), or OLI Studio ([Wang et al., 2002](#)) implement numerous electrolyte models and contain powerful capabilities for specialists. However, they are not highly accessible for routine use by others, due to a steep learning curve, difficult interoperability with other tools (such as external transport models), the lack of a freely-available version, and/or limitation to specific operating systems. Several python interfaces to the open-source PHREEQC software exist, including IPhreeqC ([Parkhurst & Appelo, 2013](#)), phreeqpython (<https://github.com/Vitens/phreeqpython>), and pyeqion2 ([Marcellos et al., 2021](#)). However, these interfaces are either not object-oriented, poorly documented, and/or only offer access to only a limited subset of the PHREEQC parameter databases. There are more subtle limitations as well. For example, phreeqpython is unable to calculate solution conductivity when used in conjunction with the PHREEQC pitzer.dat database (the most accurate for high salinity solutions). A researcher seeking quality data on common bulk properties such as density or viscosity or solute-specific properties such as diffusion coefficient, transport number, or activity coefficient is thus left to piece together outputs from disparate models and literature – a time-consuming and error-prone process.

pyEQL is designed to free researchers from the tedium of identifying and implementing the relevant models and compiling the required parameters from literature. It defines a python Solution class from which properties can be easily retrieved. It implements the Pitzer model

(May et al., 2011) for binary salts, with mixing rules (Mistry et al., 2013) for more complex solutions, and decays gracefully to more approximate models like the Debye-Huckel activity model (Stumm & Morgan, 1999) when adequate data is not available. The built-in property database includes Pitzer model parameters (May et al., 2011) for more than 100 salts, diffusion coefficients (Vanýsek, 2011) for more than 100 solutes, and an ever-expanding set of additional property data that make the best-available models transparently accessible to the end user.

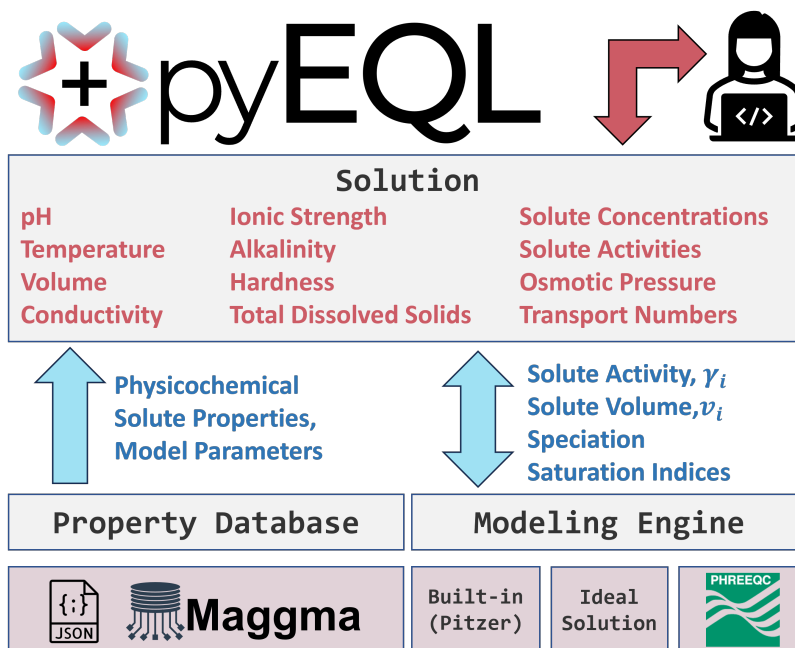


Figure 1: Overview of pyEQL's architecture. Properties such as ionic strength, conductivity, and concentrations are calculated directly by pyEQL. Modeling engines are used to calculate non-ideal effects such as activity coefficients, while property database stores necessary parameters. The modular design of the modeling engines and property database facilitate customization.

Example Use Cases

pyEQL may be useful to scientists and engineers in various fields broadly related to aqueous solution chemistry. Specific use cases include, but are not limited to:

- Calculating the osmotic pressure of concentrated salt solutions
- Estimating the speciation of complex electrolyte solutions at different pH values
- Calculating the transport number of a specific ion
- Computing bulk solution characteristics such as ionic strength, alkalinity, or total dissolved solids, given the composition of solutes
- Converting concentrations between different unit systems, e.g. moles per L, weight %, parts per million
- Looking up properties of individual ionic species, including molecular weight, diffusion coefficient, ionic, hydrated, and van der Waals radii, etc.

Design Principles

Return the best answer possible

Recognizing that accurate modeling of complex electrolyte solutions can be difficult or even impossible, pyEQL is designed to return **the best answer possible** given the data and models

available. For example, to calculate the osmotic pressure of a solution, the built-in modeling engine first attempts to use the Pitzer model, but if parameters are not available, it reverts to a more approximate formula rather than raising an error. To maintain transparency, log messages (and where appropriate, warnings) are generated throughout the codebase to document when assumptions or approximations have to be invoked or when important model parameters are missing from the database.

Interoperate with other scientific codes

pyEQL is built to be extensible, customizable, and easy to use in conjunction with widely-used scientific python libraries. Specifically, it makes use of `pint` (<https://github.com/hgrecco/pint>) to provide automatic unit conversions and leverages codes in the Materials Project (Jain et al., 2013) ecosystem – namely, `pymatgen` (Ong et al., 2013) for chemical informatics (e.g., molecular weight, parsing chemical formulae) and `maggma` (<https://github.com/materialsproject/maggma>) for accessing the built-in property database.

Architecture

The Solution class

The primary user-facing object in pyEQL is the `Solution` class. This class contains constitutive relationships for calculating most solution properties that depend on composition, such as total dissolved solids, ionic strength, density, conductivity, and many others (Figure 1). Calculations that require information about non-idealities (e.g., activity coefficients) are handled by a “modeling engine” that is stored in `Solution` as an attribute.

Modeling Engines

Every `Solution` contains a “modeling engine” which inherits from a base class defined in pyEQL. Modeling engines provide methods for calculating non-ideal thermodynamic corrections including solute activity coefficients and molar volumes as well as performing speciation. The results of these calculations are passed back to `Solution` where they can be transparently accessed by the user alongside other properties. This modular design facilitates connecting the `Solution` API to multiple modeling backends or software packages. Currently, the available modeling engines include an ideal solution approximation, a built-in implementation of the Pitzer model, and the PHREEQC modeling engine.

The Property Database and `Solute` class

pyEQL also provides `Solute`, a `dataclass` that defines a structured schema for solute property data. The database distributed with pyEQL is a list of serialized `Solute` objects stored in a `.json` file, which is accessed via the `maggma` Store API. The database used by a particular `Solution` instance can be specified by keyword argument when the object is created, which makes it possible in principle to use customized databases. Furthermore, using the Store API means that such databases can be stored in any format supported by `maggma` (e.g., Mongo Database, `.json` file, etc.).

Acknowledgements

The author gratefully acknowledges the Persson Research Group at the University of California, Berkeley, particularly Shyam Dwaraknath, Matthew K. Horton, Donny Winston, Jason Munro, and Orion Cohen, for their guidance in scientific software development practices. I also acknowledge Hernan Grecco for helpful discussions regarding unit conversion and Kirill Pushkarev, Dhruv Duseja and Andrew Rosen for recent contributions. The author acknowledges partial

financial support from Princeton University, Membrion, Inc., and Bluecell Energy, LLC over the period 2013-2023.

References

- Aqueous Solutions, L. (2023). *The geochemist's workbench, release 17*. <https://www.gwb.com/documentation.php>
- Charlton, S. R., & Parkhurst, D. L. (2011). Modules based on the geochemical model PHREEQC for use in scripting and programming languages. *Computers & Geosciences*, 37(10), 1653–1663. <https://doi.org/10.1016/j.cageo.2011.02.005>
- Jain, A., Ong, S. P., Hautier, G., Chen, W., Richards, W. D., Dacek, S., Cholia, S., Gunter, D., Skinner, D., Ceder, G., & Persson, K. A. (2013). Commentary: The materials project: A materials genome approach to accelerating materials innovation. *APL Materials*, 1(1). <https://doi.org/10.1063/1.4812323>
- Marcellos, C. F. C., Silva Junior, G. F. da, Amaral Soares, E. do, Ramos, F., & au2, A. G. B. J. (2021). *PyEquion: A python package for automatic speciation calculations of aqueous electrolyte solutions*. <https://doi.org/10.48550/arXiv.2101.07246>
- May, P. M., Rowland, D., Hefter, G., & Königsberger, E. (2011). A Generic and Updatable Pitzer Characterization of Aqueous Binary Electrolyte Solutions at 1 bar and 25 °C. *Journal of Chemical and Engineering Data*, 56(12), 5066–5077. <https://doi.org/10.1021/jc2009329>
- Mistry, K. H., Hunter, H. A., & Lienhard V, J. H. (2013). Effect of composition and nonideal solution behavior on desalination calculations for mixed electrolyte solutions with comparison to seawater. *Desalination*, 318, 34–47. <https://doi.org/10.1016/j.desal.2013.03.015>
- Ong, S. P., Richards, W. D., Jain, A., Hautier, G., Kocher, M., Cholia, S., Gunter, D., Chevrier, V. L., Persson, K. A., & Ceder, G. (2013). Python materials genomics (pymatgen): A robust, open-source python library for materials analysis. *Computational Materials Science*, 68, 314–319. <https://doi.org/10.1016/j.commatsci.2012.10.028>
- Parkhurst, D. L., & Appelo, C. A. J. (2013). *Description of input and examples for PHREEQC version 3: A computer program for speciation, batch-reaction, one-dimensional transport, and inverse geochemical calculations*. US Geological Survey. <https://doi.org/10.3133/tm6a43>
- Rowland, D., & May, P. M. (2019). Progress in Aqueous Solution Modelling: Better Data and Better Interfaces. *Journal of Solution Chemistry*, 48(7), 1066–1078. <https://doi.org/10.1007/s10953-019-00871-5>
- Stumm, W. (Ed.), & Morgan, J. J. (ed.). (1999). *Aquatic Chemistry: Chemical Equilibria and Rates in Natural Waters* (J. L. Schnoor & A. Zehnder, Eds.). Wiley Interscience. [https://doi.org/10.1016/S0016-7037\(97\)81133-7](https://doi.org/10.1016/S0016-7037(97)81133-7)
- Vanýsek, P. (2011). Ionic Conductivity and Diffusion at Infinite Dilution. In W. M. Hamner (Ed.), *CRC Handbook of Chemistry and Physics*.
- Wang, P., Anderko, A., & Young, R. D. (2002). A speciation-based model for mixed-solvent electrolyte systems. *Fluid Phase Equilibria*, 203(1-2), 141–176. [https://doi.org/10.1016/S0378-3812\(02\)00178-4](https://doi.org/10.1016/S0378-3812(02)00178-4)