# Abstraction of Linear Algebra Data Structures

**Google Summer of Code 2020 Proposal for QuTiP**

Jake Lishman*
Mentors: Eric Giguère and Alex Pitchford

13th May 2020

The core `Qobj` class in QuTiP specifically uses a custom `scipy`-derived sparse-matrix format for data storage, which allows simulation and optimisations on large open quantum systems but causes significant memory and computational overhead on smaller-dimensioned systems, and the 32-bit index size can prevent even extremely sparse systems of high numbers of qubits from being representable. This project will decouple the low-level data manipulation procedures from the algebraic manipulations performed within the rest of QuTiP, encapsulating the low-level data manipulation into a higher-level interface, which will allow multiple storage formats to be used at the appropriate situations transparently. Overall, QuTiP will be able to spread its impressive performance to all problem-size domains without compromising on future maintainability or extensibility as demands for numerical quantum simulations increase.

## 1 Technical Overview

The primary data structure in QuTiP is the `Qobj` class, which represents all linear operators, and elements of their associated vector and dual spaces. In large systems, explicitly storing a two-dimensional matrix containing every matrix element in some basis can quickly become excessively large—in dense form, a 12-qubit operator would require 256 MiB, a 15-qubit operator 16 GiB, and a 20-qubit system would be completely unrepresentable by any current computer. Instead, QuTiP has always used a "compressed sparse row" (CSR) format for matrix representation, where for each row only the non-zero elements are stored, along with the column indices for each entry and the number of non-zero elements in each row. This enables working with very large sparsely populated Hilbert spaces, even when the basis states to be represented and time-evolved may have much greater proportions of non-zero elements; an element of the vector or dual space has memory usage $\mathcal{O}(2^n)$ compared to $\mathcal{O}(2^{2n})$ for an operator on $n$ qubits.

If most elements of a matrix are zero, then in a dense format a lot of computational effort in linear operators is wasted multiplying zero by numbers and summing the

*jake@binhbar.com

results. This particular sparse format is very efficient for calculating matrix left-actions, such as $\hat{M}|\psi\rangle$, but less efficient when being acted upon *from* the left where column-lookup overheads can begin to cause performance to drop below traditional dense matrix representations (assuming the matrix can fit in memory). At small system sizes too where the proportion of non-zero elements is high, the need to store column indices and non-zero counts causes significant computational and storage overheads, generally making sparse matrices far less desirable in these regimes. In QuTiP this is already a known problem, discussed in detail in #437, with further issues raised in #818, #831, #845 and #853, amongst others, where a range of memory and calculation time concerns have been identified. In the `qutip.control` module, the `Qobj` class is bypassed in order to use a dense matrix representation, since these control problems typically concern smaller, controllable systems, but this is undesirable as it causes a very high-level portion of the interface to have to address low-level data manipulations. Several parts of QuTiP now directly manipulate the internal CSR-formatted data stored in `Qobj` for speed reasons, essentially locking-in the format, making the previous issues difficult to solve.

This project aims to overhaul all access to the QuTiP data internals, forcing them to be done through a well defined interface, and removing direct access. It will allow `Qobj` to use any representation which implements the interface, allowing the user or internal functions to switch between them as necessary to ensure high performance in all problem sizes. Aside from the improved calculation abilities, this will also simplify maintenance and debugging, since it will force a strong separation of concerns between different layers of the package.

## 2 Timeline

Project work officially starts on the 1st of June, and numFOCUS require "regular blog posts" to track progress throughout the three months. The breakdown here is approximate and (hopefully) achievable in scope, as with such a far-reaching proposal it is difficult to anticipate where the major difficulties will lie.

*Before official start date, or community bonding*

- Continue small contributions to QuTiP development.
- Familiarise myself more with the `QobjEvo` internals and Cython intricacies.

*1st of June until 21st of June (three weeks)*

- Investigate the extended effects of any abstraction.
- Discuss, specify and document the necessary interface for any underlying data representation in consultation with the core team.
- Identify the main obstacles to separation of the data storage and linear-algebraic-manipulation layers of the package stack.

*22nd of June until 19th of July (four weeks)*

- Write profiling tests of the current code to allow tracking of the resource consumption in actual use as the core objects evolve. These would not be part of the default test suite, as they would by necessity need to take a reasonable amount of time to run.
- Implement the separation of concerns of the data storage, without adding in additional storage methods or new functionality—this step is purely about ensuring that all existing code is moved to the new interface without significant degradation of performance.
- Attempt to isolate uses of Cython, and, if reasonable in the time frame, provide an alternative either manually with `numba` or Pythran, or general with `transonic`.

*19th of July until 2nd of August (two weeks)*

- Add the dense `numpy.ndarray` as an allowable data store. This is the primary goal of the project.
- Verify that `QobjEvo` in particular can use both sparse and dense matrices transparently.
- Make any necessary changes to the data-storage interface specification with the new knowledge gained.
- Add new tests for this new data abstraction.

*3rd of August until 23rd of August (two weeks)*

- Find suitable heuristics and implement matrix conversions in places in the codebase where a change in representation could result in an increase in performance.
- Add a 64-bit `int` version of the sparse matrix format to allow representation of massive problem sizes.
- Ensure all new matrix representations are documented and tested.

*24th of August until 31st of August (final week)*

- Final check over documentation and tests.
- Write final report, project summary and evaluation.

# 3 Future Extensions

The general concept of abstraction of quantum objects leads to some rather interesting prospects for enhancement of QuTiP's feature set. The most immediate would be the possibility of reducing the amount of Cython code that is absolutely necessary to the functioning of QuTiP, which is a fairly large source of the complexity in maintaining and distributing QuTiP effectively.

Looking at a much larger picture, however, and far beyond the scope of this proposal, one could imagine that a complete abstraction of the underlying data representation could pave the way to full symbolic representation and manipulation of quantum objects. This is already beginning to be brought to bear within QuTiP—the addition of `QobjEvo` allows the representation of general time-dependent objects with time left unspecified. It would be interesting to consider the unification of `Qobj` and `QobjEvo` under this idea of abstraction of storage formats, with a possible extension being to allow the symbolic representation of operators and spaces whose truncation is not yet specified. To allow these to seamlessly inter-operate, and remove the need for quadratically scaling numbers of `if isinstance(other, QobjVariant9)` blocks in interaction code, we would want to unify the object hierarchy, most likely under two or three abstract interfaces (rather than instantiatiable objects) and rely on the interfaces' interactions with each other, similarly to how we intend to work with the multiple data representations in this project.

# 4 My Experience

I am a final-year PhD student working on optimal control of trapped ions under Dr. Florian Mintert at Imperial College London, often doing numerical simulations in Python using QuTiP. I have been able to open-source several parts of my work in particular the highly-optimised Floquet formalism solver I worked on (section 4.3), and I have already made several commits to the QuTiP `master` branch. I cannot, however, immediately share my most recent PhD code because there are intellectual property issues that I am not certain about. Other more nebulous bits of my personal code without these restrictions are visible on my GitHub page[1].

## 4.1 Prior QuTiP Contributions

I have four pull requests merged to the QuTiP `master` branch already:

#1159: Implicit tensor product for `qeye`, `qzero` and `basis`
: Refactored the code paths for `qeye([2,2])` and `qzero` where a tensor-product state is created so that nested lists were not flattened. Added a similar ability to create direct tensor products in `basis`, as in `basis([2,2], [0,1])` creating the qubit state $|01\rangle$. Closed (or should be closed!) #363.

#1161: Remove duplicated test runners
: Fixed a corrupted merge resulting in both `nose` and `pytest` test runners being present in `testing.py`. Also converted `test_qobj.py` to a rough `pytest` style, due to a `nose`-specific test case. Fixed #1158.

#1164: Move tests to `pytest`
: Began the work of converting the large test suite to the more modern `pytest` framework.

[1] https://www.github.com/jakelishman

#1206: Deprecate `qutip.graph` functions
: Marked the `qutip.graph` module as deprecated, pending a move to convert it to the `scipy` versions to simplify the codebase.

I also have two more that are currently pending, and am actively contributing to:

#1181: Convert tests to `pytest`
: Continues the work of #1164, but much more in depth as I have learnt more about `pytest` and have converted more files.

#1194: Make LaTeX image conversion more resilient
: Fixes several bugs found in the generation of images of quantum circuits, including making it play nicely with IPython when optional dependencies are missing. Fixes #1179 and #1185.

### 4.2 Instrument Control

During my undergraduate master's degree in physics at the University of Warwick and for three months immediately after graduation I worked as a paid software engineer writing instrument control software in F# for the lab of Dr. Gavin Morley with two PhD students. Working in this team is where I really learnt how to use `git` properly, and the functional style we wrote in made me a more well-rounded programmer, having originally learned pure C. A lot of that code is available on the Warwick EPR Github pages[2], especially the `Endorphin.Instrument.Keysight.N5172B` and `Endorphin.Core` modules, though I was active in several more during the summers of 2015 and 2016.

### 4.3 Numerical Floquet Solver

At the very beginning of my PhD I inherited a previous Master's student's Floquet calculation code[3], which is now published under the Imperial Controlled Quantum Dynamics group on GitHub. I was able to heavily optimise this, reducing calculation times by several orders of magnitude for large systems, in part by implementing a custom column sparse matrix format that was accessible by `numba` in `57af111`[4] (I'm also especially proud of several other commits around that time, too!). Very notably for this project, I also implemented a form of abstraction of the input data types in `5dff87e`[5], although this was not as far-reaching as the changes proposed here.

[2] https://www.github.com/WarwickEPR
[3] https://github.com/ImperialCQD/floq
[4] https://github.com/ImperialCQD/floq/commit/57af111a5af768ad184b1278a0428071af6482a5
[5] https://github.com/ImperialCQD/floq/commit/5dff87e427c01b09ca0b3cb5e4ca25c64c51294f

### 4.4 Symbolic Linear Algebra

Most recently I have been working on optimised symbolic calculations with linear operators in Python as part of a recent arXiv paper[6]. We needed to perform Baker–Campbell–Hausdorff-type expansions with operator power series in a coupling parameter, and keep track of all coefficients and powers through several orders of the expansion, algebraically simplifying all sums to keep the memory and time usage manageable. Using this, we were able to calculate Magnus expansions to ninth order and perform fully analytic propagator expansions with what would have been hundreds of thousands of functional terms without simplification in pure Python.

## 5 Why This Project?

I have used QuTiP a lot throughout my PhD, and I started contributing to the development of it two months ago as a way of giving back. I am particularly interested in optimising the low-level components of the package without compromising end-user experience, and I want to gain more experience in more advanced software architecture and team development. Working on the QuTiP internals with this project satisfies all of these, and could be a valuable contribution to the project just by nature of having somebody be free enough to make the far-reaching changes in a relatively short amount of calendar time, so that the inevitable merge at the end is as painless as possible.

[6] https://www.arxiv.org/abs/2003.11718