**Program benchmarks (laptop)**

**Solid effect:**

Matlab (s): 9.29

Python + NumPy (s), Anaconda: 31.06

Python + NumPy (s), Enthought: 29.22

Python + NumPy (s), Intel: 29.64

Python + F2PY (s): 4.18

Python + F2PY + OpenMP (s): 3.19 \*

**Calculating Liouville space propagator:**

Matlab (s): 8.46

Python + NumPy (s): 26.6

Python + F2PY (s): 1.78

Python + F2PY + OpenMP (s): 0.74 \*

*\*Currently there is a bug(?) associated with access violation when running multithreaded Kronecker product function, which may be limiting performance. A fix is to apply OpenMP statements to the function, however this needs to be investigated further.*

*\* Currently there is only a partial F2PY implementation, limited to Hamiltonian and propagator calculation.*

The significantly higher performance of Matlab for all tested matrix operations is puzzling, as both Python and Matlab are linked to LAPACK Fortran library for matrix multiplication. The Fortran benchmark supports this to a degree however it appears Matlab is somehow significantly more optimised.

In conclusion Matlab is significantly faster than pure Python, however when combined with F2PY Python may perform up to an order of magnitude faster than Matlab. More work needs to be done to ensure Python speeds are representative, and to investigate other Fortran compilers and matrix multiplications.

**General benchmarks (laptop)**

**Calculating 1E7 matrix products (2x2 real):**

Matlab (s): 4.99

Python + NumPy (s): 9.69

Fortran (s): 6.97

Fortran + BLAS (s): 0.26

**Calculating 1E6 matrix products (8x8 real):**

Matlab (s): 0.73

Python + NumPy (s): 1.07

Fortran (s): 0.96

Fortran + BLAS (s): 0.35

**Calculating 1E6 matrix products (8x8 complex):**

Matlab (s): 1.33

Python + NumPy, Anaconda (s): 2.12

Python + NumPy, Enthought (s): 2.11

Python + Numpy, Intel (s): 2.15

Fortran (s): 1.90

As expected when inputs are both small real matrix operation is significantly faster, while when complex or larger are slower. The same pattern between functions and languages is present, showing that function dgemm() provided by BLAS constantly outperforms the intrinsic function matmul(). However, dgemm() only works on real matrices so its applications are limited. There appears to be no difference between the performance of three tested Python interpreters, as all are linked to Intel MKL implementation of LAPACK. The performance of other Fortran compilers such as ifort has not been examined, however gfortran is known to perform similarly despite being open source. Matlab is likely capable of the most significant pre-optimisation, hence it outperforms all other languages except for the dgemm() function. Further research is required to confirm these relationships and provide further justification. Check Fortran benchmarks on a Linux installation to ensure Cygwin/MingGW are not causing decreased performance.

**Calculating 1E5 Kronecker products:**

Matlab (s): 2.32

Python + NumPy, Anaconda (s): 4.45

Python + NumPy, Enthought (s): 4.37

Python + NumPy (s), Intel: 4.49

Fortran (s): 1.65

**Calculating 1E4 matrix exponentials:**

Matlab (s): 1.37

Python + SciPy, Anaconda (s): 4.35

Python + SciPy, Enthought (s): 4.32

Python + SciPy, Intel (s): 4.23

Fortran + Expokit (s): 0.21