**Program benchmarks (laptop Intel i7 3.2MHz)**

**Solid effect (original relaxation, 1 run):**

Matlab (s): 9.58

Python + NumPy (s): 14.19

Fortran (s): 1.87

Fortran + OpenMP (s): 0.91 **(11x faster)**

**Solid effect (Mance relaxation, 1 run):**

Matlab (s): 8.42

Python + NumPy (s): 12.48

Fortran (s): 1.82

Fortran + OpenMP (s): 0.86 (10x faster)

**Solid effect (original relaxation, 40 runs):**

Matlab (s): 286.4

Python + NumPy (s): 551.4

Fortran (s): 75.5

Fortran + OpenMP in series (s): 35.4

Fortran + OpenMP in parallel (s): 20.6 **(14x faster)**

**Cross effect (original relaxation, 1 run):**

Matlab (s): 33.5

Python + NumPy (s): 68.4

Fortran (s): 67.4

Fortran + OpenMP (s): 22.2

**Cross effect (Mance relaxation, 1 run):**

Python + NumPy (s): 67.8

Fortran (s): 63.9

Fortran + OpenMP (s): 22.3

**General benchmarks (laptop Intel i7 3.2MHz)**

**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

**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

**Performance:**

For a single run Fortran performs around 11x faster than Matlab, and for a parallel run of 40 jobs performs 14x faster. It should be noted that parallel performance of Matlab and Python has not been tested, however as neither language support ‘true’ multithreading the gains are likely to be negligible. The greater performance of Fortran over Matlab and Python is expected as Fortran is a language optimised for High Performance Computing, however it is interesting that Matlab can defeat Fortran in certain areas such as matrix multiplication. This is likely due to the highly optimised LAPACK interface used by Matlab, able to pre-optimise matrix operations. It is likely that this is also responsible for the greater performance of Matlab over Python; as the bulk of the computational cost is in performing matrix operations, the LAPACK implementation is the deciding factor in the overall speed.

**Relaxation:**

It appears that the Mance relaxion method is consistently faster than the original method, for all three tested languages and for both cross effect and solid effect. This is likely due to the high computational expense of Kronecker products used in the original method, despite the addition of many for loops in the Mance relaxation method which are typically slow in languages such as Matlab and Python. In addition, the Mance relaxation method requires ordering of eigenvalues and eigenvectors (as performed automatically by ‘Eigenshuffle’ in Matlab), however this adds a negligible cost of around 1ms for solid effect and 9ms for cross effect. Therefore, the Mance relaxation method can be recommended for use without any reservations.

**Further work:**

* Look into stability of Fortran code (appears to be a few bugs remaining)
* Look into very low performance of Fortran cross effect code
* Consider BLAS matrix multiplication
* Consider re-designing Kronecker product function, optimising for use with identity matrices
* Consider reducing precision of Fortran calculations
* Look into further Fortran optimisations
* Look into changing number of timesteps

**Extensions:**

* Powder averaging
* Cluster version
* Python GUI using F2PY
* Four spin system e-e-n-n



