## Spinach package – Hilbert space module

Familiarity with the previous sections of the manual are assumed.

The Hilbert space module makes use of parallelization techniques that are not possible in Liouville space. Such functionality requires the latest version of the *Parallel Computing Toolbox* in *MatLab.* In order to see performance gains over the serial computation one must thus first have a multicore box and open the `MatLab pool’ using the command matlabpool open *n* where *n* is the number of cores one wishes to use. N.B. the *Parallel Computing Toolbox* allows for up to 8 cores; to use more requires the *Distributed Computing Toolbox*.

1. **Similarities to the Liouville space package**

Most of the functionality of the Hilbert space module has been designed to be outwardly familiar to those used to *Spinach.* The functions are named similarly (with a prepended hs\_) and many of the same kernel functions are called.

1. **Differences with the Liouville space package**

The major differences are in the absence of functionality compared with the Liouville space simulation tools.

Relaxation and kinetics are best described (from both a theoretical and computational perspective) in Liouville space and as such implementation of either is left up to the user.

Basis restriction is not performed prior to construction of the Hamiltonian.

There are no predefined /exp/\* files. Such files may be readily constructed by aping those used for Liouville space simulations. Some caveats:

1. 2D-experiments are currently not implicitly carried out by hs\_evolution().
2. There is no *‘refocus’* option for hs\_evolution().
3. `Destination state screening’ has not been implemented in hs\_evolution().
4. Operators and states are equivalent in Hilbert space (i.e. where state() is called in the Liouville space functions one should use hs\_operator()).
5. Operators should always be Hermitian (with the exception of the *coil* operator, which may be non-Hermitian).
6. **Specific functionality**

Here follows a brief description of the functions that may be found in the /kernel/hilbert\_space directory, and how they differ from their equivalent functions in Liouville space (where an equivalent function exists).

1. **hs\_equilibrium()**

Generates the equilibrium density matrix.

Equivalent functionality and syntax to equilibrium(). N.B. that populations will be half those generated by equilibrium() (and thus spectra will be half as intense) because in the Liouville space calculations the density matrix is normalized to the trace/2.

1. **hs\_evolution()**

Evolve a given density matrix under a given Hamiltonian.

This is the function in which parallelization has been implemented. There are currently three output options: *‘final’*, *‘trajectory’*, and *‘observable’* and all three use different methods of parallelization.

*‘final’* operates in the same way as the similar option in evolution().A single density matrix is returned.

*‘trajectory’* returns a cell array of density matrices.

*‘observable’* returns a 1D array of complex numbers corresponding to the expectation values of the operator *‘coil’* at each specific point.

1. **hs\_generators()**

Obtain the generating operators for a given spin.

Function called by hs\_hamiltonian() and hs\_operator(). Expert users may call this function directly but it should normally be unnecessary: for most purposes hs\_operator()serves as an appropriate wrapper function.

1. **hs\_hamiltonian()**

Generate the Hamiltonian of the system given certain parameters pertaining to the coupling frequencies, magnetic field strength, etc.

Equivalent in terms of user syntax to h\_superop().

1. **hs\_operator()**

Generate operators (and states) in a user-friendly way.

Accepts *‘Lx’, ’Ly’, ’Lz’, ’L+’, ‘L-’* as valid operatorspecifications. Note that spherical tensor specifications of the form *‘Tl,m’* are not accepted (though may be easily constructed using operators that can be produced from well-known formulae by the user if required).

1. **hs\_step()**

Propagate a given density matrix under a given operator for a single time-step (i.e. a hard pulse).

Can currently only deal with an individual density matrix thus must be looped over for 2D-experiments.