QuaC User Manual

Matthew Otten

August 29, 2016

Abstract

Basic notes on using QuaC, a Lindblad master equation solver written in C.

1 Installation

QuaC is designed to run on Linux systems/clusters, as well as Mac OSX computers. QuaC can be downloaded via `git` by doing a clone,

```
git clone https://github.com/0tt3r/QuaC.git
```

or, if `git` is not available, via wget

```
wget https://github.com/0tt3r/QuaC/archive/master.
   zip
```

QuaC is built upon PETSc; without a PETSc installation, QuaC will fail to compile. PETSc has an extensive set of installation options (which can be viewed at http://www.mcs.anl.gov/petsc/documentation/installation.html); QuaC requires the option `-with-scalar-type=complex`. To install PETSc from scratch (which will also install BLAS/LAPACK and MPICH), do (assuming bash as the shell, and that GCC compilers are available)

```
git clone -b maint https://bitbucket.org/petsc/
   petsc petsc
cd petsc/
export PETSC_DIR=$(pwd)
export PETSC_ARCH=linux-gnu-c
./configure --with-cc=gcc --with-cxx=g++ --with-fc
   =gfortran --download-fblaslapack --download-
   mpich --with-scalar-type=complex
make all
make test
```

Furthermore, the environment variables `PETSC_DIR` and `PETSC_ARCH` will need to be set everytime QuaC is used; the easiest way to do this is by putting `export PETSC_DIR=<dir/where/petsc>` and `export PETSC_ARCH=linux-gnu-c` in a

.bashrc or .profile type file. For more installation instructions for PETSc (such as using an already installed version of BLAS or MPI), please see their website. Once PETSc is installed and QuaC is downloaded, all that is left is to run a simulation. There are several examples within the `src` directory. To run an example, try:

```
make simple_jc_test
./simple_jc_test
```

within the QuaC directory. This should print out the steady state populations for a simple Jaynes Cummings like Hamiltonian. In the following section, we will discuss creating such a system from scratch.

2 Creating a New System

QuaC was designed to allow the user to go from the physics to the computations with as little pain as possible. To illustrate this concept, we will use a simple Jaynes-Cummings Hamiltonian coupled to a thermal resonator as our physics and show the steps necessary to program the physics and run the code.

2.1 Hamiltonian

The Jaynes-Cummings model describeds the dynamics of a two level system coupled to an oscillator mode (be it a mechanical resonator, a plasmonic system, or an electromagnetic field):

$$H = \omega_a a^\dagger a + \omega_\sigma \sigma^\dagger \sigma + g(\sigma^\dagger a + \sigma a^\dagger), \tag{1}$$

where $a^\dagger$ is the creation operator of the oscillator, ω_s is the frequency of the oscillator, $\sigma^\dagger$ is the creation operator of the two level system, ω_σ is the transition frequency of the two level sistem, and g is the coupling strength between the two systems. We are using units such that $\hbar = 1$. To create this system within QuaC, we first have to create operators for each subsystem, via

```
create_op(number_of_levels,&op_name)
```

where `number_of_levels` is the number of levels of the operator and `op_name` is a previously declared variable of the `operator` type. The `operator` type is a special QuaC object that is used in the creation of the Hamiltonian and Lindblad terms. One call to `create_op`, with a single `op_name` creates the annihilation, creation, and number (i.e. $a^\dagger a$) operators. `op_name` is the annihilation operator within the subsystem's Hilbert space, `op_name->dag` in the creation operator, and `op_name->n` is the number operator. Each of these three should be thought of as opaque objects; internally, they store all the necessary information for the creation of the Hamiltonian matrix which uses those terms, but they should never be directly manipulated by the user.

For our example, we need to create two operators: one for the oscillator, and one for the two level system. Trivially, we know the number of levels for the two level system, but it is not so clear for the oscillator. Formally, the oscillator has infinite levels. Practically, there will be some maximum excitation number for a given problem. The infinite ladder should be truncated at a point in which adding more levels has minimal impact on the dynamics. In this example, we will use 25. Two create our two operators, we call

```
create_op(2,&sigma)
create_op(25,&a)
```

where `sigma` and `a` are previously declared variables of the `operator` type. With our operators defined, we can now construct the Hamiltonian matrix. In QuaC, Hamiltonians are created term by term, with calls to the `add_to_ham` family of functions. To add a term which has a single operator (including the number operator, $a^\dagger a$, as a 'single operator'), we use

```
add_to_ham(scalar,op_name)
```

where `scalar` is the scalar that multiplies the operator, `op_name`. In our Jaynes-Cummings model, we need to add two such terms to the Hamiltonian, $\omega_a a^\dagger a$ and $\omega_\sigma \sigma^\dagger \sigma$. We would call

```
add_to_ham(omega_a,a->n)
add_to_ham(omega_s,sigma->n)
```

where `omega_a` and `omega_s` are previously declared (and assigned) numbers, and `a->n` and `sigma->n` are the number operators for the oscillator and two level system. The coupling terms in our Hamiltonian have two operators ($g(\sigma^\dagger a + \sigma a^\dagger)$), so we will need to use

```
add_to_ham_mult2(scalar,op_name1,op_name2).
```

Since we have the sum of two such terms, we will need to call this function twice:

```
add_to_ham_mult2(g,sigma->dag,a)
add_to_ham_mult2(g,sigma,a->dag)
```

where g is a previously declare and assigned number. Here we have used both the annihilation and creation operators of the `operator` type. We have now taken our initial Hamiltonian of eq. (1) and told QuaC everything it needs to do a calculation using that Hamiltonian.

2.2 Lindblad Terms

QuaC is optimized for open quantum systems. As such, we need to include nonunitary evolution. QuaC does this through the Lindblad superoperator. The Lindblad superoperator is defined as

$$L(C)\rho = C\rho C^\dagger - \frac{1}{2}(C^\dagger C\rho + \rho C^\dagger C), \tag{2}$$

where C is an operator. The Lindblad superoperator can effectively describe dissipation and dephasing. For example, our TLS might have some spontaneous emission with rate γ_σ. This could be included using a Lindblad term of the form $\gamma_\sigma L(\sigma)\rho$ in our master equation. In QuaC, the general function for adding Lindblad terms is

```
add_lin(scalar,op_name)
```

where `scalar` is the rate, and `op_name` is the associated operator. In our example, we want to add $\gamma_\sigma L(\sigma)\rho$, so we call

```
add_lin(gamma_s,sigma)
```

where `gamma_s` is the (previously declared and assigned) spontaneous emission rate of our TLS and `sigma` is the annihilation operator which we previously created. Our oscillator can also have Lindblad terms. For this example, let's say that the oscillator is coupled to a thermal bath with thermal occupation number n_{th} and has a linewidth γ_a. This would give two Lindblad terms: $\gamma_s(n_{th}+1)L(a)\rho + \gamma_s n_{th} L(a^\dagger)\rho$. To add these terms in QuaC, we call

```
add_lin(gamma_a*(n_th+1),a)
add_lin(gamma_a*n_th,a->dag)
```

We could, of course, add additional Lindblad terms (such as environmental dephasing), but, for our simple example, this will suffice.

2.3 Full QuaC Code

For illustrative purposes, we collect all the calls needed here:

```
create_op(2,&sigma) //Create the TLS
create_op(25,&a)    //Create the oscillator

//Add terms to the Hamiltonian
add_to_ham(omega_a,a->n)
add_to_ham(omega_s,sigma->n)

add_to_ham_mult2(g,sigma->dag,a)
add_to_ham_mult2(g,sigma,a->dag)

//Add Lindblad terms
add_lin(gamma_s,sigma)
add_lin(gamma_a*(n_th+1),a)
add_lin(gamma_a*n_th,a->dag)
```

Neglecting variable declaration and assignment, these nine lines of code are all that is necessary to construct our simple Jaynes-Cummings system. In the next section, we will discuss how to use the constructed system to solve for both the steady state and time dependence.

3 Solving the System

QuaC has two modes for using the constructed system: solving for the steady state of the system, and solving for the full time dependence.

3.1 Steady State

Solving for the steady state in QuaC simply involves a call to

```
steady_state()
```

Once this call is executed, QuaC solves for the steady state of the system, using, as default, additive Schwarz method preconditioned GMRES. The selection of algorithm can be changed at runtime using PETSc's flexible command line interface (see section on utilizing PETSc). The initial condition is unimportant, as the solver goes to the (unique?) steady state from any initial condition rapidly (maybe?). TODO: Add information about getting population/concurrence/etc after the solve and results

3.2 Time Dependence

To solve for the time dependence, we must first set the initial conditions. Initial conditions are set per subsystem, and the system is restricted to starting in a pure state (though this limitation will be removed in a future version). If no initial condition is set for a subsystem, it is assumed to be in the $|0\rangle$ state initially. In QuaC, to set the initial population of a subsystem, you call

```
set_initial_pop(op_name,pop)
```

where `op_name` is an operator and `pop` is the population to add. `op_name` can be any of the three basic operators created by QuaC in the initial `create_op` call; there is no difference between the three for this routine. By convention, it is best to use the annihilation operator. `pop` will be understood as an integer, and must be less than the total number of levels for that subsystem. This stems from the fact that `set_initial_pop` is really setting the initial condition of the subsystem as a pure state of the form $|\texttt{pop}\rangle\langle\texttt{pop}|$. For our Jaynes-Cummings example, we would call

```
set_initial_pop(a,pop_a)
set_initial_pop(sigma,pop_sigma)
```

With the initial conditions set, we can now proceed to do the timestepping using

```
time_step(time_max,dt,steps_max)
```

where `time_max` is the maximum time to integrate to, `dt` is the initial time step, and `steps_max` is the maximum number of time steps to take. The default solver is an explicit, adaptive time step Runge-Kutta method. QuaC (through PETSc) also supports other explicit, as well as implicit, time steppers. Algorithms can be chosen at runtime; see the section on PETSc to learn more about these options. QuaC will then run the time dynamics until it either reaches the end time or completes the maximum number of steps.

3.2.1 Printing Results at Each Time Step

To get results at each time step, a user defined function must be declared and passed to QuaC via the function

```
  set_ts_monitor(function_name)
```

`function_name` must have this particular form:

```
PetscErrorCode function_name(TS ts,PetscInt step,
    PetscReal time,Vec dm,void *ctx)
```

where `ts` is an opaque object which holds information about the timestepping algorithm (and need not be touched by the user), `step` is the current step number, `time` is the current time, `dm` is the (vectorized) density matrix (and should be though of as an opaque object to be passed to other routines), and `ctx` is another opaque object that the user should not touch. Within the ts_monitor function, the user can call a variety of functions to obtain and print different observables and metrics. To print the current populations to a file name `pop`, you call

```
  get_populations(dm,time)
```

This routine will calculate the populations for each subsystem and print them to the file `pop`. TODO: Add observables (`get_observables`), concurrence, fidelity, etc The ts_monitor function must have `PetscFunctionReturn(0)` as its final line.

4 Unequally Spaced Operators

5 Using PETSc Command Line Options