Bloch sphere representation of a qubit. The probability amplitudes for the superposition state $|\psi\rangle$ is $\alpha|0\rangle + \beta|1\rangle$, are given by $\alpha = \cos(\frac{\theta}{2})$ and $\beta = e^{i\phi}\sin(\frac{\theta}{2})$.

- Quantum computing: computing using quantum-mechanical phenomena, such as superposition and entanglement.
- Quantum algorithm: algorithm that runs on a realistic model of quantum computation, such as quantum circuit model
- Quantum circuit: model in which a computation is a sequence of quantum gates.
Promise of Quantum Computing

- A paramount goal in the field of quantum computing is to perform an efficient quantum calculation that cannot be simulated in any reasonable amount of time on even the most powerful classical computer.
Quantum Optimal Control

- Quantum control:
  - Use to drive a quantum system/model to a desired state
  - Manipulate dynamical processes at the atomic or molecular scale, typically by means of external electromagnetic fields or forces.

- Quantum optimal control:
  - Devise and implement shapes of pulses of external fields or sequences of such pulses that reach a given task in a quantum system in the best way possible.
Quantum Optimal Control

Intrinsic Hamiltonian: $H_0$

An initial state: $|\psi_0\rangle$

Control operators: $H_0, \ldots, H_M$

QOC: Determine, for a sequence of timesteps $t_0, t_1, \ldots, t_n$ a set of control fields $u_{j,k}$ such that

$$H_j = H_0 + \sum_{k=1}^{m} u_{j,k} H_k$$

$$U_j = e^{-iH_j(t_j - t_{j-1})}$$

$$K_j = U_j U_{j-1} U_{j-2} \ldots U_1 U_0$$

$$|\psi_j\rangle = K_j |\psi_0\rangle.$$  

Objective: Minimize

$$F_0 = 1 - \left| \text{Tr}(K_n^T K_n) / D \right|^2$$

Quantum Optimal Control

Intrinsic Hamiltonian: $H_0$

An initial state: $|\psi_0\rangle$

Control operators: $H_0, \ldots, H_M$

QOC: Determine, for a sequence of timesteps $t_0, t_1, \ldots, t_n$ a set of control fields $u_{j,k}$ such that

$$H_j = H_0 + \sum_{k=1}^{m} u_{j,k} H_k$$

$$U_j = e^{-i H_j (t_j - t_{j-1})}$$

$$K_j = U_j U_{j-1} U_{j-2} \ldots U_1 U_0$$

$$|\psi_j\rangle = K_j |\psi_0\rangle.$$ 

Objective: Minimize

$$F_0 = 1 - |\text{Tr}(K_T^\dagger K_n)/D|^2$$

Quantum Optimal Control

Intrinsic Hamiltonian: $H_0$

An initial state: $|\psi_0\rangle$

Control operators: $H_0, \ldots, H_M$}

QOC: Determine, for a sequence of timesteps $t_0, t_1, \ldots, t_n$ a set of control fields $u_{j,k}$ such that

$$H_j = H_0 + \sum_{k=1}^{m} u_{j,k} H_k$$

$$U_j = e^{-iH_j(t_j-t_{j-1})}$$

$$K_j = U_j U_{j-1} U_{j-2} \ldots U_1 U_0$$

$$|\psi_j\rangle = K_j |\psi_0\rangle.$$ 

Objective: Minimize

$$F_0 = 1 - |Tr(K_T^\dagger K_n)/D|^2$$

Quantum Optimal Control

Intrinsic Hamiltonian: $H_0$

An initial state: $|\psi_0\rangle$

Control operators: $H_0, \ldots, H_M$

QOC: Determine, for a sequence of timesteps $t_0, t_1, \ldots, t_n$ a set of control fields $u_{j,k}$ such that

$$
\mathcal{H}_j = H_0 + \sum_{k=1}^{m} u_{j,k} H_k
$$

$$
U_j = e^{-i\mathcal{H}_j(t_j-t_{j-1})}
$$

$$
K_j = U_j U_{j-1} U_{j-2} \ldots U_1 U_0
$$

$$
|\psi_j\rangle = K_j |\psi_0\rangle.
$$

Objective: Minimize

$$
F_0 = 1 - |\text{Tr}(K_T^n)/D|^2
$$

Quantum Optimal Control

Intrinsic Hamiltonian: $H_0$

An initial state: $|\psi_0\rangle$

Control operators: $H_0, \ldots, H_M$

QOC: Determine, for a sequence of timesteps $t_0, t_1, \ldots, t_n$ a set of control fields $u_{j,k}$ such that

$$
H_j = H_0 + \sum_{k=1}^{m} u_{j,k} H_k
$$

$$
U_j = e^{-iH_j(t_j-t_{j-1})}
$$

$$
K_j = U_j U_{j-1} U_{j-2} \ldots U_1 U_0
$$

$$
|\psi_j\rangle = K_j |\psi_0\rangle.
$$

Objective: Minimize

$$
F_0 = 1 - \left| \text{Tr}(K_T^T K_n)/D \right|^2
$$

Quantum Optimal Control

Intrinsic Hamiltonian: $H_0$

An initial state: $|\psi_0\rangle$

Control operators: $H_0, \ldots, H_M$

QOC: Determine, for a sequence of timesteps $t_0, t_1, \ldots, t_n$ a set of control fields $u_{j,k}$ such that

$$\mathcal{H}_j = H_0 + \sum_{k=1}^{m} u_{j,k} H_k$$

$$U_j = e^{-i\mathcal{H}_j(t_j-t_{j-1})}$$

$$K_j = U_j U_{j-1} U_{j-2} \ldots U_1 U_0$$

$$|\psi_j\rangle = K_j |\psi_0\rangle.$$ 

Objective: Minimize

$$F_0 = 1 - |\text{Tr}(K_T^\dagger K_n)/D|^2$$

Quantum Optimal Control

Intrinsic Hamiltonian: $H_0$

An initial state: $|\psi_0\rangle$

Control operators: $H_0, \ldots, H_M$}

QOC: Determine, for a sequence of timesteps $t_0, t_1, \ldots, t_n$ a set of control fields $u_{j,k}$ such that

$$
\mathcal{H}_j = H_0 + \sum_{k=1}^{m} u_{j,k} H_k
$$

$$
U_j = e^{-i\mathcal{H}_j(t_j-t_{j-1})}
$$

$$
K_j = U_j U_{j-1} U_{j-2} \ldots U_1 U_0
$$

$$
|\psi_j\rangle = K_j |\psi_0\rangle.
$$

Objective: Minimize

$$
F_0 = 1 - |\text{Tr}(K_T^\dagger K_n)/D|^2
$$

Quantum Optimal Control

- Secondary objectives and additional constraints

\[
\min_{u_{j,k}} \left( \sum_{i=0}^{2} w_i F_i + \sum_{i=3}^{6} w_i G_i \right),
\]

\[
F_1 = 1 - \frac{1}{n} \sum_j |\text{Tr}(K_T^\dagger K_j)/D|^2
\]
\[
F_2 = 1 - \frac{1}{n} \sum_j |\langle \psi_T | \psi_j \rangle|^2
\]
\[
G_3 = 1 - |\langle \psi_T | \psi_n \rangle|^2
\]
\[
G_4 = |u|^2
\]
\[
G_5 = \sum_{j,k} |u_{j,k} - u_{j-1,k}|^2
\]
\[
G_6 = \sum_j |\langle \psi_F | \psi_j \rangle|^2
\]

where \( \psi_F \) is a forbidden state.

- Adjoint\s used in the Gradient Ascent Pulse Engineering (GRAPE) Algorithm for QOC

- UChicago researchers exploited TensorFlow for automatic differentiation (AD)
Current Implementation (in QuTiP)

- Transmon qubit capacitively coupled to a 3D microwave cavity

- Hamiltonian:

\[
H(t) = \omega_q b^\dagger b + \frac{1}{2} \alpha b^\dagger b (b^\dagger b - 1) + \omega_r a^\dagger a + g(a+a^\dagger)(b+b^\dagger) + \Omega_x(t)(b + b^\dagger) + \Omega_z(t)b^\dagger b
\]

- \(a\) and \(b\): lowering operators for photon number and transmon excitation number

```python
def qutip_verification(datafile, atol):
    #Omitted: I/O and Initialization
    # looping over each initial vector
    for init_vector_id in range(len(initial_vectors_c)):
        # initial vector
        psi0 = qt.Qobj(initial_vectors_c[init_vector_id])
        # make functions to return uks field
        def make_get_uks_func(id):
            def _function(t, args=None):
                time_id = int(t/dt)
                return uks[id][time_id]
            return _function
        # create the time-dependent Hamiltonian list
        Ht_list = []
        Ht_list.append(H0_qobj)
        for ii in range(len(Hops_qobj)):
            Ht_list.append([Hops_qobj[ii], make_get_uks_func(ii)])
        # solving the Schrodinger evolution in QuTiP’s sesolve
        output = qt.sesolve(Ht_list, psi0, tlist, [])
```

Drawback with current approach and opportunities

- **Scalability:**
  - Reverse mode AD stores each unitary matrix $U_j$
  - $U_j$ is of size $2^q \times 2^q$

- **Question:** What tool/simulator/technique can we use to scale up the number of qubits?
  - Option1: study other tools/simulators where QOC can be expressed
  - Option2: use checkpointing strategies from AD
  - Option3: exploit invertibility of Unitary matrices
Option 1: Survey of Quantum Simulators

Search for simulators where we could implement the primal

<table>
<thead>
<tr>
<th>Tool</th>
<th>Parallelism</th>
<th>Gate Operations</th>
<th>Time evolution of Hamiltonian</th>
</tr>
</thead>
<tbody>
<tr>
<td>Qrack</td>
<td>OpenCL</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>QX</td>
<td></td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>qpp</td>
<td>OpenMP</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>eqcs</td>
<td></td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>libquantum</td>
<td></td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>q++</td>
<td></td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>QDD</td>
<td></td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>qcl</td>
<td></td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>Intel-QS.</td>
<td>MPI</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>Quac</td>
<td>MPI</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>ProjectQ</td>
<td>MPI</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Cirq*</td>
<td></td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>Q# *</td>
<td></td>
<td>Y</td>
<td>N</td>
</tr>
</tbody>
</table>
Time Dependent Open Quantum Systems Solver
Developed internally at Argonne by Matthew Otten
Designed to run on Linux systems/clusters, as well as Mac OSX systems
Allows the user to specify
- gates and circuits,
- a density matrix (which enables non-pure quantum states),
- operators that are used to define a Hamiltonian matrix.

Time evolution of the Hamiltonian is possible through the use of PETSc solvers.

https://github.com/0tt3r/QuaC
create_op creates a basic set of operators, namely the creation, annihilation, and number operator.

```c
void create_op(int number_of_levels, operator *new_op)
```

add_to_ham adds a*op to the Hamiltonian

```c
void add_to_ham(PetscScalar a, operator op)
```

add_lin adds a Lindblad L(C) term to the system of equations

```c
void add_lin(PetscScalar a, operator op)
```

get_populations calculates the populations of all operators previously declared and returns the number of populations and the populations. For normal operators, this is $\text{Tr}(C^\dagger Crho)$.

```c
void get_populations(Vec x, double **populations)
```

time_step solves for the time dependence of the system

```c
void time_step(Vec x, PetscReal init_time, PetscReal time_max, PetscReal dt, PetscInt steps_max)
```
for (i=0; i<gate_steps; i++){
    //Loop through all time steps where the pulse is 'flat'
    //This is basically assuming the pulse is a bunch of short
    //time independent chunks defined by the vectors uks
    pulse_x = uks[get_1d_index(2,0,i,dims)];
    pulse_z = uks[get_1d_index(2,1,i,dims)];
    //Add the pulse terms to the hamiltonian
    // Omega_x * (b + b^\dag)
    add_to_ham_p(pulse_x,1,qubit); //Omega_x * b
    add_to_ham_p(pulse_x,1,qubit->dag); //Omega_x * b^\dag
    add_to_ham_p(pulse_z,1,qubit->n); //Omega_z * b^\dag * b
    //Time step from now to the end of this pulse step
    //Go from 0 to gate_time with an initial step size of dt
    //taking no more than steps_max
    time_step(psi,0.0,gate_time,dt,steps_max);
    //Subtract off the pulse from this step
    add_to_ham_p(-pulse_x,1,qubit); //Omega_x * b
    add_to_ham_p(-pulse_x,1,qubit->dag); //Omega_x * b^\dag
    add_to_ham_p(-pulse_z,1,qubit->n); //Omega_z * b^\dag * b
}
We computed the weighted probability of being in that quantum subsystem, or the expectation value of the number operator.

\[ a|0 \rangle + b|1 \rangle \]

pops: 1.036887e-01 2.520915e-03

The population of \(|0 \rangle\) is \(|a|^2\), the population of \(|1 \rangle\) is \(|b|^2\).
Option 2: Use binomial/binary checkpointing

Revolve / Binomial checkpointing for \( n_{\text{TimeSteps}} = 10 \)

This is a UTurn within Revolve for a single logical timestep

Write state checkpoint
Read state checkpoint
Option 3: Reversibility of Unitary Matrices

\[ UU^* = U^* U = I \]
\[ U^{-1} = U^* \]

\[ K_j = U_j U_{j-1} U_j - 2 \ldots U_1 U_0 \]
\[ K_{j-1} = U_j^* K_j \]

- For our problem, we may really need to store the controls.
- In real arithmetic, the Unitary matrices are perfectly reversible. In floating point arithmetic, they are not.

Option 3: Error Growth versus dimension of the matrix

▶ We should be able to determine the number of steps to which the unitary matrix can be reversed.
Option 3: How to exploit reversibility

- Use a similar setup to two level checkpointing.

Two level checkpointing

- nTimeSteps is 10
- nTimeSteps_L2 is 10
  - This is the outer loop

Using reversibility

- This is the use of revolve for nTimeSteps

- Write unitary matrix
- Read unitary matrix

- This is reversing of unitary matrices
Conclusion

- Quantum optimal control requires derivatives which can be computed by AD
- High memory requirement prevents the current approach from scaling
- We are exploring the reversibility of Unitary matrices as a solution
References

Acknowledgement

This work was funded in part the National Science Foundation Mathematical Sciences Graduate Internship, and by support from the U.S. Department of Energy, Office of Science, under contract DE-AC02-06CH11357.