QCHack 2021 - QCTRL Challenge

Ryan Cocuzzo
Daniel Busaba
Michael Taylor
Dominick Harasimiuk

April 10, 2021

1 Introduction

In this project we take a two stage approach to finding optimal pulses for the two gates (H, NOT). For information on how to run the code, please view our **README.md**. We first use the more realistic simulation provided to gather candidate *seed* solutions to begin our learning process. We prepare the qubits with "promising" seed values, calculated via simulation. These have a high chance of quickly converging in our learning algorithm to high-fidelity gates. Our algorithm tests several quantum optimal control solutions (optimal in the context of the simulator), picks the best among them, and feeds the locally-optimal pulse into a gaussian step genetic algorithm. This algorithm is guaranteed only to improve upon the performance of the *seed*. There is more on both of these methods in the following two sections.

2 First Step: Quantum Optimum Control Optimization with Sinc Interpolation

We were quickly able to generate a pulse that would easily satisfy the model. The issue though was this proposed pulse at scale is not very physically viable. Specifically, it would have a phase and amplitude that oscillated at incredibly fast rates. After experimenting with various different smoothing algorithms and parameters for the optimization, we settled on a method that would produce a more physically feasible pulse shape.

We did this with a combination of two modifications to the method discussed in the Twitch stream. First, we changed the definition of the amplitude function in the optimization from a bounded_ optimization_variable to an anchored_difference_bounded_variables and added a difference_bound constraint. This ensured that between the adjacent segments, the amplitude of the pulse would not change by more than a small tunable parameter. This made the pulse amplitude function more realistic and well behaved. However, this still left the phase as rapidly oscillating and the amplitude as having sharp peaks/troughs. To fix this, we purposely had the optimizer optimize over a relatively small number of segments and then upscaled it by using a sinc (aka $\sin(\pi * x)/(\pi * x)$) function interpolation algorithm onto the amplitude. This smoothed out the peaks and valleys of the amplitude function, making it a more natural loooking function. The rapidly changing phase problem is also mitigated by this because the phase is simply copied n times for an upscaling factor of n. This increases the amount of time that the microwave pulse is at a given phase.

To validate that the interpolated pulse function is still valid for the more realistic simulation, we plug the smoothed pulse back into this simulation and ensure that the error is still relatively small. By tuning the number of segments, the factor of upscaling and the amplitude difference_bound constraint, we reliably errors on the real qubit within 5-10%. This provided us with a good starting point to plug into our Quantum Learning Control Using Genetic Algorithm Postprocessing.

3 Second Step: Quantum Learning Control Using Genetic Algorithm Postprocessing

Algorithm 1: Genetic Gaussian Search

```
Result: (best pulse, best loss)
seed_pulse = quantum_opt_control_best;
population = P * [ seed_pulse ] (shape: P * segment_count * 2);
best, best_loss = 0, 3;
for 1...N do
    scores = Loss(population);
    parents = select(population, scores);
    children = [];
    for (parent1, parent2) in parents do
        child1, child2 = cross(parent1, parent2);
        child1.mutate(N(0,s1));
        child2.mutate(N(0,s2));
        children.append(child1);
        children.append(child2);
    end
    population = children;
end
return (best, best_loss)
```

Above, you can see our genetic gaussian search algorithm. This is essentially a genetic algorithm but with the mutation step consisting of a gaussian error update. Whenever two parents are crossed, there is a chance that the amplitude and phase at some point along the segment count axis will be swapped (with some probability) between the two parents as they form two children. These two children then each have a chance to incur a gaussian update (again with some probability) to one or both of their amplitude and phase. These updated children are then used to create a new population for which fitness is evaluated. Each pulse is converted into a complex representation (as opposed to amplitude and phase) and then fed into the qubit. The probabilties of a correct measurement over some number of shots are computed and then fed into a fitness function.

Fitness is evaluated with a loss function as follows:

$$L(p) = \sum_{r=1}^{R} \frac{(o(p)_0 - t_0)^2}{r} + \frac{(o(p)_1 - t_1)^2}{r}$$

Where R is the number of repetitions, t_0, t_1 are the true gate values in states 0 and 1, and $o(p)_0, o(p)_1$ are the observed probabilities for states 0 and 1 when firing the pulse onto the real qubit. This is used to assess the fit of each pulse. We do not consider duration as a trainable parameter.

4 Results and Conclusions

We have a more well defined an graphical portion of our **README.md** that is devoted to results. **Please read both files to get a good overview of our project!** For a quick summary on our gate performance here, our H gate was able to achieve probabilities between .49 to .51 for states 0,1 for any number of repetitions, up to and including 64 repetitions! The NOT gate was more challenging because it is inherently hard to transfer an entire population from one state to another completely (as opposed to the H gate which is more entropically favorable to converging). We were able to achieve probabilities between .75 and .95 reliably through 64 repetitions.