Planning in a Quantum System

Guillermo Andres Cidre

CMU-CS-17-103
December 2016

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee:
Geoff Gordon, Advisor
Gary Miller

Submitted in partial fulfillment of the requirements
for the degree of Master of Science.

Keywords: Quantum Mechanics, Planning, Quantum Planning

For the end of menial and less impactful work.

Abstract

Can we use quantum mechanics to improve our abitlity to plan approximately in classical systems? To address this question, we develop tools to model and plan in a quantum system. We review fundamental quantum mechanical ideas needed to define a quantum model. We also review classical planning models and show how to generalize them to our quantum mechanical model (QuaMDP) which can also model quantum systems. Then we show one way to construct a QuaMDP model for a system given its potential energy. Using our new tools, we run some experiments and show that our QuaMDP model can approximately model some low dimensional classical systems well, qualitatively, and plan in them. However, it is still unclear whether planning in this model is simpler than in the classical case.

Acknowledgments

I would like to acknowledge my advisor Geoff Gordon for being an all around great guy. It was a blast working for him.

I would also like to acknowledge Gary Miller for taking the time to be part of my committee and Tracy Farbacher for being my academic advisor. I am happy to have met such nice people.

I would also like to acknowledge the people who were part of this program behind the scenes for making this possible.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Planning is a problem in Computer Science with many applications. It can be used to automate robotic tasks or beat players in games, for example. However, it has been very difficult coming up with programs that can approximately plan well. Here, we tackle the problem using a different route. We want to see whether we can improve our ability to approximately plan by using quantum mechanics to model classical systems.

The idea of planning using quantum systems may have some merit. Quantum systems make different assumptions than classical systems, so to plan in a quantum system, we have to make a new model. By making a new model and planning on it, we increase the scope of problems we can plan on. Also, there is hope that planning in new quantum models might be simpler than in the classical models, especially when we use a compact basis to plan approximately.

We formulate a new quantum model for planning and give some planning algorithms. To formulate a new quantum model and planning algorithms, we review discrete quantum mechanics to get an understanding about how quantum systems work and classical planning models (ie., MDPs and POMDPs) to understand how to plan in classical systems. Then, we derive the QuaMDP planning model and show one way to plan in it. We also show one way to generate QuaMDP models from a system, given its potential energy function. Then, we test how well our algorithms can approximately plan on this new model to

find inherent weaknesses and strengths.

Chapter 2

Discrete Quantum Mechanics

The goal of this section is to introduce enough discrete quantum mechanics to understand our new QuaMDP model introduced in section 4. Discrete quantum mechanics is a subset of quantum mechanics that can be simulated by a computer using linear algebra with only rounding errors. It deals with the case that our environment has only a finite number of different states. The three foundamental topics of discrete quantum mechanics are density matrices (represent a state in a quantum system), quantum measurements (dictate how to measure outcomes) and transition matrices (dictate how the quantum state changes with time). We will review all of these topics in this section.

Before we start reviewing, we must make one concept clear. In Quantum Mechanics, we make a distinction between different kinds of uncertainty that result when we measure quantities. We define non-classical uncertainty as the uncertainty that results when we make a measurement and measure its outcome. For example, when we measure the length of an object by a ruler we are only guided by the ticks in the ruler. If the object's length falls between two ticks, we have non-classical uncertainty there because we cannot measure that part of the length exactly. We define classical uncertainty as the uncertainty that results when we make a measurement and are unsure of the outcome.

2.1 Density Matrix

We will represent the state of a quantum system, that stores the information of all the objects in the system, as a density matrix, which is a self-adjoint positive semi-definite matrix with trace 1. The diagonal entries of the density matrix correspond to a probability distribution over positions. The off-diagonal entries encode momentum and quantum weirdness. We can manipulate a density matrix by either performing observations or transitioning the density matrix forwards or backwards in time.

Because the density matrix is a positve semi-definite matrix with trace 1, all of the eigenvalues are nonnegative and sum to 1. Thus, we can interpret the eigenvalues as a probability distribution over the eigenvectors. Each eigenvector is called a wave function, and corresponds to a rank-1 density matrix (a quantum state that contains no classical uncertainty). Given a wavefunction ψ, its corresponding rank-1 density matrix is $\psi\psi^*$.

For example, consider a quantum system where an object can be in four different locations. The density matrix

$$\begin{pmatrix} \frac{1}{2} & 0 & 0 & 0 \\ 0 & \frac{1}{2} & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

represents a quantum state where the object is in the first or second position with $\frac{1}{2}$ probability each. The wave functions with nonzero eigenvalues are $(1, 0, 0, 0)$ and $(0, 1, 0, 0)$. Their corresponding rank-1 density matrices are

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \text{ and } \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix},$$

respectively. An example of a nontrivial wave function is $\left(\frac{1}{\sqrt{2}}, i\frac{1}{\sqrt{2}}, 0, 0\right)$, whose corre-

sponding rank-1 density matrix is

$$\begin{pmatrix} \frac{1}{2} & -i\frac{1}{2} & 0 & 0 \\ i\frac{1}{2} & \frac{1}{2} & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}.$$

This density matrix encodes an object that is either in the first or second position with probability $\frac{1}{2}$. But unlike the previous example, if we measure the position of the particle, the resulting expected observation density matrix (defined in the next section) is not the same as the original density matrix. The way the transition matrix evolves (defined in section 2.3) over time depends on the density matrix so by measuring the position, we may have changed how the object in the current example will move in the future. Measuring a particle in a classical system should not affect how it will move in the future, so this behavior yields quantum effects.

2.2 Quantum Measurements

To make measurements on the quantum state, we use Born's rule. In contrast to classical mechanics, the quantum state changes based on the nature and accuracy of our measurements. The more accurate we make our measurements, the more we change the quantum state. Hence, we need to trade off between accuracy of our measurements and the damage we do to the density matrix.

According to Born's rule, we represent measurements (like position and momentum) in a discrete quantum system as a self-adjoint measurement matrix. Born's rule states that the outcomes of a measurement are the eigenvalues of its corresponding measurement matrix. More concretely, let λ_i be the ith eigenvalue of a measurement matrix Q. From a measurement matrix, we can compute an outcome matrix which projects into the eigenspace of one the measurement matrix's eigenvalues. Let P_i be the outcome matrix

for λ_i. The probability of measuring outcome λ_i given density matrix D is

$$\langle P_i, D \rangle \equiv \mathrm{Tr}\left(P_i^* D\right).$$

If the outcome of the measurement is i, then the density matrix that results from the measurement is

$$\frac{1}{Z} P_i D P_i^*$$

where

$$\begin{aligned} Z &= \mathrm{Tr}\left(P_i D P_i^*\right) \\ &= \mathrm{Tr}\left(P_i D P_i\right) \\ &= \mathrm{Tr}\left(P_i^2 D\right) \\ &= \mathrm{Tr}\left(P_i D\right) \\ &= \langle P_i, D \rangle \end{aligned}$$

is the normalizing constant and probability of observing that eigenvalue. We have to normalize the trace of the next density matrix to 1 so that the probabilities sum up to one.

The expected density matrix after the measurement is

$$\begin{aligned} \sum_i \Pr\left[\text{measure outcome } i\right] \left(\frac{1}{Z} P_i D P_i^*\right) &= \sum_i \langle P_i, D \rangle \left[\frac{1}{Z} P_i D P_i^*\right] \\ &= \sum_i P_i D P_i^*. \end{aligned}$$

Using the identity $Q = \sum_i \lambda_i P_i$, the expected outcome for measurement Q on a density matrix D via Born's rule is therefore

$$\langle Q, D \rangle = \sum_i \lambda_i \langle P_i, D \rangle.$$

For example, the measurement matrix for the position is the diagonal matrix A where $A_{ii} = p_i$ and p_i is the position that entry i represents. The eigenvalues are every position

p_i and the eigenvectors are the canonical basis vectors e_i. The projection operators are $P_i = e_i e_i^*$. Thus given density matrix D, the probability that we observe position p_i is $\langle P_i, D\rangle = D_{ii}$ and the expected position of our object is $\langle A, D\rangle$. Assuming we just measured the particle to be in position p_i, our density matrix becomes $e_i e_i^*$. The expected density matrix after measuring is the diagonal matrix we get after zeroing out all of the off-diagonals in D.

Another example involves one bit measurements. One bit measurements are measurements with only two outcomes. For example, consider an object that can only be in 4 different positions. The density matrix D of the system is a 4×4 matrix. An example of a one bit observation is to measure whether the object is in the first two or second two positions. The outcome matrices are

$$P_1 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

$$P_2 = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

The probability that we measure the first outcome is $\langle P_1, D\rangle = D_{11} + D_{22}$. Similarly, the probability that we measure the second outcome is $D_{33}+D_{44}$. The expected density matrix after the observation is the original density matrix D with the off diagonal 2×2 blocks zeroed out. If the corresponding eigenvalues were 0 and 1, then the expected measurement is $1 \cdot (D_{33} + D_{44}) + 0 \cdot (D_{11} + D_{22}) = D_{33} + D_{44}$.

2.3 Transition Matrix

A transition matrix is a unitary matrix U that dictates how a quantum state changes with time. To move density matrix D forward in time, we perform

$$D \leftarrow UDU^*.$$

For example, consider an object that can only be in 4 positions. At each time step, the object moves down one coordinate and loops back to the top if it is in the border. The corresponding transition matrix for this operation is the permutation matrix

$$U = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{pmatrix}.$$

We review a way to generate transition matrices in section 5 which we use for our experiments in section 6.

Chapter 3

Classical Planning

Now that we have covered discrete quantum mechanics, we review classical planning models, which inspire the QuaMDP model introduced in section 4. In classical planning, we have an agent and an environment. The agent can interact with the environment choosing from a set of prespecified actions. The goal of planning is to develop instructions that the agent follows to complete a task in the environment.

We review two classical models called the Markov decision process (MDP) and the partially observable Markov decision process (POMDP). MDPs deal with the case that an agent knows all of the information of the environment while POMDPs deal with the case that the agent only knows the state of the environment approximately. For example, Chess is a game that can be modeled using MDPs because the players know everything about the environment (ie. the board). However, poker needs a POMDP instead because each player cannot see the other players' hands so they do not know the state of the environment completely. Both models only deal with one agent so in the above games, we assume that the other players are computers with predetermined strategies.

We cover MDPs first because they naturally extend into POMDPs, a model that is most similar to our QuaMDP model covered in section 4.

3.1 Markov Decision Process

A Markov decision process (MDP) is a 6-tuple (S, A, T, R, γ, s_0) where:

(a) S is the set of states of the environment.

(b) A is the set of actions the agent may make.

(c) $T : S \times A \times S \to \mathbb{R}$ is a function that encodes transition probabilities. $T(s, a, s')$ returns the probabilty of ending up at state s' given that the agent observed the environment was at state s and took action a.

(d) $R : S \to \mathbb{R}$, called the reward function, measures how good it is to be in a state.

(e) γ is called the discount factor.

(f) s_0 is the initial state of the system.

In this model, the agent acts in discrete time steps. At each time step, the agent observes that the environment is in some state $s \in S$ (in the first time step, the agent observes state s_0) and experiences a reward $R(s)$. Then, the agent chooses an action $a \in A$ to perform. Then, we use the probability distribution $T(s, a, \cdot)$ to determine what state s' the environment ends up at.

We will assume that S and A are finite in this thesis. Notice that an MDP assumes that the transition probabilities are the same in every step (although this assumption is easy to relax at the cost of more notation).

3.1.1 Example

Let's go over an example and see how we can write it as an MDP. Consider an agent in a one dimensional grid starting at position zero. The grid has borders at position zero and position three. Let's say we have three actions: move left one unit, move right one unit or stand still. At the borders we loop around so going left on position zero results in being in position three and vice-versa. Let's assume the goal of the agent is to get to state 2.

We define our set of states to be the positions $\{0, 1, 2, 3\}$ and the set of actions to be $\{-1, 0, 1\}$ where -1 means go left, 0 means stay, and 1 means go right. For each $s \in S$ and $a \in A$, we set

$$T\left(s, a, (s + a) \mod 4\right) = 1.$$

We set the rest of the transition probabilities to zero.

We can define the reward as

$$R(b) = \begin{cases} 1 & b = 2 \\ 0 & b \neq 2 \end{cases}$$

and the discount factor as any $\gamma \in (0, 1)$.

3.2 Planning in MDPs

We have rigorously defined an environment, so now we want to define agent strategies and assign a score or value to them. We can then define the goal of planning as finding the strategy with the highest score or value. We conclude this section by reviewing an algorithm that will yield the policy with the highest value assuming it is computationally feasible to enumerate all of the states.

3.2.1 Policy and Value Functions

In general, a strategy maps experience to an action. This strategy does not need to be deterministic. However, in this thesis we focus on a subset of general strategies called deterministic policies. a (deterministic) policy is a mapping $\pi : S \to A$ that determines what action an agent performs at each state. This class is small and contains an optimal policy that we can find [Bertsekas and Tsitsiklis, 1989].

Now we need to define a way to assign a value to a strategy by using long term rewards. In this thesis we will focus on discounted returns. Given a policy π and an arbitrary state

s_0 (not necessarily the same s_0 from the MDP definition in section 3.1), we can generate a random trajectory $s_0, s_1, \ldots, s_t, \ldots$ by performing action $\pi(s_i)$ at state s_i. We can get the corresponding rewards $r_i = R(s_i)$. We combine these rewards to get the value function

$$V^{\pi}(s_0) = \mathbb{E}\left[r_0 + \gamma r_1 + \ldots + \gamma^t r_t + \ldots\right]$$

where $0 \leq \gamma < 1$ is the discount factor specified in the MDP. Since we are dealing with finite states, the value function is defined for all states.

We can define the optimal value function

$$V^*(s) = \max_{\pi} V^{\pi}(s).$$

and the optimal policy

$$\begin{aligned}\pi^*(s) &= \arg\max_a R(s) + \gamma \mathbb{E}_{s' \sim T(s,a,\cdot)}\left[V^*(s')\right] \\ &= \arg\max_a R(s) + \gamma \sum_{s' \in S} T(s,a,s') V^*(s')\end{aligned}$$

by picking the action with the highest expected value.

It can be shown that the optimal value function follows Bellman's equation [Bertsekas and Tsitsiklis, 1989]:

$$V^*(s) = R(s) + \gamma \max_{a \in A} \sum_{s' \in S} T(s,a,s') V^*(s').$$

3.2.2 Value Iteration in MDPs

To generate the optimal policy, it suffices to compute the optimal value function and pick the action with the highest expected value as shown previously. We review the value iteration algorithm, a method to compute the optimal value function, here. The value iteration algorithm is an iterative method that converges exponentially to the optimal value function by using the Bellman equation.

Since we assume that we can enumerate the states and actions, we represent the value and reward functions as vectors where the ith component represents the value at s_i for

some enumeration of the states. We can define matrix T_a via

$$(T_a)_{ss'} = T(s, a, s')$$

Thus, the Bellman equation becomes

$$V^*(s) = R(s) + \gamma \max_a (T_a V^*)_s$$

which is

$$V^* = R + \gamma \max_{a \in A} T_a V^*$$

where $\max$ takes a component-wise max of a set of vectors. Thus, V^* is a fixed point of the Bellman operator:

$$T[v] = R + \gamma \max_{a \in A} T_a v.$$

The main idea of the value iteration algorithm is that the Bellman operator is a $\sup$-norm contraction [Bertsekas and Tsitsiklis, 1989]. Thus, there is only one fixed point to the Bellman operator, the value function, and we can find this fixed point by repeatedly applying the Bellman operator on an arbitrary starting vector. Thus, we compute the recurrence where v_0 is arbitrary and $v_{t+1} = T[v_t]$. When we set $v_0 = R$, v_t is the optimal value function when we look ahead t steps.

The optimal value function corresponds to $\lim_{t \to \infty} v_t$. Thus, the algorithm computes v_T for some T and uses that as an approximation. The error between v_T and V^* decreases exponentially in T so v_T is a good approximation of V^* for big enough T.

3.3 Partially Observable Markov Decision Processes

In many practical applications we don't know the state of the environment exactly. Instead, we have noisy observations about the environment that may come from a camera, for example. In these settings, it is inadequate to model the environment and agent using a Markov decision process because we can't measure the environment exactly — it could

be that multiple states of the environment may lead to the same observation by the agent. Instead, we will model the environment and agent using a partially observable Markov decision process.

A partially observable Markov decision process (POMDP) is a 8-tuple $(S, A, P, R, \Omega, O, \gamma, s_0)$. A POMDP extends an MDP by adding:

(a) Ω, the set of possible observations the agent can see.

(b) O, a probability measure over observations given the current state. $O(o, s', a)$ is the probability of obseving o given that the environment ended up at state s' after the agent chose action a.

In this model, the agent acts in discrete time steps like in the MDP setting. At each time step, the environment is at a state $s \in S$ which the agent doesn't observe. The agent chooses an action $a \in A$ to perform based on past observations and actions. Then, the new state $s' \in S$ of the environment is determined via the probability distribution $T(s, a, \cdot)$. From the new state s' and the action a chosen by the agent, we determine the observation the agent receives via the probability distribution $O(\cdot, s', a)$.

3.3.1 Example

Let's repeat the example is section 3.1.1 and turn it into a POMDP with the additional assumption that an agent can only observe whether a state is even or odd. The sets S, A and the transition, reward and discount factor remain the same. The set $O = \{\text{Even}, \text{Odd}\}$. We can define $O(\text{Even}, s, a) = 1$ if s is even and $O(\text{Odd}, s, a) = 1$ if s is odd for any $a \in A$ and 0 otherwise.

3.4 Planning in POMDPs

Planning in POMDPs is harder than in MDPs. We don't have complete access to the state of the system so we need to take into account the history of observations and actions taken,

not just our current observation.

3.4.1 Policy

There are two ways to represent a policy in a POMDP: using a history of observations and actions, or using a belief, a distribution over states.

History formulation A policy is a function $\pi : (\Omega \times A)^* \to A$ that chooses an action based on the history of observations and actions.

A policy can be visually represented via a policy tree. A policy tree is a rooted tree where the nodes are actions and the edges represent different observations. We interpret the root node as the action we would take for the empty history. For other histories, we use the sequence of actions and observations to follow a path from the root of the tree and pick the action at the resulting node.

Belief formulation Alternatively, we can represent our history as a probability distribution over all environment states, called a belief. In this setting, $\pi : \mathbb{R}^{|S|} \to A$ maps a probability distribution to an action. An advantage of using beliefs instead of histories is that we can summarize our experience using a finite amount of memory, assuming it is computationally feasible to enumerate all of the states. The algorithms covered in this thesis will use the belief representation.

3.4.2 Value function

There are an infinite number of beliefs so we can't represent the value function as a finite dimensional vector as we did in section 3.2.2.

In the following sections, we will define the value set v to be a set $\{\alpha_1, \ldots, \alpha_k\}$ where α_i is a vector with the same dimensions as the belief. Given a belief b and value set v, we

define the value function to be

$$v(b) = \max_{\alpha \in v} \langle \alpha, b \rangle.$$

This is a reasonable definition because the value function is convex and we can approximate any convex function arbitrarily well in this form [Pineau et al., 2003].

3.4.3 Value Iteration in POMDPs

We can interpret a POMDP as a continuous MDP, where the beliefs are the states of the MDP. Thus given v_t, the value function at time step t, the Bellman equation becomes

$$v_{t+1}(b) = \max_{a \in A} \langle R, b \rangle + \gamma \sum_{b'} T(b, a, b') v_t(b')$$

where T is the transition function between beliefs. By rewriting the equation to sum over states and observations, we get [Pineau et al., 2003]

$$v_{t+1}(b) = \max_{a \in A} \sum_{s \in S} R(s) b(s) + \gamma \sum_{o \in O} \max_{\alpha \in v_t} \sum_{s \in S} \sum_{s' \in S} T(s, a, s') O(o, s', a) \alpha(s') b(s).$$

We can implement this value iteration by doing the following steps as highlighted by [Pineau et al., 2003]:

- Define the set $\Gamma^* = \{R\}$.
- Let $\alpha^{a,o}(s) = \gamma \sum_{s' \in S} T(s, a, s') O(o, s', a) \alpha(s')$ and define the sets $\Gamma^{a,o} = \{\alpha^{a,o} \mid \alpha \in v_t\}$.
- Define the sets $\Gamma^a = \Gamma^* \oplus [\oplus_{o \in O} \Gamma^{a,o}]$ where $A \oplus B = \{a + b \mid a \in A \wedge b \in B\}$.
- Let $v_{t+1} = \cup_{a \in A} \Gamma^a$.

Each of these sets corresponds to a convex function of b. The Γ^* singleton set evaluates to

$$\begin{aligned}\Gamma^*(b) &= \max_{\alpha \in \Gamma^*} \langle \alpha, b \rangle \\ &= \langle R, b \rangle \\ &= \sum_{s \in S} R(s) b(s).\end{aligned}$$

Each $\Gamma^{a,o}$ set evaluates to

$$\begin{aligned}\Gamma^{a,o}(b) &= \max_{\alpha^{a,o} \in \Gamma^{a,o}} \langle \alpha^{a,o}, b \rangle \\ &= \max_{\alpha^{a,o} \in \Gamma^{a,o}} \sum_{s \in S} \alpha^{a,o}(s) b(s) \\ &= \max_{\alpha \in v_t} \gamma \sum_{s \in S} \sum_{s' \in S} T(s, a, s') O(o, s', a) \alpha(s') b(s).\end{aligned}$$

Each Γ^a set evaluates to

$$\begin{aligned}\Gamma^a(b) &= \left[\Gamma^* \oplus \left(\oplus_{o \in O} \Gamma^{a,o}\right)\right](b) \\ &= \Gamma^*(b) + \sum_{o \in O} \Gamma^{a,o}(b) \\ &= \sum_{s \in S} R(s) b(s) + \gamma \sum_{o \in O} \max_{\alpha \in v_t} \sum_{s \in S} \sum_{s' \in S} T(s, a, s') O(o, s', a) \alpha(s') b(s).\end{aligned}$$

So v_{t+1} evaluates to

$$\begin{aligned}v_{t+1}(b) &= \max_{a \in A} \Gamma^a(b) \\ &= \max_{a \in A} \sum_{s \in S} R(s) b(s) + \gamma \sum_{o \in O} \max_{\alpha \in v_t} \sum_{s \in S} \sum_{s' \in S} T(s, a, s') O(o, s', a) \alpha(s') b(s).\end{aligned}$$

Thus, the value function updates appropriately. We set $v_0 = R$ so that we compute the optimal value function looking a certain number of steps ahead.

After performing an update on the value function v_t, the new value set v_{t+1} will have $|A||v|^{|O|}$ elements. This fact leads to an doubly exponential growth of $|v_t|$ in t. Thus, this algorithm is computationally infeasible even with a small number of iterations.

3.4.4 Point-based Value Iteration

Point-based value iteration [Pineau et al., 2003] is an approximate value iteration algorithm with the goal of being computationally efficient. The main idea of the algorithm is to keep a set of belief points B. These belief points are used to prune the vectors in $\cup_{a \in A} \Gamma^a$. For each $b \in B$, pick $\alpha_b = \arg\max_{\alpha \in \cup_{a \in A} \Gamma^a} \langle \alpha, b \rangle$ and let $v_{t+1} = \{\alpha_b \mid b \in B\}$. The result of this approximation will keep $|v|$ fixed, resulting in a polynomial time algorithm.

Now we need to worry how to pick these belief points. Other than preselecting belief points, heuristics have been employed. The idea is to selectively add beliefs we encounter as we run the policy generated from the point based value iteration algorithm. The performance of the algorithm improves every time we rerun the algorithm with an expanded set of belief points. Therefore, we repeatedly run the point based value algorithm and expand the set of points. Since we are trying to compute the infinite horizon value function, we also can initialize our initial value function v_0 in the next step as the value function we ended up with in the current step to get a better approximate value function.

One interesting heuristic is the Stochastic Simulation with Greedy Action (SSGA) [Pineau et al., 2003]. After we compute our optimal policy with a set of points, we randomly pick a point, simulate it forward a certain number of steps using the action outputted by the policy or a random action with ϵ probability and add the output to the set of points. We repeat this step a number of times and then recompute the optimal policy.

Chapter 4

Planning in a Quantum System

We have reviewed discrete quantum mechanics and classical planning so now we have the tools to build our quantum planning model and planning algorithm.

The classical planning models are incompatible with quantum mechanics. For example, when we observe a quantity in quantum mechanics, we change the state as a result. Also, the state and state update equations are different in quantum mechanics. Thus, we must create a new model, which we will call a QuaMDP, to plan in these settings.

4.1 Quantum Markov Decision Processes

A quantum Markov decision process (QuaMDP) is a 7-tuple (S, A, U, M, R, γ, D_0) where:

(a) S is the set of states of the environment.

(b) A is the set of actions an agent can choose at each time step.

(c) $U = \{U_a \mid a \in A\}$ where U_a, a quantum transition matrix, specifies how we update a density matrix when we choose action a.

(d) M is a set of observations our agent can make. Each observation $m \in M$ is a set of projection matrices where each element in m represents a possible outcome of an observation.

(e) R is the reward matrix that is used to assign a reward to a density matrix. To get the reward of density matrix D, we compute $\langle R, D \rangle$.

(f) γ is the discount factor.

(g) D_0 is the initial quantum state.

In contrast to classical planning models, our agent chooses a measurement to make in addition to an action in each step. An example of a measurement that fits this description is the one bit measurement mentioned in section 2.2.

The agent starts out with a density matrix D_0 describing the initial state of the system. At each time step, the agent chooses an action $a \in A$ and measurement $m \in M$ to perform based on past observations, actions and measurements. Then, the agent receives an observation and updates the quantum state accordingly.

4.2 Value iteration in a Quantum Setting

The value iteration algorithm I present here is very similar to the one used in POMDPs. The density function takes the role of a belief, intuitively. The proof that the value function is convex function is analogous to the proof in the POMDP case (namely, the update below preserves convexity). Thus, we can represent the value function similarly. We represent the value set v as $\{\Lambda_1, \ldots, \Lambda_n\}$ where Λ_i is a matrix. Given a density matrix D, its value is

$$v(D) = \max_{\Lambda \in v} \langle \Lambda, D \rangle.$$

When we pick action $a \in A$ and observation $m \in M$ and observe $P \in m$, our resulting

density matrix becomes

$$\frac{1}{Z_{(D,P,a)}} P U_a D U_a^* P^*$$

where $Z_{(D,P,a)} = \langle P, U_a D U_a^* \rangle$ is the probability of seeing the outcome above. Since this is a transition between beliefs, our quantum Bellman equation becomes

$$v_{t+1}(D) = \max_{a,m} \langle R, D \rangle + \gamma \sum_{P \in m} Z_{(D,P,a)} v_t \left(\frac{1}{Z_{(D,P,a)}} P U_a D U_a^* P^* \right).$$

Using the value representation above, we can rewrite the equation as

$$\begin{aligned} v_{t+1}(D) &= \max_{a,m} \langle R, D \rangle + \gamma \sum_{P \in m} Z_{(D,P,a)} \max_{\Lambda \in v_t} \langle \Lambda, \frac{1}{Z_{(D,P,a)}} P U_a D U_a^* P^* \rangle \\ &= \max_{a,m} \langle R, D \rangle + \gamma \sum_{P \in m} \max_{\Lambda \in v_t} \langle \Lambda, P U_a D U_a^* P^* \rangle \\ &= \max_{a,m} \langle R, D \rangle + \gamma \sum_{P \in m} \max_{\Lambda \in v_t} \langle U_a^* P^* \Lambda P U_a, D \rangle. \end{aligned}$$

To compute one iteration of the value iteration algorithm, we do the following:

- Define the set $\Gamma^* \leftarrow R$.
- Define the sets $\Gamma^{a,m,P} \leftarrow \gamma U_a^* P^* \Lambda P U_a, \ \forall \Lambda \in v_t$ where $P \in m$.
- Define the sets $\Gamma^{a,m} = \Gamma^* \oplus \left[\oplus_{P \in m} \Gamma^{a,m,P} \right]$.
- Let $v_{t+1} = \cup_{(a,m) \in A \times M} \Gamma^{a,m}$.

Each of the sets above represent a convex function of D. The Γ^* singleton set evaluates to

$$\begin{aligned} \Gamma^*(D) &= \max_{\Lambda \in \Gamma^*} \langle \Lambda, D \rangle \\ &= \langle R, D \rangle. \end{aligned}$$

Each $\Gamma^{a,m,P}$ set evaluates to

$$\begin{aligned} \Gamma^{a,m,P}(D) &= \max_{\Delta \in \Gamma^{a,m,P}} \langle \Delta, D \rangle \\ &= \max_{\Lambda \in v_t} \gamma \langle U_a^* P^* \Lambda P U_a, D \rangle. \end{aligned}$$

Each $\Gamma^{a,m}$ set evaluates to

$$\begin{aligned}\Gamma^{a,m}(D) &= \left[\Gamma^* \oplus \left(\oplus_{P\in m}\Gamma^{a,m,P}\right)\right](D) \\ &= \Gamma^*(D) + \sum_{P\in m}\Gamma^{a,m,P}(D) \\ &= \langle R, D\rangle + \gamma \sum_{P\in m}\max_{\Lambda\in v_t}\langle U_a^* P^* \Lambda P U_a, D\rangle.\end{aligned}$$

So v_{t+1} evaluates to

$$\begin{aligned}v_{t+1}(D) &= \max_{a,m}\Gamma^{a,m}(D) \\ &= \max_{a,m}\langle R, D\rangle + \gamma \sum_{P\in m}\max_{\Lambda\in v_t}\langle U_a^* P^* \Lambda P U_a, D\rangle.\end{aligned}$$

Thus, the value function updates appropriately. We set $v_0 = R$ so that we compute the optimal value function looking a certain number of steps ahead.

After one iteration on the value function v_t our new value v_{t+1} function will have size $|A||M||v_t|^k$ where k is the maximum number of outcomes for every observation. Thus as in the POMDP case, the value set increases doubly exponentially in t.

4.3 Point-based value iteration on QuaMDPs

As in the POMDP case, the algorithm runs too slowly because the value set grows too fast. One possible solution is to adapt the point based value iteration method to QuaMDPs. The points D in our algorithm are now density matrices representing different quantum states. Like in the POMDP case, for each density matrix $d \in D$, pick $\Lambda_d = \arg\max_{\Lambda\in\cup_{a\in A}\Gamma^a}\langle\Lambda, b\rangle$ and let $v_{t+1} = \{\Lambda_d \mid d \in D\}$. By fixing the number of points, we fix the size of our value set so the runtime of the algorithm is polynomial. We can apply heuristics like the SSGA method in section 3.4.4.

Chapter 5

Approximating Continuous Classical Systems

Now that we have defined the QuaMDP framework, we want to generate models of real-world systems in this framework. One way to accomplish this is to turn a continuous quantum system into a discrete one. We can use the continuous quantum system to approximate any classical system with a known potential energy function.

In continuous systems, we deal with linear operators and functionals instead of matrices and vectors, respectively. A functional maps from a hyperrectangle of the reals (or a subset) into the complex numbers. We focus on the case where the domain is a hyperrectangle B of the reals. The linear operators map functionals to other functionals and preserve linearity. We also define an inner product on this space via

$$\langle f, g \rangle = \int_B \overline{f(x)} g(x) dx.$$

We start by reviewing the continuous analogs of the matrices covered in section 2. Then we show one way to transform a classical state into a wavepacket. Finally, we show how to approximate the operators and functionals by matrices and vectors.

5.1 Hamiltonian Operator

The Hamiltonian is a linear operator that encodes the total energy of the quantum system. For example, the Hamiltonian for a single particle in a potential well is

$$H = -\frac{\hbar^2}{2m}\nabla^2 + V$$

where ∇^2 is the Laplacian operator, V is a potential energy operator, m is the mass of the particle and $\hbar$ is the angular Planck's constant. The potential energy operator performs $(Vf)(x) = V(x)f(x)$. Thus,

$$(Hf)(x) = -\frac{\hbar^2}{2m}\left(\nabla^2 f\right)(x) + V(x)f(x).$$

In the case where we deal with periodic functionals on the hyperrectangle, the Hamiltonian is a self-adjoint operator. The potential energy operator is self-adjoint because the potential energy is a real quantity. When we are considering only periodic functionals, we can show that the partial derivative operator is skew-symmetric using integration by parts. Thus, every second partial derivative operator is self-adjoint. Since the Laplacian operator is a sum of second partial derivative operators, the Laplacian operator is self-adjoint. The Hamiltonian is a sum of self-adjoint operators so it is self-adjoint.

5.2 Schrodinger Equation

The Schrodinger differential equation determines how the quantum system changes over time. The equation takes in the Hamiltonian operator of the system and spews out the time derivative of the density operator. The solution to the Schrodinger differential equation is the transition operator

$$U = e^{-i\frac{1}{\hbar}Ht}$$

where the variable t is the seconds we want to move the quantum system forward by and H is the Hamiltonian operator of the system. When H is self-adjoint, U is unitary.

Quantum dynamics approaches classical dynamics as $\hbar \to 0$ [Landau et al., 1958] so we let Planck's constant be small to approximate a classical system well, but not too small to cause numerical problems or a break down of the approximation techniques we employ. To pick a reasonable $\hbar$, we manually search for an $\hbar$ that makes the system perform as expected, qualitatively.

5.3 Gaussian Wavepacket

Now that we got operators out of the way, we want to turn a classical state (position and momentum) into a quantum state (density operator). Instead of defining the density operator, we instead turn a classical state into a wavefunction, discretize this wavefunction into a vector, and then get our density matrix following section 2.1.

We represent a position and momentum by a wavepacket. There are many different types of wavepackets but we will focus on the Gaussian wavepacket. The Gaussian wavepacket is a Gaussian with a complex part expressing its momentum. A Gaussian wavepacket centered at position μ with standard deviation σ and momentum p is the function

$$f(x) = e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2 + i\frac{p}{\hbar}x}.$$

In addition to a position and momemtum, the Gaussian wavepacket encodes the uncertainty in position and momentum through the standard deviation parameter. The uncertainty in position is a Gaussian with standard deviation σ. The uncertainty in momentum is a Gaussian with standard deviation $\sigma_p = \frac{\hbar}{\sigma}$.

Notice that a very low uncertainty in position may lead to a very high uncertainty in momentum. Thus, we want to set the uncertainty of a wavepacket so that neither uncertainty is too high. This fact is an example of a global quantum phenomenon called Heisenberg's uncertainty principle, discussed in Landau et al. [1958].

5.4 Approximating the Integral

Now that we have turned a classical system into a quantum system, we want to discretize the system. The first step in discretizing the system is finding a way to compute the inner product between any two functionals. We outline one way to approximate it.

We can impose a grid of points G uniformly spaced on the domain and represent a functional as a vector of point-wise evaluations on the points on the grid. We can approximate the integral inner product in this space as

$$\langle f, g \rangle = \frac{|B|}{|G|} \sum_{x \in G} \overline{f(x)} g(x)$$

where $|B|$ is the area of the domain. In addition to approximating the inner product, we need to approximate the Laplacian operator in the Hamiltonian. We can accomplish this by using a finite difference approximation of the derivative. Given the points x_{i-1}, x_i, x_{i+1} on the grid from left to right for some i and the distance between to grid points as Δx, we can approximate the Laplacian operator at that point as

$$\nabla^2 f(x_i) = \frac{f(x_{i+1}) + f(x_{i-1}) - 2f(x_i)}{\Delta x^2}.$$

When we consider the points at the left and right end of the grid, we wrap the indices around. Thus for the left edge and right edges x_0, x_n, $x_{0-1} = x_n$ and $x_{n+1} = x0$. Using this approximation, the Laplacian can be represented by a self-adjoint matrix in this inner product space. Thus, the resulting transition matrix is unitary.

5.5 Discretizing the Quantum System

Now that we can compute inner products between functionals, we can discretize the quantum system by using an orthonormal basis of functionals. One obvious candidate for an orthonormal basis is to use an indicator functional for each point on the grid. The problem with this approach is that we need a lot of points to approximate the discrete derivative well so there might be too many features to keep track of.

Instead, we are going to show how to take any linearly independent set of functionals, project functionals and operators into that set, and turn the linearly independent set into an orthonormal basis. We are going to accomplish the above by using the normal equations and the Gram-Schmidt process. Readers that already know how to do this may safely skip to section 5.6. To avoid confusion, we define the following terms:

- Let $\langle v, w \rangle_2 = v^* w$ be the standard vector inner product.
- Let $\langle \cdot, \cdot \rangle$ be a complex inner product over functionals.
- Let $\{\phi_1, \ldots, \phi_n\}$ be a linearly independent set of functionals.
- Let A be the matrix where $A_{ij} = \langle \phi_i, \phi_j \rangle$.
- Let

$$\phi(x) = \begin{bmatrix} \phi_1(x) \\ \vdots \\ \phi_n(x) \end{bmatrix}.$$

- Let

$$\phi_f = \begin{bmatrix} \langle \phi_1, f \rangle \\ \vdots \\ \langle \phi_n, f \rangle \end{bmatrix}$$

  where f is a functional.

- Let f be a vector representing the coefficients in the linearly independent set. Thus, $f(x) = f_1\phi_1(x) + \ldots + f_n\phi_n(x) = f^T\phi(x)$.
- Let $\tilde{T}$ be the matrix $\tilde{T}_{ij} = \langle \phi_i, T\phi_j \rangle$ where T is a linear operator.
- Let $\hat{f}$ be a vector representing the coefficients in an orthonormal basis. More on this later.

5.5.1 Projecting a Functional

Given a functional f, we want to project it by interact with it by computing inner products. In the case where $f(x) = f_1\phi_1(x) + \ldots + f_n\phi_n(x)$, we can expand ϕ_f to get:

$$\begin{aligned}(\phi_f)_i &= \langle \phi_i, f \rangle \\ &= \sum_j f_j \langle \phi_i, \phi_j \rangle \\ &= \sum_j f_j A_{i,j} \\ &= A_{i,:} f.\end{aligned}$$

From this, we get the equation

$$\begin{aligned}\phi_f &= Af \\ \Rightarrow f &= A^{-1}\phi_f.\end{aligned}$$

In the case that f is not a linear combination of the linearly independent set, we turn the equation ($\phi_f = Af$) into an optimization problem:

$$\min_{f \in \mathbb{C}} \|Af - \phi_f\|_2^2 .$$

A is invertible, so the solution is $f = A^{-1}\phi_f$ which is the solution to the normal equations. We can prove this claim using a proof of contradiction. Assume for the sake of contradiction that A is not invertible. Then, we can pick vector $v \neq 0$ so that $Av = 0$. The ith row of this equation states that $\langle \phi_i, v_1\phi_1 + \ldots + v_n\phi_n \rangle = 0$. We can construct an orthonormal basis e_i from the set and we still have $\langle e_i, v_1\phi_1 + \ldots + v_n\phi_n \rangle = 0$. This implies that $v_1\phi_1 + \ldots + v_n\phi_n = 0$ which contradicts our linear independence assumption.

5.5.2 Projecting a Linear Operator

Now that we can project a functional into this space, we need to project a linear operator. Assuming that f is a linear combination of the linearly independent set, we can deduce

$$\begin{aligned}(\phi_{Tf})_i &= \langle \phi_i, Tf \rangle \\ &= \sum_j f_j \langle \phi_i, T\phi_j \rangle \\ &= \sum_j f_j \tilde{T}_{i,j} \\ &= \tilde{T} f.\end{aligned}$$

From the previous section, we can conclude that $A^{-1}\tilde{T}f$ is the coefficient vector of Tf.

5.5.3 Expressing an Inner Product Using A

Now that we can project functionals and operators into this linearly independent set, we want to turn our set into an orthonormal basis and work in that instead. Before we delve in to the orthonormalization, we need to note that we can rewrite the inner product between any two functions (f and g) in the linearly independent set as

$$\begin{aligned}\langle f, g \rangle &= \sum_{ij} \bar{f}_i g_j \langle \phi_i, \phi_j \rangle \\ &= f^* A g.\end{aligned}$$

This fact implies that for an operator T, we can also deduce

$$\begin{aligned}\langle f, Tg \rangle &= f^* A A^{-1} \tilde{T} g \\ &= f^* \tilde{T} g.\end{aligned}$$

5.5.4 Getting an Orthonormal Basis

To get an orthonormal basis, we decompose $A^{-1} = E^*E$. We can set $\hat{f} = E\phi_f$ because

$$\begin{aligned}\langle \hat{f}, \hat{g} \rangle_2 &= \phi_f^* A^{-1} \phi_g \\ &= \phi_f^* A^{-1} A A^{-1} \phi_g \\ &= f^* A g \\ &= \langle f, g \rangle.\end{aligned}$$

We set $\hat{T} = E\tilde{T}E^*$ because

$$\begin{aligned}\langle \hat{f}, \hat{T}\hat{g} \rangle_2 &= \phi_f^* A^{-1} \tilde{T} A^{-1} \phi_g \\ &= f^* \tilde{T} g \\ &= \langle f, Tg \rangle.\end{aligned}$$

5.6 Putting It All Together

The above tools are all that is needed to construct a QuaMDP. The states of the QuaMDP are the postions in our discretized space. We can project the Hamiltonian and exponentiate to get our transition matrices. We can define measurement operators by an indicator operator over positions. We can project these measurement operators to get our measurement matrices. Given a reward function over positions, we can define the reward operator as multiplying the input by that function. We project that operator to get the reward matrix. We can turn a classical state into a density matrix by using a Gaussian wavepacket.

Another thing to note is that the diagonal entries of the density matrix encode a probability distribution over the orthonormal basis functions, which are not neccessarily positions. We assumed that that the orthonormal basis functions were the indicator functions over each position when describing concepts in chapter 2.

Chapter 6

Experiments

Now that we have explained our new QuaMDP model, planning algorithm, and feature approximation, we want to show them in action through experiments. We detail two experiments: A quantum spring system and a quantum hillcar system. The quantum spring system illustrates how well our QuaMDP planning algorithm works on a simple planning problem. The quantum hillcar system showcases how well our planning algorithm works on a somewhat more realistic planning problem.

On both experiments, we use the grid inner product and the discrete Laplacian operator explained in section 5.4. The grid on each experiment will use 200 different points equally spaced between -1 and 1. Thus, the sampling frequency of the position is 100. We let $\hbar = 0.01$.

When generating our transition matrices, we advance time by $t = 0.3$ seconds. After computing the value function, we evaluate it on a bunch of Gaussian wavepackets localized at different positions and momentums to get our value function pictures.

6.1 Quantum Spring System

The quantum spring system is a very simple planning problem. We have a particle attached to a spring that starts out oscillating at a certain frequency and we want the agent to get the particle to stand still at the spring's equilibrium length.

In a classical spring system, we expect particles with positions close to the equilibrium and low speeds to have high values. As particles get further away from the equilibrium point, we expect the ones moving towards the equilibrium point to have higher values. Also as in any classical system, we expect the value function to reach higher values the more accurate observations we allow. We investigate whether we can see these classical effects in the quantum spring system.

6.1.1 Configuration

The potential energy of the system is

$$\frac{1}{2}kx^2 - F_e x$$

where x is the position of the particle and F_e is an external force added by the agent. From the potential energy, the equilibrium point is at 0. We let $k = 0.4$ in our experiment. The external force is -0.05, 0, or 0.05 depending on whether the agent chooses to move left, stand still, or move right, respectively.

One thing we have to keep in mind when applying an external force on a particle is edge effects. There are two different edge effects we have to keep in mind. The first effect is that we note is that the particle wraps around at the boundaries of the grid. Thus, if a particle is going leftward on the left side of the grid and want to get the right side, the agent may not want to slow down. Another issue we note is that when we apply a constant force, there is a huge difference in the potential energy at both edges of the grid. Because we are using the discrete approximation of the Laplacian, applying a constant force to a wavepacket on the border results in huge accelerations. Thus the value function will output inaccurate values near the border, but we will still get accurate values away from

the borders.

The goal of the system is to keep the particle still at the equilibrium point. We model that goal using the reward function

$$R(x) = -x^2.$$

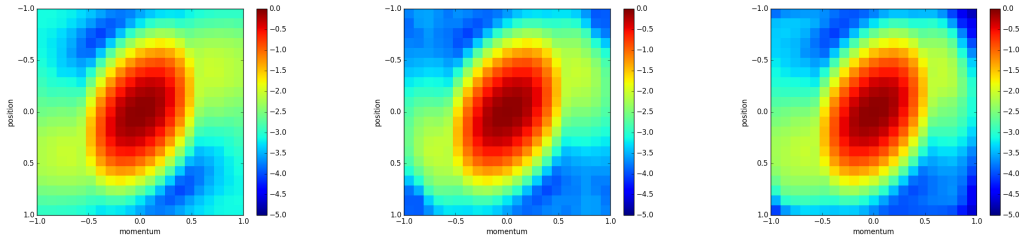
To compute the value function, we use point based value iteration with sample points that are Gaussian wavepackets centered at each point in $\{-1.0, -0.75, -0.5, \ldots, 1.0\}$, with momentums in $\{\frac{-1.0}{\hbar}, \frac{-0.75}{\hbar}, \ldots, \frac{1}{\hbar}\}$, and with standard deviation $\frac{0.25}{3}$. We run the point based value iteration algorithm for 15 steps using discount factor $\gamma = 0.9$.

We will focus on three classes of features: delta features, Fourier features, and Gabor features. Below, we show how we setup each feature set.

Delta Features Delta features are indicator functions signaling a point on the grid. We use delta features to model every point on the grid so our feature set has size 200.

Fourier Features Fourier features are functions of the form $e^{i\theta x}$ for some $\theta \in \mathbb{R}$. Our feature set is the Fourier features with the angle θ in $\{-30\pi, -29\pi, \ldots, 30\pi\}$. Our feature set size is 61. The maximum frequency we can represent is $\frac{30\pi}{2\pi} = 15$ Hz which is less than 50 Hz, the Nyquist frequency.

Gabor Features Gabor features are functions of the form $e^{-\frac{(x-\mu)^2}{2v} + i\theta x}$ for some $\mu, v, \theta \in \mathbb{R}$, which is a Gaussian function multiplied by a complex exponential. Gaussian features encode a wavepacket that is localized in both position and momentum. Our feature set consists of every Gabor feature formed from the position $\mu \in \{-1, -0.75, \ldots, 1\}$, momentum $p \in \{\frac{-1}{\hbar}, \frac{-0.75}{\hbar}, \ldots, \frac{1}{\hbar}\}$, and variance $v = 0.1^2$. Our feature set size is 81. The highest frequency we can represent here is $\frac{1}{2\pi\hbar} \approx 15.9$ Hz which is smaller than 50 Hz, the Nyquist frequency.



(a) Delta features (b) Fourier features (c) Gabor features

Figure 6.1: Value function of spring system using no observations. The colored squares represent the value function evaluated at a Gaussian wavepacket in that position and momentum with standard deviation $\sigma = 0.1$.

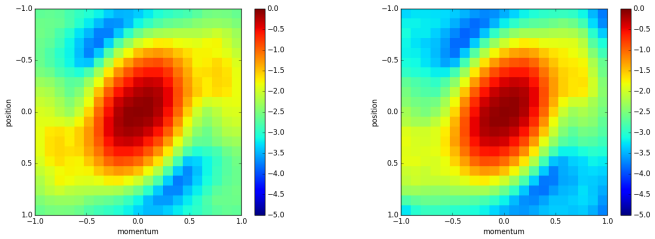
6.1.2 No Observations

In this setup, the agent can only make one measurement that has one outcome. The corresponding measurement matrix is the identity. Thus, the agent is blind in this setup. The approximate value function in this setup is shown in figure 6.1 under different feature functions.

6.1.3 One Bit Observations

In this setup, the agent can choose among four measurements. It can measure whether the particle is left or right of the -0.5 mark, 0.0 mark, or 0.5 mark. The agent can also choose to not to measure at that time step. The value function is shown in figure 6.2.

When we compare the value function to the no observation setup in figure 6.1, the tails of the value function get bigger and the value generally goes up everywhere. We expect this behavior in a classical system because we are observing more accurately.



(a) Fourier features (b) Gabor features

Figure 6.2: Value function of spring system using one bit observations. The colored squares represent the value function evaluated at a Gaussian wavepacket in that position and momentum with standard deviation $\sigma = 0.1$.

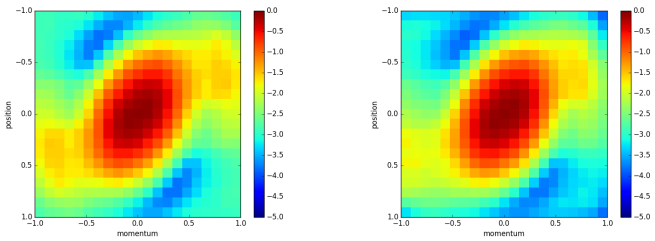
6.1.4 Three Outcome Observations

In this setup, the agent can choose among four measurements at each time step. It can measure whether the particle is left of the position -0.5, between positions -0.5 and 0.0, or right of the position 0.0. It can also measure whether the particle is left of the position 0.0, between positions 0.0 and 0.5, or right of the position 0.5. It can also measure whether the particle is left of the position -0.5, between positions -0.5 and 0.5, or right of the position 0.5. The agent can also choose to not to measure at that time step. The value function is shown in figure 6.3.

When we compare this value function to the one bit observation setup in figure 6.2, the tails of the value function get higher. We expect this behavior in a classical system because we are observing more accurately.

6.1.5 Eight Outcome Observations

In this setup, the agent can choose among two measurements at each time step. The agent can choose to not to measure at that time step or measure whether it is in one of eight intervals of equal length covering the domain $[-1, 1]$. The value function is shown in figure 6.4. We also plotted what measurements and actions our resulting policy makes in



(a) Fourier features (b) Gabor features

Figure 6.3: Value function of spring system using three outcome observations. The colored squares represent the value function evaluated at a Gaussian wavepacket in that position and momentum with standard deviation $\sigma = 0.1$.

figure 6.5 and figure 6.6 using Fourier features, respectively.

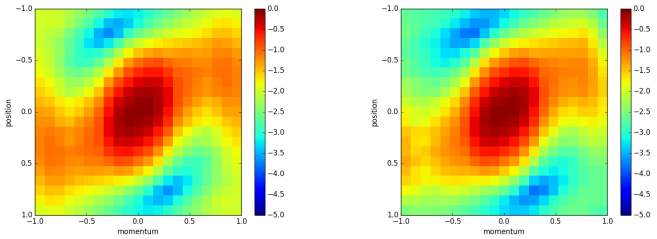
Figure 6.4 shows what we expect in a classical spring system. The value function outputs higher values than the ones that make less accurate measurements, shown in figure 6.3, figure 6.2, and figure 6.1.

Figure 6.5 shows that our policy doesn't observe roughly in areas that have low momentum or that have high value.

Figure 6.6 shows that in the region with low velocities and positions away from the borders, the agent works to steer the particle towards the center. This is exactly what we expect in the classical hillcar system. In regions of very high velocity or of positions very close to the borders, our particle is going to hit the border no matter what, so our agent takes advantage of edge effects. Thus, the actions in these regions don't reflect what we expect but can be explained through edge effects.

6.2 Quantum Hillcar System

In a quantum hillcar system, the agent controls a particle that moves in a valley with a hill on the left and right. The particle starts sliding back and forth in the valley. The agent can apply a small leftward or rightward force on the particle. The goal of the agent is to climb



(a) Fourier features (b) Gabor features

Figure 6.4: Value function of spring system using eight outcome observations. The colored squares represent the value function evaluated at a Gaussian wavepacket in that position and momentum with standard deviation $\sigma = 0.1$.

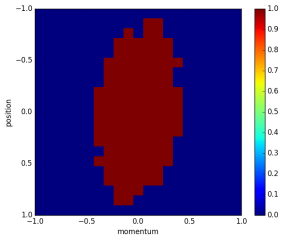


Figure 6.5: The plot shows what observation is chosen by our policy for the quantum spring system. The colored squares represent the observation chosen by the policy evaluated at a Gaussian wavepacket in that position and momentum with standard deviation $\sigma = 0.1$. The red color means that our policy chose not to observe. The blue color means that our policy chose to do an eight bit observation.

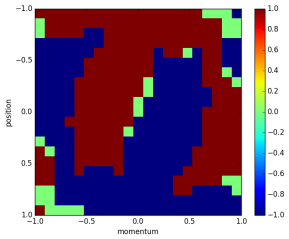


Figure 6.6: The plot shows what action is chosen by our policy for the quantum spring system. The policy chose to move back, stay, or move forward when the value is -1, 0, and 1 respectively. The colored squares represent the action chosen by the policy evaluated at a Gaussian wavepacket in that position and momentum with standard deviation $\sigma = 0.1$.

the right hill.

The agent doesn't have enough force to climb a hill by going in a constant direction. Instead, the agent has to swing back and forth to escape.

In a classical hillcar system, we expect the value to be high for particles with low speed located on top of the right hill. We also expect that the value degrades in an arc shape because a particle not at the top of the hill needs more speed to reach it. We investigate whether we see this behavior in the quantum hillcar system.

6.2.1 Configuration

The potential energy of the system is

$$-m \cos\left(\frac{\pi}{0.5}x\right) - F_e x$$

where x is the position of the particle, F_e is an external force added by the agent, and $m = 0.1$ is a constant. From the potential energy, the bottom of the valley is at position zero and the top of the two hills are at -0.5 and 0.5. The external force is -0.01, 0, or 0.01 depending on whether the agent chooses to move left, stand still, or move right, respectively.

The goal of the agent is to escape the valley via the right hill. We model that goal using

the reward function

$$R(x) = e^{-\frac{1}{2}\left(\frac{x-0.5}{0.05}\right)^2}$$

To compute the value function, we use point based value iteration with sample points that are Gaussian wavepackets centered at each point in $\{-0.5, -0.4, \ldots, 0.5\}$, with momentums in $\{\frac{-1.0}{\hbar}, \frac{-0.8}{\hbar}, \ldots, \frac{1.0}{\hbar}\}$, and with standard deviation 0.1. As before, we run the point based value iteration algorithm for 15 steps using discount factor $\gamma = 0.9$.

We will focus on three classes of features (as before): delta features, Fourier features, and Gabor features. Below we show how we setup each feature set.

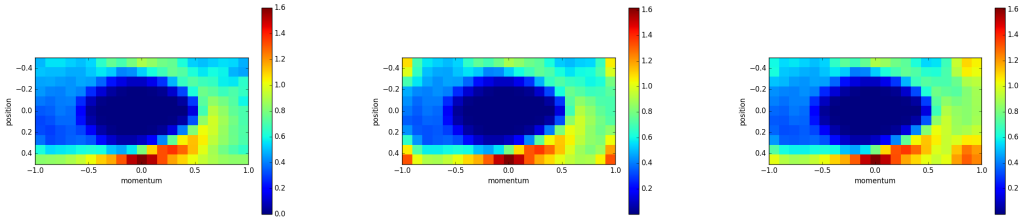
Delta Features Delta features are indicator functions signaling a point on the grid. We use delta features to model every point on the grid so our feature set has size 200.

Fourier Features Fourier features are functions of the form $e^{i\theta x}$ for some $\theta \in \mathbb{R}$. Our feature set is the Fourier features with the angle $\theta \in \{-35\pi, -29\pi, \ldots, 35\pi\}$. Our feature set size is 71. The highest frequency we can represent under these features is $\frac{35\pi}{2\pi} = 17.5$ Hz which is less than 50 Hz, the Nyquist frequency.

Gabor Features Gabor features are functions of the form $e^{-\frac{(x-\mu)^2}{2v}+i\theta x}$ for some $\mu, v, \theta \in \mathbb{R}$, which is a Gabor function multiplied by a complex exponential. Gabor features encode a wavepacket that is localized in both position and momentum. Our feature set consists of every gabor feature formed from the position $\mu \in \{-1, -0.75, \ldots, 1\}$, momentum $p \in \{\frac{-1}{\hbar}, \frac{-0.75}{\hbar}, \ldots, \frac{1}{\hbar}\}$, and variance $v = 0.1^2$. Our feature set size is 81. As in the quantum spring setup, the highest frequency we can represent with Gabor features is less than the Nyquist frequency.

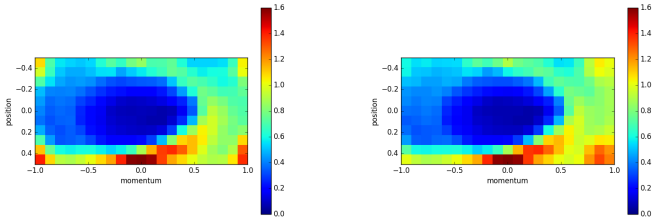
We perform the same measurement setups as in section 6.1. The resulting value functions are shown in figure 6.7, figure 6.8, figure 6.9, figure 6.10, figure 6.11, and figure 6.12.

The value functions in figure 6.7, figure 6.8, figure 6.9, figure 6.10 show what we expect the shape of our value function to look like in the classical hillcar system. Figure 6.11



(a) Delta features (b) Fourier features (c) Gabor features

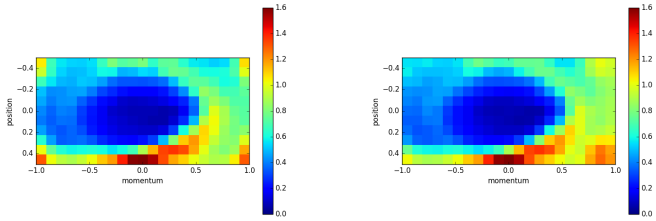
Figure 6.7: Value function of the quantum hillcar system using no observations. The colored squares represent the value function evaluated at a Gaussian wavepacket in that position and momentum with standard deviation $\sigma = 0.1$.



(a) Fourier features (b) Gabor features

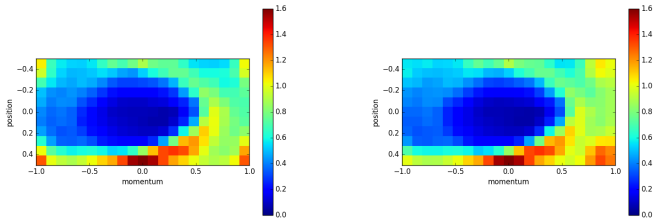
Figure 6.8: Value function of the quantum hillcar system using one bit observations. The colored squares represent the value function evaluated at a Gaussian wavepacket in that position and momentum with standard deviation $\sigma = 0.1$.

suggests that our policy chooses not to observe in states that have good value. Figure 6.12 shows that in areas of low momentum and positions away from the borders of the grid, the agent moves the particle in the direction of its momentum. This is exactly what we expect in the classical hillcar system. In other regions, we can explain bizarre behavior using edge effects.



(a) Fourier features (b) Gabor features

Figure 6.9: Value function of the quantum hillcar system using three outcome observations. The colored squares represent the value function evaluated at a Gaussian wavepacket in that position and momentum with standard deviation $\sigma = 0.1$.



(a) Fourier features (b) Gabor features

Figure 6.10: Value function of the quantum hillcar system using eight outcome observations. The colored squares represent the value function evaluated at a Gaussian wavepacket in that position and momentum with standard deviation $\sigma = 0.1$.

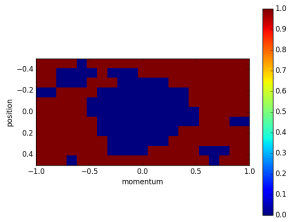


Figure 6.11: The plot shows what observation is chosen by our policy for the quantum hillcar system. The colored squares represent the observation chosen by the policy evaluated at a Gaussian wavepacket in that position and momentum with standard deviation $\sigma = 0.1$. The red color means that our policy chose not to observe. The blue color means that our policy chose to do an eight bit observation.

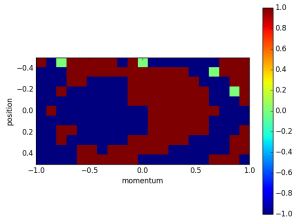


Figure 6.12: The plot shows what action is chosen by our policy for the quantum hillcar system. The policy chose to move back, stay, or move forward when the value is -1, 0, and 1 respectively. The colored squares represent the action chosen by the policy evaluated at a Gaussian wavepacket in that position and momentum with standard deviation $\sigma = 0.1$.

Chapter 7

Conclusion

We introduced the QuaMDP model to formalize quantum systems for planning. We showed how to perform value iteration on a QuaMDP to generate an optimal policy and how to adapt point based value iteration, an approximate planning technique from POMDPs, to QuaMDPs.

We set up and ran experiments that approximated a continuous classical system with a discrete quantum system and computed value functions. These results agree with what we expect, qualitatively, if we were simulating the classical system directly. At the very least, we have shown that there are low dimensional classical systems that can be modeled well, qualitatively, using a quantum system.

We have expanded the set of problems we can plan in but it is still unclear whether approximate planning is easier in this model than in classical models.

There are a couple of things we leave for future work. One thing is to figure out how to build quantum models directly from data and not from the potential energy function. This way we can gather useful statistics about how well our quantum approximation performs and model quantum systems where we don't really know the potential energy. Another thing is to gauge how well our planning algorithm works on quantum systems with very pronounced quantum effects like entanglement and interference.

Bibliography

Dimitri P Bertsekas and John N Tsitsiklis. *Parallel and distributed computation: numerical methods*, volume 23. Prentice Hall Englewood Cliffs, NJ, 1989. 3.2.1, 3.2.2

Lev Davidovich Landau, Evgenii Mikhailovich Lifshitz, JB Sykes, John Stewart Bell, and ME Rose. Quantum mechanics, non-relativistic theory. *Physics Today*, 11:56, 1958. 5.2, 5.3

Joelle Pineau, Geoff Gordon, Sebastian Thrun, et al. Point-based value iteration: An anytime algorithm for pomdps. In *IJCAI*, volume 3, pages 1025–1032, 2003. 3.4.2, 3.4.3, 3.4.4