Moving Least-squares Approximations for Linearly-solvable MDP

Mingyuan Zhong
Department of Applied Mathematics
University of Washington
Seattle, Washington 98195
Email: zhongmy@u.washington.edu

Emanuel Todorov
Department of Applied Mathematics
Department of Computer Science and Engineering
University of Washington
Seattle, Washington 98195
Email: todorov@cs.washington.edu

***Abstract*—By introducing Linearly-solvable Markov Decision Process (LMDP), a general class of nonlinear stochastic optimal control problems can be reduced to solving linear problems. However, in practice, LMDP defined on continuous state space remain difficult due to high dimensionality of the state space. Here we describe a new framework for finding this solution by using a moving least-squares approximation. We use efficient iterative solvers which do not require matrix factorization, so we could handle large numbers of bases. The basis functions are constructed based on collocation states which change over iterations of the algorithm, so as to provide higher resolution at the regions of state space that are visited more often. The shape of the bases is automatically defined given the collocation states, in a way that avoids gaps in the coverage and avoids fitting a tremendous amount of parameters. Numerical results on test problems are provided and demonstrate good behavior when scaled to problems with high dimensionality.**

I. Introduction

In control theory, it is difficult to solve for a nonlinear stochastic optimal control problem globally. This motivates exploring more restricted formulations leading to more efficient algorithms. Previous work [4], [5], [7] has identified a class of nonlinear stochastic optimal control problems which reduced to solving a linear problem in terms of the exponentiated optimal cost-to-go function. In this paper, we primarily focus on the problems with the infinite-horizon average-cost setting, which involve computing the principal eigenfunction of a linear operator. Despite this linearity, the state space of many physical systems is high-dimensional, so the curse of dimensionality is still an issue. Thus, carefully designed approximation schemes are needed for such problems.

Some numerical methods applicable to this problem class have previously been developed, in particular direct MDP discretization [5] and function approximation using Gaussian bases [3]. Discretization is useful in terms of obtaining "ground-truth" solutions in low-dimensional problems and comparing to the results of more advanced algorithms that need fine-tuning, but it is not applicable to higher-dimensional problems. Gaussian bases are promising, but they have some disadvantages. First, the resulting problem is weighted (in the average-cost setting it is in the form $\lambda F\mathbf{w} = G\mathbf{w}$ instead of $\lambda\mathbf{w} = G\mathbf{w}$) which slows down the solver. Second, when λ is also unknown, this method might converge to the wrong eigenvector. Third, positivity of the solution (which is required since we are solving for the exponent of a function) is hard to enforce without introducing inequality constraints. Fourth, Gaussians have too many shape parameters that need to be adjusted: it takes $O\left(n^2\right)$ scalars to specify a covariance matrix in an n-dimensional space.

Our new method avoids the above limitations. It is motivated by the moving least-squares (MLS) methods [1], also known as local regression or LOWESS (locally weighted scatterplot smoothing). The new approximation scheme developed here leads to simple eigen-problems in the form $\lambda\mathbf{w} = G\mathbf{w}$ (here $G = QP$ in (21)) and guarantees the positivity of the solution by enforcing the positivity of G. It also adapts the shape of the basis functions automatically given a selection of collocation states (see Section III). On the downside, the bases are defined implicitly, as the solution to a linear system with $O\left(n\right)$ equations, thus evaluating all bases at one state involves an $O\left(n^3\right)$ operation (Cholesky decomposition). This by itself is not a major disadvantage because finding the inverse of a covariance matrix (which is needed when working with Gaussian bases) is also an $O\left(n^3\right)$ operation. However, the bases developed here need to be evaluated at more points, because the lack of an analytical expression leads to the use of cubature formulas to compute integrals (see below).

While our new method has its limitations, it is up to par with existing methods. Our results to date do not show conclusively that any one method dominants the rest, and we believe that different methods are tailored to different problems. Thus, it is important to have multiple tools available.

A. Outline of the rest of the paper

Section 2 provides a summary of the Linearly-solvable Markov Decision Process. Section 3 describes the Moving Least-squares approximation process. Section 4 demonstrates other key components of our methods. Numerical results are provided in Section 5.

II. Linearly-solvable optimal control problems

In this section, based on [3], [5], we summarize the Linearly-solvable Markov Decision Process (LMDP) framework in continuous space and time. While we focus on average-cost settings, we will also address first-exit settings in later parts.

A. Linearly-solvable MDPs

Consider an MDP with state $\mathbf{x} \in X \subseteq \mathbf{R}^n$. Let $p(\mathbf{x}'|\mathbf{x}, u) = u(\mathbf{x}'|\mathbf{x})$ denote the transition probability given a certain control signal $\mathbf{u}$, and $p(\mathbf{x}'|\mathbf{x})$ denote the transition probability without any control, also known as the passive dynamics. Note that u is a transition probability distribution rather than a control vector, and the mean of this distribution will correspond to the control vector $\mathbf{u}$. The optimal cost-to-go function is given by the Bellman equation

$$v(\mathbf{x}) = \min_u \{l(\mathbf{x}, u) + E_{\mathbf{x}' \sim p(\cdot|\mathbf{x},u)}[v(\mathbf{x}')]\}. \tag{1}$$

For this problem class the immediate cost function is defined as

$$\tilde{l}(\mathbf{x}, u) = \tilde{q}(x) + KL(u(\cdot|\mathbf{x})||p(\cdot|\mathbf{x})), \tag{2}$$

where KL denotes the Kullback-Leibler divergence between two probability distributions. Here $\tilde{q}(x)$ is a state cost function encoded the task goal, while the term $KL(u(\cdot|\mathbf{x})||p(\cdot|\mathbf{x}))$ penalize the controller for controlling the system away from passive dynamics. Define the *desirability* function $z(\mathbf{x}) = \exp(-v(\mathbf{x}))$, where $v(\mathbf{x})$ is the optimal cost-to-go function. Then the optimal control law is

$$u^*(\mathbf{x}'|\mathbf{x}) = \frac{p(\mathbf{x}'|\mathbf{x})z(\mathbf{x}')}{\mathcal{G}[z](\mathbf{x})}, \tag{3}$$

where the operator $\mathcal{G}$ is defined as

$$\mathcal{G}[z](\mathbf{x}) = \int_{\mathbf{x}' \in X} p(\mathbf{x}'|\mathbf{x})z(\mathbf{x}')d\mathbf{x}'. \tag{4}$$

The Bellman equation for infinite-horizon average-cost problems can be written as

$$\exp\left(\tilde{q}(\mathbf{x}) - \tilde{c}\right) z(\mathbf{x}) = \mathcal{G}[z](\mathbf{x}). \tag{5}$$

The desirability function is the principal eigenfunction and is guaranteed to be positive. The corresponding eigenvalue is $\lambda = \exp(-\tilde{c})$ where $\tilde{c}$ is the average cost per step. For the first-exit formulation we have $\tilde{c} = 0$ and $z(\mathbf{x}) = \exp(-q_T(\mathbf{x}))$ at terminal states. This makes the problem a linear equation rather than an eigenfunction problem (see [5]).

B. Linearly-solvable controlled diffusions

Here we consider a class of continuous-time optimal control problems with the following stochastic dynamics:

$$d\mathbf{x} = \mathbf{a}(\mathbf{x})dt + B(\mathbf{x})(\mathbf{u}dt + \sigma d\omega), \tag{6}$$

where $\omega(t)$ represents Brownian motion, and σ is the noise magnitude. The cost function is in the form

$$l(\mathbf{x}, \mathbf{u}) = q(\mathbf{x}) + \frac{1}{2\sigma^2}\|\mathbf{u}\|^2. \tag{7}$$

where $q(\mathbf{x})$ is a state cost function. Note that here $l(\mathbf{x}, \mathbf{u})$ is defined on control vector $\mathbf{u}$ rather than $u(\mathbf{x}'|\mathbf{x})$ defined in II-A. The noise is assumed to lie in the same subspace as the control. The fact that the noise amplitude also appears in the cost function is unusual; however, $l(\mathbf{x}, \mathbf{u})$ can be scaled by σ^2 without changing the optimal control law, and this scaling factor can be absorbed in the state cost $q(\mathbf{x})$, so this is not a restriction. Now we can discretize this dynamical system. The one-step transition probability under the passive dynamics is approximated as a Gaussian distribution

$$p(\mathbf{x}'|\mathbf{x}) = \mathcal{N}(\mathbf{x} + h\mathbf{a}(\mathbf{x}), h\Sigma(\mathbf{x})), \tag{8}$$

where $h\Sigma(\mathbf{x}) = h\sigma^2 B(\mathbf{x})B(\mathbf{x})^T$ is the noise covariance matrix. The transition probability with control $\mathbf{u}$ is

$$p(\mathbf{x}'|\mathbf{x}, \mathbf{u}) = \mathcal{N}(\mathbf{x} + h\mathbf{a}(\mathbf{x}) + hB(\mathbf{x})\mathbf{u}, h\Sigma(\mathbf{x})). \tag{9}$$

The formula for KL divergence between Gaussians gives

$$KL(p(\mathbf{x}'|\mathbf{x}, \mathbf{u})||p(\mathbf{x}'|\mathbf{x})) = \frac{h}{2\sigma^2}\|\mathbf{u}\|^2. \tag{10}$$

Thus, the familiar quadratic energy cost is a special case of the KL divergence cost defined earlier. It can be shown that in the limit $h \to 0$, the solution to the above discrete-time problem converges to the solution of the underlying continuous-time problem. Thus, if we define

$$\tilde{q}(\mathbf{x}) = hq(\mathbf{x}), \tilde{c} = hc, \tag{11}$$

the continuous problem is approximated as a continuous-space discrete-time LMDP. In the infinite-horizon average-cost setting, (5) becomes

$$\exp\left(hq(\mathbf{x}) - hc\right) z(\mathbf{x}) = \mathcal{G}[z](\mathbf{x}). \tag{12}$$

III. Moving least square basis functions

A. Discretizing LMDP problem with basis functions

Approximate solutions to optimal control problems are often represented as linear combinations of predefined basis functions. Our method is similar, except that the basis functions here are constructed in an unusual way, as explained here and in later sections.

Let ϕ_j denote (to-be-defined) basis functions with weights w_j, $j = 1, \cdots, N$. Then the desirability function $z(\mathbf{x})$ can be approximated by

$$z(\mathbf{x}) = \Sigma_{j=1}^N w_j \phi_j(\mathbf{x}). \tag{13}$$

As mentioned earlier, for the average-cost setting, we aim to solve the eigenfunction problem

$$\lambda z(\mathbf{x}) = \exp(-hq(\mathbf{x}))\mathcal{G}[z](\mathbf{x}). \tag{14}$$

Since $\mathcal{G}[z]$ is a linear operator, we have

$$\mathcal{G}[z](\mathbf{x}) = \Sigma_{j=1}^N w_j \mathcal{G}[\phi_j](\mathbf{x}). \tag{15}$$

Therefore the equation we need to solve becomes

$$\lambda \Sigma_{j=1}^N w_j \phi_j(\mathbf{x}) = \exp(-hq(\mathbf{x}))\Sigma_{j=1}^N w_j \mathcal{G}[\phi_j]. \tag{16}$$

We will solve this equation approximately, by introducing collocation points $\mathbf{x}_i, i = 1, \cdots, M$ and enforce the equation at those points. This yields the generalized eigenvalue problem

$$\lambda F\mathbf{w} = QP\mathbf{w} \tag{17}$$

Here F and P are $M \times N$ matrices with entries

$$F_{ij} = \phi_j(\mathbf{x}_i), P_{ij} = \mathcal{G}[\phi_j](\mathbf{x}_i), \quad i = 1, \cdots, M, j = 1, \cdots, N, \tag{18}$$

Q is a $M \times M$ diagonal matrix with diagonal entries $Q_{ii} = \exp(-hq(\mathbf{x_i}))$, $\mathbf{w}$ is a vector of all w_j.

The above outline applies generally to many basis function approximators that are linear in the unknown parameters w_j. For example, in [3], the functions ϕ_j are Gaussians.

Here we will design bases and select collocation states so as to satisfy the following additional conditions.

1) Equal number of collocation points and basis functions, i.e., $N = M$.
2) F in (17) is an identity matrix, or equivalently

$$\phi_j(x_i) = \begin{cases} 1, & \text{when } i = j, \\ 0, & \text{when } i \neq j. \end{cases} \tag{19}$$

3) The basis functions are everywhere non-negative, i.e.

$$\phi_j(x_i) \geq 0 \tag{20}$$

There are two reasons for such restrictions. First, when N is large and F is an identity matrix, we can avoid matrix factorization and take advantage of the sparsity of matrices. Secondly, $z(\mathbf{x}) = \exp(-v(\mathbf{x})) > 0$ is difficult to enforce directly. We'll discuss the details about how we enforced those constraints on basis functions.

Thus in the average-cost setting, the resulting discretized problem becomes

$$\lambda \mathbf{w} = QP\mathbf{w} \tag{21}$$

We will discuss the details of the basis function construction below.

B. Moving-least-squares (MLS) approximation

MLS approximation [1] or locally-weighted regression is a way to approximate a continuous function $z(\mathbf{x})$ when data are assigned to discrete points. MLS is a variation of weighted least squares fitting. Compared to ordinary least squares, both the weight function and coefficients here are no longer constant but are functions of the space variable $\mathbf{x}$. In MLS, the weight function $\omega(\mathbf{x})$ is designed to reconstruct $z(\mathbf{x})$ based mainly on the neighboring nodes, in other words, weights would be higher at nearby nodes.

The reconstructed function can be expressed as

$$z^{MLS}(\mathbf{x}) = \mathbf{h}(\mathbf{x})^T \alpha(\mathbf{x}). \tag{22}$$

Here, the elements of the vector $\mathbf{h}(\mathbf{x})$ are given bases. For example, in one dimensional space, $\mathbf{h}(\mathbf{x})$ can be expressed as $\mathbf{h}(\mathbf{x}) = [1, x]^T$ in linear fitting. $\mathbf{h}(\mathbf{x})$ should not be confused with the basis functions $\phi_i(\mathbf{x})$ in section III-A,III-C. $\mathbf{h}(\mathbf{x})$ is a by-product of MLS approximation process, while we finally use $\phi_i(\mathbf{x})$ in our solving-procedure, which will be outlined in subsequent parts.

Elements of $\alpha(\mathbf{x})$ are the unknown coefficients of those bases $\mathbf{h}(\mathbf{x})$. Unlike ordinary least squares, here they are functions of $\mathbf{x}$ rather than constants. They are obtained by minimizing the Euclidean norm between the approximation and the given function values z_I at nodes $\mathbf{X}_I$.

$$\alpha(\mathbf{x}) = \arg\min_{\alpha} \Sigma_I \omega(\mathbf{x_I} - \mathbf{x})(\mathbf{h}(\mathbf{x_I})^T \alpha - z_I)^2. \tag{23}$$

Here $\omega(\mathbf{x})$ is a given weight function. It is used to make the function approximator local, i.e. fit the value at each state using only the (given) values at nearby states $\mathbf{x}_I$. The minimization process treats the vector α as a free variable at each $\mathbf{x}$. It can be easily shown that the resulting $z^{MLS}(\mathbf{x})$ is linear in the vector $\mathbf{z}$, whose elements are the given function values z_I. This can be represented as $z^{MLS}(\mathbf{x}) = \Sigma_I \tilde{\phi}_I(\mathbf{x}) z_I$, making the MLS process a natural candidate for the construction of basis functions. (We can also see that the basis functions defined in MLS approximation process represent the relationship between a reconstructed function and original data points, while the bases $\mathbf{h}(\mathbf{x})$ in (22) is a way to restrict the fitting process)

C. Construction of basis functions

Now we begin to construct basis functions for solving our problem. In addition to the above criteria (19) and (20), we also need $\Sigma_i \phi_i(x) = 1$. This property guarantees consistency, e.g. if all data indicate that we have a constant function, the approximator will recover a constant function. These properties come for free when using the least-squares bases and $\mathbf{h}(\mathbf{x})$ contains a constant. In summary, we seek basis functions $\phi_i(x)$ that satisfy the following conditions:

(a) $\phi_j(x_i) = \begin{cases} 1, & \text{when } i = j, \\ 0, & \text{when } i \neq j. \end{cases}$
(b) $\phi_i(\mathbf{x}) \geq 0$ everywhere.
(c) $\Sigma_i \phi_i(x) = 1$.

To achieve this goal, we first use MLS to generate potential functions $\tilde{\phi}_i(\mathbf{x})$. Then, we truncate and renormalize them to construct basis functions $\phi_i(\mathbf{x})$. Details are described below.

Making P sparse can significantly improve computational efficiency. In order to achieve sparseness, $z(\mathbf{x})$ will only depend on the values at the K-nearest neighbor nodes (also collocation states). Here K is a manually tuned parameter that determines the trade-off between computer time and smoothness of the approximator. Note that very large values of K are undesirable even if we ignore computer time, because they induce too much coupling and effectively decrease the approximating power of the method. Thus $\phi_i(\mathbf{x}) = 0$ if $\mathbf{x}_i$ is not among the K nearest nodes to $\mathbf{x}$. In the following parts, we will use $\mathbf{x}_{(1)}, \mathbf{x}_{(2)} \cdots, \mathbf{x}_{(K)}$ to represent the K nearest neighboring nodes of $\mathbf{x}$. $\mathbf{x}_{(1)}, \mathbf{x}_{(2)} \cdots, \mathbf{x}_{(K)}$, are sorted in increasing order by distance from $\mathbf{x}$, so we have $\|\mathbf{x} - \mathbf{x}_{(1)}\| \leq \|\mathbf{x} - \mathbf{x}_{(2)}\| \leq \cdots \leq \|\mathbf{x} - \mathbf{x}_{(K)}\|$.

Constraint (a) above or equivalently (19) can be easily obtained by enforcing a linear constraint $\mathbf{h}(\mathbf{x_1})^T \alpha = z_{(1)}$. This will not significantly affect the computational efficiency, since it can be done by introducing a Lagrange multiplier.

To maintain scalability to higher dimensional systems, we use linear basis

$$\mathbf{h}(\mathbf{x}) = [1, x_1, \cdots, x_n]^T. \tag{24}$$

Here $x_i, i = 1, \cdots, n$ are elements of $\mathbf{x} \in \Re^n$. Even when the number n of dimensions increases, the time complexity of the algorithm in this step will not increase significantly.

We would like to use a weight function that can automatically adjust to areas with different densities. We currently use $\omega_i(\mathbf{x}) = \mu_i(\mathbf{x})^2$, with

$$\mu_i = C_\mu\left(\frac{1}{\|\mathbf{x}-\mathbf{x}_{(i)}\| - \|\mathbf{x}-\mathbf{x}_{(1)}\| + \epsilon_\mu} - \frac{1}{\|\mathbf{x}-\mathbf{x}_{(K)}\| - \|\mathbf{x}-\mathbf{x}_{(1)}\| + \epsilon_\mu}\right). \quad (25)$$

Here, $i = 2, \cdots, K$. C_μ is a normalization constant adjusted to yield $\Sigma_i \mu_i = 1$, and ϵ_μ is a small constant used to avoid numerical difficulties.

It is possible to enforce the non-negativity constraints in (26) using quadratic programming, however this is slow. Instead we simply set any negative values to 0 and renormalize to ensure $\Sigma_i \phi_i(\mathbf{x}) = 1$.

To recapitulate, we define $\alpha(\mathbf{x})$ as the solution to the optimization problem

$$\begin{aligned} &\alpha(\mathbf{x}) = \arg\min_\alpha \Sigma_{I=2}^K \omega(\mathbf{x_I} - \mathbf{x})(\mathbf{h}(\mathbf{x_I})^T\alpha - z_{(I)})^2 \\ &\text{s.t.} \quad \mathbf{h}(\mathbf{x_1})^T\alpha = z_{(1)} \end{aligned} \quad (26)$$

to obtain $\tilde{\phi}_i(\mathbf{x})$ in

$$z^{MLS}(\mathbf{x}) = \mathbf{h}(\mathbf{x})^T\alpha(\mathbf{x}) = \Sigma_i \tilde{\phi}_i(\mathbf{x}) z_i \quad (27)$$

This process is done without knowing z_i or explicitly solving for $\alpha(\mathbf{x})$. Instead, (26) is converted into linear equations, and by comparison with (27), an algorithm for finding $\tilde{\phi}_i(\mathbf{x})$ is obtained.

After setting any negative values to 0 and renormalizing, the solution is given by

$$\phi_i(\mathbf{x}) = \begin{cases} C_\phi(\mathbf{x})\tilde{\phi}_i(\mathbf{x}), & \text{if } \tilde{\phi}_i(\mathbf{x}) \geq 0, \\ 0, & \text{if } \tilde{\phi}_i(\mathbf{x}) < 0. \end{cases} \quad (28)$$

Note that $C_\phi(\mathbf{x})$ is adjusted to ensure $\Sigma_i \phi_i(\mathbf{x}) = 1$

Fig.1 gives an example of what these basis functions look like. There are 16 nodes on the plane. We can see that the basis functions have an irregular shape, but nevertheless their support is concentrated around the node. In Fig.1(a) we see a basis that stretches to the right, since the right side is "empty". In Fig.1(b) we see a base stretching out to infinity. Suppose the true $z(\mathbf{x})$ is the Gaussian shown in Fig.1(c). The function $z^{MLS}(\mathbf{x})$ fitted by our approximator is shown in Fig.1(d). The reconstructed result is not entirely smooth but captures the shape of the original function.

In summary, we used an MLS method with truncation to construct basis functions whose shape is automatically adapted to the set of collocation states. This method can be viewed as an interpolation scheme, but it is modified to ensure a fast yet reliable algorithm to solve the eigenfunction problem.

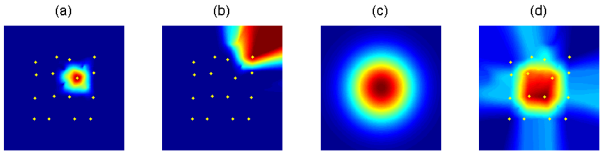

Fig. 1. Illustration of basis functions. (a,b) shows basis functions, (c) shows a Gaussian function and (d) shows the fitted value of the Gaussian function. Dots represent nodes.

IV. Solution Method

In this section we describe how we solve for the desirability function using the above function approximator. First, we show how to get the discretized linear operator. Then, we describe how to determine the weights of the basis functions. Next, we describe how the (approximately) optimal control law is found from the approximate desirability function. After that, we describe a procedure for adding collocation states so at to improve the solution in critical regions (i.e. regions that are visited often under the resulting control law but do not yet contain enough collocation states). Finally, we review the cubature formula.

A. Computing the discretized integral operator

Starting from our discretized problem, as mentioned before, we use the nodes themselves as collocation points. Then we have

$$P_{ij} = \mathcal{G}[\phi_j](\mathbf{x}_i) = \int p(\mathbf{x}'|\mathbf{x})\phi_j(\mathbf{x}')d\mathbf{x}'. \quad (29)$$

Since $p(\mathbf{x}'|\mathbf{x})$ is a Gaussian, this integral can be approximated using cubature formulas as discussed in section IV-E. Cubature formulas can be thought of as deterministic sampling, designed to match as many moments of the Gaussian distribution as possible.

B. Solving for the desirability function

Recall that in the average-cost setting, we need to solve for the leading eigenvalue and eigenvector of a matrix. Many numerical methods are available for solving this kind of problem, e.g. [6]. Our current implementation uses the classic power iteration method. The reasons for choosing this algorithm are: (i) it does not require matrix factorization, which would be very slow when we have a large amount of nodes; (ii) the result is always positive as long as the initialization is positive and all elements of P are nonnegative (which they are by design). Thus the algorithm is:

(0) Let $\mathbf{w}^0$ be an initial guess
(1) $\mathbf{w}^n = \frac{QP\mathbf{w}^{n-1}}{\|QP\mathbf{w}^{n-1}\|}$
(2) Repeat step (1) until $\|\mathbf{w}^n - \mathbf{w}^{n-1}\| < \epsilon$

After this step, we obtain an approximate cost-to-go function for the optimal control problem. We know that the eigenfunction corresponding to the leading eigenvalue of the linear operator $\mathcal{G}$ is unique. Thus the discretization QP is also likely to have a unique leading eigenvector, unless the placement of nodes is pathological (which we have not observed in practice). The speed of convergence is governed

by the difference between the top two eigenvalues of QP. Alternatively, since an iteration can be viewed as running backwards with step h, the speed of convergence is roughly related to the mixing rate of the Markov decision process. In practice we have found this algorithm to converge very quickly (in a few iterations) in our test problems. The time needed to run power iteration is a small fraction of the overall CPU time; most of the CPU time is spent in constructing the discretized operator.

In other settings, the discretized problem is no longer an eigenvector problem. However, iterative algorithms without matrix factorization are still available.

C. Finding the optimal control u^*

Recall that the optimal control law (in terms of probability densities) is given by (3)

$$u^*(\mathbf{x}'|\mathbf{x}) = \frac{p(\mathbf{x}'|\mathbf{x})z(\mathbf{x}')}{\mathcal{G}[z](\mathbf{x})}.$$

The above probability distribution function is known, and its mean can be calculated. To calculate a control vector $\mathbf{u}^*$, we can solve the following equation:

$$\mathbf{x}+h\mathbf{a}(\mathbf{x})+hB(\mathbf{x})\mathbf{u}^* = E_{\mathbf{x}'\sim u^*(\cdot|\mathbf{x})}[\mathbf{x}'] = \frac{\int p(\mathbf{x}'|\mathbf{x})\mathbf{x}'z(\mathbf{x}')d\mathbf{x}'}{\int p(\mathbf{x}'|\mathbf{x})z(\mathbf{x}')d\mathbf{x}'}. \tag{30}$$

Both the numerator and the denominator of the right hand side of (30) are the integral of a function multiplied by a normal distribution. Again, we can use the cubature formula to calculate them approximately as discussed in section IV-E.

Numerical errors may be problematic when the denominator of (30) is very small. However, that happens when $z(\mathbf{x})$ is very small, i.e. the cost-to-go function $v(\mathbf{x})$ is very large, in other words $\mathbf{x}$ is far away from the region of interest. In that case we cannot trust the above u^*; instead we can replace it with some default linear feedback control law. Another issue is that the cubature formula restricts the left hand side of (30), so that the mean of $u^*(\mathbf{x}'|\mathbf{x}))$ lies in the convex hull of the cubature sampling points. This effectively restricts the magnitude of the control signal – which in practice may be a good thing.

D. Adapting the nodes

In previous sections we described how our method works once the nodes/collocation states are chosen. This choice affects performance significantly, and needs to be done in a way suitable for the problem at hand. We have developed an automated procedure for generating problem-specific node placements that optimize performance. Note that the state space is usually very large, so our nodes can only cover a small fraction of it. Ideally the part that is covered will correspond to the region where the optimally-controlled system spends most of its time (i.e. where the good states are found). Thus we use an adaptive procedure, placing new nodes in regions that are visited most often under the control law obtained on the previous iteration of the algorithm. The details are as follows.

The method only adds nodes, with the restriction that every new node must be sufficiently far away from any existing node. If nodes are further restricted within a predefined volume, then this method is guaranteed to terminate in a finite number of iterations. After solving for the optimal control law with the current selection of nodes, we generate prospective nodes based on a stochastic simulation starting from the current nodes (or from given initial states). Meanwhile we also introduce random perturbations to the current nodes. Note that regions with lower $z(\mathbf{x})$, or equivalently higher cost-to-go, will end up with smaller node density under this scheme. We also impose a heuristic restriction to avoid generating prospective nodes in regions where it is impossible to add new nodes (because the density is already too high). The distance between nodes should approximately match the characteristic distance of the system determined by $h\mathbf{a}(\mathbf{x})$ and $\sqrt{h}\sigma B(\mathbf{x})$, which are the magnitude of passive dynamics and transition probabilities. This feature means the time step should be big enough if one wants to solve the problem with a small number of nodes.

In summary, we used a heuristic strategy to put place nodes in regions that are visited more often under the optimal control law.

E. Cubature formula

As mentioned, we want to evaluate the expectation of a function for a normal distribution. This is equivalent to calculating the integral of the product of an arbitrary function and a normal distribution numerically. Here we used the cubature scheme [2].

The goal is to calculate

$$E[g(\mathbf{x})] = \int g(\mathbf{y})p(\mathbf{y}|\mathbf{x})d\mathbf{y}. \tag{31}$$

As in (8),

$$p(\mathbf{y}|\mathbf{x}) = \mathcal{N}(\mathbf{x}+h\mathbf{a}(\mathbf{x}), h\Sigma(\mathbf{x})), h\Sigma(\mathbf{x}) = CC^T, \tag{32}$$

where $C = \sigma B(\mathbf{x})$ for continuous formulation.

We need a variable transformation to convert the integral into the standard form of cubature formula.

$$E[g(\mathbf{x})] = C_0 \int f(\xi)exp(-\xi^T\xi)d\xi, \tag{33}$$

$$C_0 = \frac{1}{(\pi)^{N/2}}, f(\xi) = g(C\sqrt{2}\xi + \mathbf{x} + h\mathbf{a}(\mathbf{x})). \tag{34}$$

Now we can apply the cubature scheme in [2], which would use the following formula to approximate the integral.

$$\int f(\xi)d\xi \approx Q[f] = \Sigma_{j=1}^{N} w_j f(\xi^j), \tag{35}$$

where w_j are weights and ξ^j are points. Several choices of weights and points exist, and it is not clear which cubature formula is optimal for our problems. We observed that formula (36) in [2] works better in some test problems. This is a degree 5 formula with $2n_C^2 + 1$ points, n_C is the number of columns of the $C(\mathbf{x})$ matrix in (32), which is the number of dimensions in control space (for under-actuated systems this number may be significantly smaller than the number of

dimensions in the system). Here, "full sym." means all possible index permutations and reflections.

$$\begin{aligned} Q[f] &= \frac{n^2-7n+18}{18}\pi^{n/2}f(\mathbf{0}) \\ &+\frac{4-n}{18}\pi^{n/2}\Sigma_{full\,sym.}f(\sqrt{3/2},0,..,0) \\ &+\frac{1}{36}\pi^{n/2}\Sigma_{full\,sym.}f(\sqrt{3/2},\sqrt{3/2},..,0). \end{aligned} \quad (36)$$

V. Numerical Results

In this section we present numerical results. First we show that the results given by the MLS approximation are meaningful. Then we show the numerical issues. Finally we will show the results in the first-exit setting.

A. Test problems, solution and dynamical simulation

Here we use MLS approximation to solve our test problems. We focus on the desirability function $z(\mathbf{x})$, cost-to-go function $v(\mathbf{x})$, optimal control law $u^*(\mathbf{x})$, and dynamical simulations based on $u^*(\mathbf{x})$. When possible, we compare these results with dense MDP discretization [3], a grid-based approximation scheme with extreme amount of sampling.

1) Example 1: Car-on-the-hill: This test problem is adapted from [5]. It has a 2D state space (position and velocity) and 1D control space. This dynamical system simulates a point mass (a car) moving along a curve (inverted Gaussian) in the presence of gravity. The control signal is the force acting in the tangential direction. One interesting property of this model is that the continuous dynamics are augmented with the following rule. When the car hits the "walls" at x_{min} or x_{max} , its speed becomes 0. Such a discontinuity cannot be captured by the diffusion model (6), yet it can easily be captured by our LMDP [3]. The reason for constructing a model with collisions is that we hope our methods will work for more complex tasks such as locomotion and hand manipulation, where contact phenomena and discontinuity are essential.

We design a cost-function $q(\mathbf{x})$ (Fig.2(b)), to keep the car passing those two targets with non-zero desired velocities – resulting in a limit cycle behavior. This is done with the average-cost infinite-horizon setting. As shown in Fig.2(c-h), the cost-go function and optimal control law are consistent with the results of dense MDP discretization.

2) Example 2: Coupling a series of masses on ideal springs: This model simulates several identical masses attached to frictionless springs, which are dynamically independent. Each mass can oscillate with any amplitude. The control objective is to generate movements such that: (1) the energy of each mass-spring equals a constant; (2) mass number (i) moves with phase $\pi/2$ ahead of mass number (i+1). We use the average-cost setting to solve the problem, with state cost function designed to enforce the above goals. This model is rather simple, but it has advantages when tuning the algorithm – namely it can be defined for any number of dimensions (masses), and the optimal behavior can be computed analytically: cosine functions $\cos(Ct+\phi)$ with appropriate phase changes.

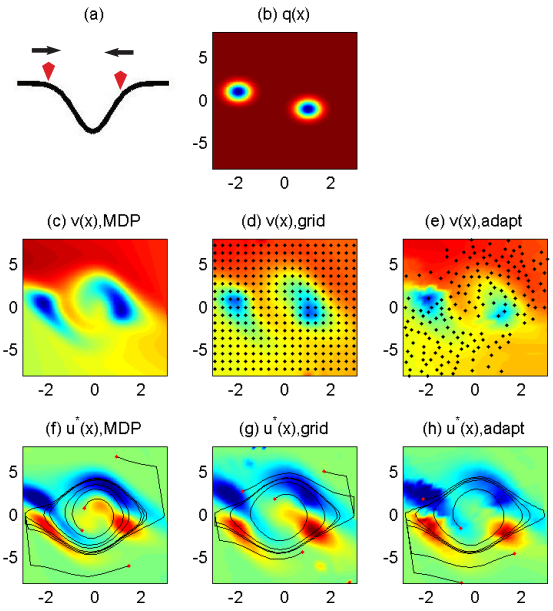

Fig. 2. Results for the car-on-the-hill model with average-cost setting. (a) illustrates the model and (b) shows the state cost function $q(\mathbf{x})$. (c)-(e) (the second row) show the cost-to-go function $v(\mathbf{x}) = -\log(z(\mathbf{x}))$, while (f)-(h) (the third row) show the optimal control law obtained. In (f-h), dynamical simulations are shown by black curves, while their initial states are shown by red dots. (c),(f) show the results from the MDP method with 151×151 grid. (d),(g) shows the results from the MLS approximation with 21×21 nodes on a regular grid. (e,h) shows the results from the MLS approximation with 189 nodes generated using our adaptive scheme. Black dots in (d,e) represent nodes.

Here, we applied a modification that includes a decay factor when the state is too far away from existing nodes. (This effectively applies an additional control to move the system back to nodes, or a hybrid method with the Gaussian approximator in [3]) . The formula is shown below. r is the distance to nearest node,

$$\tilde{z}^{MLS}(x) = \begin{cases} z^{MLS}(x), & r < R, \\ z^{MLS}(x)\exp(-\lambda(r-R)^2), & r \geq R. \end{cases} \quad (37)$$

We found that with some prior information (i.e. initializing some nodes near the optimal trajectory), our approximation scheme generates good results even in a 14-dimensional state space (7 masses) and only requires a few thousand nodes, which is less than a regular grid with two nodes per dimension! This is because most of the nodes end up being places around the limit cycle. Fig. 3 shows the emergence of an attractive trajectory for systems composed of 2 masses and 7 masses respectively.

B. Numerical issues

1) Convergence: First we consider convergence for fixed bases. The power iteration method converges geometrically when the leading eigenvalue is real and its corresponding eigenvector is unique. This holds for the original problem (before the approximation), and in practice we have observed that it holds for the approximation as well. The rate of convergence corresponds to the mixing time of the MDP and

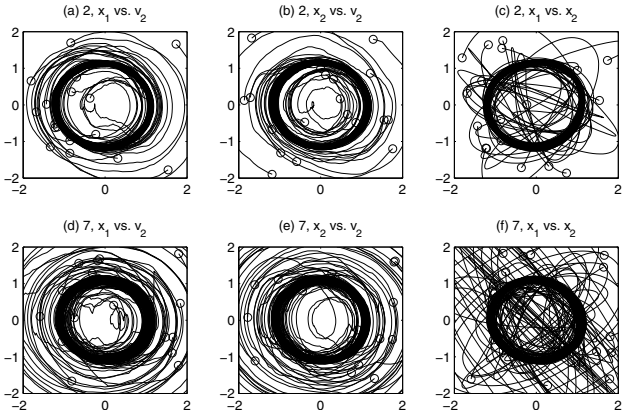

Fig. 3. Stochastic simulation for Example 2 with different dimensionalities. The trajectories are shown by projecting to different dimensions. The first row(a-c) shows the results for 2 masses (4-dimensional state space) with 166 nodes. The second row(d-f) shows the results for 7 masses (14-dimensional state space) with 2912 nodes. The first column(a, d) shows the trajectories of the first mass. The second column(b, e) shows the trajectoriers of the second mass. The last column(c, f) shows the positions of the first and second masses. Note that the analytical solution in each case is a circle with radius 1.

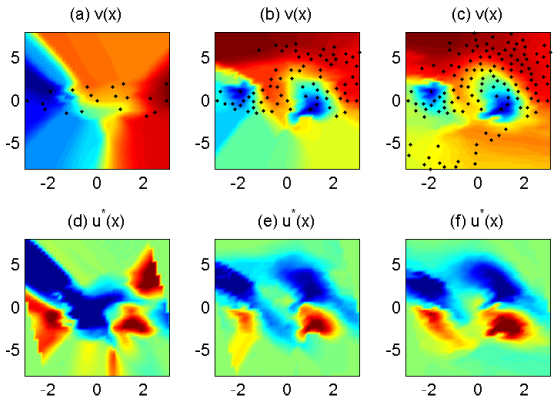

Fig. 4. Demonstration of the adaptive scheme. Here, we demonstrate the process of the adaptive scheme by showing the intermediate cost-to-go function and desirability functions for the average-cost setting of Example 1. The final result is shown in Fig. 2 (e,h), which is obtained with 17 iterations of our adaptive scheme. Here, the first column (a,d) shows results with initial nodes. The second column (b,e) shows the 4th iteration, while the third column(c,f) shows the 9th iteration. The first row (a-c) shows the cost-to-go function, where black dots represent the nodes. The second row (d-f) shows the optimal control law obtained.

is problem specific, but normally it is fast enough. We are guaranteed to get a positive solution. In our numerical tests, the CPU time spent on power iteration is always a small fraction of the total CPU time. Another issue is how accurate the solution is. Due to the nature of cubature formula we cannot make our nodes distributed too densely, however as shown in the previous part, the solution is good in the region covered by the nodes.

Second we consider convergence of the adaptive scheme. The method is trivially guaranteed to terminate in a finite number of iterations, because the approximation volume and node density are limited. In our numerical tests, we observed that the solution improves when the number of nodes increases. The adaptive scheme usually terminates in tens of iterations with proper parameters.

2) Scalability to high-dimensional system: Scalability to high-dimensional systems is the primary challenge in solving Bellman equations numerically. A successful method has to satisfy several conditions. First, the method should give a controller not far from the true solution. Second, the computational complexity of the method should scale well with the number of parameters (weights, grid points, etc). Third, the number of parameters needed to achieve good accuracy should scale well with dimensionality. The first and second conditions hold for most methods, including MDP discretization. The real challenge however is the third condition – which is rarely met in practice. Our numerical results illustrated that the new method has a lot of potential in terms of scaling. This is because the function approximator is automatically adapted to the problem at hand, and because for a fixed set of bases the problem becomes linear and is solved very efficiently. We now discuss these two points in more detail.

First we discuss the case of fixed bases/nodes. Our empirical results show that most of the CPU time is spend in constructing the discretized operator (Section IV-A) using the MLS method. The time complexity of this construction is $O(n^3)$ where n is the number of state space dimensions. The cubature formula has time complexity $O(n_c^2)$ where n_c is the number of conrol space dimensions. Thus, when the number of nodes is N and the number of nearest neighbors being considered is K, the overall time complexity is $O(NKn^3n_c^2)$. The CPU time for constructing the discretized operator in Example 2, with the final set of nodes, was as follows. For 2 masses, with 100-200 nodes, it takes around 0.02 second. For 7 masses, with 2000-4000 nodes, it takes 40-160 seconds. These results are consistent with our theoretical analysis.

Second we discuss the time complexity of the adaptive scheme. We do not yet have theoretical results, however with our current settings, the number of iterations did not increase significantly with state space dimensionality (this is possible because multiple nodes are added in each iteration, and the number of new nodes per iteration typically increases with dimensionality). In Example 2, the algorithm converges in 10-30 iterations, regardless of the number of masses. Each iteration involves (1) constructing the discretized operator, (2) finding the principal eigen-pair and corresponding control law, and (3) running a dynamical simulation starting from current nodes for a short period of time(described in section IV-D). Part (2) takes negligible time, while parts (1) and (3) are comparable. The total CPU time for solving Example 2 with the adaptive scheme is as follows. For 2 masses it takes 1-3 seconds. For 7 masses it takes 400-4000 seconds. Our code is a mixture of Matlab and C mex files. The speed can be improved significantly by utilizing parallel processing; indeed both the construction of the discretized operator and the stochastic simulation (starting from many different states) are easily parallelized.

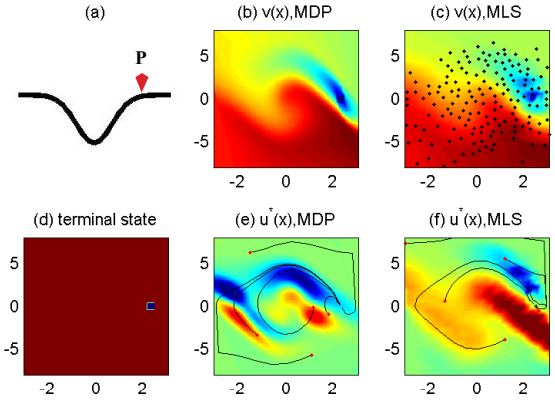

Fig. 5. Results for the car-on-the-hill model with first-exit setting. (a) illustrates the model and (d) shows the terminal states vs. nonterminal states. Blue shows terminal states. (b-c) show the cost-to-go function $v(\mathbf{x}) = -\log(z(\mathbf{x}))$. (e-f) show the optimal control law obtained. In (e-f), dynamical simulations are shown by black curves, while their initial states are shown by red dots. (b),(e) show the results from the MDP method with 151×151 grid. (c),(f) show the results from the MLS approximation with 161 nodes generated by our adaptive scheme. Black dots in (c) represent nodes. The MLS solution is not identical to the MDP solution, yet the resulting control laws are similar.

C. First-exit setting

Although our method is tailored to the average-cost infinite-horizon problems, it works for other settings. Finite-horizon settings and discounted-cost settings are computationally similar to average-cost setting. For the first-exit setting, the integral operator $\mathcal{G}$ (29) is no longer taken in the entire space but only in non-terminal states, thus more numerical integration tricks are needed. Here we present the result for a first-exit problem.

We use the same model as in example 1. The terminal cost function is adjusted to park the car(velocity being 0) at where the horizontal position is 2.4, with state cost function $q(\mathbf{x})$ is constant everywhere. As shown in Fig.5, the cost-go function and optimal control law are similar to the results of dense MDP discretization.

VI. Conclusion

Here we developed a new function approximation method for linearly-solvable stochastic optimal control problems. Numerical tests show its effectiveness. Furture work includes fine-tunning the method and applying it to more complicated and practical problems, as well as devising an algorithm to estimate the error in high-dimensional problems.

Acknowledgment

This work was supported by the US National Science Foundation.

References

[1] T. Belytschko, Y. Krongauz, D. Organ, M. Fleming, P. Krysl, *Meshless methods: An overview and recent development*, Comput. Mech. Engrg. 139, 3-47 (1996).

[2] J Lu, DL Darmofal, *Higher-dimensional integration with Gaussian weight for applications in probabilistic design* ,SIAM J. SCI. COMPUT, Vol. 26, No. 2, pp. 613C624, 2005

[3] E. Todorov, *Eigen-function approximation methods for linearly-solvable optimal control problems*, IEEE International Symposium on Adaptive Dynamic Programming and Reinforcemenet Learning, 2009.

[4] E. Todorov, *Linearly-solvable Markov decision problems*, Advances in Neural Information Processing Systems, 2006.

[5] E. Todorov, *Efficient computation of optimal actions*, PNAS, July 14, 2009 vol. 106 no. 28 11478-11483 .

[6] LN Trefethen, D Bau , *Numerical Linear Algebra*, SIAM1997

[7] H. Kappen, *Linear theory for control of nonlinear stochastic systems*. Physics Review Letter 95: 200201 (2005).