J Control Theory Appl 2011 9 (3) 451–463
DOI 10.1007/s11768-011-0275-0

Moving least-squares approximations for linearly-solvable stochastic optimal control problems

Mingyuan ZHONG [1], Emanuel TODOROV [1,2]

1.Department of Applied Mathematics, University of Washington, Seattle, WA 98195, U.S.A.;

2.Department of Computer Science, University of Washington, Seattle, WA 98195, U.S.A.

Abstract: Nonlinear stochastic optimal control problems are fundamental in control theory. A general class of such problems can be reduced to computing the principal eigenfunction of a linear operator. Here, we describe a new method for finding this eigenfunction using a moving least-squares function approximation. We use efficient iterative solvers that do not require matrix factorization, thereby allowing us to handle large numbers of basis functions. The bases are evaluated at collocation states that 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. Numerical results on test problems are provided.

Keywords: Stochastic optimal control; Bellman equations

1 Introduction

Nonlinear stochastic optimal control problems are fundamental in control theory, yet they remain difficult to solve. This motivates exploring more restricted formulations leading to more efficient algorithms. Previous work [1–3] has identified a class of nonlinear stochastic optimal control problems that is reduced to solve a linear problem in terms of the exponentiated optimal cost-to-go function. In this paper, we primarily focus on infinite-horizon average-cost settings, which involve computing the principal eigenfunction of a linear operator. Despite this linearity, the state space of many physical systems is high-dimensional, and 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 [3] and function approximation using Gaussian bases [4]. 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 $\lambda F\boldsymbol{w} = G\boldsymbol{w}$ instead of $\lambda\boldsymbol{w} = G\boldsymbol{w}$), 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 $\mathrm{O}(n^2)$ 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 [5–7], which is 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 $\lambda\boldsymbol{w} = G\boldsymbol{w}$ (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 the set of collocation states (see Section 3). The downside of this is that the bases are defined implicitly as the solution to a linear system with $\mathrm{O}(n)$ equations, thus evaluating all bases at one state involves an $\mathrm{O}(n^3)$ operation (Cholesky decomposition). This by itself is not a major disadvantage because of finding the inverse of a covariance matrix (which is needed when working with Gaussian bases) is also an $\mathrm{O}(n^3)$ 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).

In Section 2, we review the linearly solvable optimal control problems. In Section 3, we demonstrate the moving-least-square approximation process. In Section 4, we show other key components of our method. Numerical results will be presented in Section 5.

2 Linearly solvable optimal control problems

In this section, based on [3,4], we summarize the linearly solvable Markov decision process (LMDP) framework in

Received 10 November 2010; revised 2 March 2011.
This work was supported by the US National Science Foundation.

continuous space and time. While we focus on average-cost settings, we will also address first-exit and other settings in later parts.

2.1 Linearly solvable MDPs

Consider an MDP with state $\boldsymbol{x} \in X \subseteq \mathbb{R}^n$. Let

$$p(\boldsymbol{x}'|\boldsymbol{x}, u) \equiv u(\boldsymbol{x}'|\boldsymbol{x})$$

denote the transition probability given a certain control signal u, and $p(\boldsymbol{x}'|\boldsymbol{x})$ denote the transition probability without any control, which is also known as passive dynamics. The optimal cost-to-go function is given by the Bellman equation:

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

Note that u under $\min$ is the transition probability distribution $u(\boldsymbol{x}'|\boldsymbol{x})$ rather than a control vector.

For this problem class, the immediate cost function is defined as

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

where KL denotes the Kullback-Leibler divergence between two probability distributions. Define the *desirability* function $z(\boldsymbol{x}) = \exp(-v(\boldsymbol{x}))$, where $v(\boldsymbol{x})$ is the optimal cost-to-go function. Then, (1) and (2) become (shown in [3])

$$-\log(z(\boldsymbol{x})) = \tilde{q}(x) \min_u \{E_{\boldsymbol{x}' \sim u(\cdot|\boldsymbol{x})}[\frac{u(\boldsymbol{x}'|\boldsymbol{x})}{p(\boldsymbol{x}'|\boldsymbol{x})z(\boldsymbol{x}')}]\}.$$

The minimizing expression resembles KL divergence between $u(\boldsymbol{x}'|\boldsymbol{x})$ and $p(\boldsymbol{x}'|\boldsymbol{x})z(\boldsymbol{x}')$ except that $p(\boldsymbol{x}'|\boldsymbol{x})z(\boldsymbol{x}')$ is not normalized to 1. Since KL divergence get its global minimum 0 when the two distributions are equal, therefore the optimal control law can be expressed in the following form without minimization:

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

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

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

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

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

The desirability function is the principal eigenfunction of $\exp\left(-\tilde{q}(\boldsymbol{x})\right)\mathcal{G}[z](\boldsymbol{x})$ 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 $z(\boldsymbol{x}) = \exp(-q_{\mathrm{T}}(\boldsymbol{x}))$ at terminal states. This makes the problem a linear equation rather than an eigenfunction problem (see [3]). Reference [3] developed the Bellman equations for other formulations, and they are reviewd in Appendix A2.

2.2 Linearly solvable controlled diffusions

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

$$\mathrm{d}\boldsymbol{x} = \boldsymbol{a}(\boldsymbol{x})\mathrm{d}t + B(\boldsymbol{x})(\boldsymbol{u}\mathrm{d}t + \sigma\mathrm{d}\omega), \tag{6}$$

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

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

where $q(\boldsymbol{x})$ is a state cost function. Note that here $l(\boldsymbol{x}, \boldsymbol{u})$ is defined on control vector $\boldsymbol{u}$ rather than $u(\boldsymbol{x}'|\boldsymbol{x})$ defined in Section 2.1. 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, $l(\boldsymbol{x}, \boldsymbol{u})$ can be scaled by σ^2 without changing the optimal control law, and this scaling factor can be absorbed in the state cost $q(\boldsymbol{x})$, 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

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

where h is the time duration of this step,

$$h\Sigma(\boldsymbol{x}) = h\sigma^2 B(\boldsymbol{x})B(\boldsymbol{x})^{\mathrm{T}}$$

is the noise covariance.

The transition probability with control $\boldsymbol{u}$ is

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

while the formula for KL divergence between Gaussians gives

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

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. Therefore, if we define

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

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

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

3 Moving least-squares basis functions

3.1 Discretizing LMDPs 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 the later sections.

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

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

From (12), for the average-cost setting, we aim to solve the following eigenfunction problem with $\lambda = \exp(-hc)$

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

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

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

Therefore, if (13) holds, (14) becomes

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

We will solve this equation approximately, by introducing collocation points $\boldsymbol{x}_i, i = 1, \ldots, M$, and enforce the equa-

tion at those points. This yields the generalized eigenvalue problem:

$$\lambda F\boldsymbol{w} = QP\boldsymbol{w}, \tag{17}$$

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

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

Q is an $M \times M$ diagonal matrix with diagonal entries $Q_{ii} = \exp(-hq(\boldsymbol{x}_i))$, and $\boldsymbol{w}$ 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 [4], the functions ϕ_j are Gaussians.

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

1) Equal number of collocation points and basis functions, i.e., $N = M$.

2) F in (17) is the 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 nonnegative, i.e.,

$$\phi_j(x_i) \geqslant 0. \tag{20}$$

There are two reasons for these restrictions. First, when N is large and F is the identity matrix, we can avoid matrix factorization and take advantage of sparsity. Second, $z(\boldsymbol{x}) = \exp(-v(\boldsymbol{x})) > 0$ is difficult to enforce if the bases can be negative.

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

$$\lambda\boldsymbol{w} = QP\boldsymbol{w}. \tag{21}$$

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

3.2 Moving-least-squares (MLS) approximation

MLS approximation [5] or locally weighted regression is a way to approximate a continuous function $z(\boldsymbol{x})$ 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 $\boldsymbol{x}$. In MLS, the weight function $\omega(\boldsymbol{x})$ is designed to reconstruct $z(\boldsymbol{x})$ based mainly on the neighboring node; in other words, weights would be higher at nearby nodes.

The reconstructed function can be expressed as

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

Here, the elements of the vector $\boldsymbol{h}(\boldsymbol{x})$ are given bases. For example, in one-dimensional space, $\boldsymbol{h}(\boldsymbol{x})$ can be expressed as $\boldsymbol{h}(\boldsymbol{x}) = [1\ \ x]^{\mathrm{T}}$ in linear fitting. $\boldsymbol{h}(\boldsymbol{x})$ should not be confused with the basis functions $\phi_i(\boldsymbol{x})$ in Sections 3.1 and 3.3. $\boldsymbol{h}(\boldsymbol{x})$ is a by-product of MLS approximation process, while we finally use $\phi_i(\boldsymbol{x})$ in our solving-procedure, which will be outlined in subsequent parts.

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

$$\boldsymbol{\alpha}(\boldsymbol{x}) = \arg\min_{\alpha} \sum_I \omega(\boldsymbol{x}_I - \boldsymbol{x})(\boldsymbol{h}(\boldsymbol{x}_I)^{\mathrm{T}}\boldsymbol{\alpha} - z_I)^2. \tag{23}$$

Here, $\omega(\boldsymbol{x})$ 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 $\boldsymbol{x}_I$. The minimization process treats the vector α as a free variable at each $\boldsymbol{x}$. It can be easily shown that the resulting $z^{\mathrm{MLS}}(\boldsymbol{x})$ is linear in the vector $\boldsymbol{z}$, whose elements are the given function values z_I. This can be represented as $z^{\mathrm{MLS}}(\boldsymbol{x}) = \sum_I \tilde{\phi}_I(\boldsymbol{x}) z_I$, 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 $\boldsymbol{h}(\boldsymbol{x})$ in (22) is a way to restrict the fitting process).

3.3 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 $\sum_i \phi_i(x) = 1$. This property guarantees consistency, e.g., if all data indicate that we have a constant function, and the approximator will recover a constant function. These properties come for free when using the least-squares bases, and $\boldsymbol{h}(\boldsymbol{x})$ 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) $\phi_i(\boldsymbol{x}) \geqslant 0$ everywhere.

c) $\sum_i \phi_i(x) = 1$.

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

Making P sparse can significantly improve computational efficiency. In order to achieve sparseness, $z(\boldsymbol{x})$ 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, $\phi_i(\boldsymbol{x}) = 0$ if $\boldsymbol{x}_i$ is not among the K-nearest nodes to $\boldsymbol{x}$. In the following parts, we will use $\boldsymbol{x}_{(1)}, \boldsymbol{x}_{(2)}, \ldots, \boldsymbol{x}_{(K)}$ to represent the K-nearest neighboring nodes of $\boldsymbol{x}$. $\boldsymbol{x}_{(1)}, \boldsymbol{x}_{(2)}, \ldots, \boldsymbol{x}_{(K)}$, which are sorted in the increasing order by distance from $\boldsymbol{x}$, so we have $\|\boldsymbol{x} - \boldsymbol{x}_{(1)}\| \leqslant \|\boldsymbol{x} - \boldsymbol{x}_{(2)}\| \leqslant \ldots \leqslant \|\boldsymbol{x} - \boldsymbol{x}_{(K)}\|$.

Constraint (a) or (19) can be easily obtained by enforcing a linear constraint $\boldsymbol{h}(\boldsymbol{x}_1)^{\mathrm{T}}\boldsymbol{\alpha} = z_{(1)}$. 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 $\boldsymbol{h}(\boldsymbol{x}) = [1\ \ x_1\ \ \cdots\ \ x_n]^{\mathrm{T}}$. Here, $x_i, i = 1, \ldots, n$ are elements of $\boldsymbol{x} \in \mathbb{R}^n$. 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 $\omega_i(\boldsymbol{x}) = \mu_i(\boldsymbol{x})^2$, with

$$\mu_i = C_\mu\left(\frac{1}{\|\boldsymbol{x}-\boldsymbol{x}_{(i)}\| - \|\boldsymbol{x}-\boldsymbol{x}_{(1)}\| + \epsilon_\mu} - \frac{1}{\|\boldsymbol{x}-\boldsymbol{x}_{(K)}\| - \|\boldsymbol{x}-\boldsymbol{x}_{(1)}\| + \epsilon_\mu}\right). \quad (24)$$

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

It is possible to enforce the nonnegativity constraints in (25) using quadratic programming, however, this is slower than solving a linear system. Instead, we simply set any negative values to 0 and renormalize them to ensure

$$\sum_i \phi_i(\boldsymbol{x}) = 1.$$

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

$$\boldsymbol{\alpha}(\boldsymbol{x}) = \arg\min_{\alpha} \sum_{I=2}^{K} \omega(\boldsymbol{x}_I - \boldsymbol{x})(\boldsymbol{h}(\boldsymbol{x}_I)^{\mathrm{T}}\boldsymbol{\alpha} - z_{(I)})^2,$$
$$\text{s.t. } \boldsymbol{h}(\boldsymbol{x}_1)^{\mathrm{T}}\boldsymbol{\alpha} = z_{(1)} \quad (25)$$

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

$$z^{\mathrm{MLS}}(\boldsymbol{x}) = \boldsymbol{h}(\boldsymbol{x})^{\mathrm{T}}\boldsymbol{\alpha}(\boldsymbol{x}) = \sum_i \tilde{\phi}_i(\boldsymbol{x}) z_i. \quad (26)$$

This process is done without knowing z_i or explicitly solving for $\boldsymbol{\alpha}(\boldsymbol{x})$. Instead, (25) is converted into linear equations, and by comparison with (26), an algorithm for finding $\tilde{\phi}_i(\boldsymbol{x})$ is obtained. Details of this process are shown in Appendix A3.

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

$$\phi_i(\boldsymbol{x}) = \begin{cases} C_\phi(\boldsymbol{x})\tilde{\phi}_i(\boldsymbol{x}), & \text{if } \tilde{\phi}_i(\boldsymbol{x}) \geqslant 0, \\ 0, & \text{if } \tilde{\phi}_i(\boldsymbol{x}) < 0. \end{cases} \quad (27)$$

Note that $C_\phi(\boldsymbol{x})$ is adjusted to ensure that $\sum_i \phi_i(\boldsymbol{x}) = 1$.

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.

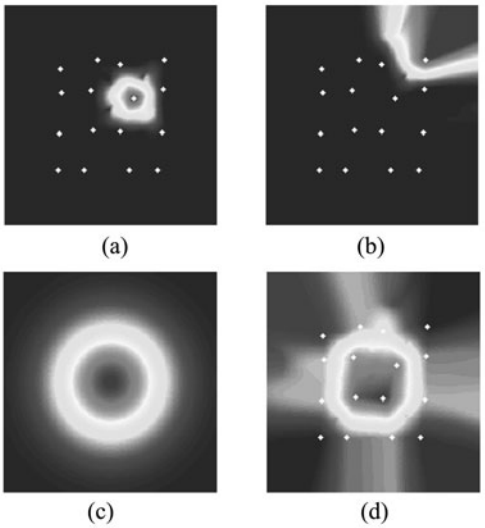


Fig. 1 Illustration of basis functions. (a) and (b) show example basis functions, (c) shows a Gaussian function, and (d) shows the fit to the Gaussian function. Dots represent nodes.

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 $z(\boldsymbol{x})$ is the Gaussian shown in Fig. 1 (c). The function $z^{\mathrm{MLS}}(\boldsymbol{x})$ 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.

4 Solution method

In this section, we describe how we solve 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. Finally, we describe a procedure for adding collocation states so as 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).

4.1 Computing the discretized integral operator

Starting from our discretized problem, as mentioned before, we choose basis functions as (27) in Section 3.3, and we use the nodes themselves as collocation points. Then, we have $P_{ij} = \mathcal{G}[\phi_j](\boldsymbol{x}_i) = \int p(\boldsymbol{x}'|\boldsymbol{x})\phi_j(\boldsymbol{x}')\mathrm{d}\boldsymbol{x}'$.

Since $p(\boldsymbol{x}'|\boldsymbol{x})$ is a Gaussian, this integral can be approximated using cubature formulas, as discussed in Appendix A1. Cubature formulas can be thought of as deterministic sampling, which is designed to match as many moments of the Gaussian distribution as possible.

4.2 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., [8]. Our current implementation uses the classic power iteration method. The reasons for choosing this algorithm are 1) it does not require matrix factorization, which would be very slow when we have a large number of nodes; 2) the result is always positive as long as the initialization is positive, and all elements of P are nonnegative (which is required by design). Thus, the algorithm is gived as follows:

0) Let $\boldsymbol{w}^0$ be an initial guess.

1) $\boldsymbol{w}^n = \dfrac{QP\boldsymbol{w}^{n-1}}{\|QP\boldsymbol{w}^{n-1}\|}$.

2) Repeat step 1) until $\|\boldsymbol{w}^n - \boldsymbol{w}^{n-1}\| < \epsilon$.

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 backward with step h in a finite horizon formulation (see Appendix A2), the speed of convergence corresponds to the mixing rate of the Markov chain. 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. Details are discussed in Appendix A2.

4.3 Finding the optimal control u^*

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

The above probability distribution function is known, and its mean can be calculated. In linearly solvable diffusion process, $u^*(\boldsymbol{x}'|\boldsymbol{x})$ should be a gaussian distribution whose mean moved $hB(\boldsymbol{x})\boldsymbol{u}^*$ from mean of passive dynamics, thus to calculate a control vector $\boldsymbol{u}^*$, we can solve $\boldsymbol{x} + h\boldsymbol{a}(\boldsymbol{x}) + hB(\boldsymbol{x})\boldsymbol{u}^* = E_{\boldsymbol{x}'\sim u^*(\cdot|\boldsymbol{x})}[\boldsymbol{x}']$.

From (3), we have

$$E_{\boldsymbol{x}'\sim u^*(\cdot|\boldsymbol{x})}[\boldsymbol{x}'] = \int u^*(\boldsymbol{x}'|\boldsymbol{x})\boldsymbol{x}'\mathrm{d}\boldsymbol{x}' = \frac{\int p(\boldsymbol{x}'|\boldsymbol{x})\boldsymbol{x}'z(\boldsymbol{x}')\mathrm{d}\boldsymbol{x}'}{\int p(\boldsymbol{x}'|\boldsymbol{x})z(\boldsymbol{x}')\mathrm{d}\boldsymbol{x}'}. \quad (28)$$

Both the numerator and the denominator of the right-hand side of (28) 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 Appendix A1. By this method, both the numerator and the denominator can be expressed as a weighted sum of $z(\boldsymbol{\xi}_i)$, which are the z values at the cubature points.

Numerical errors introduced by (3) and (28) may be problematic when the denominator is very small. However, that happens when $z(\boldsymbol{x})$ is very small, i.e., the cost-to-go function $v(\boldsymbol{x})$ is very large; in other words, $\boldsymbol{x}$ 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 (28), so that the mean of $u^*(\boldsymbol{x}'|\boldsymbol{x})$ lies in the convex hull of the cubature sampling points. This effectively restricts the magnitude of the control signal, which in practice can be a good thing.

4.4 Adapting the nodes

In the 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 to 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 $z(\boldsymbol{x})$, 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 $h\boldsymbol{a}(\boldsymbol{x})$ and $\sqrt{h}\sigma B(\boldsymbol{x})$, which are the magnitude of passive dynamics and transition probabilities. This feature means that 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.

5 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 that the method scales to high-dimensional systems. Finally, we will compare them with other methods.

5.1 Test problems, solution, and dynamical simulation

Here, we use MLS approximation to solve our test problems. We focus on the desirability function $z(\boldsymbol{x})$, cost-to-go function $v(\boldsymbol{x})$, optimal control law $u^*(\boldsymbol{x})$, and dynamical simulations based on $u^*(\boldsymbol{x})$. When possible, we compare these results with dense MDP discretization [4]. Test problems are fomulated as linearly solvable controlled diffusions, as in Section 2.2. Additional details on the test problems are provided in Appendix A5.

5.1.1 Example 1: Car-on-the-hill

This test problem is adapted from [3]. 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 LMDP [4]. 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.

For the average-cost setting, we design a cost function $q(\boldsymbol{x})$ (Fig. 2 (b)) that encourages the car to pass repeatedly via two targets with non zero desired velocity, resulting in a

limit cycle behavior. As shown in Fig. 2 (c)–(h), the cost-go function and optimal control law are consistent with the results of dense MDP discretization. (a) illustrates the model,

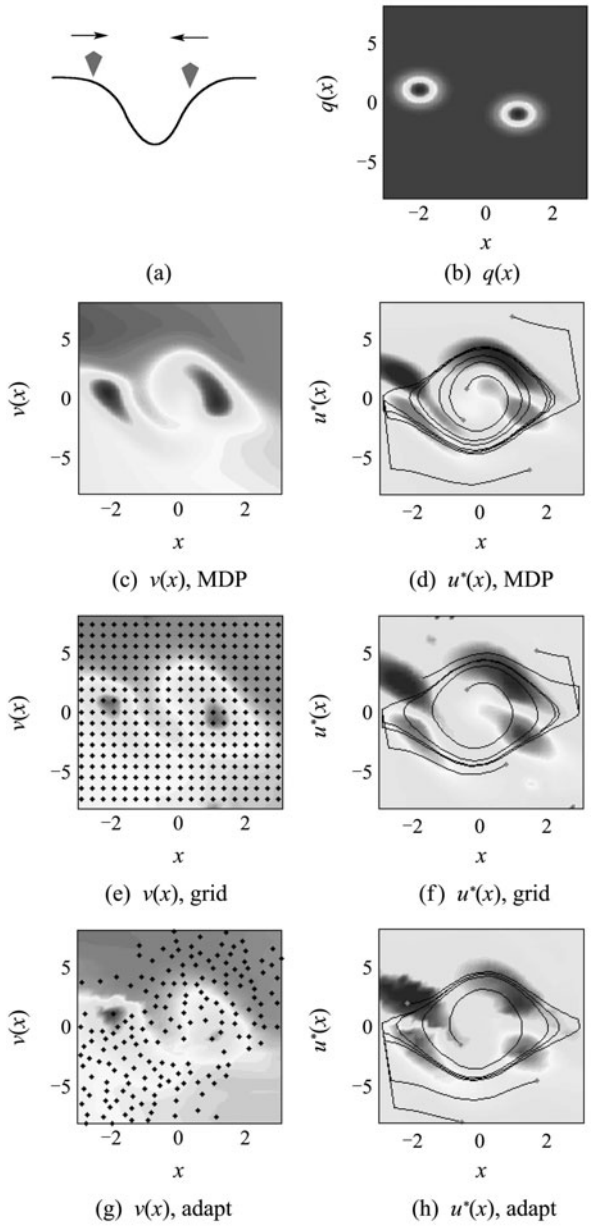


Fig. 2 Results for the car-on-the-hill model with average-cost setting.

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

We can also use this dynamical system to define other optimal control problems: discounted, first exit, and finite horizon. For the discounted setting the cost function is the same as before. For the first exit and finite horizon settings, the goal (encoded as a final cost) is to park the car at horizontal position 2.4; in that case, the running state cost $q(\boldsymbol{x})$ is constant. As shown in Figs. 3–5, the results are again similar to those obtained by dense MDP discretization.

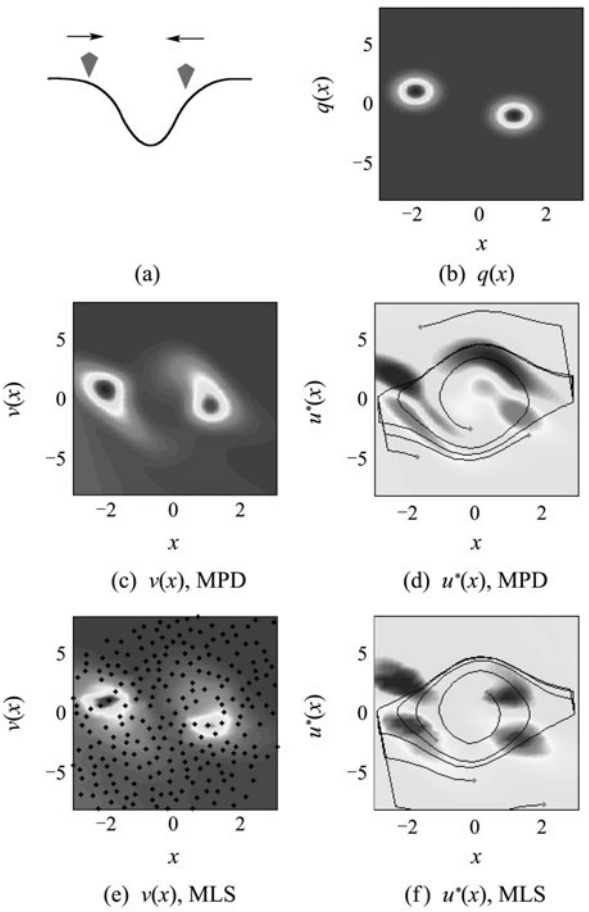
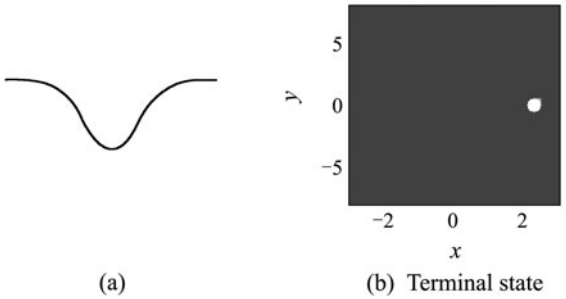


Fig. 3 Results for the car-on-a-hill probelm, discounted cost setting.

Fig. 3 (a) illustrates the model and (c) shows the state cost function $q(\boldsymbol{x})$. (c) and (e) show the cost-to-go function $v(\boldsymbol{x}) = -\log(z(\boldsymbol{x}))$. (d) and (f) show the optimal control law obtained. In (d) and (f), dynamical simulations are shown by black curves, while their initial states are shown by gray dots. (c) and (d) show the results from the MDP method with 151×151 grid. (e) and (f) show the results from the MLS approximation with 251 nodes generated by our adaptive scheme. Black dots in (e) represent nodes. Compared with the average-cost setting, here the optimal control law seems to take advantage of walls (hit the wall, then keep going). MLS with the adaptive scheme is able to capture this property.



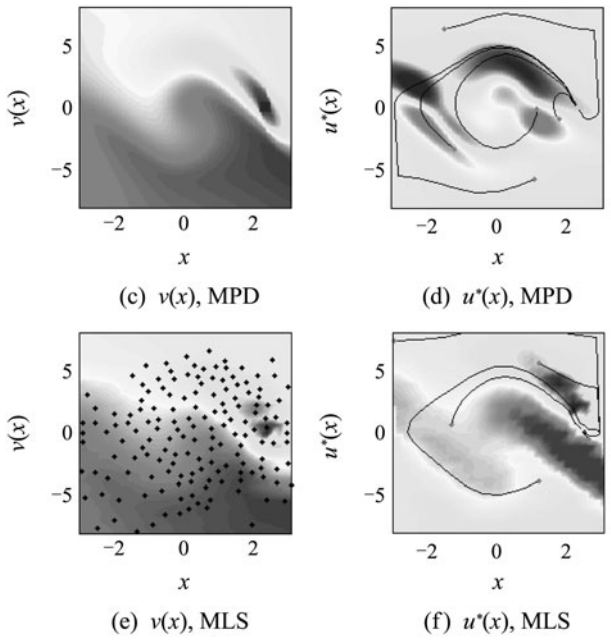


Fig. 4 Results for the car-on-the-hill model with first-exit setting.

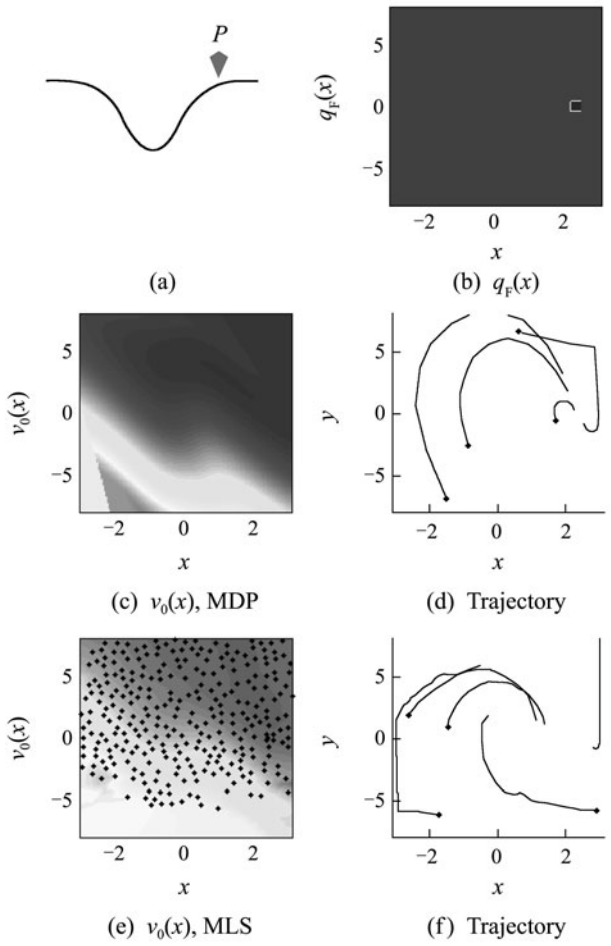


Fig. 5 Results for the car-on-the-hill model with finite-horizon setting.

Fig. 4 (a) illustrates the model and (b) shows the terminal states vs. nonterminal states. Write shows terminal states. (c) and (e) show the cost-to-go function $v(\boldsymbol{x}) = -\log(z(\boldsymbol{x}))$. (d) and (f) show the optimal control law obtained. In (d) and (f), dynamical simulations are shown by curves, while their initial states are shown by gray dots. (c) and (d) show the results from the MDP method with 151×151 grid. (e) and (f) show the results from the MLS approximation with 161 nodes generated by our adaptive scheme. Black dots in (e) represent nodes. The MLS solution is not identical to the MDP solution, yet the resulting control laws are similar.

Fig. 5 (a) illustrates the model and (b) shows the final cost function $q_{\mathrm{F}}(\boldsymbol{x})$. (c) and (e) show the cost-to-go function $v(\boldsymbol{x}) = -\log(z(\boldsymbol{x}))$ at time 0. In (d) and (f), dynamical simulations are shown by black curves, while their initial states are shown by gray dots. (c) and (d) show the results from the MDP method with 151×151 grid. (e) and (f) show the results from the MLS approximation with 334 nodes generated by our adaptive scheme. Black dots in (e) represent nodes. Different from previous settings, here both desirability functions and optimal control law are functions of time, so we focus on showing the dynamical simulation. MLS tends to smooth out the desirability functions and gives a slightly different optimal control law.

5.1.2 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 and 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 achieve 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.

Here, we applied a modification that includes a decay factor when the state is too far away from the existing nodes (This effectively uses an additional control to move the system back to nodes or a hybrid method with the Gaussian approximator in [4]). The formula is shown below. r is the distance to nearest node:

$$\tilde{z}^{\mathrm{MLS}}(\boldsymbol{x}) = \begin{cases} z^{\mathrm{MLS}}(\boldsymbol{x}), & r < R, \\ z^{\mathrm{MLS}}(\boldsymbol{x}) \exp(-\lambda(r-R)^2), & r \geqslant R. \end{cases} \quad (29)$$

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 (seven 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 in places around the limit cycle. Fig. 6 shows the emergence of an attractive trajectory for systems composed of two and seven masses, respectively. The trajectories are shown by projecting to different dimensions. Fig. 6 (a), (c) and (e) shows the results for 2 masses (4-dimensional state space) with 166 nodes. (b), (d) and (f) shows the results for 7 masses (14-dimensional state space) with 2912 nodes. (a) and (b) shows the trajectories of the first mass. (c) and (d) shows the trajectories of the second mass. (e) and (f) shows

the positions of the first and second masses. Note that the analytical solution in each case is a circle with radius 1.

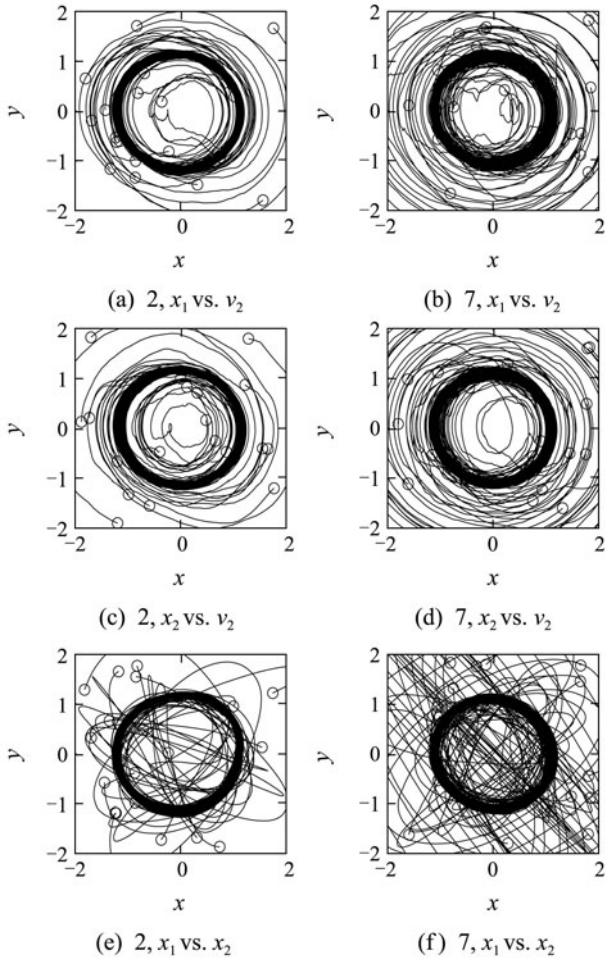


Fig. 6 Stochastic simulation for Example 2 with different dimensionalities.

5.2 Numerical issues

5.2.1 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 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 be 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.

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 (d) and (h), which is obtained with 17 iterations of our adaptive scheme. Here, Fig. 7 (a) and (b) shows results with initial nodes. The second column (c) and (d) shows the fourth iteration, while (e) and (f) show the ninth iteration. (a), (c) and (e) shows the cost-to-go function, where black dots represent the nodes. (b), (d) and (f) show the optimal control law obtained.

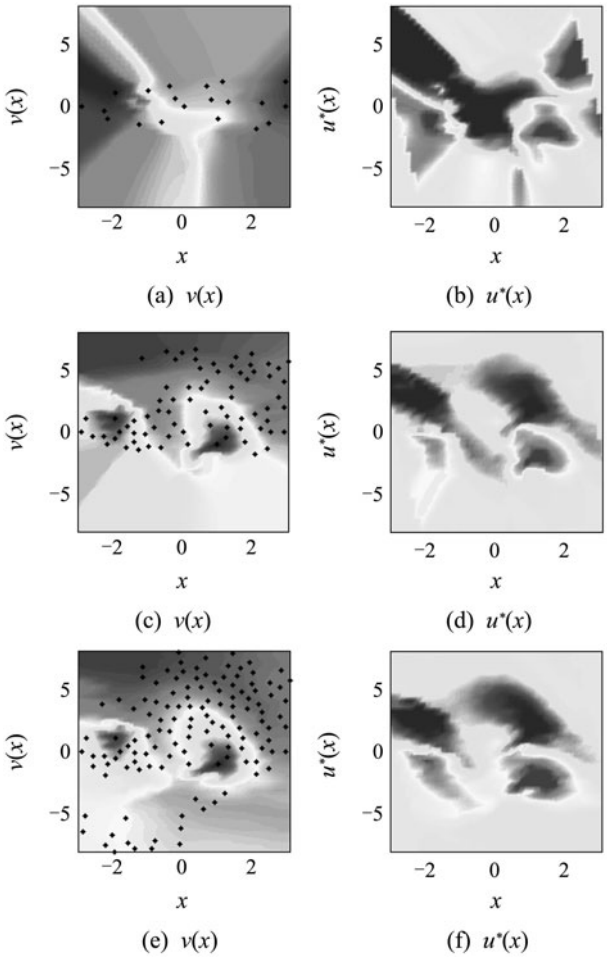


Fig. 7 Demonstration of the adaptive scheme.

5.2.2 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 be-

comes 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 4.1) using the MLS method. The time complexity of this construction is $\mathrm{O}(n^3)$, where n is the number of state space dimensions. The cubature formula has time complexity $\mathrm{O}(n_{\mathrm{c}}^2)$, 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 $\mathrm{O}(NKn^3n_{\mathrm{c}}^2)$. The CPU time for constructing the discretized operator in Example 2, with the final set of nodes, was done as follows. For two masses, with 100–200 nodes, it takes around 0.02 second. For seven 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 each current node. 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 discussed as follows. For two masses, it takes 1–3 seconds. For seven 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.

5.3 Comparison with other methods

5.3.1 Comparison with MDP method

Previous results have demonstrated the similarity of results given by both methods. Theoretically, MDP can naturally give a 'ground-truth' solution, while MLS tends to yield bigger error.

However, MLS would outperform MDP in practice. First of all, normally, MLS costs much less CPU time than MDP. For example, in the example shown in Fig. 2, MDP method (Fig. 2 (c) and (d)) takes an hour in MATLAB, while MLS method (Fig. 2 (e) and (f)) takes 0.1 seconds without adaption 0.09 seconds with adaption (Fig. 2(g) and (h)) (Programming details and complexity of model may affect those estimates, but normally, MDP would run much slower than MLS). In both cases, most of CPU time is spent in constructing a matrix (probability transition matrix for MDP, discretized operator for MLS). In high-dimensional practical control problems, the MDP method is not numerically possible, since it needs dense grids on every dimension on state space. Second, since the MDP method is to approximate a problem within a finite box in state space, it may yield suspicious results near boundaries (thus, in Section 5.1, we perform MDP in a bigger region than shown). MLS tends to have less suspicious results near boundaries due to its exploration power.

5.3.2 Comparison with Gaussian method

The method presented here differs significantly from the method using gaussians presented in [4] besides different choices of bases functions. First, the method here uses power iteration rather than quadratic optimization or nonlinear optimization. Second, the method here uses nonparametric way to adapt the bases. Our comparison need to take these factors into account.

First, we test the gaussian method on Example 2 with similar nonparametric adaptive scheme, as described here in Section 4.4. The covariances of gaussians are allocated based on distance between each other, thus provides a rather fair comparison with the MLS method. MATLAB built-in optimizer 'quadprog' is used to perform quadratic programming. The results are similar to that in Section 5.1.2 with 2, 3, and 4 masses but fail for more masses. We can conclude that MLS bases have the following advantages over gaussian bases. First, solving the resulting eigen problem cost less CPU time than gaussian bases. Second, MLS method is less prone to converge to the wrong eigenvector. The disadvantage of MLS bases is that constructing MLS bases tends to comsume more time than constructing gaussian bases. The total CPU time consumed by either method is similar. In conclusion, when scaling to high dimensionality, if the number of bases grows, or the results cannot easily be verified, MLS tends to outperform gausian bases.

Then, we test the adaptive scheme in Section 4.4 by comparing above results with results from gaussian bases with gradient descent. MATLAB built-in optimizer 'fmincon' is used to perform nonlinear optimization. When gaussian bases are initially placed properly (near trajectories), the gradient descent method may normally converge to acceptable results. The results are similar to that in Section 5.1.2 with 2, 3, 4, and 5 masses, but fail for more masses. The time consumed with gradient descent methods is much higher than non parametric adaptation if they converge to acceptable results. For example, with four masses, gradient descent bases adaptation needs 46 seconds with 50 bases, while adaptive acheme in Section 4.4 with 161 gaussian bases needs 0.17 seconds, and adaptive scheme in Section 4.4 with 191 MLS bases needs 0.06 seconds. This is likely due to the need to repeatly construct gaussian bases. Moreever, gradient descent method is very likely to converge to wrong eigenvector (λ very small) when, initially, bases are not put properly. To conclude, the method presented here outperform the gaussian approximation method with gradient descent presented in [4].

6 Conclusions

Here, we developed a new function approximation method for linearly solvable stochastic optimal control problems. Numerical tests show its effectiveness. Future 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.

Acknowledgements

We would like to thank Yuval Tassa for helpful discussion.

References

[1] H. Kappen. Linear theory for control of nonlinear stochastic systems. *Physics Review Letters*, 2005, 95(20): 200 – 201.

[2] E. Todorov. Linearly-solvable Markov decision problems. *Advances in Neural Information Processing Systems*, Cambridge: MIT press, 2006: 1369 – 1376.

[3] E. Todorov. Efficient computation of optimal actions. *PNAS*, 2009, 106(28): 11478 – 11483.

[4] E. Todorov. Eigen-function approximation methods for linearly-solvable optimal control problems. *IEEE International Symposium on Adaptive Dynamic Programming and Reinforcemenet Learning*. New York: IEEE, 2009: 161 – 168.

[5] T. Belytschko, Y. Krongauz, D. Organ, et al. Meshless methods: An overview and recent development. *Computer Mechanics Engineering*, 1996, 139(1/4): 3 – 47 .

[6] C. G. Atkeson, A. W. Moore, S. Schaal. Locally weighted learning. *Artificial Intelligence Review*, 1997, 11(1/5): 11 – 73.

[7] C. G. Atkeson, A. W. Moore, S. Schaal. Locally weighted learning for control. *Artificial Intelligence Review*, 1997, 11(1/5): 75 – 113.

[8] LN Trefethen, D. Bau. *Numerical Linear Algebra*. Philadelphia: SIAM, 1997

[9] J. Lu, D. L. Darmofal. Higher-dimensional integration with Gaussian weight for applications in probabilistic design. *SIAM Journal Science Computer*, 2005, 26(2): 613 – 624.

Appendix

A1 Calculating the expectation of an arbitrary function under a normal distribution using the cubature method

We want to evaluate the expectation of a function under 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 [9].

The goal is to calculate

$$E[g(\boldsymbol{x})] = \int g(\boldsymbol{y}) p(\boldsymbol{y}|\boldsymbol{x}) \mathrm{d}\boldsymbol{y}. \tag{a1}$$

As in (8),

$$p(\boldsymbol{y}|\boldsymbol{x}) = \mathcal{N}(\boldsymbol{x} + h\boldsymbol{a}(\boldsymbol{x}), h\Sigma(\boldsymbol{x})), \;\; h\Sigma(\boldsymbol{x}) = CC^{\mathrm{T}}, \tag{a2}$$

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

We need a variable transformation

$$\boldsymbol{y} = C\sqrt{2}\boldsymbol{\xi} + \boldsymbol{x} + h\boldsymbol{a}(\boldsymbol{x}) \tag{a3}$$

to convert the integral into the standard form of cubature formulas.

$$\begin{cases} E[g(\boldsymbol{x})] = C_0 \int f(\boldsymbol{\xi}) \exp(-\boldsymbol{\xi}^{\mathrm{T}}\boldsymbol{\xi}) \mathrm{d}\boldsymbol{\xi}, \\ C_0 = \dfrac{1}{(\pi)^{\frac{N}{2}}}, \;\; f(\boldsymbol{\xi}) = g(C\sqrt{2}\boldsymbol{\xi} + \boldsymbol{x} + h\boldsymbol{a}(\boldsymbol{x})). \end{cases} \tag{a4}$$

Now, we can apply the cubature scheme in [9], which uses the following formula to approximate the integral.

$$\int f(\boldsymbol{\xi}) \mathrm{d}\boldsymbol{\xi} \approx Q[f] = \sum_{j=1}^{N} w_j f(\boldsymbol{\xi}^j), \tag{a5}$$

where w_j are weights, and $\boldsymbol{\xi}^j$ 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 (a6) in [9] works better in some test problems. This is a degree 5 formula with $2n_{\mathrm{C}}^2 + 1$ points, n_{C} is the number of columns of the $C(\boldsymbol{x})$ matrix in (a2), 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.

$$\begin{aligned} Q[f] = {} & \frac{n^2 - 7n + 18}{18} \pi^{\frac{n}{2}} f(\boldsymbol{0}) \\ & + \frac{4 - n}{18} \pi^{\frac{n}{2}} \sum_{\text{full sym.}} f(\sqrt{3/2}, 0, \ldots, 0) \\ & + \frac{1}{36} \pi^{\frac{n}{2}} \sum_{\text{full sym.}} f(\sqrt{3/2}, \sqrt{3/2}, \ldots, 0). \end{aligned} \tag{a6}$$

In summary, to calculate the integral (a1) approximately, we used (a4) and (a2); therefore, the following formula represents this process:

$$E[g(\boldsymbol{x})] = \int g(\boldsymbol{y}) p(\boldsymbol{y}|\boldsymbol{x}) \mathrm{d}\boldsymbol{y} = C_0 \sum_{j=1}^{N} w_j f(\boldsymbol{\xi}^j). \tag{a7}$$

More work is needed to estimate the error introduced in this step, and the effects of using different cubature formulas.

A2 Discretized problems and algorithms for LMDP problems in different settings

This paper mainly addresses the LMDP problem in the average-cost setting. Nevertheless, the proposed method is applicable to other settings as well. Here, we describe the Bellman equation for all settings, their discretized versions, and the corresponding algorithms. Table a1 is provided to summarize the results.

As in the main text, $\boldsymbol{x}_i$, $i = 1, \ldots, N$ are the nodes/collocation points, and $\phi_i(\boldsymbol{x})$ are their corresponding basis functions. Q is an $N \times N$ diagonal matrix with diagonal entries $Q_{ii} = \exp(-hq(\boldsymbol{x}_i))$. We still have

$$P_{ij} = \mathcal{G}[\phi_j](\boldsymbol{x}_i), \;\; i = 1, \ldots, N, \; j = 1, \ldots, N. \tag{a8}$$

A2.1 Finite horizon

In the finite horizon setting, costs are accumulated from time 0 to time T. $q_{\mathrm{F}}(\boldsymbol{x})$ represents the final cost ealuated at time T.

The Bellman equation for the finite-horizon setting [3] is

$$z_t(\boldsymbol{x}) = \exp(-hq(\boldsymbol{x})) \mathcal{G}[z_{t+h}(\boldsymbol{x})], \tag{a9}$$

$$z_T(\boldsymbol{x}) = \exp(-q_{\mathrm{F}}(\boldsymbol{x})). \tag{a10}$$

If we represent the desirability function with the same set of basis functions at each time step, then

$$z_t(\boldsymbol{x}) = \sum_{i=1}^{N} w_{t,i} \phi_i(\boldsymbol{x}). \tag{a11}$$

Defining a column vector $\boldsymbol{w}_t$ with elements $w_{t,i}$, (a9) becomes

$$\boldsymbol{w}_t = QP\boldsymbol{w}_{t+h}. \tag{a12}$$

Due to the nature of our basis functions, we can use the following approximation($\approx$ would become $=$ if $\boldsymbol{x} = \boldsymbol{x}_i$):

$$z(\boldsymbol{x}) \approx \sum_{i=1}^{N} z(\boldsymbol{x}_i) \phi_i(\boldsymbol{x}). \tag{a13}$$

Applying (a13) to (a10), we will have

$$\boldsymbol{w}_{\mathrm{T}} = \exp(-q_{\mathrm{F}}(\boldsymbol{x}_i)). \tag{a14}$$

Using (a12) iteratively for T/h times from (a14), we would get the approximated desirability function at time 0:

$$z_0(\boldsymbol{x}) \approx \sum_{i=1}^{N} w_{0,i} \, \phi_i(\boldsymbol{x}). \tag{a15}$$

A2.2 Infinite horizon, discounted cost

In the discounted-cost setting, costs are accumulated from time 0 to infinity, however, future costs decay exponentially as $\exp(-t/\tau)$. Let $\alpha = \exp(-h/\tau) < 1$ represent the corresponding discretized discount factor, where h is the time step.

The Bellman equation for the discounted-cost setting [3] is

$$z(\boldsymbol{x}) = \exp(-hq(\boldsymbol{x})) \mathcal{G}[z^{\alpha}(\boldsymbol{x})]. \tag{a16}$$

We will use our basis functions to approximate the desirability function:

$$z(\boldsymbol{x}) = \sum_{i=1}^{N} w_i \phi_i(\boldsymbol{x}). \tag{a17}$$

Due to the nature of our basis functions, we can use the following approximation($\approx$ would become $=$ if $\boldsymbol{x} = \boldsymbol{x}_i$):

$$z^{\alpha}(\boldsymbol{x}) \approx \sum_{i=1}^{N} w_i^{\alpha}\phi_i(\boldsymbol{x}). \tag{a18}$$

Thus, the problem is discretized as

$$\boldsymbol{w} = QP\boldsymbol{w}^{\alpha}. \tag{a19}$$

Here, $\boldsymbol{w}$ is a column vector with elements w_i.

There exist alternative ways to solve the equation (a19), but here, we will focus on the iterative methods to avoid matrix factorization.

One may naturally try to use the following iterative formula (a20) to solve this problem:

$$\boldsymbol{w}^{k+1} = QP(\boldsymbol{w}^k)^{\alpha}. \tag{a20}$$

However, this may not work well when α is close to 1. There are two possibilities: 1) the decay of cost is slow; 2) the time step h is too small. In these cases, elements of $\boldsymbol{w}$ would become undistinguishable from 0, and the rate of convergence would become very slow. This situation is not so important since the results are close to what the average-cost setting may give. However, if one wants to overcome the numerical difficulty, the following algorithm would work:

$$\begin{cases} \tilde{\boldsymbol{w}}^{k+1} = \tilde{\lambda}_k QP(\tilde{\boldsymbol{w}}^k)^{\alpha}, \\ \tilde{\lambda}_k = \|QP(\tilde{\boldsymbol{w}}^k)^{\alpha}\|^{-1}, \ \ \boldsymbol{w}^k = (\tilde{\lambda}^k)^{\frac{1}{\alpha-1}}\tilde{\boldsymbol{w}}^k. \end{cases} \tag{a21}$$

This is similar to the power iteration method in the average-cost case. In our simulation, (a20) performs no worse than the simple power iteration in average-cost setting. On the other hand, if we only need the optimal control law, since it is invariant when the desirability function is multiplied by a constant, there is no need to evaluate $\boldsymbol{w}^k$ explicitly.

A2.3 First exit

In the first-exit setting, costs are accumulated from time 0 to infinity, however, accumulation would stop when the system reaches a terminal state. Here, $\mathcal{T} \in \mathbb{R}^n$ represents the subset of terminal states, and $\mathcal{N} \in \mathbb{R}^n$ represents the subset of nonterminal states. $q_{\mathcal{T}}(\boldsymbol{x})$ represents the cost at terminal states.

The Bellman equation would become a boundary value problem [3]:

$$z(\boldsymbol{x}) = \exp(-hq(\boldsymbol{x}))\mathcal{G}[z(\boldsymbol{x})], \tag{a22}$$

$$z(\boldsymbol{x}) = \exp(-q_{\mathcal{T}}(\boldsymbol{x})), \ \ x \in \mathcal{T}. \tag{a23}$$

(a22) can be further rewritten as

$$\begin{aligned} &\exp(hq(\boldsymbol{x}))z(\boldsymbol{x}) \\ &= \mathcal{G}_{\mathcal{N}}[z(\boldsymbol{x})] + \mathcal{G}_{\mathcal{T}}[\exp(-q_{\mathcal{T}}(\boldsymbol{x}))], \ \ x \in \mathbb{N}, \end{aligned} \tag{a24}$$

where operators are defined as integrals on finite regions

$$\mathcal{G}_{\mathcal{N}}[z(\boldsymbol{x})] = \int_{\boldsymbol{x}\in\mathbb{N}} z(\boldsymbol{x}')p(\boldsymbol{x}'|\boldsymbol{x})\mathrm{d}\boldsymbol{x}', \tag{a25}$$

$$\mathcal{G}_{\mathcal{T}}[z(\boldsymbol{x})] = \int_{\boldsymbol{x}\in\mathcal{T}} z(\boldsymbol{x}')p(\boldsymbol{x}'|\boldsymbol{x})\mathrm{d}\boldsymbol{x}'. \tag{a26}$$

We can construct a matrix $P_{\mathcal{N}}$ with integrals only at nonterminal states:

$$P_{\mathcal{N},ij} = \mathcal{G}_{\mathcal{N}}[\phi_j](\boldsymbol{x}_i), \ \ i = 1,\ldots,N, j = 1,\ldots,N, \tag{a27}$$

and a vector $\boldsymbol{r}$ with elements as $r_i = \mathcal{G}_{\mathcal{T}}[\exp(-q_{\mathcal{T}}(\boldsymbol{x}_i))]$. $\int_{\boldsymbol{x}\in\mathbb{N}}$ is usually not different than the integral on the entire space, because the bases are localized, and terminal states are normally limited in a small region. Therefore, most $P_{\mathcal{N},ij}$ can be evaluated with the cubature formula, as in Appendix A1. Otherwise, if the collocations state is close to the terminal states, discretization in grids are needed to evaluate $P_{\mathcal{N},ij}$.

If we use basis functions to approximate the desirability function (a17), we will obtain the discretized formula, which is a linear equation:

$$\boldsymbol{w} = Q(P_{\mathcal{N}}\boldsymbol{w} + \boldsymbol{r}). \tag{a28}$$

Many algorithms are available for solving this kind of problems (backslash in MATLAB, for example). One iterative method will take the form

$$\boldsymbol{w}^{k+1} = QP_{\mathcal{N}}\boldsymbol{w}^k + Q\boldsymbol{r}. \tag{a29}$$

This algorithm would terminate when the modulus of the leading eigenvalue of $QP_{\mathcal{N}}$ is smaller than 1, which is true when $q(\boldsymbol{x}) > 0$ everywhere, i.e., the first exit problem is well defined.

A2.4 Summary

Table a1 summarizes the Bellman equations, their discretized correspondences, and algorithms. Some details are omitted.

In summary, it is possible to use the MLS approximation to solve the LMDP problems posed in all four settings [3]. Iterative algorithms are available, so no matrix factorization are needed when matrix P ($P_{\mathcal{N}}$ for first-exit) is constructed. This feature provides efficiency when dealing with large number of base functions.

Table a1 Discretized problems and algorithms.

Setting	Bellman equation	Discretized	Algorithm
Average	$z(\boldsymbol{x}) = \exp(hc - hq(\boldsymbol{x}))\mathcal{G}[z(\boldsymbol{x})]$	$\boldsymbol{w} = \lambda QP\boldsymbol{w}$	$\boldsymbol{w}^{k+1} = QP\boldsymbol{w}^k/(\|QP\boldsymbol{w}^k\|)$
Finite	$z_t(\boldsymbol{x}) = \exp(-hq(\boldsymbol{x}))\mathcal{G}[z_{t+h}(\boldsymbol{x})]$	$\boldsymbol{w}_t = QP\boldsymbol{w}_{t+h}$	$\boldsymbol{w}_t = QP\boldsymbol{w}_{t+h}$
Discount	$z(\boldsymbol{x}) = \exp(-hq(\boldsymbol{x}))\mathcal{G}[z^{\alpha}(\boldsymbol{x})]$	$\boldsymbol{w} = QP\boldsymbol{w}^{\alpha}$	$\boldsymbol{w}^{k+1} = QP(\boldsymbol{w}^k)^{\alpha}$
First exit	$z(\boldsymbol{x}) = \exp(-hq(\boldsymbol{x}))\mathcal{G}[z(\boldsymbol{x})]$	$\boldsymbol{w} = Q(P_{\mathcal{N}}\boldsymbol{w} + \boldsymbol{r})$	$\boldsymbol{w}^{k+1} = QP_{\mathcal{N}}\boldsymbol{w}^k + Q\boldsymbol{r}$

A3 Details of MLS basis function construction

In the main text, we omitted the details about how to obtain the basis functions before truncation and renormalization, i.e., $\tilde{\phi}_i(\boldsymbol{x})$ in (25) and (26). The optimization problem (25) is solved via a linear equation, the same way as in the ordinary least square methods.

Let us start from the optimization problem (25)

$$\begin{aligned} &\boldsymbol{\alpha}(\boldsymbol{x}) = \arg\min_{\alpha} \sum_{I=2}^{K} \omega(\boldsymbol{x}_I - \boldsymbol{x})(\boldsymbol{h}(\boldsymbol{x}_I)^{\mathrm{T}}\boldsymbol{\alpha} - z_{(I)})^2, \\ &\text{s.t. } \boldsymbol{h}(\boldsymbol{x}_1)^{\mathrm{T}}\boldsymbol{\alpha} = z_{(1)}. \end{aligned}$$

Using a Lagrange multiplier $\tilde{\lambda}$, we can rewrite the optimization problem as

$$\begin{aligned} \boldsymbol{\alpha}(\boldsymbol{x}) = \arg\min_{\alpha} &\sum_{I=2}^{K} \omega(\boldsymbol{x}_I - \boldsymbol{x})(\boldsymbol{h}(\boldsymbol{x}_I)^{\mathrm{T}}\boldsymbol{\alpha} - z_{(I)})^2 \\ &+\tilde{\lambda}(\boldsymbol{h}(\boldsymbol{x}_1)^{\mathrm{T}}\boldsymbol{\alpha} - z_{(1)}). \end{aligned} \tag{a30}$$

This is equivalent to solving for $\boldsymbol{\alpha}, \tilde{\lambda}$ in the linear equation:

$$M_1(\boldsymbol{x})[\boldsymbol{\alpha}(\boldsymbol{x})^{\mathrm{T}} \ \ \tilde{\lambda}]^{\mathrm{T}} - M_2(\boldsymbol{x})\boldsymbol{z} = 0, \tag{a31}$$

where

$$\boldsymbol{z} = [z_{(1)} \ \ \cdots \ \ z_{(K)}]^{\mathrm{T}}, \tag{a32}$$

$$M_1 = \begin{pmatrix} H_2^{\mathrm{T}}WH_2 & H_1^{\mathrm{T}} \\ H_1 & 0 \end{pmatrix}, \tag{a33}$$

$$M_2 = \begin{pmatrix} \boldsymbol{0} & H_2^{\mathrm{T}}W \\ 1 & \boldsymbol{0} \end{pmatrix}, \tag{a34}$$

$$H_1 = \boldsymbol{h}(\boldsymbol{x}_1)^{\mathrm{T}} = (1 \ \ \boldsymbol{x}_1^{\mathrm{T}}), \tag{a35}$$

$$H_2 = \begin{pmatrix} \boldsymbol{h}(\boldsymbol{x_2})^{\mathrm{T}} \\ \vdots \\ \boldsymbol{h}(\boldsymbol{x_K})^{\mathrm{T}} \end{pmatrix} = \begin{pmatrix} 1 & \boldsymbol{x}_2^{\mathrm{T}} \\ \vdots & \vdots \\ 1 & \boldsymbol{x}_K^{\mathrm{T}} \end{pmatrix}, \tag{a36}$$

and W is a diagonal matrix representing weights:

$$W = \begin{pmatrix} \omega(x-x_2) & 0 & \cdots & 0 \\ 0 & \omega(x-x_3) & \cdots & 0 \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \cdots & \omega(x-x_K) \end{pmatrix}. \quad \text{(a37)}$$

Therefore, from (a31), we have

$$[\boldsymbol{\alpha}(\boldsymbol{x})^{\mathrm{T}} \ \tilde{\lambda}]^{\mathrm{T}} = M_1^{-1}(\boldsymbol{x}) M_2(\boldsymbol{x}) \boldsymbol{z}. \quad \text{(a38)}$$

Comparing with (26)

$$z^{\mathrm{MLS}}(\boldsymbol{x}) = \boldsymbol{h}(\boldsymbol{x})^{\mathrm{T}} \boldsymbol{\alpha}(\boldsymbol{x}) = \sum_i \tilde{\phi}_i(\boldsymbol{x}) z_i,$$

we have

$$[\tilde{\phi}_{(1)}(\boldsymbol{x}) \ \cdots \ \tilde{\phi}_{(K)}(\boldsymbol{x})] = \boldsymbol{h}(\boldsymbol{x})^{\mathrm{T}} \left(M_1(\boldsymbol{x})^{-1} M_2(\boldsymbol{x})\right)_{n+1}, \quad \text{(a39)}$$

where $(\cdot)_{n+1}$ denotes the operation that takes the first $n+1$ rows of a matrix (since we ignore the Lagrange multiplier), and n is the number of dimensions in the dynamical system.

(a39) gives how we construct basis functions from the optimization problem (25). Since M_1 matrix is an $(n+2)\times(n+2)$ matrix, when we perform $M_1^{-1}M_2$ with Cholesky decomposion, the computational complexity of (a39) is $\mathrm{O}(n^3)$.

A4 Details of the adaptive node-placement scheme

In the main text, we omitted the details of the implementation of our adaptive scheme. Details of the algorithm are given here.

Several general rules apply: 1) All nodes are classified into normal nodes and 'pioneer' nodes. New nodes can only be generated from 'pioneer' nodes. 2) All new nodes should lie sufficiently far away from old nodes. This distance lower bound is basically a constant, but when an old node has a very small $z(x)$ value, it would 'occupy' larger space to avoid new nodes being created. 3) The user has to specify a range for states and assume that the range is large enough to cover the region of interest. 4) Initially, all nodes are pioneer nodes.

a) If any 'pioneer' nodes lie outside of the specified range, they would turn to normal nodes.

b) If any 'pioneer' nodes satisfy $z(x) < \theta_n$ for several iterations, where θ_n is a manually defined parameter, then they would turn to normal nodes.

c) If any normal nodes satisfy $z(x) < \theta_{\mathrm{c}}$ for several times, where θ_{c} is a manually defined parameter, then they would occupy a larger region to avoid new nodes to be created.

d) If any 'pioneer' nodes failed to generate new nodes for several iterations, they would turn to normal nodes.

e) Generate new nodes from 'pioneer' nodes. They may come from both dynamics and random perturbation. (optional) Run dynamical simulation from known initial points, and use the resulting trajectory to generate nodes.

Here, a) is used to avoid exploring less interesting region. b) and c) is used to avoid exploring the region where the cost-to-go function is high. d) is used to avoid exploring where the nodes is dense enough. In e), we generate new nodes both from calculated optimal control and random perturbation, since the calculated optimal control law might not be accurate. If some information about the system is provided, nodes may be generated from dynamical simulation.

This method would terminate in finite iterations, which is because only those nodes in the specific region can be 'pioneer' nodes, and the space in this region is limited; then, finally, all of those 'pioneer' nodes would turn to normal thus stop iterations. In practice, our method hardly terminate in this way, since filling a high dimensional means a tremendous amount of nodes. On the other hand, we lack theoretical results about whether the resulting optimal control law will converge to the true solution or not.

A5 Details of test problems

Our test problems are linearly solvable controlled diffusions (Section 2.2). Details are listed here.

A5.1 Car-on-the-hill

This test problem is adapted from [4] with 2D state space and 1D control space. This dynamical system is a point mass(a car) moving along a valley-shaped curve(an inverted Gaussian) with the existence of gravity. Thus, $\boldsymbol{x} = [x_{\mathrm{p}} \ x_{\mathrm{v}}]^{\mathrm{T}}$, where x_{p} denotes horizontal position, and x_{v} denotes tangential velocity. The passive dynamics can be defined as

$$\boldsymbol{a}(\boldsymbol{x}) = \begin{pmatrix} x_{\mathrm{v}}(1+s(x_{\mathrm{p}})^2)^{-\frac{1}{2}} \\ -9.8\mathrm{sgn}(x_{\mathrm{p}})(1+s(x_{\mathrm{p}})^{-2})^{-\frac{1}{2}} \end{pmatrix}, \quad \text{(a40)}$$

where $s(x_{\mathrm{p}}) = x_{\mathrm{p}}\exp(-x_{\mathrm{p}}^2/2)$ is the slope of the inverted gaussian 'hill' at x_{p} position. $B(\boldsymbol{x}) = [0 \ \ 1]^{\mathrm{T}}$. Only the velocity of the car is controlled. Magnitude of noise is $\sigma = 3$, while time step is set as h=0.1. If $x_{\mathrm{p}} < x_{\min}$, at the next time step, $x_{\mathrm{p}} = x_{\min}$. If $x_{\mathrm{p}} > x_{\max}$, at the next time step, $x_{\mathrm{p}} = x_{\max}$. In both cases, $B = [0;0]$(controller would not work when 'hit the wall'). The time step is $h = 0.1$ (finite-horizon setting uses $h = 0.04$). Cost functions for each formulation are discussed in the following sections.

1) Infinite-horizon average cost formulation.

State cost function is defined as

$$q(\boldsymbol{x}) = 2(2 - \exp((x_{\mathrm{p}}-1)^2 + (x_{\mathrm{v}}+1)^2) - \exp((x_{\mathrm{p}}+2)^2 + (x_{\mathrm{v}}-1)^2). \quad \text{(a41)}$$

It would keep the car passing those two targets ($x_{\mathrm{p}} = -2, 1$ with velocities $x_{\mathrm{v}} = 1, -1$) and obtain a limit cycle behavior.

2) Discounted cost formulation.

State cost function is the same as before. Costs decay in rate of $\exp(-t/\tau)$, where $\tau = 0.5$.

3) A car-on-the-hill problem, first-exit formulation.

State cost and exit cost are defined as

$$q(\boldsymbol{x}) = 2, \ \ q_{\mathrm{T}}(\boldsymbol{x}) = 0 \quad \text{(a42)}$$

The terminal states are within $[-2.2, 2.6] \times [-0.4, 0.4]$.

4) A car-on-the-hill problem, finite-horizon formulation.

State cost and terminal cost are defined as

$$\begin{cases} q(\boldsymbol{x}) = 2, \\ q_{\mathrm{F}}(\boldsymbol{x}) = \begin{cases} 0, & \text{when } \boldsymbol{x} \in \text{goal states}, \\ 100, & \text{when } \boldsymbol{x} \notin \text{goal states}. \end{cases} \end{cases} \quad \text{(a43)}$$

The goal states are within $[-2.1, 2.7] \times [-.4, .4]$.

A5.2 Coupling independent masses

This model simulates M masses attached on ideal frictionless springs (or equivalently, electrical oscillators), which are dynamically independent. Each mass yields oscillation on any amplitude, whose determinant dynamical behavior can be expressed as $\dot{x}_{p,i} = Cx_{v,i}, \dot{x}_{v,i} = -Cx_{p,i}$, where $C = 2\pi$ is a constant. The control is in the form of a force that would only apply on the velocity. The time step h is 0.01, $\sigma = 1$. Hence, the state space is $2M$ dimensional, with M freedom of control. Thus, $\boldsymbol{a}(\boldsymbol{x})$ and $B(\boldsymbol{x})$ can be easily written as a linear function and a constant matrix, respectively.

The state cost function is designed to achieve the goal, which are 1) fixing the amplitude to 1 and 2) making the (i)th mass be $\pi/2$ phase ahead of the $(i+1)$th mass. We would like to use the

average-cost formulation to solve the problem.

The state cost function is

$$q(\boldsymbol{x}) = \sum_{i=1}^{m} q_a([x_i \;\; v_i]^{\mathrm{T}}) + \sum_{i=1}^{m-1} q_b([x_i \;\; v_i]^{\mathrm{T}}, [x_{i+1} \;\; v_{i+1}]^{\mathrm{T}}). \tag{a44}$$

Here, first kind of parts makes $x^2 + v^2$ be one, i.e.,

$$q_a([x_i \;\; v_i]^{\mathrm{T}}) = 20(\sqrt{x_i^2 + v_i^2} - 1)^2. \tag{a45}$$

The second kind of parts tries to create the $\pi/2$ phase shift

$$q_b([x_i, v_i]^{\mathrm{T}}, [x_{i+1}, v_{i+1}]^{\mathrm{T}}) = 50 f_\epsilon(x_i^2 + v_i^2) f_\epsilon(x_{i+1}^2 + v_{i+1}^2) f_\theta(\theta_{i+1}, \theta_i), \tag{a46}$$

$$\begin{cases} x_i = \cos\theta_i, \\ v_i = \sin\theta_i, \end{cases} \quad f_\epsilon(\eta) = \begin{cases} 1, \eta \geqslant 0.1, \\ 0, \eta < 0.1, \end{cases} \tag{a47}$$

$$f_\theta(\theta_{i+1}, \theta_i) = (1 - \cos(\theta_{i+1} - \theta_i - \pi/2)). \tag{a48}$$

Here, $f_\theta(\theta_{i+1}, \theta_i)$ is a function designed to achieve desired phase shift, and $f_\epsilon(\eta)$ is a function employed to avoid numerical difficulties when $x_i^2 + v_i^2$ is too close to zero (where angles θ_i are not well defined).

Mingyuan ZHONG received his B.S. degree in Physics from Peking University, China in 2006, and M.S. degree in Applied Mathematics from University of Washington, Seattle, U.S.A. He is currently a Ph.D. candidate at the Department of Applied Mathematics, University of Washington, Seattle, U.S.A. His research interests include optimal control problems and their corresponding approximation methods. E-mail: zhongmy@u.washington.

Emanuel TODOROV obtained his Ph.D. from the MIT in 1998. He was an assistant professor at the University of California San Diego, and is now an associate professor at the University of Washington. His research focus is control in biology and engineering. E-mail: todorov@cs.washington.edu.–
– – – – – – – – – – – – – – –