Eigenfunction approximation methods for linearly-solvable optimal control problems

Emanuel Todorov

***Abstract*— We have identified a general class of nonlinear stochastic optimal control problems which can be reduced to computing the principal eigenfunction of a linear operator. Here we develop function approximation methods exploiting this inherent linearity. First we discretize the time axis in a novel way, yielding an integral operator that approximates not only our control problems but also more general elliptic PDEs. The eigenfunction problem is then approximated with a finite-dimensional eigenvector problem – by discretizing the state space, or by projecting on a set of adaptive bases evaluated at a set of collocation states. Solving the resulting eigenvector problem is faster than applying policy or value iteration. The bases are adapted via Levenberg-Marquardt minimization with guaranteed convergence. The collocation set can also be adapted so as to focus the approximation on a region of interest. Numerical results on test problems are provided.**

I. Introduction

Optimal control in its generic formulation is a hard problem. This motivates a search for more structured formulations making the problem easier. We and others [4], [5], [1], [3] have identified a class of nonlinear stochastic optimal control problems which are inherently linear, in the sense that the exponentiated optimal cost-to-go function can be found as the principal eigenfunction of a certain linear operator. The optimal control law can then be computed analytically given the optimal cost-to-go. Despite its inherent linearity this problem class is surprisingly general; the only unusual restriction is that the noise and the controls must act in the same subspace. Computing eigenfunctions in high-dimensional spaces is still hard, yet it is more tractable than approximating non-linear HJB equations or using other ADP/RL methods aimed at generic problem formulations.

The goal of this paper is to develop function approximation methods that have the potential to handle complex control problems within our class. Section II summarizes the necessary mathematical background. Section III describes a grid-based approximation scheme which we use to solve low-dimensional test problems with high accuracy. Section IV adapts the Gaussian RBF approximation framework to our problem class and shows how relevant quantities can be computed analytically. Section V develops fitting methods that exploit the ideas of collocation, eigensolvers, quadratic programming and Levenberg-Marquardt optimization. Section VI presents numerical results.

Emanuel Todorov is with the Department of Cognitive Science, University of California San Diego, todorov@cogsci.ucsd.edu

This work was supported by the US National Science Foundation. Thanks to Yuval Tassa for comments on the manuscript.

II. Linearly-solvable optimal control problems

The control problems we aim to solve are continuous in space and time. Yet the best approximations are obtained by discretizing the time axis and solving the corresponding continuous-space discrete-time problem. Therefore we need both continuous and discrete-time problem formulations.

A. Continuous-time formulation

Consider a system with state $\mathbf{x} \in \mathbb{R}^n$, control $\mathbf{u} \in \mathbb{R}^m$ and continuous-time stochastic dynamics

$$d\mathbf{x} = \mathbf{a}(\mathbf{x})\, dt + B(\mathbf{x})\,(\mathbf{u}dt + \sigma d\boldsymbol{\omega}) \tag{1}$$

$\boldsymbol{\omega}(t)$ is Brownian motion. The cost rate function[1] is

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

The functions $q(\mathbf{x}) \geq 0$, $\mathbf{a}(\mathbf{x})$ and $B(\mathbf{x})$ can be arbitrary. Let $v(\mathbf{x})$ denote the optimal cost-to-go and $v_{\mathbf{x}}(\mathbf{x})$ its gradient. It can be shown that the optimal control law is

$$\mathbf{u}^*(\mathbf{x}) = -\sigma^2 B(\mathbf{x})^{\mathsf{T}} v_{\mathbf{x}}(\mathbf{x}) \tag{3}$$

v satisfies the minimized HJB equation

$$c = q(\mathbf{x}) + \mathcal{L}[v](\mathbf{x}) - \frac{1}{2} v_{\mathbf{x}}(\mathbf{x})^{\mathsf{T}} \Sigma(\mathbf{x})\, v_{\mathbf{x}}(\mathbf{x}) \tag{4}$$

where the linear differential operator $\mathcal{L}$ is defined as

$$\mathcal{L}[v](\mathbf{x}) = \mathbf{a}(\mathbf{x})^{\mathsf{T}} v_{\mathbf{x}}(\mathbf{x}) + \frac{1}{2}\operatorname{trace}\left(\Sigma(\mathbf{x})\, v_{\mathbf{xx}}(\mathbf{x})\right) \tag{5}$$

and Σ is the noise covariance matrix:

$$\Sigma(\mathbf{x}) = \sigma^2 B(\mathbf{x})\, B(\mathbf{x})^{\mathsf{T}} \tag{6}$$

In an infinite-horizon average-cost-per-step setting, $c > 0$ is the unknown average cost-per-step and v is the differential cost-to-go[2]. In a first-exit setting, $c = 0$ and v is the actual cost-to-go. Our methods will apply to both settings.

Although (4) is nonlinear in v, it becomes linear when written in terms of the following function:

$$z(\mathbf{x}) = \exp(-v(\mathbf{x})) \tag{7}$$

[1]The noise amplitude σ and control weight $1/2\sigma^2$ must be related in this way to make the math work. If we wish to find the optimal control law for a different control weight, say r, we simply replace $q(\mathbf{x})$ with $\widetilde{q}(\mathbf{x}) = q(\mathbf{x})/2r\sigma^2$. The cost rate (and therefore the optimal cost-to-go) are now scaled by $1/2r\sigma^2$ relative to what we want, however scaling by a constant does not affect the optimal control law.

[2]Suppose t starts negative and ends at 0 and $\widehat{v}(\mathbf{x}, t)$ is the optimal cost-to-go for the corresponding finite horizon problem. Then the average cost-per-step is $c = -\lim_{t \to -\infty} \widehat{v}(\mathbf{x}, t)/t$, and $\widehat{v}(\mathbf{x}, t) \approx v(\mathbf{x}) - ct$.

Exponentiating (4) and expressing it in terms of z yields

$$(q(\mathbf{x}) - c)\, z(\mathbf{x}) = \mathcal{L}[z](\mathbf{x}) \tag{8}$$

This linear differential equation will be the point of departure for the differential approximation methods developed later. From the Krein-Rutman theorem, (8) has a unique positive solution $z(\mathbf{x})$ corresponding to the smallest eigenvalue c. The constraint $q(\mathbf{x}) \geq 0$ was needed to apply this theorem.

B. Discrete-time formulation

The time axis is now discretized with step h. Thus $\mathbf{x}(k)$ in discrete time corresponds to $\mathbf{x}(hk)$ in continuous time. The controller specifies the transition probability $\pi(\mathbf{y}|\mathbf{x})$ directly. Thus the discrete-time stochastic dynamics are

$$\mathbf{x}(k+1) \sim \pi(\cdot|\mathbf{x}(k)) \tag{9}$$

We also define the transition probability $p(\mathbf{y}|\mathbf{x})$ corresponding to the notion of passive dynamics. The control cost is then defined as the Kullback-Leibler (KL) divergence[3] between the controlled and passive dynamics. Thus the cost rate is

$$\ell(\mathbf{x}, \pi(\cdot|\mathbf{x})) = hq(\mathbf{x}) + \mathrm{KL}(\pi(\cdot|\mathbf{x}) \,||\, p(\cdot|\mathbf{x})) \tag{10}$$

As before let v be the optimal cost-to-go and $z = \exp(-v)$. Then the optimal control law is

$$\pi^*(\mathbf{y}|\mathbf{x}) = \frac{p(\mathbf{y}|\mathbf{x})\, z(\mathbf{y})}{\mathcal{G}[z](\mathbf{x})} \tag{11}$$

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

$$\mathcal{G}[z](\mathbf{x}) = \int p(\mathbf{y}|\mathbf{x})\, z(\mathbf{y})\, d\mathbf{y} \tag{12}$$

The minimized Bellman equation (not shown) can now be exponentiated and expressed in terms of z as

$$\exp(hq(\mathbf{x}) - hc)\, z(\mathbf{x}) = \mathcal{G}[z](\mathbf{x}) \tag{13}$$

This linear integral equation will be the point of departure for the integral approximation methods developed later. We can show [4] that it has a unique positive solution $z(\mathbf{x})$ corresponding to the largest eigenvalue $\lambda = \exp(-hc)$.

C. Relation between continuous and discrete time

In order to construct a discrete-time approximation to a given continuous-time problem we have to make sure that the passive dynamics match. This is done by setting $p(\mathbf{y}|\mathbf{x})$ to the (Euler approximation of the) h-step transition probability of the diffusion (1) with $\mathbf{u} = 0$:

$$p(\mathbf{y}|\mathbf{x}) = \mathcal{N}(\mathbf{y}; \mathbf{x} + h\mathbf{a}(\mathbf{x}), h\Sigma(\mathbf{x})) \tag{14}$$

$\mathcal{N}$ denotes a Gaussian. To see why this construction leads to a valid approximation, recall the notion of a generator of a stochastic process. The generator computes the expected directional derivative of a function along trajectories $\mathbf{y}(\cdot)$ sampled from the stochastic process:

$$\lim_{h \to 0} \frac{\mathrm{E}[z(\mathbf{y}(h)) \,|\, \mathbf{y}(0) = \mathbf{x}] - z(\mathbf{x})}{h} \tag{15}$$

[3] Recall that $\mathrm{KL}(r(\cdot) \,||\, p(\cdot)) = \int r(x) \log \frac{r(x)}{p(x)} dx$.

It is known that the generator of the diffusion process (1) with $\mathbf{u} = 0$ is the operator $\mathcal{L}$ defined in (5). At the same time the expectation in (15) is the operator $\mathcal{G}$ defined in (12). Thus $\mathcal{L}$ is a functional linearization of $\mathcal{G}$:

$$\mathcal{G}[z](\mathbf{x}) = z(\mathbf{x}) + h\mathcal{L}[z](\mathbf{x}) + o(h^2) \tag{16}$$

If we now substitute (16) in (13), subtract z from both sides, divide by h and take the limit $h \to 0$, we will recover (8).

The relation between the discrete and continuous formulations is further illuminated by noting the similarity between the KL divergence control cost in (10) and the quadratic control cost in (2). Consider any control vector $\mathbf{u}$ and the resulting h-step transition probability under (1):

$$\pi^{(\mathbf{u})}(\mathbf{y}|\mathbf{x}) = \mathcal{N}(\mathbf{y}; \mathbf{x} + h\mathbf{a}(\mathbf{x}) + hB(\mathbf{x})\mathbf{u}, h\Sigma(\mathbf{x})) \tag{17}$$

The formula for KL divergence between Gaussians yields

$$\mathrm{KL}\left(\pi^{(\mathbf{u})}(\cdot|\mathbf{x}) \,||\, p(\cdot|\mathbf{x})\right) = \frac{h}{2\sigma^2} \|\mathbf{u}\|^2 \tag{18}$$

which is just the quadratic cost accumulated over time h.

III. Approximation via discrete-state MDPs

Before discussing the function approximation methods which are the focus of this paper, we outline a different method based on discrete MDPs. This is only applicable to low-dimensional problems, but when applicable it yields highly accurate results because the approximating MDP can be solved exactly. The MDP is constructed [2] by choosing a set (usually grid) of states $\{\mathbf{x}_n\}$ and adjusting the probabilities P_{nk} of transitioning from $\mathbf{x}_n$ to $\mathbf{x}_k$ so that the mean and covariance of $p(\mathbf{y}|\mathbf{x}_n)$ are matched:

$$\begin{gathered} \overline{\mathbf{y}}_n = \textstyle\sum_k P_{nk}\mathbf{x}_k \approx \mathbf{x}_n + h\mathbf{a}(\mathbf{x}_n) \\ \textstyle\sum_k P_{nk}(\mathbf{x}_k - \overline{\mathbf{y}}_n)(\mathbf{x}_k - \overline{\mathbf{y}}_n)^{\mathsf{T}} \approx h\Sigma(\mathbf{x}_n) \end{gathered} \tag{19}$$

Such matching is not exact for a finite grid, but if the mismatch decreases sufficiently rapidly as the grid becomes more dense, the MDP solution converges to the solution of the continuous-state problem [2]. Define the vector $\mathbf{z}$ with elements $z(\mathbf{x}_n)$, and the diagonal matrix Q with elements $\exp(-hq(\mathbf{x}_n))$ on its main diagonal. Then (13) becomes

$$\lambda \mathbf{z} = QP\mathbf{z} \tag{20}$$

Once we find the principal eigenvector λ, the average cost-per-step can be computed as $c = -\log(\lambda)/h$. For small h the variance of $p(\mathbf{y}|\mathbf{x})$ is small and so QP is sparse, allowing us to take advantage of sparse eigensolvers.

One way to solve (20) is the power iteration method – which in the present context is equivalent to value iteration in exponentiated form. Despite this equivalence, however, solving (20) can be orders of magnitude faster than value iteration for generic MDPs, because: (i) the optimal controls are found analytically; (ii) value iteration can converge slowly while here convergence is linear; (iii) eigensolvers use methods such as Rayleigh quotient iteration that are faster than power iteration.

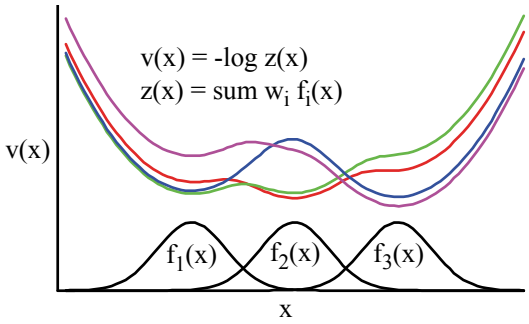

Fig. 1. Illustration of the function approximator. Colored curves correspond to different random sets of w's.

IV. Gaussian RBFs and collocation

We now turn to methods that use smooth approximation instead of discretization. Here we describe the function approximator and its properties in the context of our problem class, and leave the fitting procedures for the next section.

Define a basis set $\{f_i\}_{i=1\cdots K}$ of unnormalized Gaussians:

$$f_i(\mathbf{x}) = \exp\left(-\tfrac{1}{2}(\mathbf{x}-\mathbf{m}_i)^{\mathsf{T}} S_i (\mathbf{x}-\mathbf{m}_i)\right) \tag{21}$$

z is approximated as a linear combination of these bases:

$$\widehat{z}(\mathbf{x};\boldsymbol{\theta},\mathbf{w}) = \textstyle\sum_i w_i f_i(\mathbf{x}) \tag{22}$$

$\boldsymbol{\theta}$ contains the means $\mathbf{m}_i$ and inverse covariances S_i. Note that (22) is linear in $\mathbf{w}$ and nonlinear in $\boldsymbol{\theta}$. We will consider linear methods that fix $\boldsymbol{\theta}$ and solve for $\mathbf{w}$, as well as nonlinear methods that adapt $\boldsymbol{\theta}$.

There are several reasons for choosing to work with Gaussian RBFs: (i) they have been widely used with considerable success; (ii) they allow analytical computation of the operator $\mathcal{G}$ as well as the control laws π^* and $\mathbf{u}^*$; (iii) we have shown that the function z behaves like a probability density [5] thus it makes sense to approximate it with a density model; (iv) this function approximator has appealing default behavior as illustrated in **Fig. 1**. Varying w_i affects the shape of the function in the region where the bases are clustered. Away from the bases $\widehat{v} = -\log(\widehat{z})$ gradually becomes quadratic, giving rise to linear feedback control laws that push the state back towards the bases.

A. Computing the integral operator $\mathcal{G}$

Since $\mathcal{G}$ is linear and $\widehat{z}$ is as a linear mixture, we have

$$\mathcal{G}[\widehat{z}](\mathbf{x}) = \textstyle\sum_i w_i \mathcal{G}[f_i](\mathbf{x}) \tag{23}$$

Both p and f_i are Gaussians so the integral in (12) can be solved analytically. One complication however is that the covariance matrix $h\Sigma(\mathbf{x})$ of $p(\mathbf{y}|\mathbf{x})$ may be singular. This occurs in 2nd-order systems, as follows. Consider a state vector $\mathbf{x}$ which includes the positions $\mathbf{x}_p$ and velocities $\mathbf{x}_v$ of all degrees of freedom. The (deterministic) dynamics are

$$\mathbf{u} = M(\mathbf{x}_p)\dot{\mathbf{x}}_v + \boldsymbol{\rho}(\mathbf{x}_p, \mathbf{x}_v) \tag{24}$$

where $\mathbf{u}$ is the vector of applied torques, M the inertia matrix and $\boldsymbol{\rho}$ the vector of Coriolis, centripetal and gravitational forces. Writing (24) in general 1st-order form yields $B(\mathbf{x}) = \left[0; M(\mathbf{x}_p)^{-1}\right]$ and thus $\operatorname{rank}(\Sigma) = n/2$.

To handle this complication we first compute a symmetric matrix square root

$$h\Sigma(\mathbf{x}) = C(\mathbf{x})\, C(\mathbf{x})^{\mathsf{T}} \tag{25}$$

where C is an n-by-$\operatorname{rank}(\Sigma)$ matrix. The obvious choice is $C(\mathbf{x}) = \sigma h^{1/2} B(\mathbf{x})$. Let $\overline{\mathbf{y}}$ be the mean of p:

$$\overline{\mathbf{y}}(\mathbf{x}) = \mathbf{x} + h\mathbf{a}(\mathbf{x}) \tag{26}$$

Then the random variable $\mathbf{y}$ can be expressed as

$$\mathbf{y} = C\boldsymbol{\varepsilon} + \overline{\mathbf{y}}, \quad \boldsymbol{\varepsilon} \sim \mathcal{N}(\cdot; 0, I) \tag{27}$$

and the integral in (12) becomes

$$\begin{aligned} \mathcal{G}[f_i](\mathbf{x}) &= \int \mathcal{N}\left(\mathbf{y}; \overline{\mathbf{y}}, CC^{\mathsf{T}}\right) f_i(\mathbf{y})\, d\mathbf{y} \\ &= \int \mathcal{N}(\boldsymbol{\varepsilon}; 0, I)\, f_i(C\boldsymbol{\varepsilon} + \overline{\mathbf{y}})\, d\boldsymbol{\varepsilon} \end{aligned} \tag{28}$$

The latter integral involves the exponent of a quadratic function of $\boldsymbol{\varepsilon}$. Completing the squares in the exponent yields

$$\begin{aligned} &\boldsymbol{\varepsilon}^{\mathsf{T}}\boldsymbol{\varepsilon} + (C\boldsymbol{\varepsilon} + \overline{\mathbf{y}} - \mathbf{m}_i)^{\mathsf{T}} S_i (C\boldsymbol{\varepsilon} + \overline{\mathbf{y}} - \mathbf{m}_i) \\ &= (\overline{\mathbf{y}} - \mathbf{m}_i)^{\mathsf{T}} H_i (\overline{\mathbf{y}} - \mathbf{m}_i) + (\boldsymbol{\varepsilon} - \mathbf{d}_i)^{\mathsf{T}} V_i (\boldsymbol{\varepsilon} - \mathbf{d}_i) \end{aligned} \tag{29}$$

where $V_i, H_i, \mathbf{d}_i$ are given by

$$\begin{aligned} V_i &= I + C^{\mathsf{T}} S_i C \\ H_i &= S_i - S_i C V_i^{-1} C^{\mathsf{T}} S_i \\ \mathbf{d}_i &= V_i^{-1} C^{\mathsf{T}} S_i (\mathbf{m}_i - \overline{\mathbf{y}}) \end{aligned} \tag{30}$$

Putting the results together we have

$$\mathcal{G}[f_i](\mathbf{x}) = |V_i|^{-\frac{1}{2}} \exp\left(-\tfrac{1}{2}(\overline{\mathbf{y}} - \mathbf{m}_i)^{\mathsf{T}} H_i (\overline{\mathbf{y}} - \mathbf{m}_i)\right) \tag{31}$$

B. Computing the optimal control law $\mathbf{u}^*$

Suppose we started with a continuous-time problem, constructed its discrete-time counterpart and approximated $z(x)$ and $\pi^*(\mathbf{y}|\mathbf{x})$. How can we now recover $\mathbf{u}^*(\mathbf{x})$? The obvious approach is to compute $v = -\log(z)$ and apply (3). From (7) it follows that $v_{\mathbf{x}}(\mathbf{x}) = -z_{\mathbf{x}}(\mathbf{x})/z(\mathbf{x})$. Thus

$$\mathbf{u}^*(\mathbf{x}) = \sigma^2 B(\mathbf{x})^{\mathsf{T}} \textstyle\sum_i n_i(\mathbf{x})\, S_i(\mathbf{m}_i - \mathbf{x}) \tag{32}$$

where the weights n_i are given by

$$n_i(\mathbf{x}) = \frac{w_i f_i(\mathbf{x})}{\sum_k w_k f_k(\mathbf{x})} \tag{33}$$

This resembles the way functions are represented in locally-weighted regression.

The drawback of (32) is that it involves differentiation of the basis functions, which can have undesirable effects even though the differentiation is carried out analytically. In particular the resulting control laws often have high-frequency components. Fortunately there is an alternative –

which is to match the mean of π^* to the drift caused by $\mathbf{u}^*$, and solve (in a least squares sense) the equation

$$\mathbf{x} + h\mathbf{a}(\mathbf{x}) + hB(\mathbf{x})\mathbf{u}^*(\mathbf{x}) = \int \mathbf{y}\pi^*(\mathbf{y}|\mathbf{x})\,d\mathbf{y} \tag{34}$$

From (29) we see that $p(\mathbf{y}|\mathbf{x})\,f_i(\mathbf{y})$ is a Gaussian which is scaled by $\mathcal{G}[f_i](\mathbf{x})$ and has mean $C\mathbf{d}_i + \overline{\mathbf{y}}$. Thus the passive dynamics ($\overline{\mathbf{y}}$ and $\mathbf{x} + h\mathbf{a}$) cancel and (34) becomes

$$hB(\mathbf{x})\mathbf{u}^*(\mathbf{x}) = C(\mathbf{x})\textstyle\sum_i \eta_i(\mathbf{x})\mathbf{d}_i(\mathbf{x}) \tag{35}$$

where the weights η_i are given by

$$\eta_i(\mathbf{x}) = \frac{w_i\mathcal{G}[f_i](\mathbf{x})}{\sum_k w_k\mathcal{G}[f_k](\mathbf{x})} \tag{36}$$

When we use $C = \sigma h^{1/2}B$ equation (35) becomes

$$\mathbf{u}^*(\mathbf{x}) = \sigma h^{-1/2}\textstyle\sum_i \eta_i(\mathbf{x})\mathbf{d}_i(\mathbf{x}) \tag{37}$$

C. Collocation

The solutions to (8) and (13) will be approximated via collocation: choose a set of collocation states $\{\mathbf{x}_n\}_{n=1\cdots N}$ and enforce equality at those states. In the over-determined case ($N > K$) this will be done using least squares. Define the matrices $F(\boldsymbol{\theta})$, $G(\boldsymbol{\theta})$ and $L(\boldsymbol{\theta})$ with elements

$$\begin{aligned} F_{ni} &= f_i(\mathbf{x}_n) \\ G_{ni} &= \exp(-hq(\mathbf{x}_n))\,\mathcal{G}[f_i](\mathbf{x}_n) \\ L_{ni} &= q(\mathbf{x}_n)f_i(\mathbf{x}_n) - \mathcal{L}[f_i](\mathbf{x}_n) \end{aligned} \tag{38}$$

Equations (8, 13) restricted to $\{\mathbf{x}_n\}$ then become

$$cF(\boldsymbol{\theta})\mathbf{w} = L(\boldsymbol{\theta})\mathbf{w} \tag{39a}$$

$$\lambda F(\boldsymbol{\theta})\mathbf{w} = G(\boldsymbol{\theta})\mathbf{w} \tag{39b}$$

Recall that we seek the smallest eigenvalue c in (39a) and the largest eigenvalue λ in (39b).

V. Optimization methods

A. Fixed bases: eigensolvers and quadratic programming

Here we assume fixed basis functions and compute $\lambda, \mathbf{w}$. For concreteness we focus on (39b) although the same methods apply to (39a). The obvious approach is to use an eigensolver. When $N = K$ the matrices $F(\boldsymbol{\theta}), G(\boldsymbol{\theta})$ are square and the generalized eigenvalue problem (39b) can be reduced to the standard form[4]

$$\lambda\mathbf{w} = F(\boldsymbol{\theta})^{-1}G(\boldsymbol{\theta})\mathbf{w} \tag{40}$$

One problem with (40) is that $F(\boldsymbol{\theta})$ may be ill-conditioned – which occurs when the bases have large overlap. In that case we can use

$$\lambda\mathbf{w} = \left(F(\boldsymbol{\theta})^{\mathsf{T}}F(\boldsymbol{\theta}) + \alpha I\right)^{-1}F(\boldsymbol{\theta})^{\mathsf{T}}G(\boldsymbol{\theta})\mathbf{w} \tag{41}$$

where α is a regularization parameter. Note the relation to Gaussian process regression. Apart from dealing with ill-conditioning, (41) can handle the case $N > K$.

[4]Of course one should use matrix factorization and backsubstitution instead of matrix inversion.

While the eigensolver approach is computationally the most efficient, it has two drawbacks. First, if the elements of $\mathbf{w}$ are allowed to be negative the approximation $\widehat{z}$ may be negative for some states – which is a problem because the machinery we developed assumes positive z. Second, in the over-determined case the eigensolver does not minimize the residual but instead its projection on $F(\boldsymbol{\theta})$. This becomes problematic when we make $\boldsymbol{\theta}$ adjustable later. Both issues can be addressed using quadratic programming (QP) as follows. Define the vector of residuals

$$\mathbf{r}(\lambda, \boldsymbol{\theta}, \mathbf{w}) = \lambda F(\boldsymbol{\theta})\mathbf{w} - G(\boldsymbol{\theta})\mathbf{w} \tag{42}$$

and the objective function

$$e(\lambda, \boldsymbol{\theta}, \mathbf{w}) = \frac{1}{2}\left\|\mathbf{r}(\lambda, \boldsymbol{\theta}, \mathbf{w})\right\|^2 \tag{43}$$

Since e is quadratic in $\mathbf{w}$, we can now apply QP while imposing the constraints $w_i \geq 0$.

Eigenvectors are defined up to a scaling factor, thus every eigensolver needs to impose a norm constraint to avoid the trivial 0 solution. One way to do this in QP is to impose the constraint $\mathbf{1}^{\mathsf{T}}\mathbf{w} = const$. This is sufficient for fixed bases but is not sufficient for adaptive bases. If the bases move away from the collocation states we can have $\widehat{z}(\mathbf{x}_n) \approx 0$ even when $w_i \gg 0$. Indeed the adaptive algorithm below exploits this loophole. Thus a more sensible constraint is

$$\mathbf{1}^{\mathsf{T}}F(\boldsymbol{\theta})\mathbf{w} = \textstyle\sum_n \widehat{z}(\mathbf{x}_n; \boldsymbol{\theta}, \mathbf{w}) = const \tag{44}$$

Next we need to compute λ. The minimum of e over λ is

$$\lambda = \frac{\mathbf{w}^{\mathsf{T}}F(\boldsymbol{\theta})^{\mathsf{T}}F(\boldsymbol{\theta})\mathbf{w}}{\mathbf{w}^{\mathsf{T}}F(\boldsymbol{\theta})^{\mathsf{T}}G(\boldsymbol{\theta})\mathbf{w}} \tag{45}$$

We now iterate between (45) and QP. Convergence is guaranteed because each step reduces e. The solution to (40) or (41) can be used for initialization. In practice convergence only takes a few iterations.

B. Adaptive bases: constrained Levenberg-Marquardt

Next we adapt the means and covariances of the Gaussian basis functions[5]. This is done by minimizing e over $\boldsymbol{\theta}$. Define the Jacobian of $\mathbf{r}$ w.r.t. $\boldsymbol{\theta}$:

$$J(\lambda, \boldsymbol{\theta}, \mathbf{w}) = \frac{\partial\mathbf{r}(\lambda, \boldsymbol{\theta}, \mathbf{w})}{\partial\boldsymbol{\theta}} \tag{46}$$

The gradient is $e_{\boldsymbol{\theta}} = J^{\mathsf{T}}\mathbf{r}$ which can be used in a gradient descent algorithm. However, since e is in least-squares form, a better approach is to approximate the Hessian as $J^{\mathsf{T}}J$ and apply the Gauss-Newton algorithm with the Levenberg-Marquardt refinement:

$$\boldsymbol{\theta}' = \boldsymbol{\theta} - \left(J^{\mathsf{T}}J + \gamma I\right)^{-1}J^{\mathsf{T}}\mathbf{r} \tag{47}$$

For $\gamma = 0$ this is exactly Gauss-Newton, while for large γ it approximates gradient descent with step size $1/\gamma$. If the candidate $\boldsymbol{\theta}'$ reduces e then we set $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta}'$ and reduce γ, otherwise we increase γ and compute a new candidate.

[5]It now becomes essential to have $N \gg K$ or else we can overfit badly.

One caveat here is that we want to optimize e subject to the constraint (44). The vector

$$\mathbf{g} = \frac{\partial \left(\mathbf{1}^{\mathsf{T}} F(\boldsymbol{\theta}) \mathbf{w} \right)}{\partial \boldsymbol{\theta}} \tag{48}$$

gives the direction in which $\boldsymbol{\theta}$ should not change. Thus we need to modify (47) so that $\mathbf{g}^{\mathsf{T}} \boldsymbol{\delta} = 0$ where $\boldsymbol{\delta} = \boldsymbol{\theta}' - \boldsymbol{\theta}$. This leads to the following constrained minimization problem:

$$\min_{\boldsymbol{\delta}: \mathbf{g}^{\mathsf{T}} \boldsymbol{\delta} = 0} \boldsymbol{\delta}^{\mathsf{T}} J^{\mathsf{T}} \mathbf{r} + \frac{1}{2} \boldsymbol{\delta}^{\mathsf{T}} \left(J^{\mathsf{T}} J + \gamma I \right) \boldsymbol{\delta} \tag{49}$$

Using Lagrange multipliers, $\boldsymbol{\delta}$ is found by solving

$$\left[\begin{array}{cc} J^{\mathsf{T}} J + \gamma I & \mathbf{g} \\ \mathbf{g}^{\mathsf{T}} & 0 \end{array} \right] \left[\begin{array}{c} \boldsymbol{\delta} \\ \nu \end{array} \right] = \left[\begin{array}{c} -J^{\mathsf{T}} \mathbf{r} \\ 0 \end{array} \right] \tag{50}$$

One could write down an explicit formula, however computing $\boldsymbol{\delta}$ by solving the above linear equation is more accurate and efficient. Once $\boldsymbol{\delta}$ is available, the candidate is $\boldsymbol{\theta}' = \boldsymbol{\theta} + \boldsymbol{\delta}$. We now alternate between improving $\boldsymbol{\theta}$ and solving for $\lambda, \mathbf{w}$. Since each update reduces e, the iteration converges.

C. Approximate value iteration

Instead of adjusting the parameters on both sides of (39b) simultaneously, i.e. treating the problem as an eigenvalue problem, we could use a method that resembles value iteration. Each iteration involves two steps: (i) compute the vector $\widetilde{\mathbf{z}} = G(\boldsymbol{\theta}) \mathbf{w}$ for the current parameters; (ii) redefine

$$\mathbf{r}(\lambda, \boldsymbol{\theta}, \mathbf{w}) = \lambda F(\boldsymbol{\theta}) \mathbf{w} - \widetilde{\mathbf{z}} \tag{51}$$

and update the parameters so as to reduce e for this new residual. The update for $\boldsymbol{\theta}$ has the same form as above. The update for $\lambda, \mathbf{w}$ is simplified because we are no longer solving an eigenvalue problem. The value of λ is computed from the constraint (44).

Since we are redefining e at each iteration it is not obvious that the algorithm will converge. Convergence is guaranteed if we keep $\boldsymbol{\theta}$ fixed – indeed in that case the algorithm reduces to power iteration. But if $\boldsymbol{\theta}$ is adapted online, convergence cannot be guaranteed and needs to be assessed empirically.

VI. Numerical results

A. Test problems

We focus on two test problems (**Fig. 2a**) with 2D state space and 1D control space. The dynamical systems are point masses constrained to move along a circle (corresponding to a pendulum) and an inverted Gaussian (corresponding to a car-on-a-hill). Thus $\mathbf{x} = [x_p; x_v]$ and $B = [0; 1]$. There is also gravity. The pendulum has passive dynamics

$$\mathbf{a}(\mathbf{x}) = \left[\begin{array}{c} x_v \\ \sin(x_p) \end{array} \right] \tag{52}$$

with $\sigma = 2$. The car-on-a-hill has passive dynamics

$$\mathbf{a}(\mathbf{x}) = \left[\begin{array}{c} x_v \left(1 + s(x_p)^2 \right)^{-1/2} \\ -9.8 \operatorname{sign}(x_p) \left(1 + s(x_p)^{-2} \right)^{-1/2} \end{array} \right] \tag{53}$$

with $\sigma = 1$. Here $s(x_p) = x_p \exp\left(-x_p^2/2\right)$ is the slope of the inverted Gaussian hill at position x_p. The latter dynamics have unusual form because x_p denotes horizontal position while x_v denotes tangential velocity. The pendulum coordinates are angular position and velocity as usual.

One interesting aspect of our simulations is that we model inelastic collisions at the positional limits $x_{\min}, x_{\max}$. The dynamics are augmented with the following rule:

$$\begin{array}{c} \text{if } x_p < x_{\min} \\ \text{then } x_p = x_{\min},\ x_v = 0 \end{array} \tag{54}$$

and similarly for $x_{\max}$. This causes a discontinuity in continuous time and cannot be captured by the diffusion model (1). Indeed such systems are known as hybrid. In discrete time however collisions are easily handled. All we have to do is redefine the transition probabilities $p(\mathbf{y}|\mathbf{x})$ accordingly. In this simple example, collisions cause p to become a delta function centered at $\left[x_{\min / \max}; 0 \right]$. For more general 3D collisions p may become a mixture of Gaussians and delta functions. Note that the analytical methods in the previous section are trivially extended to such mixtures. The reason for emphasizing collisions is that we want to eventually apply our methods to locomotion and hand manipulation where contact phenomena are essential.

The cost functions $q(\mathbf{x})$ are chosen so that the optimal behavior is a limit cycle. The task for the pendulum is to move at constant velocity in either direction. The task for the car-on-a-hill is to be at one of two states – which have non-zero desired velocities thus being stationary at those states is not an option. **Fig. 2a** shows stochastic trajectories sampled under the optimal controls computed via discretization.

B. MDP solutions

The state space was discretized with a 200-by-200 grid. The passive transition probabilities P_{nk} were computed as in (19). For each state (away from the boundary) there were 38 next states with non-zero transition probability. Thus the 40000-by-40000 matrix P had about 1.4 million non-zero elements. Finding the principal eigenvector of QP with the "eigs" function in Matlab took 2 and 8 seconds respectively on the two problems. The resulting approximations to $z(\mathbf{x})$ are shown in **Fig. 2b** top. On the bottom we show the corresponding $u^*(\mathbf{x})$ computed with the mean-matching method. The solid curve is the noise-free limit cycle resulting from each control law. The values of c show the average cost-per-step computed via extensive sampling: 5000 stochastic trajectories with random starting states, 5500 time steps per trajectory, simulation time step $h = 0.01$, the first 500 time steps removed to avoid transients. The results from function approximation will be shown in the same format.

C. Differential and integral methods, fixed bases

First we compared the differential and integral methods using 20-by-20 regular grids of collocation states. The 400 fixed bases were centered at the collocation states. The standard deviations of the bases were adjusted to 0.7 of the center-to-center distance in each direction. This value was found to be near-optimal via manual experimentation. The principal eigenpair of (40) – largest or smallest depending

on the method – was used to initialize $\mathbf{w}$ and λ, which were then improved iteratively using quadratic programming while imposing $w_i \geq 0$ and (44). This converged in less than 5 iterations. The results are shown in **Fig. 2c,d**. The integral method always works well. On problem 2 we see some differences from the MDP solution, but these differences are at states far from the limit cycle and have the effect of stabilizing the system more than is necessary. The differential method gives plausible results on problem 2 even though the limit cycle has somewhat different shape and the solution has some higher-frequency components. The latter are hard to eliminate because increasing the width of the bases quickly leads to singularity of F. On problem 1 however the differential method fails. This is presumably because the collision dynamics are essential – the optimal controller is counting on the collision to break the movement before accelerating in the opposite direction. Thus we focus on the integral method in the rest of the paper.

D. Eigen-iteration and value iteration, adaptive bases

Recall that the integral method can be applied in eigen-iteration or value-iteration mode. Here we test both, using 40 adaptive bases with random initial positions and the same sizes as before. The collocation states are still on a 20-by-20 grid. Results are shown in **Fig. 3a,c**. The eigen-iteration still works well even though the number of bases has been reduced by a factor of 10. The dots in **Fig. 3a** show the final positions of the bases. They appear random but apparently are far from random. Indeed, truly random positions (as we had at initialization) produce results that are unrelated to the correct answer. We noticed that small changes in the position and shape of the bases could lead to large changes in the solution. **Fig. 3c** illustrates the convergence of the two iteration methods. As expected from the theory, eigen-iteration monotonically reduces the residual. This is not the case for value iteration (and is not expected to be) although there is a clear trend. The control laws found by eigen-iteration achieved lower cost. One advantage of value-iteration however was that it could very quickly find good solutions – only to wander away from them. Thus one could use value-iteration to initialize eigen-iteration. Note that both methods are trying to minimize Bellman error (in exponentiated form) and there is no guarantee that the resulting control law will actually improve at each iteration. We observed (somewhat anecdotally) that the control law almost always improved, but there were exceptions – enough to discourage attempts to prove theorems to the contrary.

E. Localized collocation set

Thus far we used a collocation set that covers the entire state space. In high-dimensional problems this is clearly not feasible. Instead we need methods that can approximate functions over a region of state space. **Fig. 3b** shows such results for 400 collocation states sampled from the stochastic trajectories under the optimal MDP controller. The collocation states are now shown with dots. The 40 adaptive bases are no longer shown. The results (from eigen-iteration) are still very reasonable. This is only a first step towards developing an adaptive collocation-set method which approximates the control law, moves the collocation states according to this control law, constructs another approximation etc. Differential dynamic programming does something related however it is limited to a single trajectory.

F. Convergence to the wrong eigenfunction

Finally, there is potential problem with all our methods except for the simple eigensolver. Although we are minimizing a sensible residual, there is no guarantee that we will converge to the principal eigenpair. Indeed any other eigenpair will achieve zero residual. There are two causes for hope. First, we are initializing close to the principal eigenpair found by an eigensolver. Second, only the principal eigenfunction is positive and real – and our function approximator is constrained to positive real functions. But given the approximations, little can be guaranteed mathematically. **Fig. 3d** illustrates a problem where eigen-iteration indeed converges to something that resembles the 2nd eigenfunction found by MDP. This problem involves balancing an inverted pendulum at desired state $[0; 0]$. Interestingly, value-iteration converged to a more plausible solution. We studied a similar task with the car-on-a-hill and observed similar results. Thus, it seems that when the optimal behavior involves a point attractor, the principal eigenvalue is not separated by a large enough gap and our eigen-iteration method can find the wrong answer. In such problems value-iteration may be preferred; it is actually more stable here than in limit-cycle problems. A way to fix the eigen-iteration method is to apply it once, and then apply it again with the constraint that the second solution should be orthogonal to the first solution. This is a linear constraint and is easily imposed.

VII. Summary

Here we developed a range of function approximation methods for solving stochastic optimal control problems that are reducible to eigenfunction problems. We focused on simple 2D examples where the correct solution could be computed accurately via discretization. Now that we have a reasonable understanding of what these methods are doing, we can venture into more complex higher-dimensional problems. Apart from that, future work includes development of the adaptive collocation-set method and dealing with the issue of occasional convergence to the wrong eigenfunction. On the theory front we would like to understand why the eigenspectrum seems to have different properties in stationary vs. limit-cycle tasks.

References

[1] H. Kappen. Linear theory for control of nonlinear stochastic systems. *Physical Review Letters*, 95, 2005.

[2] H.J. Kushner and P. Dupuis. *Numerical Methods for Stochastic Optimal Control Problems in Continuous Time*. Springer, New York, 2001.

[3] S. Mitter and N. Newton. A variational approach to nonlinear estimation. *SIAM J Control Opt*, 42:1813–1833, 2003.

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

[5] E. Todorov. General duality between optimal control and estimation. *IEEE Conference on Decision and Control*, 2008.

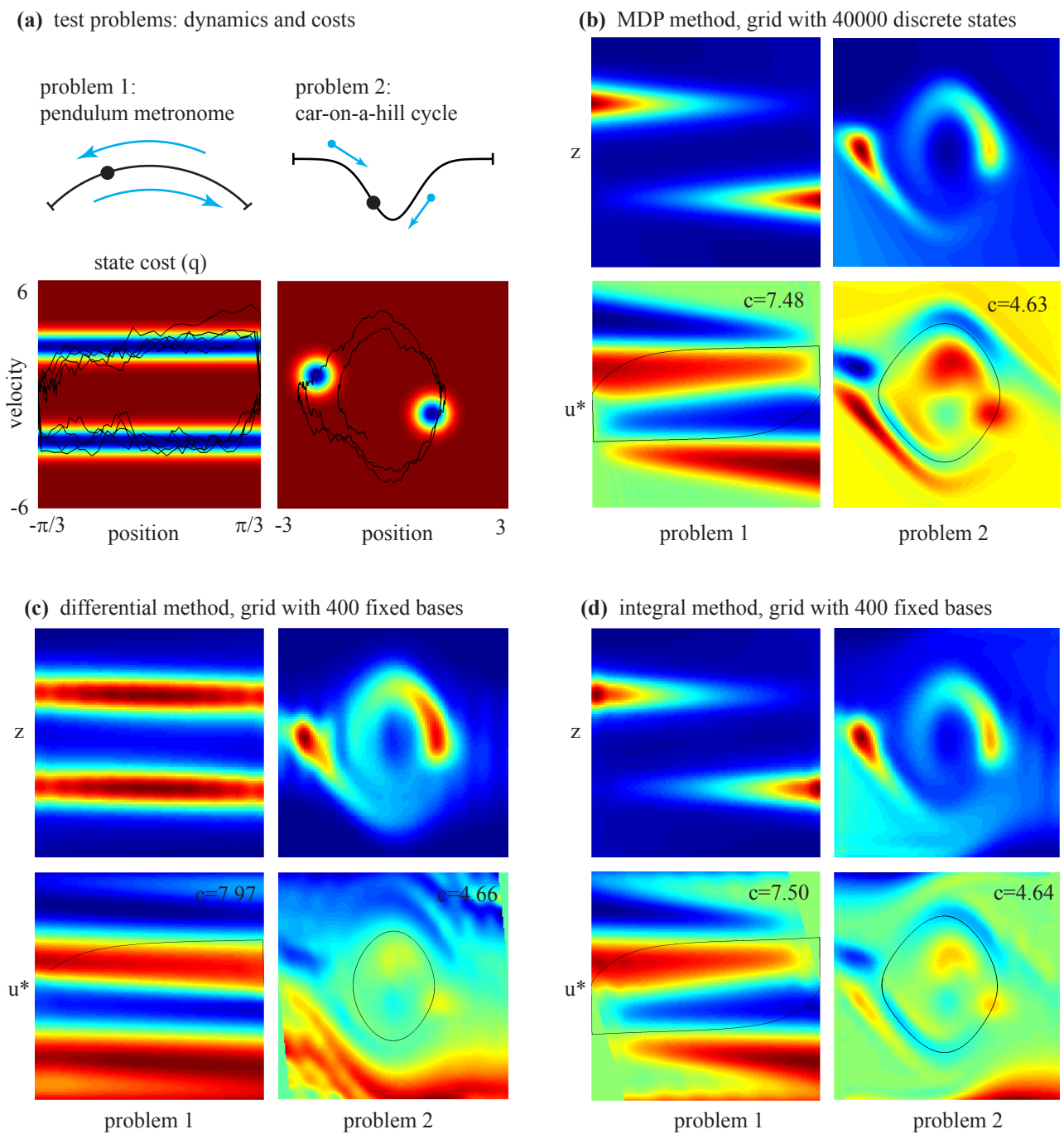

Fig. 2. **(a)** - Illustration of the two dynamical systems and cost functions. In all plots blue correspond to smaller (possibly negative) values and red to larger values. The traces are stochastic trajectories sampled under the optimal MDP controller. **(b)** - Results from the MDP method. z(x) is on the top, u*(x) on the bottom, problem 1 on the left, problem 2 on the right. The value of c shows the average cost per step computed via extensive sampling. The traces are the limit cycles resulting from each control law in the absence of noise. **(c)** - Results for the differential method, with 20-by-20 grid of collocation states and fixed basis functions. The limit cycle for problem 1 is incomplete because in the absence of noise this system gets stuck in the collision states; noise can make it continue along the cycle. **(d)** - Results for the integral method, again with 20-by-20 grid of collocation states and fixed bases.

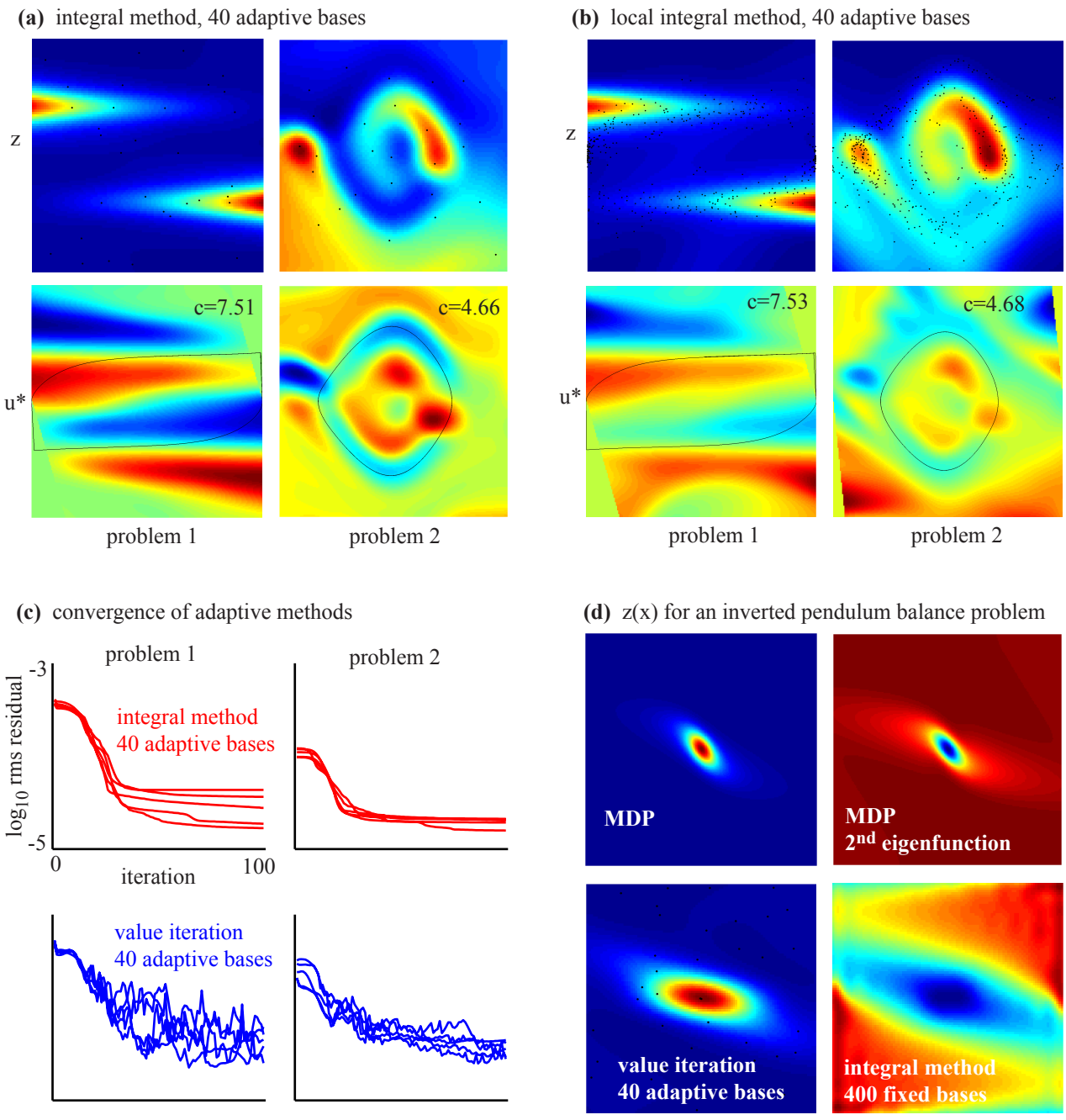

Fig. 3. **(a)** - Results for the integral method with eigen-iteration, 40 adaptive bases (shown with dots), 20-by-20 grid of collocation states. The method converges in about 100 iterations and takes about 10 seconds of CPU time. **(b)** - Results for the integral method with eigen-iteration, 40 adaptive bases (not shown), this time with 400 collocation states (dots) sampled from stochastic trajectories under the optimal MDP controller. The bases are initialized randomly but most of them move to the region covered by collocation states. **(c)** - Log residual over iterations, for 5 runs of eigen-iteration and value-iteration on the two problems. **(d)** - Results for a different problem which uses the pendulum dynamics but the desired state is [0;0]. Only z(x) is shown. Value iteration converges to a plausible solution, while eigen-iteration converges to something closer to the 2nd eigenfunction found by the MDP method.