2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
Daejeon Convention Center
October 9-14, 2016, Daejeon, Korea

Sequential Alternating Least Squares for Solving High Dimensional Linear Hamilton-Jacobi-Bellman Equation

Elis Stefansson[1] and Yoke Peng Leong[2]

Abstract— **This paper presents a technique to efficiently solve the Hamilton-Jacobi-Bellman (HJB) equation for a class of stochastic affine nonlinear dynamical systems in high dimensions. The HJB solution provides a globally optimal controller to the associated dynamical system. However, the curse of dimensionality, commonly found in robotic systems, prevents one from solving the HJB equation naively. This work avoids the curse by representing the linear HJB equation using tensor decomposition. An alternating least squares (ALS) based technique finds an approximate solution to the linear HJB equation. A straightforward implementation of the ALS algorithm results in ill-conditioned matrices that prevent approximation to a high order of accuracy. This work resolves the ill-conditioning issue by computing the solution sequentially and introducing boundary condition rescaling. Both of these additions reduce the condition number of matrices in the ALS-based algorithm. A MATLAB tool, Sequential Alternating Least Squares (SeALS), that implements the new method is developed. The performance of SeALS is illustrated using three engineering examples: an inverted pendulum, a Vertical Takeoff and Landing aircraft, and a quadcopter with state up to twelve.**

I. Introduction

The Hamilton-Jacobi-Bellman (HJB) equation is a nonlinear partial differential equation (PDE) whose solution provides a globally optimal controller for the associated stochastic dynamical system [1]. Solving the HJB equation is nontrivial because the equation is a second order nonlinear PDE. Nonetheless, for stochastic affine nonlinear systems, by placing a minor restriction on the cost function and applying a log transformation, the associated nonlinear HJB equation reduces to a linear PDE [2], [3]. This reduction enables the use of off-the-shelf numerical PDE solvers to compute a solution of the linear HJB equation. However, the curse of dimensionality quickly causes the problem to become intractable for systems with modest dimensions [4] because the number of degrees of freedom required to solve the optimal control problem grows exponentially with dimension. Yet, robots or other engineering systems commonly have at least six degrees of freedom.

Previous research has considered multiple approaches to alleviate this curse of dimensionality. These techniques include using sparse grid [5], Taylor polynomial approximation [6], max-plus method [7], and model reduction [8]. In contrast to the previous work, this paper focuses on using a *tensor decomposition* technique to represent and solve the linear HJB equation. The complexity of this approach grows linearly with the number of dimensions [9]. Others [10] have considered using a tensor decomposition based technique. However, that technique solves for an optimal controller by using value iteration, instead of the HJB equation.

This work develops a numerical technique based on tensor decomposition and the Alternating Least Squares (ALS) to solve the linear HJB equation associated with the first exit problem of a large class of affine nonlinear systems. A tensor decomposition represents the equation efficiently in order to avoid the curse of dimensionality. A new MATLAB tool, Sequential ALS (SeALS), is developed to compute the approximate solution to the linear HJB equation represented in tensor decomposition form. This work is an extension of [11], where the idea of using ALS and tensor decomposition to solve the HJB equation was first considered.

The major contributions of this paper are the improvement of the ALS algorithm from the original work [11] and the development of the SeALS tool in MATLAB. Specifically, the new algorithm mitigates the ill-conditioning issue arising in [11], which prevents the algorithm from computing the approximate solution to high accuracy. This paper introduces sequential computation of the solution and boundary condition rescaling to alleviate the orignal ALS ill-conditioning issues. As a result, SeALS can solve high dimensional linear HJB equations more accurately with shorter computation time. This tool is used to compute optimal controllers for three simulated examples – balancing an inverted pendulum, landing a Vertical Takeoff and Landing (VTOL) aircraft, and stabilizing a quadcopter. The ability to compute the solution of 12-dimensional linear HJB equation of a quadcopter on a personal laptop suggests that SeALS has a great potential for use in robotics applications.

Section II presents notation and a brief review of the linear HJB equation, tensor decomposition, and the ALS algorithm. Section III provides an illustrative example, discusses limitations of the original algorithm [11], and describes the improvements using sequential computation and boundary condition rescaling. The MATLAB tool, SeALS, is summarized in Section IV. A more detailed description of SeALS and its implementation may be found in [12].

II. Preliminaries

A. Notation

The positive integers, real numbers, non-negative real numbers, n-dimensional real vectors, and $m \times n$ real matrices are represented as $\mathbb{Z}^+$, $\mathbb{R}$, $\mathbb{R}^+$, $\mathbb{R}^n$, and $\mathbb{R}^{m\times n}$. The boundary of a compact domain $\Omega \subset \mathbb{R}^d$ is denoted $\partial\Omega$. A sequence of points from x_1 to x_k is written as $\{x_i\}_{i=1}^k$.

[1]E. Stefansson is with KTH Royal Institute of Technology, Sweden elisst@kth.se.
[2]Y. P. Leong is with the Control and Dynamical Systems, California Institute of Technology, Pasadena, CA 91125, USA ypleong@caltech.edu.

The symbols ∇_x and ∇_{xx} represents the gradient and the hessian with respect to x respectively. The matrix trace is represented as Tr, and $\|\cdot\|$ represents the Frobenius norm. The tensor product of two vectors u and v is written as $u \bigotimes v \triangleq w$ where $w_{ij} = u_i v_j$. The inner product of two vectors u and v is written as $\langle u, v\rangle$ where $\langle u, v\rangle = \sum_i u_i v_i$. A function $f(\cdot)$ is abbreviated as f when its arguments are clear from the context.

B. Linear Hamilton-Jacobi-Bellman (HJB) Equation

Consider the stochastic affine dynamical system

$$dx(t) = f(x(t))\ dt + B(x(t))(u(t)\ dt + dw(t)) \quad (1)$$

defined on a compact domain $\Omega \subset \mathbb{R}^d$, where $x(t) \in \Omega$ is the state, $u(t) \in \mathbb{R}^m$ is the control inputs, f and B are smoothly differentiable functions with respect to $x(t)$, and $w(t)$ is Gaussian noise with covariance matrix Σ_ε. Note that this work applies to a more general form of affine nonlinear systems [2], [11]. The current form is chosen for notational simplicity. This class of dynamical systems arises in many robotic systems including quadcopter and other examples shown in the Section V.

The objective is to compute a controller $u(t)$ that minimizes the following cost functional

$$J(x,u) = \mathbb{E}_{\omega(t)}\left[\phi(x(T)) + \int_0^T q(x(t)) + \frac{1}{2}u(t)^T R u(t)\ dt\right] \quad (2)$$

subject to (1) where $\mathbb{E}_{\omega(t)}$ denotes the expected value with respect to noise $\omega(t)$, $\phi : \Omega \to \mathbb{R}^+$ is the final state cost, $q : \Omega \to \mathbb{R}^+$ is the accumulating state cost, and $\frac{1}{2}u(t)^T R u(t)$ is the accumulating control cost with positive definite matrix $R \in \mathbb{R}^{m\times m}$. The state reaches the boundary of Ω or a compact goal region $\Lambda \subset \Omega$ at an a priori unknown final time T. This problem is generally known as the *first exit* problem.

Assume that there exists a $\lambda > 0$ for a control penalty cost R in (2) satisfying $\lambda R^{-1} = \Sigma_\varepsilon$ [2], [3], then this controller synthesis problem is associated with the linear PDE

$$\begin{aligned}\mathcal{A}(\Psi) &\triangleq -\frac{1}{\lambda}q\Psi + f^T(\nabla_x\Psi) + \frac{1}{2}Tr((\nabla_{xx}\Psi)B(x)\Sigma_\varepsilon B(x)^T) \\ &= 0 \end{aligned} \quad (3)$$

with boundary conditions $\Psi(x) = e^{-\frac{\phi(x)}{\lambda}} \triangleq \mathcal{G}(x)$. Solving (3) gives the optimal control

$$u^* = R^{-1}B^T\frac{\nabla_x\Psi}{\Psi} \quad (4)$$

for the system (1). The solution Ψ is deemed the *desirability function* [2, Table 1]. The restriction on the cost function is quite general, and can be thought of as a design principle to ensure enough control authority is available for the subspace with high noise. For a more complete treatment on the linear HJB equation, refer to [2].

This paper focuses on numerical techniques to solve (3) in high dimensions. Henceforth, (3) is written compactly as

$$\begin{aligned}\mathcal{A}(\Psi)(x) = 0,\ x \in \Omega\backslash\Lambda \\ \Psi(x) = \mathcal{G}(x),\ x \in \partial\Omega \cup \Lambda.\end{aligned} \quad (5)$$

C. Tensor Decomposition

To compute the solution of (5) in high dimensions, the PDE is represented in the CANDECOMP/PARAFAC tensor decomposition form which scales linearly with the dimension of the system [13], [14]. Here we present a summary of tensor decompositions of functions and operators. Refer to [9] for a detailed discussion.

Consider a d-dimensional hyper-rectangle Ω and discretize Ω by M_i grid points in the i-th dimension. The notation $X_i(k)$ represents the k-th grid point in the i-th dimension. Given a real-valued function f defined on Ω, one can approximate f as

$$f(x_1, x_2, ..., x_d) \approx \sum_{l=1}^{r_F}\phi_1^l(x_1)\phi_2^l(x_2)\dots\phi_d^l(x_d) \quad (6)$$

where $\phi_i^l(x_i)$ is a single variable function and r_F is the total number of approximation terms. Then, the tensor decomposition of f is given by

$$F = \sum_{l=1}^{r_F} F_1^l \otimes F_2^l \otimes \cdots \otimes F_d^l. \quad (7)$$

where F_i^l is a M_i-length vector that represents the discretized evaluation of ϕ_i^l at the discretize points, that is $F_i^l(k) = \phi_i^l(X_i(k))$ for $k = 1, \dots, M_i$. If the vectors F_i^l are normalized to unit norm, we arrive at the CANDECOMP/PARAFAC tensor decomposition [13], [14].

Definition 1. *Given a real-valued function f defined on Ω, the CANDECOMP/PARAFAC tensor decomposition of f, denoted as a **tensor function**, is*

$$f \approx F \triangleq \sum_{l=1}^{r_F} s_l^F \bigotimes_{i=1}^{d} F_i^l \quad (8)$$

*where the **normalization constants** s_l^F are arranged in descending order according to $s_1^F \geq s_2^F \geq \cdots \geq 0$ and vector $F_i^l \in \mathbb{R}^{M_i}$ has unit norm. Each F_i^l is a **basis function** in dimension i, each summand $s_l^F \bigotimes_{i=1}^d F_i^l$ is a **tensor term**, and the total number of tensor terms, r_F, is the **separation rank**.*

By approximating the function f with a tensor function F, the number of points for storage increases linearly with dimension d for a given r_F, and linearly with r_F for a given d. Dimension d is determined by the application, as it is the dimensionality of problem (1). Hence, obtaining low rank approximations (small r_F) is vital for feasible computations. Nonetheless, a rank that is too low results in inaccurate approximations. Therefore, a balance between feasible computations and accurate approximations is a necessary consideration when determining suitable ranks.

Linear operators can similarly be approximated by tensor decomposition.

Definition 2. *Given a linear operator $\mathcal{A}$ defined on Ω, the CANDECOMP/PARAFAC tensor decomposition of $\mathcal{A}$, denoted as a* ***tensor operator****, is*

$$\mathcal{A} \approx \mathbb{A} \triangleq \sum_{l=1}^{r_A} s_l^A \bigotimes_{i=1}^{d} A_i^l \tag{9}$$

where the normalization constants s_l^F are arranged in descending order $s_1^A \geq s_2^A \geq \cdots \geq 0$, $A_i^l \in \mathbb{R}^{M_i \times M_i}$ is a unit norm matrix that represents the discretized operator, and the total number of tensor terms, r_A, is the separation rank.

Given functions and operators as tensor functions and tensor operators respectively, operations scale linearly with dimension d. For example, the multiplication operation is

$$\mathbb{A}F = \sum_{m=1}^{r_A} \sum_{l=1}^{r_F} s_m^A s_l^F \bigotimes_{i=1}^{d} A_i^m F_i^l \tag{10}$$

where the computation cost is $O(r_A r_F d M^2)$ assuming $M_i = M$ for all i. However, the separation rank often increases after such an operation. For example, (10) increases the rank from r_F to $r_A r_F$. Hence, after performing an operation, finding a low rank approximation of the resulting tensor is vital for feasible computations. Next, the algorithm used to produce low rank approximations is discussed.

Tensor decomposition is implemented numerically using the MATLAB Tensor Toolbox [15], [16].

D. Alternating Least Squares (ALS)

This section provides an overview of the main algorithm, ALS, introduced by [9]. The ALS solves (5) in the tensor decomposition form and computes low rank approximations for tensor functions and tensor operators.

Given a tensor function G and a tensor operator $\mathbb{A}$, ALS solves for F in

$$\mathbb{A}F = G \tag{11}$$

by minimizing $\|\mathbb{A}F - G\|$, termed the *residual*, for a fixed rank of F in which $\mathbb{A}$, F and G are represented in tensor decomposition form

$$F = \sum_{l=1}^{r_F} \bigotimes_{i=1}^{d} F_i^l, \quad \mathbb{A} = \sum_{l=1}^{r_A} \bigotimes_{i=1}^{d} A_i^l, \quad G = \sum_{l=1}^{r_G} \bigotimes_{i=1}^{d} G_i^l$$

where $F_i^l \in \mathbb{R}^{M_i}$, $G_i^l \in \mathbb{R}^{M_i}$, and $A_i^l \in \mathbb{R}^{M_i \times M_i}$. Note that here we do not require F_i^l, G_i^l, and A_i^l to have unit norm.

A minimum of $\|\mathbb{A}F - G\|$ satisfies $\nabla_F \|\mathbb{A}F - G\|^2 = 0$, where ∇_F denotes the gradient with respect to all vector elements $F_i^l(k)$ for $i = 1, \dots, d$, $l = 1, \dots, r_F$ and $k = 1, \dots M_i$. Unfortunately, $\nabla_F \|\mathbb{A}F - G\|^2 = 0$ is nonlinear with respect to the vector elements $F_i^l(k)$. Hence, ALS first fixes all vectors F_i^l except the vectors $\{F_k^l\}_{l=1}^{r_F}$ in dimension k. The algorithm then minimizes the residual with respect to $\{F_k^l\}_{l=1}^{r_F}$ using the now linear equation $\nabla_F \|\mathbb{A}F - G\|^2 = 0$, known as the *normal equation* [9], [17].

At each iteration for $k = 1, \dots, d$, ALS solves the normal equation and updates F. If the algorithm stagnates and the predetermined tolerance residual is not achieved, a preconditioned random tensor term is added to F. The procedure continues until the average point residual $\frac{\|\mathbb{A}F - G\|}{\sqrt{\prod_{i=1}^{d} M_i}}$ is lower than a prescribed tolerance residual ϵ.

A locally optimal solution F computed from the ALS can exhibit ill-conditioning. A regularization term $\alpha \sum_{l=1}^{r_F} ||F_k^l||^2$ with $\alpha > 0$ is therefore added to the objective function $\|\mathbb{A}F - G\|$ of the optimization [9] to prevent F from becoming ill-conditioned. The modified normal equation becomes

$$\mathcal{M}\mathcal{F}_k = \mathcal{N} \tag{12}$$

where

$$\mathcal{M} = \begin{pmatrix} M_{1,1} + \alpha I & M_{1,2} & \cdots & M_{1,r_F} \\ M_{2,1} & M_{2,2} + \alpha I & \cdots & M_{2,r_F} \\ \vdots & \vdots & \ddots & \vdots \\ M_{r_F,1} & M_{r_F,2} & \cdots & M_{r_F,r_F} + \alpha I \end{pmatrix},$$

$$\mathcal{F}_k = (F_k^1 \ F_k^2 \ \cdots \ F_k^{r_F})^T, \ \mathcal{N} = (N_1 \ N_2 \ \cdots \ N_{r_F})^T$$

and $M_{i,j}$ and N_i are given by

$$M_{i,j} = \sum_{i_A=1}^{r_A} \sum_{j_A=1}^{r_A} (A_k^{j_A})^T A_k^{i_A} \prod_{m \neq k} \langle A_m^{i_A} F_m^j, A_m^{j_A} F_m^i \rangle$$

$$N_i = \sum_{i_A=1}^{r_A} \sum_{i_G=1}^{r_G} (A_k^{i_A})^T G_k^{i_G} \prod_{m \neq k} \langle A_m^{i_A} F_m^i, G_m^{i_G} \rangle. \tag{13}$$

Henceforth, (12) is termed the normal equation, and α is set to 10^{-12} for all numerical examples so that it will not introduce too much errors.

Algorithm 1 summarizes the ALS procedure. In the rest of this paper, an iteration of this algorithm refers to one iteration of the for-loop (line 3-5).

The special case $\mathbb{A} = \mathbb{I}$, an identity operator, is used to find low rank approximations for both tensor functions and tensor operators. The latter can be achieved by storing the operator matrices A_i^l as vectors, and performing the ALS algorithm as if it was a tensor function. Refer to [9] for details.

Algorithm 1 Original ALS

```
Input: tensor operator $\mathbb{A}$ and tensor function $G$
Output: tensor function $F$
1: generate random vectors $F_i^1 \in \mathbb{R}^{M_i}$ and set $F = F_1^1 \otimes \cdots \otimes F_d^1$
2: while $\frac{\|\mathbb{A}F-G\|}{\sqrt{\prod_{i=1}^{d} M_i}} > \epsilon$ do
3:     for $k = 1, 2, \dots, d$ do
4:         solve (12) for the vector $\mathcal{F}_k$ and update $F$
5:     end for
6:     if no residual decrease then
7:         add a preconditioned random tensor term to $F$
8:     end if
9: end while
```

E. Solving the Linear HJB Equation

The linear HJB equation (5) is numerically solved using tensor decomposition and the ALS following these steps:

1) Discretize the domain Ω and set the goal region Λ.
2) Approximate the spatial HJB operator $\mathcal{A}$ by a tensor operator $\mathbb{A}'$, and the boundary conditions $\mathcal{G}$ by a tensor function G.
3) Form a new tensor operator $\mathbb{A}$ that includes the boundary condition $\mathcal{G}$ and the tensor operator $\mathbb{A}'$.
4) Recover the approximate solution F to the linear HJB equation by solving $\mathbb{A}F = G$ using ALS.

A detailed description is available in [11] and [12].

III. Issues and Improvements to the ALS

This section begins with describing an example that is used to illustrate the main points for the rest of this section. Section III-B discusses the issue of Algorithm 1, and Section III-C presents the techniques that will mitigate the issue.

A. An Illustrative Example

We first introduce a two-dimensional example from [18] whereby the true solution can be computed for comparisons. The dynamics of this system is given by

$$\begin{aligned} dx_1 &= (2(x_1^5 - x_1^3 - x_1 + x_1 x_2^4) + x_1 u_1)\ dt + d\omega_1 \\ dx_2 &= (2(x_2^5 - x_2^3 - x_2 + x_2 x_1^4) + x_2 u_2)\ dt + d\omega_2 \end{aligned} \quad (14)$$

on the domain $\Omega = \{(x_1, x_2) | -1 \leq x_1 \leq 1, -1 \leq x_2 \leq 1\}$. Set $q(x) = x_1^2 + x_2^2$, $R = 2I$, to represent that the goal is to reach the origin, $\Lambda = \{(0,0)\}$. The boundary conditions at $x_1 = \pm 1$ and $x_2 = \pm 1$ are set to $\phi(x_1, x_2) = 5$, and the boundary condition at the origin is set to $\phi(0,0) = 0$. The noise is normalized as $\Sigma_\epsilon = I$. We use a sixth order difference scheme to discretize the derivatives.

A MacBook Pro with 2.5 GHz i5 processor and 4 GB ram-memory is used to perform the computation in MATLAB.

B. Ill-Conditioning of ALS

When solving for the solution of (3), the matrix $\mathcal{M}$ in (12) often becomes ill-conditioned. In other words, κ the condition number of $\mathcal{M}$ exceeds 10^{13} where $\kappa = \frac{\sigma_{max}}{\sigma_{min}}$, and σ_{max} and σ_{min} are the maximal and minimal singular value of $\mathcal{M}$ respectively. A poorly conditioned linear equation will result in an inaccurate estimation of the solution, independent of the algorithm used to find the solution. Hence, the accuracy of the solution F in (12) will be limited by the condition number of the matrix $\mathcal{M}$ [19]. This effect may prevent ALS from iterating further to produce a lower residual resulting in a less accurate solution.

We identify two sources that cause $\mathcal{M}$ to be ill-conditioned, the operator $\mathbb{A}$ and the solution F. According to (13), $M_{i,j}$ is a function of both $\mathbb{A}$ and F. The majority of the matrices in $\mathbb{A}$ originate from discretizing the differential operators in (3) using finite differencing which tend to have coefficients with large magnitude. Particularly, elements in $\mathbb{A}$ that are near the boundary $\partial\Omega$ and the goal region Λ tend to have large values because of finite differencing on the edges.

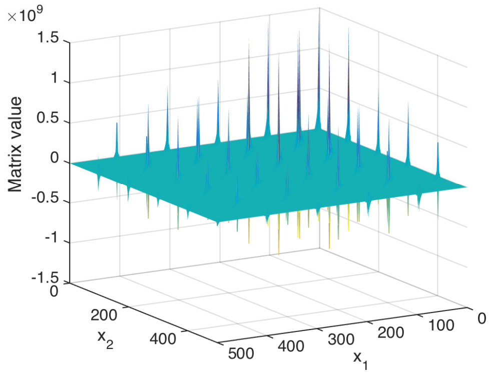

Fig. 1. An example of an ill-conditioned matrix $\mathcal{M}$ in (12) for the example system (14). The axes x_1 and x_2 denote the indices of the matrix $\mathcal{M}$, and the vertical axis is the value of $\mathcal{M}$. The peaks originate from the operator $\mathbb{A}$ and the solution F. The condition number of this matrix is 1.02×10^{13}.

However, other parts of $\mathbb{A}$ that corresponds to the boundary condition tends to be small because they consists of identity operators. As a result, the $\mathcal{M}$ that contains the tensor terms of $\mathbb{A}$ usually has a high condition number.

Furthermore, the magnitude difference among the tensor terms of the most currently computed solution F generally increases as its rank increases. New tensor terms that are added by the algorithm are generally smaller than previous tensor terms in F because the new terms are added to account for the residual of the previous tensor terms. A regularizer added to the cost function, as described in Section II-D, prevents ill-conditioning of the solution F. However, if the regularizer α is too large, the resulting solution F will give a large residual $\|\mathbb{A}F - G\|$. Therefore, limited by a moderate α to prevent a large residual, different $M_{i,j}$ in $\mathcal{M}$ that contains different tensor terms of F often have large magnitude differences, causing $\mathcal{M}$ to be ill-conditioned. Fig. 1 illustrates an example of an ill-conditioned matrix $\mathcal{M}$ for (14) that shows the effects from both $\mathbb{A}$ and F.

C. Improvements on ALS

To mitigate the ill-conditioning issue in Algorithm 1, we introduce two new improvements.

1) Sequential Computation of Solution: The linearity of (11) can be used to reduce the magnitude differences among the elements of $\mathcal{M}$ in (13) originating from the current solution F. The key idea is to subtract dominant tensor terms of F in equation (11) whenever $\mathcal{M}$ becomes ill-conditioned, and keep iterating with the remaining smaller terms. Large magnitude differences in F are therefore avoided, allowing the algorithm to realize a lower residual.

More precisely, note that

$$\begin{aligned} G = \mathbb{A}F = \mathbb{A}\sum_{l=1}^{r_F} F^l = \sum_{l=1}^{r_F} \mathbb{A}F^l \\ \iff G - \sum_{l=1}^{p-1} \mathbb{A}F^l = \sum_{l=p}^{r_F} \mathbb{A}F^l = \mathbb{A}\sum_{l=p}^{r_F} F^l \end{aligned}$$

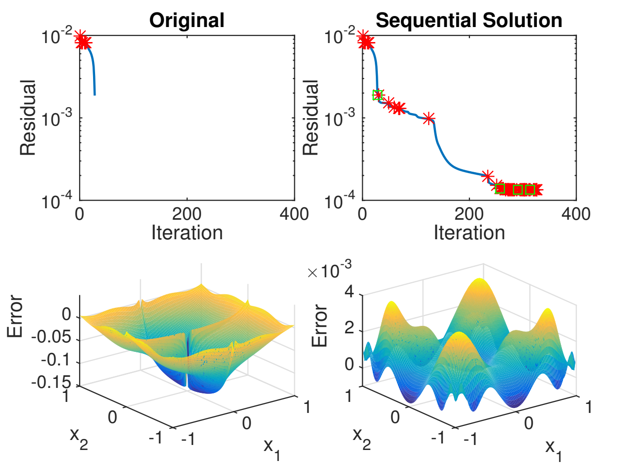

Fig. 2. Results of using Algorithm 1 on (14) with and without sequential solutions. Each dimension has $n_g = 101$ discretization points. The first row shows that the residual $\|\mathbb{A}F - G\|$ decreases in each iteration, and the second row plots the error between the computed solution and the true solution. A red star indicates when a new tensor term is added. The solution F is reset at a green square. The modified ALS achieves a more accurate solution. The original ALS quits when $\mathcal{M}$ is ill-conditioned. The algorithm with sequential solutions is terminated when the residual stops improving.

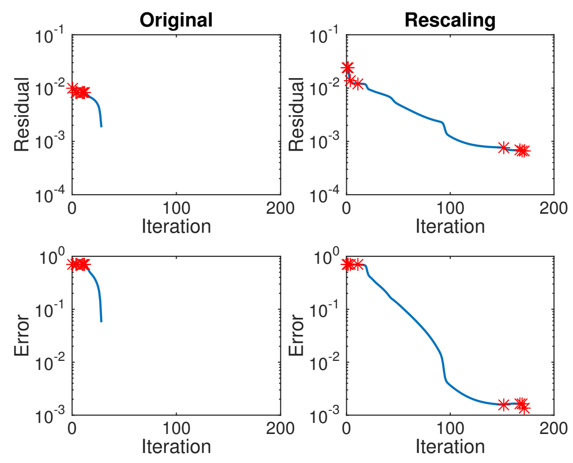

Fig. 3. Results of using Algorithm 1 on (14) with and without boundary condition rescaling. Each dimension has $n_g = 101$ discretization points. The first row shows that the residual $\|\mathbb{A}F - G\|$ decreases in each iteration, and the second row plots the mean squared error between the computed solution and the true solution in each iteration. A red star indicates when a new tensor term is added. Boundary condition rescaling yields a lower residual before $\mathcal{M}$ becomes ill-conditioned, and thus achieves a more accurate solution. Both runs quit once $\mathcal{M}$ becomes ill-conditioned.

where F^l represents the l-th tensor term in the tensor function F. Let $F_p = \sum_{l=p}^{r_F} F^l$ and $G_p = G - \sum_{l=1}^{p-1} \mathbb{A}F^l$. Then, the previous equation becomes

$$\mathbb{A}F_p = G_p$$

which is a linear equation of the form (11) that Algorithm 1 can solve. Intuitively, this technique removes the dominant terms in F from the equation allowing the algorithm to compute other smaller terms in F, avoiding ill-conditioning of $\mathcal{M}$. Hence, if the current computed solution $\hat{F}$ causes $\mathcal{M}$ to be ill-conditioned, we record the $\hat{F}$ as F_j, reset the G to $G - \mathbb{A}\hat{F}$, and restart the algorithm using the new G. As a result, we obtain a sequence of solutions F_j in which the sum of the sequence returns the full approximate solution F.

Fig. 2 shows the performance of the ALS with sequential solutions using the example (14). The modified ALS realizes a lower residual $\|\mathbb{A}F - G\|$, achieving a more accurate solution. The original ALS quits when $\mathcal{M}$ is ill-conditioned. The residual of the ALS with sequential computation stagnates when the computation reaches the MATLAB precision. At every reset of F, G is reset by subtracting $\hat{F}$ from the current G. Each subtraction reduces the magnitude of G, and eventually, the magnitude of G becomes so small that the magnitude difference between $\mathbb{A}$ and G is on the order of 10^{13}, approximately the MATLAB precision. In this case, the algorithm can no longer compute a high accuracy solution.

2) Boundary Condition Rescaling: The linearity of (11) allows for rescaling of elements in $\mathcal{M}$ to decrease its condition number. Rescaling reduces the magnitude differences resulting from the operator $\mathbb{A}$ at the start of Algorithm 1 (before line 1). Specifically, the elements in A_i^l that correspond to the boundary conditions are rescaled so that the overall condition number of $\mathcal{M}$ is decreased. The specific rescaling constants can be any constants that effectively decrease the condition number of $\mathcal{M}$. For example, the rescaling constant in dimension i can be computed from the weighted summation of the matrices $\{A_i^l\}_{l=1}^{r_A}$

$$S_i = \frac{\sum_{l=1}^{r_A} A_i^l \|A_i^l\|}{\sum_{l=1}^{r_A} \|A_i^l\|}$$

to rescale boundary conditions in dimension i to approximately the same magnitude as the elements not corresponding to the boundary conditions. Note that G is rescaled to ensure that (11) still holds. Different elements in A_i^l can have different rescaling constants. See [12] for more details on how to implement and choose the rescaling constants.

The performance of this boundary rescaling technique is demonstrated using the example system (14) and the result is shown in Fig. 3. By rescaling the boundary conditions, the modified ALS algorithm can solve to a lower residual $\|\mathbb{A}F - G\|$ achieving a more accurate solution (see second row of Fig. 3) before $\mathcal{M}$ becomes ill-conditioned. Both runs quit once $\mathcal{M}$ becomes ill-conditioned.

Furthermore, Fig. 4 shows that sequential computation combined with boundary condition rescaling produces a more accurate solution and decreases computation time dramatically from about 77 minutes (580 iterations) to about 7 minutes (190 iterations) before the residuals stabilized. The residuals for both cases stabilize once the computations reach MATLAB precision. The separation rank of the computed solutions are 17 and 126 for algorithm with and without operator rescaling respectively. Note the residual is lower for the algorithm without boundary condition rescaling (see first row of Fig. 4), but the solution is more accurate for the algorithm with boundary condition scaling (see second row of Fig. 4). This observation is not unexpected because the residual is calculated after the boundary conditions are rescaled. Thus, when comparing the two, one should consider

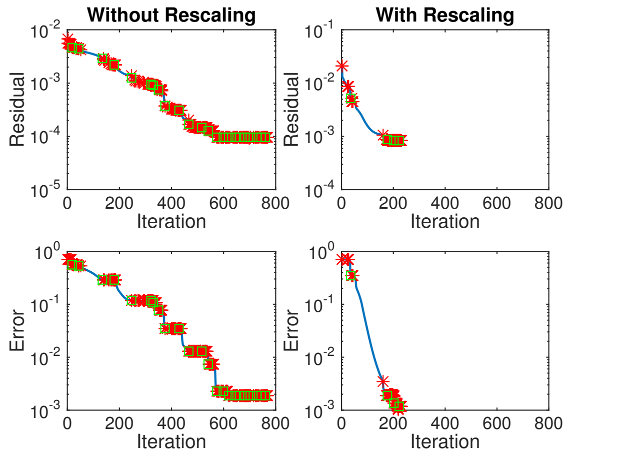

Fig. 4. Results of implementing sequential computation on (14) with and without boundary condition rescaling. We chose $n_g = 151$ discretization points for each dimension to distinguish the two cases better. The first row shows how the residual $\|\mathbb{A}F - G\|$ decreases with each iteration, and the second row shows the mean squared error between the computed solution and the true solution with each iteration.

that the residuals are not exactly equivalent.

IV. Sequential Alternating Least Squares (SeALS)

Combining the sequential solution and boundary condition rescaling, we arrive at the SeALS algorithm (Algorithm 2), which can compute more accurate solutions for the linear HJB equation in a shorter amount of time.

Given a tensor operator $\mathbb{A}$ and a tensor function G derived from (5), SeALS implements the original ALS algorithm with two additional steps. First, before solving (12), elements in $\mathbb{A}$ that correspond to the boundary conditions are rescaled as described in Section III-C.2, and G is also rescaled.

Algorithm 2 Sequential ALS (SeALS)

```
Input: tensor operator $\mathbb{A}$ and tensor function $G$
Output: tensor function $F$
 1: rescale boundary conditions in $\mathbb{A}$ and $G$ accordingly
 2: generate random vectors $F_i^1 \in \mathbb{R}^{M_i}$ and set $F = F_1^1 \otimes \cdots \otimes F_d^1$
 3: set $j = 1$
 4: while $\frac{\|\mathbb{A}F - G\|}{\sqrt{\prod_{i=1}^d M_i}} > \epsilon$ do
 5:     for $k = 1, 2, \dots, d$ do
 6:         solve (12) for the vector $\mathcal{F}_k$ and update $F$
 7:     end for
 8:     if $\mathcal{M}$ is ill-conditioned then
 9:         set $F_j = F$ and increment $j$
10:         set $G$ to $G - \mathbb{A}F$ (subtract $F$)
11:         set $F$ to a preconditioned random rank 1 tensor term
12:     else if no residual decrease then
13:         add a preconditioned random tensor term to $F$
14:     end if
15: end while
16: set $F = \sum_j F_j$
```

Second, when $\mathcal{M}$ is ill-conditioned, the current F is recorded as F_j. Then, G is reset to $G - \mathbb{A}F$, and F is reset to a preconditioned random rank 1 tensor term. In the end, the algorithm returns F, which is the sum of all previously recorded F_j.

This SeALS algorithm is implemented in MATLAB [20]. A more detailed description and other additional features of SeALS are available in the tool's user's guide [21].

V. Examples

The capability of SeALS is demonstrated using three examples from [11]. Note that simulations are performed with the indicated noise, whereas simulations in [11] were noiseless. A MacBook Pro with 2.5 GHz i5 processor and 4 GB RAM is used in the inverted pendulum and quadcopter examples, and a quadcore computer with 3.0 Ghz i7 processor and 64 GB RAM is used in the VTOL aircraft example.

A. Inverted Pendulum

The dynamics of an inverted pendulum adapted from [22] is given by

$$dx_1 = x_2\, dt + d\omega \tag{15}$$
$$dx_2 = \frac{\frac{g}{l}\sin(x_1) - \frac{1}{2}m_r x_2^2 \sin(2x_1) - \frac{m_r}{ml}\cos(x_1)u}{\frac{4}{3} - m_r\cos^2(x_1)}\, dt + d\omega$$

where x_1 is the angle from upright position of the pendulum and x_2 is the angular velocity. The goal is to keep the pendulum upright. Hence, we set the goal region $\Lambda = \{(x_1, x_2) \in \mathbb{R}^2 \mid |x_1| \le 0.18,\ |x_2| \le 0.3\}$ around the origin with zero cost. We also set $q = 0.1x_1^2 + 0.05x_2^2$, $R = 0.02$ and choose the domain $\Omega = \{(x_1, x_2) \in \mathbb{R}^2 \mid |x_1| \le \pi,\ |x_2| \le 11\}$. We place periodic boundary conditions for x_1, $\phi(x_1, \pm 11) = 10$ for x_2, and $\Sigma_\epsilon = 10I$ for noise.

The final solution and the first five basis functions $\{F_i^l\}_{l=1}^5$ in each dimension are shown in Fig. 5. Residual $\|\mathbb{A}F - G\|$ and normalization constants s_l^F are shown in Fig. 6. The algorithm quits with a separation rank of 230 when it exceeds a prescribed 2000 iterations, with a total run time of 37 hours, achieving a lower residual compared to [9]. Shorter runs can be achieved by optimizing the SeALS tool for speed or by quitting earlier when the basis weights are small enough for

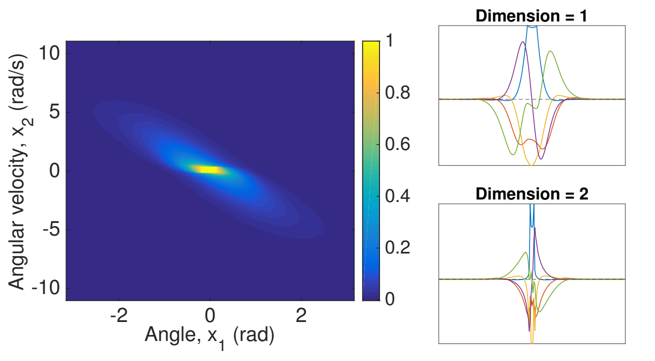

Fig. 5. Computed solution (left) and the basis functions $\{F_i^l\}_{l=1}^5$ in each dimension (right) for the inverted pendulum.

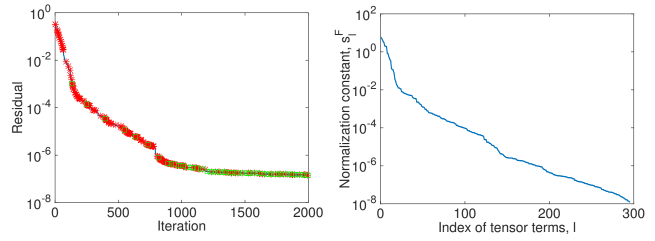
Fig. 6. Residual $\|\mathbb{A}F - G\|$ (left) and the normalization constants s_l^F (right) for the inverted pendulum. The algorithm quits when it exceeds the prescribed maximum number of iterations 2000.

the specific application. A simulation is shown in Fig. 7 in which the pendulum reaches the goal region as desired when using a controller computed by SeALS.

B. Vertical Takeoff and Landing (VTOL) Aircraft

The dynamics of a VTOL aircraft (from [23]) in the translation plane (x, y) with tilt angle θ are given by

$$\begin{aligned}
dx &= v_x \; dt \\
dv_x &= -\sin\theta(u_1 \; dt + d\omega_1) + \epsilon\cos\theta(u_2 \; dt + d\omega_2) \\
dy &= v_y \; dt \\
dv_y &= (-g + \cos\theta)(u_1 \; dt + d\omega_1) + \epsilon\sin\theta(u_2 \; dt + d\omega_2) \\
d\theta &= v_\theta \; dt \\
dv_\theta &= u_2 \; dt + d\omega_2
\end{aligned}$$

where g is the gravitational constant, and $\epsilon = 0.01$. The domain Ω is given by $x \in [-4, 4]$, $y \in [0, 2]$, $v_x \in [-8, 8]$, $v_y \in [-1, 1]$, $v_\theta \in [-5, 5]$, and periodic $\theta \in [-\pi, \pi]$. We set $q(x) = 1$ and $R = 2I$. The goal is to land the aircraft at $y = 0$. Hence, the goal region Λ is set to the origin so that the aircraft reaches $y = 0$ with moderate velocities v_x, v_y and v_θ and deviations in x and θ. We impose boundary conditions $\Psi|_{\partial\Omega} = 0$ for all non-periodic states except $y = 0$, where we set $\Psi|_{y=0} = \prod_{i=1}^{d}(1 - s_i^2)$ and s_i is the normed coordinate in direction i. The noise is set as $\Sigma_\epsilon = 3I$. The algorithm took 30 hours, and the separation rank is 134.

Fig. 8 shows a simulation of the VTOL Aircraft using controller computed from the solution produced by the SeALS algorithm. The controller successfully lands the aircraft ($y = 0$) with moderate speed and horizontal deviations, comparable to [9] despite the noise.

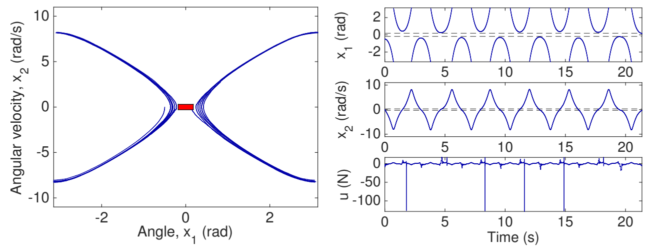
Fig. 7. A simulation of the inverted pendulum using controller computed from the solution produced by the SeALS algorithm. Left plot shows the state trajectory in space, and right plots shows the state and control trajectories in time. The red box is the goal region. The angle x_1 is periodic.

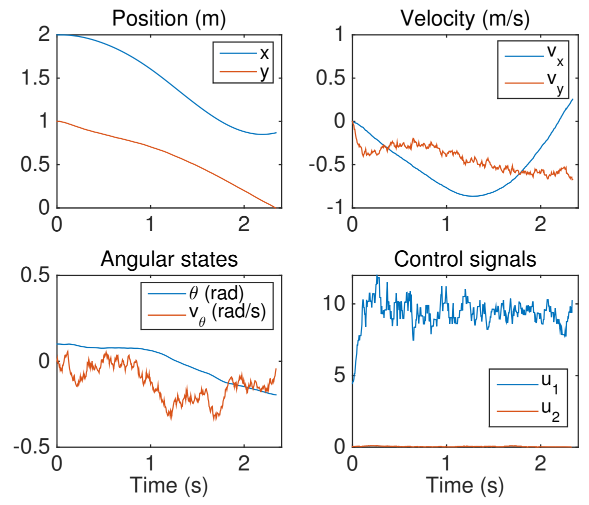
Fig. 8. A simulation of the VTOL aircraft using controller computed from the solution produced by the SeALS algorithm. The controller successfully lands the aircraft ($y = 0$) with moderate speed and horizontal deviations.

C. Quadcopter

The quadcopter (from [24]) moves in three-dimensional space (x, y, z) with orientation given by the yaw angle ψ, pitch angle θ and roll angle ϕ. The control inputs $u(t) = (\mu, \tau_\psi, \tau_\theta, \tau_\phi)^T$ are the main thrust μ, the yawing moment τ_ψ, the pitching moment τ_θ and the rolling moment τ_ψ. The dynamics are given by (1) in which

$$f = \begin{pmatrix} v_x & v_y & v_z & v_\psi & v_\theta & v_\phi & 0 & 0 & -g & 0 & 0 & 0 \end{pmatrix}^T$$

$$B = \begin{pmatrix} 0_{6\times1} & 0_{6\times3} \\ \sin\phi\sin\psi + \cos\phi\cos\psi\sin\theta & \\ \cos\phi\sin\theta\sin\psi - \cos\psi\sin\theta & 0_{3\times3} \\ \cos\theta\cos\phi & \\ 0_{3\times1} & I_{3\times3} \end{pmatrix}$$

where $x(t) = (x, y, z, \psi, \theta, \phi, v_x, v_y, v_z, v_\psi, v_\theta, v_\phi)^T$, $0_{m\times n}$ represents a m by n zero matrix, $I_{m\times n}$ represents a m by

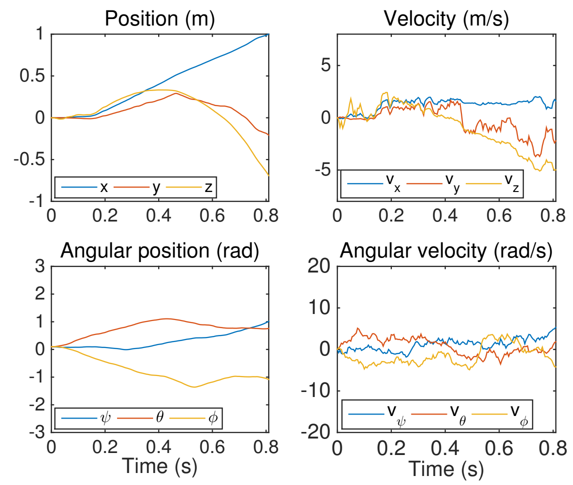
Fig. 9. A simulation of the quadcopter controlled using the solution produced by the SeALS. The quadcopter reaches $x = 1$ as desired.

n identity matrix, and g is the gravitational constant. The domain Ω is given by $x, y, z \in [-1, 1]$, periodic $\psi, \theta, \phi \in [-\pi, \pi]$, $v_x, v_y, v_z \in [-8, 8]$ and $v_\psi, v_\theta, v_\phi \in [-10\pi, 10\pi]$. We set $q = 2$ and $R = 2I$. Since the goal is to reach $x = 1$, the goal region Λ is set to the point $x = 1$ and zero for the remaining coordinates. We impose boundary conditions $\Psi|_{\partial\Omega} = 0$ except $x = 1$ where we set $\Psi|_{x=1} = \prod_{i=1}^{d}(1-s_i^2)$. The noise is set as $\Sigma_\epsilon = 100I$. The algorithm took 62 hours, and the separation rank is 52.

Fig. 9 shows a sample simulation of the quadcopter controlled using the solution produced by SeALS. The trajectory reaches $x = 1$ as desired, comparable to [9] despite the noise.

VI. Conclusion and Future Work

This paper presents a method to solve the high dimensional linear HJB equation for the first exit problems of a large class of stochastic affine nonlinear dynamical systems. This work significantly improves upon [11] where the concept of tensor decomposition and ALS are considered for solving the linear HJB equation. The original framework's ill-conditioning issue is mitigated through sequential computation of solutions and boundary condition rescaling. Consequently, the new algorithm that incorporates both methods achieves significantly lower error compared to the original implementation, resulting in more accurate solutions and better controllers. A MATLAB tool, SeALS, that implements the new algorithm is created. Three engineering examples including stabilizing a quadcopter are presented to illustrate its performance. The ability to compute the solution to the linear HJB equation for a quadcopter with twelve dimensions using a personal laptop and produce a controller that achieve the objective is a strong indicator of SeALS's great potential for implementation in other robotics and engineering systems.

Future work includes optimizing the SeALS tool for speed. Other numerical techniques that improve the algorithm's stability are currently under investigation, including adding artificial diffusion [25]. Different kinds of differentiation schemes such as Chebyshev differentiation [26] and upwind finite difference scheme [27] are under exploration. Future improvements of the SeALS tool will include time varying linear HJB equations.

VII. Acknowledgements

This work was supported by Caltech Summer Undergraduate Research Fellowship. The authors would like to thank Joel Burdick, John Doyle and Matanya Horowitz for useful feedback on this work.

References

[1] D. P. Bertsekas, *Dynamic programming and optimal control*. Athena Scientific Belmont, MA, 1995, vol. 1, no. 2.

[2] E. Todorov, "Efficient computation of optimal actions," *Proc. the Nat. Acad. Scien.*, vol. 106, no. 28, pp. 11 478–11 483, 2009.

[3] Y. P. Leong, M. Horowitz, and J. Burdick, "Suboptimal stabilizing controllers for linearly solvable system," in *IEEE Int. Conf. on Decision and Control (CDC)*, Dec 2015.

[4] R. E. Bellman and S. E. Dreyfus, *Applied dynamic programming*. Princeton university press, 2015.

[5] J. Garcke and A. Kröner, "Suboptimal feedback control of PDEs by solving HJB equations on adaptive sparse grids," INRIA Saclay, Research Report, 2015. [Online]. Available: https://hal.archives-ouvertes.fr/hal-01185912

[6] C. O. Aguilar and A. J. Krener, "Numerical solutions to the Bellman equation of optimal control," *J. optimization theory and applications*, vol. 160, no. 2, pp. 527–552, 2014.

[7] W. M. McEneaney, "Convergence rate for a curse-of-dimensionality-free method for Hamilton-Jacobi-Bellman PDEs represented as maxima of quadratic forms," *SIAM J. Control and Optimization*, vol. 48, no. 4, pp. 2651–2685, 2009.

[8] S. Gombao, "Approximation of optimal controls for semilinear parabolic PDE by solving Hamilton-Jacobi-Bellman equations," in *Proc. of the 15th Int. Symposium on the Mathematical Theory of Networks and Systems, University of Notre Dame, South Bend, Indiana, USA*, 2002.

[9] G. Beylkin and M. Mohlenkamp, "Algorithms for numerical analysis in high dimensions," *SIAM J. Scientific Computing*, vol. 26, no. 6, pp. 2133–2159, 2005.

[10] A. Gorodetsky, S. Karaman, and Y. Marzouk, "Efficient high-dimensional stochastic optimal motion control using tensor-train decomposition," in *Proc. of Robotics: Science and Systems*, Rome, Italy, July 2015.

[11] M. B. Horowitz, A. Damle, and J. W. Burdick, "Linear Hamilton Jacobi Bellman equations in high dimensions," in *IEEE Int. Conf. on Decision and Control (CDC)*. IEEE, 2014, pp. 5880–5887.

[12] E. Stefansson and Y. P. Leong, "Sequential alternating least squares for solving high dimensional linear Hamilton-Jacobi-Bellman equation," 2015. [Online]. Available: http://www.cds.caltech.edu/~yleong/docs/Research_Papers/SeALS.pdf

[13] J. D. Carroll and J.-J. Chang, "Analysis of individual differences in multidimensional scaling via an n-way generalization of eckart-young decomposition," *Psychometrika*, vol. 35, no. 3, pp. 283–319, 1970.

[14] R. A. Harshman, "Foundations of the PARAFAC procedure: Models and conditions for an "explanatory" multi-modal factor analysis," *University of California at Los Angeles*, 1970.

[15] B. W. Bader, T. G. Kolda *et al.*, "Matlab tensor toolbox version 2.6," Available online, February 2015. [Online]. Available: http://www.sandia.gov/~tgkolda/TensorToolbox/

[16] B. W. Bader and T. G. Kolda, "Algorithm 862: MATLAB tensor classes for fast algorithm prototyping," *ACM Trans. Mathematical Software*, vol. 32, no. 4, pp. 635–653, December 2006.

[17] Y. Sun and M. Kumar, "Numerical solution of high dimensional stationary Fokker-Plank equations via tensor decomposition and Chebyshev spectral differentian," *Computers and Mathematics with Applications*, vol. 65, no. 10, pp. 1960–1977, 2014.

[18] Y. P. Leong, M. Horrowitz, and J. Burdick, "Linearly solvable stochastic control Lyapunov functions," *arXiv preprint arXiv:1410.0405*, 2014.

[19] G. H. Golub and C. F. Van Loan, *Matrix computations*. JHU Press, 2012, vol. 3.

[20] E. Stefansson and Y. P. Leong, "Sequential alternating least squares (SeALS) in MATLAB," 2015. [Online]. Available: http://www.cds.caltech.edu/~yleong/research.php

[21] ——, "Sequential alternating least squares (SeALS) in MATLAB user's guide," 2015. [Online]. Available: http://www.cds.caltech.edu/~yleong/codes/SeALS_usersguide.pdf

[22] H. M. Osinga and J. Hauser, "The geometry of the solution set of nonlinear optimal control problems," *J. Dynamics and Differential Equations*, vol. 18, no. 4, pp. 881–900, 2006.

[23] J. Hauser, S. Sastry, and G. Meyer, "Nonlinear control design for slightly non-minimum phase systems: application to V/STOL aircraft," *Automatica*, vol. 28, no. 4, pp. 665–679, 1992.

[24] L. R. G. Carrillo, A. E. D. López, R. Lozano, and C. Pégard, "Modeling the quad-rotor mini-rotorcraft," in *Quad Rotorcraft Control*. Springer, 2013, pp. 23–34.

[25] A. Jameson, "Analysis and design of numerical schemes for gas dynamics, 1: artificial diffusion, upwind biasing, limiters and their effect on accuracy and multigrid convergence," *Int. J. Comp. Fluid Dynamics*, vol. 4, no. 3-4, pp. 171–218, 1995.

[26] L. N. Trefethen, *Spectral methods in MATLAB*. SIAM, 2000, vol. 10.

[27] B. Sun and B.-Z. Guo, "Convergence of an upwind finite-difference scheme for Hamilton-Jacobi-Bellman equation in optimal control," *IEEE Trans. oAutomatic Control*, vol. 60, no. 11, pp. 3012–3017, 2015.