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 solve the linear Hamilton-Jacobi-Bellman (HJB) equation for a class of stochastic affine nonlinear dynamical systems in high dimensions. The solution of a HJB equation provides a globally optimal controller to the associated dynamical system. However, the curse of dimensionality, which commonly presents in robots and other engineering systems, prevents one from solving the linear HJB equation naively. In this work, the curse is avoided 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 dimensions two, six and twelve respectively.

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

---

\(^1\)E. Stefansson is with KTH Royal Institute of Technology, Sweden elisst@kth.se.

\(^2\)Y. P. Leong with the Control and Dynamical Systems, California Institute of Technology, Pasadena, CA 91125, USA ypleong@caltech.edu.
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 number of dimensions [9]. Another group of researchers [10] also considered utilizing a tensor decomposition based technique. However, that technique solves for an optimal controller by using a value iteration scheme instead of the HJB equation.

In this work, a numerical technique based on tensor decomposition and the Alternating Least Squares (ALS) is developed to solve the linear HJB equation associated with the first exit problem of a large class of affine nonlinear systems in high dimensions. Tensor decomposition is used to represent 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 a higher accuracy. This paper introduces sequential computation of the solution and a boundary condition rescaling concept to alleviate the ill-conditioning issue of the original ALS. As a result of these improvements, 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 a twelve-dimensional linear HJB equation of a quadcopter on a personal laptop suggests that SeALS has great potential for use in robotics or engineering applications. Lastly, in addition to solving the HJB equation, this tool can potentially be adapted to solve general linear PDEs in high dimension.

A. Outline

The rest of this paper is organized as follows. 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 the issue of the original algorithm [11], and describes the improvements using sequential computation of solution and boundary condition rescaling. The MATLAB tool, SeALS, is summarized in Section IV. Section V shows three numerical examples that demonstrate the performance of SeALS. Conclusion and future work are given in Section VI.

II. Preliminaries

A. Notation

The set of 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 written as $\partial \Omega$. A sequence of points from $x_1$ to $x_k$ is written as $\{x_i\}_{i=1}^k$. The continuous time trajectory from time $t$ to time $T$ is denoted as $x_{t:T}$.

The symbols $\nabla_x$ and $\nabla_{xx}$ represent the gradient and the hessian respectively, and $\nabla_k$ represents the directional derivative in $k$-th dimension.

The norm $\|\cdot\|$ represents the Frobenius norm. Tensor product is represented as $\otimes$. A function $f(\cdot)$ is abbreviated as $f$ when the arguments of the function are clear from the context.

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

Consider the stochastic affine dynamical system

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

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 $\Sigma_e$. 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 Section V.

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

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

subject to (1) where \( \mathbb{E}_{\omega(t)} \) denotes the expected value with respect to the 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 end time \( T \), unknown a priori, is the time when the state reaches the boundary of \( \Omega \) or a compact goal region \( \Lambda \subset \Omega \). 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 \( \lambda R^{-1} = \Sigma_e \) [2], [3], then this controller synthesis problem is associated with the linear PDE

\[
\mathcal{A}(\Psi) \triangleq -\frac{1}{\lambda} q \Psi + f^T (\nabla_x \Psi) + \frac{1}{2} \text{Tr}((\nabla_{xx} \Psi) B(x) \Sigma_e B(x)^T) = 0
\]

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

\[
u^* = R^{-1} B^T \nabla_x \Psi
\]

for the system (1). The solution \( \Psi \) is deemed the desirability function [2, Table 1]. The restriction on the cost function is quite general. In fact, the equality can be thought 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].

The focus of this paper is on the numerical technique to solve (3) in high dimensions. Henceforth, (3) is written more compactly as

\[
\mathcal{A}(\Psi)(x) = 0, \ x \in \Omega \setminus \Lambda
\]

\[
\Psi(x) = G(x), \ x \in \partial \Omega \cup \Lambda.
\]

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 [12], [13]. Here we presents a summary of tensor decompositions of functions and operators. Refer to [9] for a detailed discussion.
Consider a $d$-dimensional hyper-rectangle $\Omega$ and discretize $\Omega$ 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 $\Omega$, one can approximate $f$ as

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

(6)

where $\phi_i^l(x_i) \text{ is a single variable function and } r_F \text{ is the total number of approximation terms. The right hand side of } (6) \text{ is called a separation representation} \cite{9}. \text{ Then, the tensor decomposition of } f \text{ is given by}

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

(7)

where $F_i^l$ is a $M_i$-length vector that represents the discretized evaluation of $\phi_i^l$ at the discretize points, that is $F_i^l(k) = \phi_i^l(X_i(k))$ for $k = 1, \ldots, M_i$. If the vectors $F_i^l$ are normalized to unit norm, we arrive at the CANDECOMP/PARAFAC tensor decomposition \cite{12}, \cite{13}.

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

$$f \approx F \triangleq \sum_{l=1}^{r_F} s_F^l \bigotimes_{i=1}^d F_i^l$$

(8)

where the normalization constants $s_F^l$ are arranged in descending order according to $s_F^1 \geq s_F^2 \geq \cdots \geq 0$ and $F_i^l \in \mathbb{R}^{M_i}$ is a vector with unit norm. Each $F_i^l$ is called a basis function in dimension $i$, each summand $s_F^l \bigotimes_{i=1}^d F_i^l$ is called a tensor term, and the total number of tensor terms, $r_F$, is called 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 usually determined by applications. 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.

Similarly, linear operators can also be approximated using tensor decomposition.
**Definition 2.** Given a linear operator \( A \) defined on \( \Omega \), the CANDECOMP/PARAFAC tensor decomposition of \( A \), denoted as a **tensor operator**, is

\[
A \approx A = \sum_{l=1}^{r_A} s_l^A \bigotimes_{i=1}^d A_i^l
\]  

(9)

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

\[
A F = \sum_{m=1}^{r_A} \sum_{l=1}^{r_F} s_m^A s_l^F \bigotimes_{i=1}^d A_m^l F_i^l
\]

(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, a low rank approximation of the resulting tensor is vital for feasible computations. Next, the algorithm used to produce low rank approximation is discussed.

Tensor decomposition is implemented numerically using the MATLAB Tensor Toolbox [14], [15].

**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 \( A \), ALS solves for \( F \) in

\[
A F = G
\]

(11)

by minimizing \( \| A F - G \| \), termed the residual, for a fixed rank of \( F \) in which \( A, F \) and \( G \) are represented in tensor decomposition form

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

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 $\|AF - G\|$ satisfies $\nabla_F \|AF - G\|^2 = 0$, where $\nabla_F$ denotes the gradient with respect to all vector elements $F_i^l(k)$ for $i = 1, \ldots, d$, $l = 1, \ldots, r_F$ and $k = 1, \ldots, M_i$. Unfortunately, $\nabla_F \|AF - G\|^2 = 0$ 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 $\nabla_F \|AF - G\|^2 = 0$, known as the normal equation [9, 16].

At each iteration for $k = 1, \ldots, d$, 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 $\frac{\|AF - G\|}{\sqrt{M_i}}$ is lower than a prescribed tolerance residual $\epsilon$.

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 $\|AF - G\|$ of the optimization [9] to prevent $F$ from becoming ill-conditioned. The modified normal equation becomes

$$\mathcal{M}F_k = \mathcal{N}$$  \hspace{1cm} (12)

---

**Algorithm 1** Original ALS

**Input:** tensor operator $\mathcal{A}$ and tensor function $G$

**Output:** tensor function $F$

1: generate random vectors $F_1^1 \in \mathbb{R}^{M_i}$ and set $F = F_1^1 \otimes \cdots \otimes F_1^d$
2: while $\frac{\|AF - G\|}{\sqrt{M_i}} > \epsilon$ do
3:     for $k = 1, 2, \ldots, d$ do
4:         solve (12) for the vector $F_k$ 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
where

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

\[F_k = \begin{pmatrix}
F^1_{k} \\
F^2_{k} \\
\vdots \\
F^{r_F}_{k}
\end{pmatrix},
\]

\[N = \begin{pmatrix}
N_1 \\
N_2 \\
\vdots \\
N_{r_F}
\end{pmatrix},
\]

and \(M_{i,j}\) and \(N_i\) are given by

\[
M_{i,j} = \sum_{i_A=1}^{r_A} \sum_{j_A=1}^{r_A} (A^{i_A}_k)^T A^{j_A}_k \prod_{m \neq k} \langle A^{i_A}_m F^j_m, A^{j_A}_m F^i_m \rangle,
\]

\[
N_i = \sum_{i_A=1}^{r_A} \sum_{i_G=1}^{r_G} (A^{i_A}_k)^T G^{i_G}_k \prod_{m \neq k} \langle A^{i_A}_m F^i_m, G^{i_G}_m \rangle.
\] (13)

Henceforth, (12) is termed the normal equation, and \(\alpha\) is set to \(10^{-12}\) for all numerical examples.

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 \(A = I\), 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.

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 \(\Omega\) and set the goal region \(\Lambda\):** Given a \(d\)-dimensional hyper-rectangle \(\Omega = \{(x_1, \ldots, x_d) \in \mathbb{R}^d \mid a_k \leq x_k \leq b_k, \ k = 1, \ldots, d\}\) where \(a_k < b_k\). The subdomain of \(\Omega\) given by \(x_k = a_k\) is termed the lower boundary in dimension \(k\), and the subdomain given by \(x_k = b_k\) is termed the upper boundary in dimension \(k\). The union of these two subdomains are called the boundary in dimension \(k\).

Discretize \(\Omega\) by \(M_i\) grid points in the \(i\)-th dimension, and represent the discretized domain using a multi-dimensional array \(T \in \mathbb{R}^{M_1 \times M_2 \times \cdots \times M_d}\). Each point \((m_1, m_2, \ldots, m_d)\) in the array \(T\) corresponds to a grid point in the discretized domain, denoted by \((X_1(m_1), X_2(m_2), \ldots, X_d(m_d))\). For example, the hyper-rectangle \(\Omega\) with \(d = 2\) is an ordinary rectangle and the multi-dimensional array \(T\) is a matrix.
The goal region $\Lambda$ is a hyper-rectangular subdomain of $\Omega$ defined by the specific application. It uses the same discretization grid as $\Omega$.

2) Approximate the spatial operator $A$ by a tensor operator $A'$, and the boundary conditions $G$ by a tensor function $G$: To approximate the spatial operator $A$ by a tensor operator $A'$, first approximate matrix valued-functions $q, f, B$ and $\Sigma_\epsilon$ from the spatial operator $A$ as matrix-valued tensor functions. Then, convert these matrix-valued tensor functions into matrix-valued tensor operators. Lastly, approximate the gradient $\nabla_x$ and the Hessian $\nabla_{xx}$ in the spatial operator $A$ as matrix-valued tensor operators.

Specifically, matrix-valued functions $q, f, B$ and $\Sigma_\epsilon$ consist of real-valued functions. Assume each real-valued function has an exact separated representation. Then, the real-valued function can be approximated by the tensor function as in (7). As a result, tensor decompositions of $q, f, B$ and $\Sigma_\epsilon$ are matrices composed of corresponding tensor functions, denoted matrix-valued tensor functions.

**Remark 1.** For a smooth function $f$ that does not have an exact separated representation, one can always find a separated representation with arbitrary accuracy such as the Fourier series. The approximate separated representation is then used to obtain a tensor decomposition for $f$.

Next, convert matrix-valued tensor functions of $q, f, B$ and $\Sigma_\epsilon$ into tensor matrix operators. For each tensor function in a given matrix-valued tensor function, diagonalize the basis functions according to

$$
\sum_{l=1}^{r_F} F_l^1 \otimes \cdots \otimes F_d^l \rightarrow \sum_{l=1}^{r_F} \text{diag}(F_l^1) \otimes \cdots \otimes \text{diag}(F_d^l)
$$

where $\text{diag}(F)$ is the diagonal matrix with vector $F$ as the diagonal. The matrix-valued tensor functions in this new form is called a matrix-valued tensor operator. Let $q_{op}, f_{op}, B_{op}$ and $\Sigma_{\epsilon,op}$ denote the matrix-valued tensor operators of $q, f, B$ and $\Sigma_\epsilon$ respectively.

In addition, approximate the gradient $\nabla_x$ and the Hessian $\nabla_{xx}$ in the spatial operator $A$ by matrix-valued tensor operators $D_1$ and $D_2$ respectively. The tensor decomposition of the partial derivative in $k$-th dimension is given by

$$
\nabla_k = I_1 \otimes \ldots I_{k-1} \otimes D \otimes I_{k+1} \otimes \cdots \otimes I_d
$$

where $D$ is an ordinary finite-difference matrix. The matrix-valued tensor operator $D_1$ of the gradient $\nabla_x$ is formed with entries $\nabla_k$, while the matrix-valued tensor operator $D_2$ of the hessian $\nabla_{xx}$ is formed with entries $\nabla_{k,j} = \nabla_k \cdot \nabla_j$. 
Lastly, multiply and add all matrix-valued tensor operators to form the tensor operator $A'$

$$A'(\cdot) = -\frac{1}{\lambda} q_{op}(\cdot) + f_{op}^T D_1(\cdot) + \frac{1}{2} Tr(D_2(\cdot)B_{op}\Sigma_{e,op}B_{op}^T).$$

The tensor function $G$ for the boundary condition $\mathcal{G}$ is constructed analogously to the construction of the tensor operator $A'$ for the spatial operator $A$. The elements in $G$ that correspond to the boundary points $x \in \partial \Omega$ is set to $e^{-\frac{\phi(x)}{\lambda}}$, and the elements that correspond to the goal region $x \in \Lambda$ is set to one (i.e. zero-cost at the goal). The rest of the elements of the tensor function $G$ in the interior $x \in \Omega \setminus \Lambda$ is set to zero.

3) Form a new tensor operator $\tilde{A}_e$ that includes the boundary condition $\mathcal{G}$ and the tensor operator $A'$: Now, we combine the boundary condition $\mathcal{G}$ with the operator $A' = \sum_{l=1}^{r_A} A_l \otimes \cdots \otimes A_{d}$. The procedure is first illustrated for Dirichlet boundary conditions of $\partial \Omega$. Given a Dirichlet boundary condition in the $k$-th dimension, replace the first and last row in the matrices $\{A_k^{l}\}_{l=1}^{r_A}$ by zeros. The operator now acts as zero on the boundary in the $k$-th dimension. Then, add

$$I_1 \otimes \cdots \otimes I_{k-1} \otimes C \otimes I_{k+1} \cdots I_d$$

(16)

to $A'$, where $C$ is a zero matrix except for two ones in the first and last diagonal elements. The new operator now acts as an identity on the boundary in the $k$-th dimension. Repeat this procedure for all dimensions $k = 1, \ldots, d$. Then, the new operator $\tilde{A}_e$ sets the boundary values according to $G$ in $\tilde{A}_e F = G$.

So far, only Dirichlet boundary conditions on $\partial \Omega$ is considered. A slight modification in the current procedure also enables Neumann boundary conditions and periodic boundary conditions on $\partial \Omega$ to be included. For Neumann conditions, the first and last row of $C$ are finite difference schemes calculating the directional derivative at the boundary. For periodic conditions, only the first row of $\{A_k^{l}\}_{l=1}^{r_A}$ is replaced by zeros. The zero matrix $C$ has 1 and -1 as first and last element in the first row. The points in $G$ corresponding to periodic boundaries are set to zero to achieve the periodicity.

To incorporate the goal region $\Lambda$, let $\tilde{A}_\Lambda$ be the tensor operator acting as $\tilde{A}$ within $\Lambda$ and zero everywhere else, and let $\mathbb{I}_\Lambda$ be the tensor operator acting as identity within $\Lambda$ and zero everywhere else. Then, the new operator $\tilde{A}$ is given by

$$\tilde{A} = \tilde{A}_e - \tilde{A}_\Lambda + \mathbb{I}_\Lambda,$$

(17)

where the operator $\tilde{A}$ acts as an identity on the goal region $\Lambda$. The value at the goal region $\Lambda$ is then given by $G$ according to $\tilde{A} F = G$. 

4) Recover the approximate solution $F$ to the linear HJB equation by solving $\mathbb{A}F = G$ using ALS: The discrete version of the linear HJB equation (5) becomes

$$\mathbb{A}F = G$$ (18)

where $F$ is the desirability function in tensor decomposition\(^1\). The operator $\mathbb{A}$ often has a high separation rank, and therefore, it is compressed to a low rank approximation using the ALS. Equation (18) is then solved using the ALS by minimizing $\|\mathbb{A}F - G\|$ with respect to $F$. The obtained solution $F$ is the approximate solution to the linear HJB equation (5) given as a tensor function, as in Definition 1.

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 [17] which is used as an illustrative example, because we can compute the true solution of the system for comparisons. The dynamics of this system is given by

$$\begin{align*}
dx_1 &= (2(x_1^5 - x_1^3 - x_1 + x_1x_2^4) + x_1u_1) dt + d\omega_1 \\
dx_2 &= (2(x_2^5 - x_2^3 - x_2 + x_2x_1^4) + x_2u_2) dt + d\omega_2
\end{align*}$$ (19)

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 pick $n_g = 101$ discretization points in each dimension and 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.

\(^1\)Observe that the PDE and the boundary conditions in (5) can be combined to the equation (18) because $G = 0$ for interior points.
B. Ill-Conditioning of ALS

When solving for the solution of (3), the matrix \( \mathcal{M} \) in (12) often becomes ill-conditioned (i.e. condition number is above \( 10^{13} \)). A poorly conditioned linear equation will result in an inaccurate estimation of the solution, independent of the algorithm used to find the solution. The accuracy of the solution \( F \) in (12) will be limited by the condition number of the matrix \( \mathcal{M} \) [18]. This effect may prevent the ALS algorithm (Algorithm 1) from iterating further to produce a lower approximation residual, and the overall solution may not achieve a specific residual.

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 \( \Lambda \) tend to have large values because of finite differencing on the edges. 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. A new tensor term that is added by the algorithm is generally smaller than the previous tensor terms in \( F \) because the new tensor term is added to account for the residual of the previous tensor terms. A regularizer is added into the cost function as described in Section II-D to prevent ill-conditioning of the solution \( F \). However, if the regularizer \( \alpha \) is too large, the resulting solution \( F \) will give a large residual \( \| \mathbb{A} F - G \| \). Therefore, limited by a moderate \( \alpha \) 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.
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
\[ G = \mathbf{A} \mathbf{F} = \mathbf{A} \sum_{l=1}^{r_F} \mathbf{F}^l = \sum_{l=1}^{r_F} \mathbf{A} \mathbf{F}^l \iff G - \mathbf{A} \sum_{l=1}^{p-1} \mathbf{A} \mathbf{F}^l = \sum_{l=p}^{r_F} \mathbf{A} \mathbf{F}^l = \mathbf{A} \sum_{l=p}^{r_F} \mathbf{F}^l \]
where \( \mathbf{F}^l \) represents the \( l \)-th tensor term in the tensor function \( \mathbf{F} \). Let \( F_p = \sum_{l=p}^{r_F} \mathbf{F}^l \) and \( G_p = G - \sum_{l=1}^{p-1} \mathbf{A} \mathbf{F}^l \). Then, the previous equation becomes
\[ \mathbf{A} F_p = G_p \] (20)
which is a linear equation of the form (11) that Algorithm 1 can solve. Intuitively, this technique removes the dominant terms in \( \mathbf{F} \) from the equation allowing the algorithm to compute other smaller terms in \( \mathbf{F} \) avoiding ill-conditioning of \( \mathcal{M} \). Hence, if the current computed solution \( \mathbf{F} \) causes \( \mathcal{M} \) to be ill-conditioned, we record the \( \mathbf{F} \) as \( \mathbf{F}_j \), reset the \( \mathcal{G} \) to \( \mathcal{G} - \mathbf{A} \mathbf{F} \), and restart the algorithm using the new \( \mathcal{G} \). As a result, we obtain a sequence of solutions \( \mathbf{F}_j \) in which the sum of the sequence returns the full approximate solution \( \mathbf{F} \).

Fig. 2 shows the performance of the ALS with sequential solutions using the example system (19). The modified ALS realizes a lower residual \( \| \mathbf{A} \mathbf{F} - \mathbf{G} \| \), 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 in the following sense. At every reset of \( \mathbf{F} \), \( \mathbf{G} \) is reset by subtracting \( \mathbf{F} \) from the current \( \mathbf{G} \). Each subtraction reduces the magnitude of \( \mathbf{G} \), and eventually, the magnitude of \( \mathbf{G} \) becomes so small that the magnitude difference between \( \mathbf{A} \) and \( \mathbf{G} \) is on the order of \( 10^{13} \), approximately the MATLAB precision. In this case, the algorithm can no longer compute a solution with higher accuracy.
2) Boundary Condition Rescaling: The linearity of (11) also allows for rescaling of the elements in $\mathcal{M}$ to decrease its condition number. Rescaling reduces the magnitude differences resulting from the operator $A$ in the beginning 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 $\mathcal{M}$. Note that $G$ is rescaled accordingly to ensure that (11) still holds. Note that there can be different rescaling constants for different elements in $A^i_l$. 

Fig. 2. Results of using Algorithm 1 on (19) with and without sequential solutions. The first row shows that the residual $\|AF - G\|$ 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 (19) with and without boundary condition rescaling. The first row shows that the residual $\|A F - G\|$ 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. Boundary condition rescaling yields a lower residual before $M$ becomes ill-conditioned, and thus achieves a more accurate solution. Both runs quit once $M$ becomes ill-conditioned.

Small elements in $A^l$ tend to originate from the boundary conditions, and they are therefore rescaled. To scale the Dirichlet boundary conditions of $\partial \Omega$ in the $k$-th dimension, the first and last diagonal elements in the matrix $C$ in (16) are replaced with scalars $a_{k}^{lo}$ and $a_{k}^{up}$ respectively to rescale the elements in $A$ corresponding to the lower and upper boundary conditions in the $k$-th dimension. Then, $G$ is rescaled accordingly in the $k$-th dimension to ensure that the scaled $A$ and $G$ still satisfy the equation $A F = G$. In the case of Neumann and periodic boundary conditions of $\partial \Omega$, the rescaling method is essentially the same. Elements in $A$ that corresponds to the goal region $\Lambda$ are rescaled by multiplying $I_{A}$ in (17) with a scalar $a_{\Lambda}$. Then, $G$ is again
rescaled accordingly to ensure that the scaled $A$ and $G$ still satisfy the equation $AF = G$.

The performance of this boundary rescaling technique is demonstrated using the example system (19) and the result is shown in Fig. 3. By rescaling the boundary conditions, the modified ALS algorithm can solve to a lower residual $\|AF - G\|$ 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 when computing the solution sequentially together with boundary condition rescaling, the computation time decreases dramatically, and at the same time, the computation produces a more accurate solution. The residuals for both cases stabilize once the

Fig. 4. Results of implementing sequential computation on (19) with and without boundary condition rescaling. The first row shows how the residual $\|AF - G\|$ decreases with each iteration, and the second row plots the error between the computed solution and the true solution. The algorithm terminates when the residuals stop improving.
computations are on the order of MATLAB precision. We pick \( n_g = 151 \) discretization points in each dimension for these two examples to make the distinctions between the two cases clearer. Without operator scaling, the total number of iterations before residual flatting is at about 600, and the total computation time is approximately an hour. On the other hand, with operator scaling, the total number of iterations before residual flatting is at about 130, and the total computation time is approximately a minute. The separation rank of the computed solutions before residual flatting are 17 and 123 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 keep in mind 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 new ALS algorithm, SeALS (Algorithm 2). This algorithm can compute a more accurate solution for the linear HJB equation using a shorter amount of time.

Given a tensor operator \( A \) and a tensor function \( G \) derived from the linear HJB equation, SeALS implements the original ALS algorithm with two additional steps. First, before the iterations to solve (12), elements in \( A \) that correspond to the boundary conditions are rescaled as described in Section III-C2, and \( G \) is rescaled accordingly. Second, when \( M \) is ill-conditioned, the current \( F \) is recorded as \( F_j \). Then, \( G \) is reset to \( G - AF \), and \( F \) is reset to a random rank 1 tensor term. In the end, the algorithm returns \( F \), which is the sum of all previously recorded \( F_j \).

This algorithm is implemented as the SeALS tool in MATLAB [19]. A more detailed description and other additional features of the SeALS tool are available in the documentation for the tool [20].
Algorithm 2 Sequential ALS (SeALS)

**Input:** tensor operator $A$ and tensor function $G$

**Output:** tensor function $F$

1: rescale boundary conditions in $A$ and $G$ accordingly
2: generate random vectors $F^1_i \in \mathbb{R}^{M_i}$ and set $F = F^1_1 \otimes \cdots \otimes F^1_d$
3: set $j = 0$
4: while $\frac{\|AF - G\|}{\sqrt{\prod_{i=1}^d M_i}} > \epsilon$ do
5:   for $k = 1, 2, \ldots, d$ do
6:     solve (12) for the vector $F_k$ 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 $G - AF$ (subtract $F$)
11:   end if
12:   else if no residual decrease then
13:     add a preconditioned random tensor term to $F$
14:   end if
15: end while
16: set $F = \sum_j F_j$

The first column of Fig. 5 shows the solution obtained by SeALS for system (19). The error from the true solution is on the similar order of magnitude, $10^{-3}$, as the uncertainty of the true solution. The solution is therefore considered accurate. The computation was performed with $n_g = 151$ discretization points in each dimension and sixth order differentiation schemes for discretizing the derivatives. The residual tolerance for compressing the operator was set to $10^{-6}$ decreasing the rank from 37 to 6. The first five basis functions $\{F^1_i\}_{i=1}^5$ in each dimension is shown on the second column of Fig. 5. The functions are smooth with peaks at the origin due to the zero-cost. Residual convergence and the normalization constants $s^F_i$ is shown in Fig. 6. The residual stabilizes in the end when the computation reaches the Matlab precision. The algorithm is thus terminated. The rank of the solution before the stagnant residual is 17, and the iteration time is 89 s. Fig. 7 shows six simulations of the system controlled using the controller produced from the SeALS tool. All trajectories reach the origin as expected.
Fig. 5. Computed solution (left) and the basis functions in each dimension (right) for system (19). The uncertainty of the true solution is of magnitude $10^{-3}$, the same magnitude as the error. The obtained solution is therefore considered accurate.

Fig. 6. Error $\|A_F - G\|$ (first row) and the normalization constants $s_l^F$ (second row) for system (19). The algorithm is terminated when the residual stagnants.
V. Examples

The capability of SeALS (Algorithm 2) are demonstrated using three engineering examples, an inverted pendulum, a VTOL aircraft, and a quadcopter. A MacBook Pro with 2.5 GHz i5 processor and 4 GB ram-memory is used in the inverted pendulum and quadcopter examples, and a computer cluster at California Institute of Technology with quadcore 3.0 Ghz i7 processor and 64 GB ram-memory is used in the VTOL aircraft example. All simulations are performed with the indicated noise level.

A. Inverted Pendulum

The dynamics of an inverted pendulum adapted from [21] is given by

\[
\begin{align*}
    dx_1 &= x_2 \, dt + d\omega \\
    dx_2 &= \frac{g}{l} \sin(x_1) - \frac{1}{2} m_r x_2^2 \sin(2x_1) - \frac{m_r}{ml} \cos(x_1) u \\
    &\quad \frac{4}{3} - m_r \cos^2(x_1) \, dt + d\omega
\end{align*}
\]

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 \( q(x) = 0.1 x_1^2 + 0.05 x_2^2 \), \( R = 0.02 \) and choose the domain \( \Omega = \{(x_1, x_2) \in \mathbb{R}^2 \mid |x_1| \leq \pi, |x_2| \leq 11\} \). Periodic boundary conditions
are placed for $x_1$, a high velocity penalty $\phi(x_1, \pm 11) = 10$ is set in $x_2$, and a goal region
$\Lambda = \{(x_1, x_2) \in \mathbb{R}^2 \mid |x_1| \leq 0.18, |x_2| \leq 0.3\}$ is set around the origin with zero cost. The noise
is modeled as $\Sigma_e = 10I$.

We pick $n_g = 201$ discretization points in each dimension and a sixth order difference scheme
for discretizing the derivatives. The operator was compressed from rank 29 to 11 for a relative
residual $10^{-7}$. Maximum rank of the solution was set to 20 for speed. Upon reaching the
maximum rank, the algorithm records the current solution and restarts with a new rank one
tensor (see Algorithm 2 line 8-12). Once the computation completes, the final solution is the
sum of all recorded solutions. The final solution and the first five basis functions $\{F_i\}_{i=1}^5$ in
each dimension are shown in Fig. 8. Residual $\|A F - G\|$ and normalization constants $s_i^F$ are
shown in Fig. 9 as a function of iteration number. The algorithm quits when it exceeds the
prescribed maximum number of iterations 2000 with a total run time of 37 hours. Shorter runs
can be achieved by optimizing the SeALS tool for speed or by quitting earlier when the basis
weights are small enough for the specific application. A simulation is shown in Fig. 10 in which
the pendulum reaches the goal region as desired when using a controller computed from the
solution produced by SeALS. The control impulse increases the angular velocity, enabling the
pendulum to reach the goal region.

![Dimension = 1](image1)

![Dimension = 2](image2)

Fig. 8. Computed solution (left) and the basis functions in each dimension (right) for system (21).
Fig. 9. Residual $\|A^F - G\|$ (left) and the normalization constants $s^F_l$ (right) for system (21). The algorithm quits when it exceeds the prescribed maximum number of iterations 2000.

Fig. 10. A simulation of system (21) 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 control impulse increases the angular velocity, enabling the pendulum to reach the goal region.

B. VTOL Aircraft

The next example is a Vertical Takeoff and Landing (VTOL) Aircraft from [22]. It is a simplified model of an aircraft considered in the translation plane $(x, y)$ with tilt angle $\theta$. The
parameter $x$ is the horizontal position, and $y$ is the vertical position. The dynamics are given by

\[
\begin{align*}
    dx_1 &= x_2 \, dt \\
    dx_2 &= -\sin(x_5)(u_1 \, dt + d\omega_1) + \epsilon \cos(x_5)(u_2 \, dt + d\omega_2) \\
    dx_3 &= x_4 \, dt \\
    dx_4 &= (-g + \cos(x_5))(u_1 \, dt + d\omega_1) + \epsilon \sin(x_5)(u_2 \, dt + d\omega_2) \\
    dx_5 &= x_6 \, dt \\
    dx_6 &= u_2 \, dt + d\omega_2
\end{align*}
\]

where $[x_1, x_2, x_3, x_4, x_5, x_6] = [x, v_x, y, v_y, \theta, v_\theta]$, $g$ is the gravitational constant, and $\epsilon = 0.01$. The domain $\Omega$ 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 for the aircraft is to land at $y = 0$. Therefore, we impose boundary conditions $\Psi|_{\partial \Omega} = 0$ for all non-periodic states except $y = 0$ where we set $\Psi|_{y=0} = \prod_{i=1}^{d}(1 - s_i^2)$, where $s_i$ is the normed coordinate in direction $i$. The goal region $\Lambda$ is set to the origin so that the aircraft reaches $y = 0$ with moderate velocities $v_x$, $v_y$ and $v_\theta$ and deviations in $x$ and $\theta$. The noise is set to be $\Sigma_\epsilon = 3I$.

We pick $n_g = 100$ discretization points in each dimension and second order differentiation schemes for discretizing the derivatives. The operator was compressed from rank 50 to 13 for a relative error $10^{-5}$. Fig. 11 presents the first ten basis functions $\{F_i\}^{10}_{i=1}$ in each dimension. Fig. 12 shows residual of the SeALS tool and the normalization constants $s_i^F$ of the basis functions. Fig. 13 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.
Fig. 11. The first ten basis functions in each dimension for the VTOL aircraft example.

Fig. 12. Residual $\|A_F - G\|$ (left) and normalization constants $s_F^l$ (right) for the VTOL aircraft.
Fig. 13. 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 last example is a model of a quadcopter from [23]. The quadcopter moves in three-dimensional space \((x, y, z)\) with orientation given by the yaw angle \(\psi\), pitch angle \(\theta\) and roll angle \(\phi\). The control inputs are the main thrust \(u\), the yawing moment \(\tau_{\psi}\), the pitching moment
\( \tau_\theta \) and the rolling moment \( \tau_\psi \). The dynamics are given by (1) in which
\[
x_t = \begin{bmatrix} x & y & z & \psi & \theta & \phi & \dot{x} & \dot{y} & \dot{z} & \dot{\psi} & \dot{\theta} & \dot{\phi} \end{bmatrix}^T
\]
\[
u_t = \begin{bmatrix} u_1 & u_2 & u_3 & u_4 \end{bmatrix}^T \triangleq \begin{bmatrix} u & \tau_\psi & \tau_\theta & \tau_\phi \end{bmatrix}^T
\]
\[
f(x_t) = \begin{bmatrix} \dot{x} & \dot{y} & \dot{z} & \dot{\psi} & \dot{\theta} & \dot{\phi} & 0 & 0 & -g & 0 & 0 & 0 \end{bmatrix}^T
\]
\[
B(x_t) = \begin{bmatrix} 0_{6x1} & 0_{6x3} \\
\sin \phi \sin \psi + \cos \phi \cos \psi \sin \theta & \cos \phi \sin \theta \sin \psi - \cos \psi \sin \theta & 0_{3x3} \\
\cos \theta \cos \phi & \cos \theta \cos \phi & I_{3x3} \\
0_{3x1} & 0_{3x3} \\
\end{bmatrix}
\]

where \( 0_{m\times n} \) represents a zero matrix of size \( m \) by \( n \), and \( I_{m\times n} \) represents an identity matrix of size \( m \) by \( n \). The domain \( \Omega \) is given by \( x, y, z \in [-1, 1], \) periodic \( \psi, \theta, \phi \in [-\pi, \pi], \) \( \dot{x}, \dot{y}, \dot{z} \in [-8, 8] \) and \( \dot{\psi}, \dot{\theta}, \dot{\phi} \in [-10\pi, 10\pi] \). We set \( q(x_t) = 2 \) and \( R = 2I \). The goal is to reach \( x = 1 \). Hence, 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 goal region \( \Lambda \) is set to the point \( x = 1 \) and zero for the remaining coordinates so that the quadcopter can reach \( x = 1 \) with moderate deviations in the other coordinates. The noise is set as \( \Sigma_\epsilon = 100I \).

We pick \( n_g = 100 \) discretization points in each dimension and a second order difference scheme for discretizing the derivatives. The operator was compressed from rank 270 to 25 with a relative error \( 10^{-5} \). Fig. 14 shows a sample simulation of the quadcopter controlled using the solution produced by SeALS. The trajectory reaches \( x = 1 \) as desired. The corresponding control signals are shown in Fig. 15.
Fig. 14. A simulation of the quadcopter controlled using the solution produced by the SeALS. The quadcopter reaches $x = 1$ as desired.
VI. Conclusion and Future Work

This paper presents a method to solve the high dimensional linear HJB equation for the first exit problems of stochastic affine nonlinear dynamical systems. This work is a significant improvement from the previous work [11] where the concept of tensor decomposition and ALS are considered for solving the linear HJB equation arising from a large class of dynamical systems. The ill-conditioning issue in the original framework 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 the performance of SeALS. 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. Furthermore, other types of numerical technique that improves the algorithm’s stability are currently under investigation including adding artificial diffusion [24]. Different kinds of differentiation schemes such as Chebyshev differentiation [25] and upwind finite difference scheme [26] are also explored. Future iterations of the SeALS tool will include time varying linear HJB equation.

More generally, this tool can be used as a method to provide approximate basis functions for the sums of squares (SOS) based technique [3] to solve for optimal controller in high dimensions. The number of monomials in the SOS technique grows exponentially with dimension. However, the SOS technique provides a performance guarantees that SeALS does not provide. To benefit from both, SeALS can provide the SOS technique with a smaller set of monomials, and the SOS technique can compute for a suboptimal controller with performance guarantees. Lastly, apart from solving the HJB equation, this tool may be adapted to solve general linear PDEs in high dimensions.

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


