Numerical Verification of Affine Systems with up to a Billion Dimensions

Stanley Bak
Safe Sky Analytics
Manlius, NY
stanleybak@gmail.com

Hoang-Dung Tran
Vanderbilt University
Nashville, TN
dung.h.tran@vanderbilt.edu

Taylor T. Johnson
Vanderbilt University
Nashville, TN
taylor.johnson@vanderbilt.edu

ABSTRACT

Affine systems reachability is the basis of many verification methods. With further computation, methods exist to reason about richer models that have inputs, nonlinear differential equations, and hybrid dynamics. As such, the scalability of affine systems verification is a prerequisite to the scalability of analysis methods for more complex systems. In this paper, we strive to improve the scalability of affine systems verification, in terms of the number of dimensions (variables) in the system.

One benefit of affine systems is that their reachable states can be written in terms of the matrix exponential, and safety checking can be performed at specific time steps with linear programming. Unfortunately, for large systems with many state variables, this direct approach requires an intractable amount of memory while using an intractable amount of computation time. We overcome these two problems by combining several methods that leverage common problem structure. Memory demands can be reduced by taking advantage of both initial states that are not full-dimensional, and safety properties (outputs) that only need a few linear projections of the state variables. Computation time is saved by using numerical simulations to compute only projections of the matrix exponential relevant for the verification problem. Since large systems often have sparse dynamics, we use fast Krylov-subspace simulation methods based on the Arnoldi or Lanczos iterations. Our implementation produces accurate counter-examples when properties are violated and, in the extreme case with sufficient problem structure, is shown to analyze a system with one billion real-valued state variables.

1 INTRODUCTION

An affine system is modeled with the ordinary differential equation $\dot{x} = Ax + b$, where x is a vector of n state variables, A is the $n \times n$

dynamics matrix, and b is an $n \times 1$ vector of constant forcing terms. Given a set of initial states, a set of unsafe states, and a time bound, the time-bounded safety verification problem is to check if there exists an initial state and a time within the bound such that the solution of the affine system enters the unsafe set.

One way to solve the verification problem is to construct the reachable set of states. The reachable set contains all states that lie along any solution to the differential equation $\dot{x} = Ax + b$, starting from any initial state up to the time bound. If the reachable set does not intersect with the unsafe states, then the system is verified as safe. In the discrete-time setting, we construct the reachable set at each time instant, and then perform the unsafe check once per step using linear programming (LP).

This discrete-time approach forms the core of many verification methods for richer classes of systems. For example, when systems have time-varying inputs, an additional input-effects term can be computed at each step and added to the discrete-time reachable set using a Minkowski sum operation [11, 29]. Overapproximation of the continuous-time reachable set is possible by noting that, in the finite time between time steps, the system can only go a bounded distance from the discrete-time solution. Based on this observation, methods exist that perform bloating from the discrete-time solution in order to guarantee an overapproximation of the continuous-time reachable set [26, 38, 39]. The reachable set for nonlinear dynamical systems can also be overapproximated with techniques based on affine methods, by linearizing the nonlinear dynamics and then adding uncertain terms to account for mismatch between the linear and nonlinear systems. In hybridization methods [2, 8, 18], this process is repeated in multiple domains to reduce the overapproximation error. Finally, methods to verify hybrid systems that combine continuous dynamics and discrete behaviors, such as a physical system controlled by software, also build upon the core operations needed to analyze affine systems [3, 41], in conjunction with additional techniques to handle combinatorial aspects. All of these powerful methods build on the core computations used for affine systems reachability. In this paper, we focus on the scalability of this fundamental computation.

Verification approaches for systems that have real numbers can be categorized into *validated* methods and *numerical* methods. Validated methods, such as interval analysis [46], maintain guaranteed bounds on values used throughout the computation. Numerical methods, on the other hand, accept using finite-precision floating-point numbers and algorithms that perform operations up to any user-desired accuracy, such as finite series expansions to compute a matrix exponential. Although desirable, validated methods are typically slower and often fail to work on large systems to due the accumulation of overapproximation error. In this work, we focus
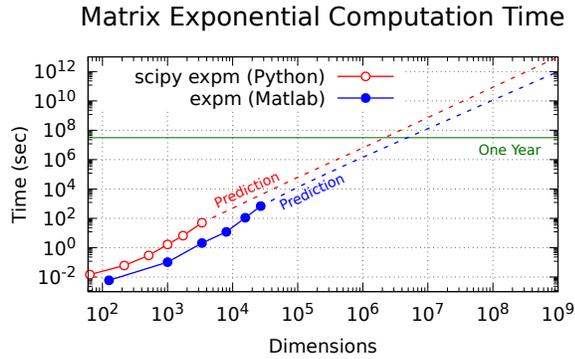
Matrix Exponential Computation Time

Figure 1: Ignoring memory issues, off-the-shelf methods require intractable computation time to compute a single high-dimensional matrix exponential for the 3D Heat Diffusion system used in our evaluation.

on numerical verification methods, as the scale of systems we want to analyze would make current validated approaches infeasible.

As time advances, the behaviors of affine systems can be written in terms of the matrix exponential, which can be used to compute the reachable set. For high-dimensional systems, however, computing the matrix exponential becomes both the runtime and memory bottleneck. A simple experiment demonstrating the runtime problem is shown in Figure 1, where extrapolation predicts it would take over a year to compute a single matrix exponential once a system has over ten million dimensions. A second problem is memory: although the A matrix for large systems can often be sparse, the matrix exponential of A will be dense. The amount of memory needed to simply store the matrix exponential result, a dense $n \times n$ matrix, can greatly exceed what is available on modern computers, even if its computation time was instant. For a million-dimensional system, this matrix would have 10^{12} numbers and need about 8 TB of main memory.

The main contribution of this paper is an approach for affine systems verification that can, with sufficient problem structure, scale to extremely high-dimensional systems, thousands to millions of times larger than existing methods. We overcome the memory and computation time problems through an original *combination* of new and existing techniques. The memory improvements are possible through a method that uses both aspects of reachability with support functions [39] (projecting the reachable set onto an output space) and affine representations [36] / zonotopes [28] (low-dimensional initial spaces). The run-time improvements use simulations to compute parts of the matrix exponential [21], possibly optimized using a new approach that simulates using the transpose system dynamics A^T. Since large dynamics matrices are often sparse (and must be sparse to simply fit into memory), we can perform numerical simulations using efficient Krylov subspace methods [27, 36]. We use a recently-published a posteriori error bound [51] to determine when the dimension of the Krylov subspace is sufficient for an accurate simulation result. Further, we propose a modification to the Lanczos iteration (used by Krylov subspace methods) that reduces memory requirements, allowing us to

significantly increase the number of iterations before memory is exhausted. Although some of the techniques have been used individually before, no existing method for affine systems has demonstrated scalability beyond a few thousand dimensions. In our evaluation, the largest system we analyze has one billion dimensions.

The research presented here builds off a previous workshop paper [7], with several new developments: (i) this paper focuses on scalability in high dimensions (up to 10^9 dimensions), whereas the earlier work performed a runtime improvement evaluation on comparatively smaller systems (up to 10^4 dimensions); (ii) Section 4 provides a detailed description and pseudo-code for the modified Krylov methods, which use an a posteriori error bound (Lemma 4.1), as opposed to the relative error estimate in the earlier paper; (iii) we provide memory improvements to the projected Lanczos iteration in Section 4.3 which, in our evaluation in Section 5.3, is shown to increase scalability by two orders of magnitude.

Section 2 first reviews affine discrete-time safety verification, which uses an $n \times n$ matrix exponential at each time step in the analysis. Next, Section 3 presents memory improvements followed by Section 4, which focuses on reducing computation time. An evaluation on several large benchmarks, including a 3D Heat Diffusion system with one billion dimensions, is given in Section 5, followed by a review of related work and a conclusion.

2 AFFINE VERIFICATION REVIEW

An affine, discrete-time, bounded safety verification problem is defined by the system dynamics $\dot{x} = Ax + b$, a set of initial states $\mathcal{I}$ defined as all states x_0 where the linear constraints $\mathcal{I}_x x_0 \leq \iota_x$ hold, unsafe states $\mathcal{U}$ defined with linear constraints $\mathcal{U}_x x \leq \upsilon_x$, a step size δ and time bound T. The system is called unsafe if and only if there exists a time $t = k\delta \leq T$ such that $x_0 \in \mathcal{I}$, $x = e^{At}x_0$, and $x \in \mathcal{U}$. The goal is to prove a system is safe or find a counter-example, which can be defined by an initial state x_0 and time t.

2.1 Basic Verification Approach

An affine system with dynamics $\dot{x} = Ax + b$ can be verified by first converting it to a linear system (without the b term), by adding a fresh variable to account for the effects of the forcing term b. The new A matrix has an extra column consisting of the entries of the b vector, and an extra row of all zeros. The initial value of the new variable is assigned to 1, and, since the row in A defining its differential equation is all zeros, the new variable's value remains at 1 at all times. In this way, the effect of the extra column in the A matrix is the same as the b vector in the original system. In the remainder of this paper, we consider linear systems after this transformation, assuming the form $\dot{x} = Ax$.

Safety can be checked by constructing a LP at each discrete time t that contains two copies of the state variables, x_0 and x, and encodes the initial state conditions $\mathcal{I}_x x_0 \leq \iota_x$, the unsafe state conditions $\mathcal{U}_x x \leq \upsilon_x$, and the linear relationship (for a fixed t) between the initial and final variables $x = e^{At}x_0$. If the LP is feasible, the solver provides an assignment to the variables that can be used to construct the counter-example. The bulk of the computation time is spent on these two operations: (i) computing e^{At} and (ii) solving the LP.

2.2 Timed Harmonic Oscillator Example

We will use an example of a timed harmonic oscillator to demonstrate the methods in this paper. The timed harmonic oscillator is
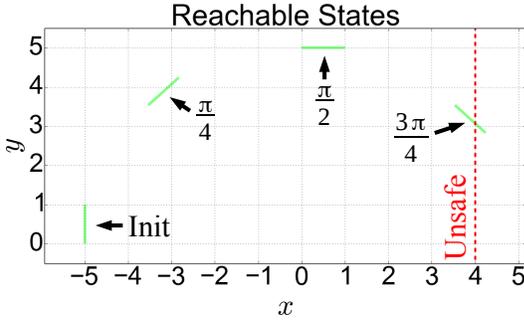
Figure 2: The timed harmonic oscillator system can reach an unsafe state at time $\frac{3\pi}{4}$.

a system with dynamics $\dot{x} = y$, $\dot{y} = -x$, and $\dot{t} = 1$. For the initial set of states, take $x_0 = -5$, $y_0 \in [0, 1]$, and $t_0 = 0$. The unsafe set of states consists of all states where $x = 4$. We attempt to verify the system with a discrete time step of $\delta = \frac{\pi}{4}$ and a time bound of $T = \pi$.

On the x-y plane, solutions of the system rotate clockwise around the origin. The reachable set is shown in Figure 2. From the figure, it is apparent that at time $\frac{3\pi}{4}$, the unsafe states are reachable.

We can show this computationally and find the initial state that leads to the violation. First, we convert the three-variable affine system (the t variable has an affine term), to a four-variable linear system using the affine-to-linear transformation described before. The variables in the transformed system are $\vec{x} = (x, y, t, a)^T$, where a is the newly-introduced variable, which is initially 1 and remains constant at all times. The transformed dynamics now form the four-dimensional linear system $\dot{\vec{x}} = A\vec{x}$, with

$$A = \begin{pmatrix} 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}.$$

Next, at each discrete time step, we construct a set of linear constraints. The constraints have two copies of the variables, one set at the initial time $\{x_0, y_0, t_0, a_0\}$, and one set at the current time step $\{x, y, t, a\}$. The linear constraints at the time of the violation, $\frac{3\pi}{4}$, are shown in Figure 3a. The only constraints that change between time steps are the ones encoding the matrix exponential at the current time (the values surrounded by a red rounded rectangle), for which we reuse terminology [9] and refer to as the *basis matrix*.

The linear constraints are then passed to an LP solver to check if they are feasible. For time steps 0, $\frac{\pi}{4}$, and $\frac{\pi}{2}$, the LP solver returns that no solution exists. At time $\frac{3\pi}{4}$, which has the constraints shown in the figure, the LP solver finds a feasible solution, and provides an assignment to the variables. In particular, its output indicates that starting from initial state $(x_0, y_0, t_0, a_0)^T = (-5, 0.66, 0, 1)^T$, the system can reach the unsafe state $(x, y, t, a)^T = (4, 3.07, 2.36, 1)^T$.

3 MEMORY IMPROVEMENTS

Although the basic verification approach works, it does not scale to very high dimensions. As mentioned in the introduction, computing and storing the basis matrix (the matrix exponential) is typically the bottleneck to verification scalability. In this section we focus

on the memory problem, and show how we can reduce the height (Section 3.1) and width (Section 3.2) of the basis matrix, by taking advantage of common problem structure.

3.1 Projecting onto the Output Space

First, we reduce the *height* of the basis matrix (compare the basis matrices in Figure 3a and Figure 3b). This is done by a method similar to the use of support functions with a fixed number of directions for reachability analysis [39]. The common problem structure exploited is that the verification result often only depends on a small number of directions, much smaller than the number of system variables.

Depending on the type of problem being solved (linear verification, plotting, or hybrid automaton reachability), these directions arise from different sources. For a safety verification problem for linear systems, these directions come from each of the constraints in the conjunction defining the unsafe states. For a plot, we only need to compute a projection onto the two or three plot dimensions. In this case, the important directions are the unit vectors in each of these dimensions. Plots can then be produced efficiently by running multiple optimizations over projections of the convex reachable set at each time step [32, 40]. For the hybrid automaton setting, additional directions can come from the constraints in the mode invariants, as well as from the guard conditions.

We can combine these directions into an output matrix C, where the output variables are $y = Cx$, and the height of the matrix is the number of output directions, o, needed for the current problem. The unsafe states, $\mathcal{U}$, are then redefined in the output space, $\mathcal{U}_y y \leq v_y$. Finally, the basis matrix in the constraints is the $o \times n$ projection of the matrix exponential onto the output space, Ce^{At}.

Consider applying this approach to the timed harmonic oscillator system of Section 2.2, where the unsafe states are defined by $x = 4$. The other three dimensions, y, t, and a, do not impact the result of the safety check, and so they (and their corresponding constraints) can be removed from the set of linear constraints, as is done in Figure 3b. In this case, the output matrix for this system is the 1×4 matrix $C = (\begin{smallmatrix} 1 & 0 & 0 & 0 \end{smallmatrix})$. We then define the unsafe states in terms of the single output space variable o_x, and replace the basis matrix by the projected matrix exponential Ce^{At}.

3.2 Projecting from the Initial Space

Next, we reduce the *width* of the basis matrix (compare the basis matrix in Figure 3b and Figure 3c). This is done with a method similar to reachability using affine representations [36], and is also similar to reachability with zonotopes [28] with a small number of generators. The common problem structure exploited is that the initial states are often low-dimensional. For example, there may not be uncertainty in every variable, or the initial states of variables may be related.

In this case, we can define an i-dimensional initial space using an $n \times i$ matrix E, where the initial states z are related to the original variables by $x = Ez$. The initial states $\mathcal{I}$ are then redefined with constraints in the initial space, $\mathcal{I}_z z \leq \iota_z$. The $o \times i$ basis matrix is now computed using both projections, $Ce^{At}E$.

In the timed harmonic oscillator system of Section 2.2, we can define the initial states using $i = 2$ dimensions. These are i_y, which

Basis Matrix

$$\left(\begin{array}{cccc|cccc}
-1 & 0 & 0 & 0 & -0.707 & 0.707 & 0 & 0 \\
0 & -1 & 0 & 0 & -0.707 & -0.707 & 0 & 0 \\
0 & 0 & -1 & 0 & 0 & 0 & 1 & 2.36 \\
0 & 0 & 0 & -1 & 0 & 0 & 0 & 1 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & & & 0 & 1 & 0 & 0 & 0 \\
0 & \text{Unsafe} & & 0 & 0 & -1 & 0 & 0 \\
0 & \text{Condition} & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{array}\right)
\left(\begin{array}{c}
x \\ y \\ t \\ a \\ x_0 \\ y_0 \\ t_0 \\ a_0
\end{array}\right)
\begin{array}{c}
= 0 \\ = 0 \\ = 0 \\ = 0 \\ = 4 \\ = -5 \\ \le 0 \\ \le 1 \\ = 0 \\ = 1
\end{array}$$

Initial State Conditions

(a) Basic approach with full linear constraints (Section 2.1)

Basis Matrix

$$\left(\begin{array}{c|cccc}
-1 & -0.707 & 0.707 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 \\
0 & 0 & -1 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 1
\end{array}\right)
\left(\begin{array}{c}
o_x \\ x_0 \\ y_0 \\ t_0 \\ a_0
\end{array}\right)
\begin{array}{c}
= 0 \\ = 4 \\ = -5 \\ \le 0 \\ \le 1 \\ = 0 \\ = 1
\end{array}$$

Initial State Conditions

(b) Projecting onto the output space (Section 3.1)

Basis Matrix

$$\left(\begin{array}{c|cc}
-1 & 0.707 & 3.54 \\
1 & 0 & 0 \\
0 & -1 & 0 \\
0 & 1 & 0 \\
0 & 0 & 1
\end{array}\right)
\left(\begin{array}{c}
o_x \\ i_y \\ i_f
\end{array}\right)
\begin{array}{c}
= 0 \\ = 4 \\ \le 0 \\ \le 1 \\ = 1
\end{array}$$

Initial State Conditions

(c) Projecting from the initial space onto the output space (Section 3.2)
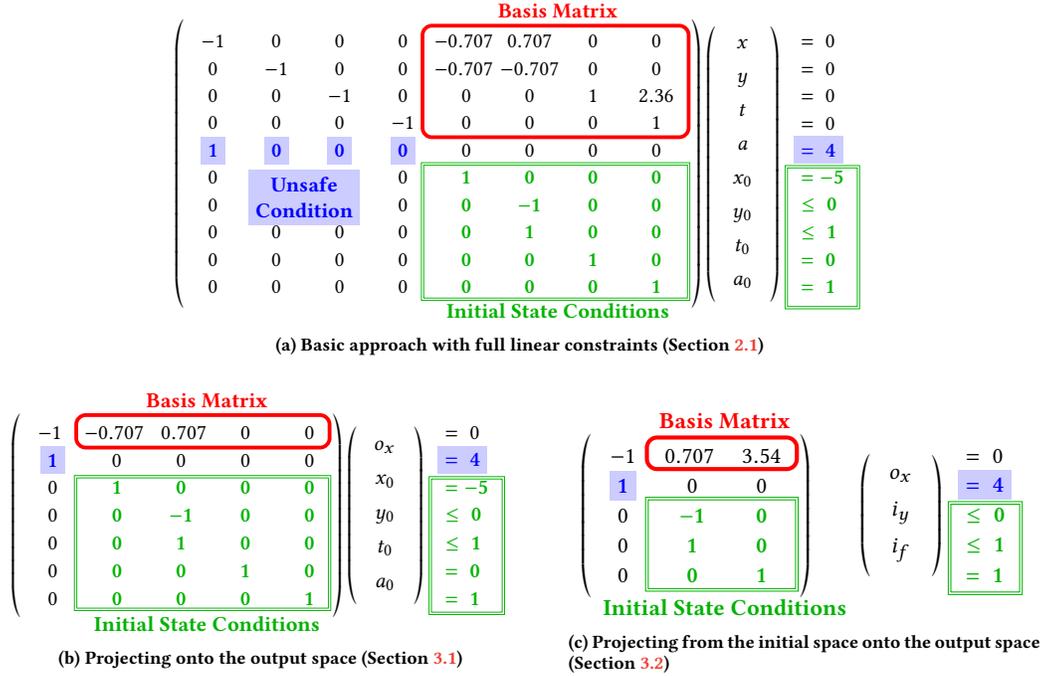
Figure 3: The linear constraints at time $\frac{3\pi}{4}$ for the timed harmonic oscillator example described in Section 2.2 can be encoded in different ways. At each time step, only the basis matrix changes in the constraints.

corresponds to the initial y value, and i_f which is the fixed initial values of all the other variables. The E matrix is the 4×2 matrix $\left(\begin{smallmatrix} 0 & 1 & 0 & 0 \\ -5 & 0 & 0 & 1 \end{smallmatrix}\right)^T$, and the initial constraints are $0 \le i_y \le 1$ and $i_f = 1$. The basis matrix is the product $Ce^{At}E$ at each step.

Using both methods, we have reduced the basis matrix from an $n \times n$ matrix to a $o \times i$ matrix. Importantly, we do not need both o and i to be very small for this reduction to be useful, only their *product*. Given, say, 800 MB to store the basis matrix (10^8 double-precision numbers), the original approach would fill the memory when $n = 10^4$, a ten-thousand dimensional system. In contrast, a million-dimensional system with every dimension initially independent and uncertain, $i = 10^6$, could still be analyzed as long as the unsafe states are defined using less than 100 output directions.

4 COMPUTATION TIME IMPROVEMENTS

Although we can define the smaller basis matrix using $Ce^{At}E$, this does not help in terms of computation time if we use the direct approach of computing e^{At} at each step and then multiplying by C and E. In this section, we describe a series of improvements targeting the computational efficiency of the method.

4.1 Basis Matrix using Numerical Simulations

There are many ways to compute the matrix exponential [43]. Generally, the methods implemented in off-the-shelf libraries use a combination of squaring and scaling and Pade approximation (methods 2 and 3 [43]), which compute the entire matrix at once.

Here, we instead use an alternative method to compute the matrix exponential, using a series of numerical simulations (method 5 [43]). The matrix exponential is computed one column at time by using the fact that $e^{At} = e^{At} \mathbf{I}_{n \times n} = e^{At} (\mathbf{e}_1|\mathbf{e}_2| \ldots |\mathbf{e}_n)$. The jth column of e^{At} is equal to $e^{At}\mathbf{e}_j$, where $\mathbf{e}_j$ is the jth column of the identity matrix. The value of $e^{At}\mathbf{e}_j$, however, is just the solution of the linear system $\dot{x} = Ax$ at time t from initial state $x(0) = \mathbf{e}_j$. To compute this, we can perform a numerical simulation with an off-the-shelf numerical method such as Runge-Kutta. This process is repeated for each column of the identity matrix to compute the full matrix exponential. For the verification problem, we need the value of e^{At} at multiple time steps, and so we run the numerical simulations up to the time bound T, recording the value at each multiple of the step size δ. The values from each column are then combined at each multiple of time step to form the basis matrix in the LP [21].

We propose to adapt this method to take advantage of initial and output spaces. Since we need to compute the basis matrix $Ce^{At}E$, rather than using each column of the identity matrix, we can instead compute simulations from each column of the E matrix, and then project the state in the simulation using the C matrix. We compute $e^{At}E$ by noting that $e^{At} E = e^{At} (E_{*,1}|E_{*,2}| \ldots |E_{*,i})$, where $E_{*,j}$ is the jth column of E. As before with the identity matrix, each column j can be computed separately with a numerical simulation of the linear system $\dot{x} = Ax$ at time t from initial state $x(0) = E_{*,j}$. There are i columns in E, corresponding to the i dimensions of the initial states. If i is much smaller than n, this approach will be significantly faster than computing the full matrix exponential and then doing the multiplication with E. For the timed-harmonic

---

Algorithm 1 Original Arnoldi algorithm

Input: normalized init $n \times 1$ vector v, $n \times n$ matrix A, iterations k
Output: $n \times k$ matrix V, $k \times k$ matrix H

1:   $V_{*,1} \leftarrow v$ ▷ assign to first column of V
2:   **for** i **from** 2 **to** $k + 1$ **do**
3:      $V_{*,i} \leftarrow A\,V_{*,i-1}$
4:      **for** j **from** 1 **to** i **do**
5:         $H_{j,i-1} \leftarrow (V_{*,j})^T V_{*,i}$
6:         $V_{*,i} \leftarrow V_{*,i} - H_{j,i-1}V_{*,j}$
7:      $H_{i,i-1} \leftarrow ||V_{*,i}||$
8:      $V_{*,i} \leftarrow \frac{V_{*,i}}{H_{i,i-1}}$
9:   $H \leftarrow H_{1:k,*}$ ▷ discard extra row of H
10: $V \leftarrow V_{*,1:k}$ ▷ discard extra column of V

---

oscillator system constraints in Figure 3c, for example, since the dimension of the initial space $i = 2$, the basis matrix could be computed in this fashion using two numerical simulations.

If the initial state dimension is large, the computation may still require a large number of simulations. In this work, we propose a new approach that can reduce the required number of simulations if the output space is small. The method works by performing simulations using the transpose system dynamics. Since $Ce^{At}E = ((Ce^{At}E)^T)^T = (E^T(e^{At})^T C^T)^T = (E^T e^{A^T t} C^T)^T$, the basis matrix can also be computed by performing o simulations (one for each column of C^T). We compute $e^{A^T t} C^T$ by noting that $e^{A^T t} C^T = e^{A^T t} (C^T_{*,1}|C^T_{*,2}|\ldots|C^T_{*,o})$, and performing a numerical simulation of the linear system $\dot{x} = A^T x$ up to time t from initial state $x(0) = C^T_{*,j}$, for each column j of C^T. The results are then multiplied by E^T, and transposed to recover the basis matrix. This allows us to compute values of the basis matrix one *row* at a time, and so we can compute the basis matrix using only o numerical simulations. In practice, only one of these is necessary, and so we can choose the minimum of i and o and perform that many numerical simulations, rather than computing an $n \times n$ matrix exponential.

In the timed-harmonic oscillator system, for example, since $o = 1$, the entire basis matrix at each step can be computed with a single numerical simulation. Starting from the state corresponding to the single output direction $(1, 0, 0, 0)^T$, we can simulate using the transpose dynamics A^T up to time $\frac{3\pi}{4}$ to get the state $(-0.707, 0.707, 0, 0)$. This is then projecting with $E^T = \begin{pmatrix} 0 & 1 & 0 & 0 \\ -5 & 0 & 0 & 1 \end{pmatrix}$ to get $(0.707, 3.54)^T$, which is transposed to get the basis matrix in Figure 3c.

4.2 Simulations using the Krylov Subspace

When the system matrix A is high-dimensional, it is also often sparse (in fact, if A has more than tens of thousands of dimensions and can fit in memory, it must be sparse or otherwise compressed). We can exploit this structure to speed up numerical simulations.

The Krylov subspace simulation method [27] computes an approximation of $e^A v$, where v is some initial state. This is done by finding the element of the k-dimensional Krylov subspace $K_k \equiv span\{v, Av, \ldots, A^{k-1}v\}$ that best approximates $e^A v$. Intuitively, the k-dimensional Krylov subspace can exactly represent the first k terms of the Taylor expansion of $e^A v$, making it a good candidate

---

Algorithm 2 Arnoldi algorithm with a posteriori error control

---

Input: normalized init $n \times 1$ vector v, $n \times n$ matrix A, error target ϵ
Output: $n \times k$ matrix V, $k \times k$ matrix H

1:   $V_{*,1} \leftarrow v$
2:   $k \leftarrow 4$
3:   **for** i **from** 2 **to** ∞ **do**
4:      $V_{*,i} \leftarrow A\,V_{*,i-1}$
5:      **for** j **from** 1 **to** i **do**
6:         $H_{j,i-1} \leftarrow (V_{*,j})^T V_{*,i}$
7:         $V_{*,i} \leftarrow V_{*,i} - H_{j,i-1}V_{*,j}$
8:      $H_{i,i-1} \leftarrow ||V_{*,i}||$
9:      $V_{*,i} \leftarrow \frac{V_{*,i}}{H_{i,i-1}}$
10:     **if** $i = k$ **then**   ▷ check error upon reaching k iterations
11:        **if** compute-error$(A, H_{1:i-1,*}) < \epsilon$ **then**
12:          **break**
13:        **else**
14:          $k \leftarrow$ ceil$(1.1 * k)$
15: $H \leftarrow H_{1:k,*}$ ▷ discard extra row of H
16: $V \leftarrow V_{*,1:k}$ ▷ discard extra column of V

---

for accurate approximation. We do not review the full theory here, but instead focus on computational aspects as they relate to the verification problem.

Note, however, that each simulation has a different initial state, and different initial states v will have different Krylov subspaces. This is important, since it means we are **not** proposing to verify the system through an abstraction of the dynamics matrix A by a single lower-dimensional system matrix.

The approximation uses a fixed number of iterations of the well-known Arnoldi algorithm [6, 50]. The pseudocode is shown in Algorithm 1. The Arnoldi algorithm computes an orthonormal basis for the Krylov subspace K_k by starting with a normalized version of v as both the first orthonormal direction and the current vector and, at each iteration, (1) multiplying the current vector by A (line 3), (2) projecting out the previous orthonormal directions from the current vector (the loop on lines 4-6), (3) normalizing the current vector (lines 7-8), and (4) adding it to the list of orthonormal directions (line 8). If the norm computed on line 7 is ever zero, the loop can terminate early (not shown) and the approximation will be exact. The memory needed for the Arnoldi iteration, which can be obtained by looking at the sizes of the outputs, is thus:

$$k \times (n + k) \times \texttt{sizeof(double)} \tag{1}$$

After k iterations complete, the outputs are two matrices V and H, which we refer to as V_k and H_k. V_k is the $n \times k$ matrix of orthonormal basis vectors and H_k is the $k \times k$ matrix that is a projection of the linear transformation A in the Krylov subspace K_k.

The outputs of the Arnoldi algorithm can be used to approximate $e^A v$. This is done by projecting the initial n-dimensional state onto the smaller, k-dimensional Krylov subspace, computing the matrix exponential using the projected linear transformation H_k, and then projecting the result back to the original n-dimensional space using V_k. By the design of the Krylov subspace, the projection of the initial state v is just the length of v multiplied by the first unit vector in the subspace, $\mathbf{e}_1$. Further, since for any time t, the

---

Algorithm 3 Original Lanczos algorithm

Input: normalized init $n \times 1$ vector v, $n \times n$ matrix A, iterations k
Output: $n \times k$ matrix V, $k \times k$ matrix H
1: $V_{*,1} \leftarrow v$
2: **for** i from 2 **to** $k + 1$ **do**
3:      $V_{*,i} \leftarrow A V_{*,i-1}$
4:      **if** i > 2 **then**
5:          $H_{i-2,i-1} \leftarrow H_{i-1,i-2}$
6:          $V_{*,i} \leftarrow V_{*,i} - H_{i-2,i-1}V_{*,i-2}$
7:      $H_{i-1,i-1} \leftarrow (V_{*,i-1})^T V_{*,i}$
8:      $V_{*,i} \leftarrow V_{*,i} - H_{i-1,i-1}V_{*,i-1}$
9:      $H_{i,i-1} \leftarrow ||V_{*,i}||$
10:     $V_{*,i} \leftarrow \frac{V_{*,i}}{H_{i,i-1}}$
11: $H \leftarrow H_{1:k,*}$ ▷ discard extra row of H
12: $V \leftarrow V_{*,1:k}$ ▷ discard extra column of V

Krylov subspaces associated with A and At are identical (because $span\{v, Av, \ldots, A^{k-1}v\}$ is the same as $span\{tv, tAv, \ldots, tA^{k-1}v\}$), we can use the same V_k and H_k to compute the approximation at any point in time. The formula for the approximation is:

$$e^{At}v \approx ||v|| V_k e^{H_k t} \mathbf{e}_1 \qquad (2)$$

Equation 2 is especially useful when the size of A is huge, e.g., millions of dimensions, since it transforms the computation with a large matrix A to a problem with a much smaller matrix H_k. For fast computation, we would like to minimize the size of H_k by using a small number of Arnoldi iterations k, but this has the effect of reducing the approximation accuracy. Thus, it is critical to select k large enough to be accurate, but small enough to be fast.

Earlier work on reachability with Krylov subspace methods [36] used an a priori error bound [27] to determine k. The error of the approximation for a fixed k is bounded by

$$||v|| \frac{||At||^k e^{||At||}}{k!}. \qquad (3)$$

Unfortunately, the a priori error bound can often be unusably pessimistic. For example, one of the models we will use in our evaluation is a 100x100x100 3D Heat Diffusion system (one million dimensions). At time 50, this system has matrix norm $||At|| = 32771611$. For an initial unit vector with $||v|| = 1$, even using a full dimensional Krylov subspace ($k = 10^6$), the computed a priori error bound from Equation 3 is effectively unusable, $10^{16182319}$.

In this work, we instead use a recently-developed a posteriori error bound [51], which uses information from the H matrix as well as the extreme eigenvalues of A to compute a bound on the error. The bound works by creating an error function using the log norm of A and looking at the derivative of this error over time.

LEMMA 4.1 ([51]). *Let* $A \in \mathbb{R}^{n \times n}$ *and* $v \in \mathbb{R}^n$ *with* $||v|| = 1$. *Let* V_k *be the orthogonal matrix and* H_k *be the upper Hessenberg matrix generated by the Arnoldi process for A and v. Let* $\omega_k(\tau) = V_k e^{-\tau H_k} e_1$ *be the Arnoldi approximation to* $\omega(\tau) = e^{-\tau A}v$. *Then the approximation error satisfies*

$$||\omega(\tau) - \omega_k(\tau)|| \le h_{k+1,k} e^{-min\{v(A),0\}\tau} \int_0^\tau |h(t)|dt, \qquad (4)$$

Algorithm 4 Lanczos algorithm with projection and error control

---

Input: normalized init $n \times 1$ vector v, $n \times n$ matrix A, $o \times n$ projection matrix C, error target ϵ
Output: $o \times k$ projected output matrix $\mathcal{P} = CV$, $k \times k$ matrix H
1: $V_{*,1} \leftarrow v$
2: $\mathcal{P}_{*,1} = CV_{*,1}$
3: $k \leftarrow 4$
4: **for** i from 2 **to** ∞ **do**
5:      $V_{*,i} \leftarrow A V_{*,i-1}$
6:      **if** i > 2 **then**
7:          $H_{i-2,i-1} \leftarrow H_{i-1,i-2}$
8:          $V_{*,i} \leftarrow V_{*,i} - H_{i-2,i-1}V_{*,i-2}$
9:          free-memory($V_{*,i-2}$)
10:     $H_{i-1,i-1} \leftarrow (V_{*,i-1})^T V_{*,i}$
11:     $V_{*,i} \leftarrow V_{*,i} - H_{i-1,i-1}V_{*,i-1}$
12:     $H_{i,i-1} \leftarrow ||V_{*,i}||$
13:     $V_{*,i} \leftarrow \frac{V_{*,i}}{H_{i,i-1}}$
14:     $\mathcal{P}_{*,i} = CV_{*,i}$
15:     **if** $i = k$ **then**   ▷ check error upon reaching k iterations
16:         **if** compute-error($A, H_{1:i-1,*}$) < ϵ **then**
17:             **break**
18:         **else**
19:             $k \leftarrow \text{ceil}(1.1 * k)$
20: $H \leftarrow H_{1:k,*}$ ▷ discard extra row of H
21: $\mathcal{P} \leftarrow \mathcal{P}_{*,1:k}$ ▷ discard extra column of $\mathcal{P}$

---

where $h(t) := e_k^T e^{-tH_k} e_1$ *is the* $(k, 1)$ *entry of the matrix* e^{-tH_k} *and* $v(A)$ *is the smallest eigenvalues of* $(\frac{A+A^T}{2})$.

The above lemma computes the error bound of approximating $\omega(\tau) = e^{-\tau A}v$ with the Arnoldi algorithm. In our application, we want to approximate $e^{\tau A}v = e^{-\tau(-A)}v$. To do that, we only need to feed $B = -A$ as an input to the Arnoldi algorithm and use the lemma with the matrix B when computing the error bound.

Since this error bound uses values of the H matrix which is an output of the Arnoldi algorithm, we cannot determine k ahead of time, as we could with an a priori bound. However, with this bound we can provide an accuracy guarantee with significantly fewer iterations.

We use Lemma 4.1 by incrementally increasing the number of Arnoldi iterations performed, k, until the approximation error is smaller than a target accuracy. The implemented algorithm starts with a small $k = 4$, and computes the corresponding error bound defined in Lemma 4.1. If the error bound satisfies the required accuracy, we use that value of k. If not, we continue iterating, increasing k by a factor of 1.1 before the error is checked again. The pseudocode is shown in Algorithm 2, with red lines indicating changes from the original Arnoldi process. The compute-error function implements the error computation from Lemma 4.1, which uses the H matrix.

In our implementation, we target a simulation accuracy of 10^{-6}, the tolerance used in our LP solver. In practice, the observed accuracy of the counter-examples produced upon reaching an unsafe state has been significantly higher, as we will show in the evaluation.

4.3 Krylov Simulations of Symmetric Matrices

A further improvement is possible when the system matrix is both sparse and *symmetric*. This may be the case when the dynamics matrix comes from a physical system due to the symmetry of many physical laws. In this case, the Arnoldi iteration can be replaced by the more efficient Lanczos iteration [37, 50]. The difference between the two is that H matrix in the symmetric case is both *symmetric* and *tridiagonal*. This means that step (2) in the algorithm, projecting out the previous orthonormal directions from the current vector, only needs to be done for the previous two directions, and requires only a single dot product. The Lanczos iteration is shown in Algorithm 3. Notice that the loop which projected out all the previous directions on lines 4-6 of the original Arnoldi iteration in Algorithm 1 is replaced by the $j = i - 2$ case on lines 4-6 and the $j = i - 1$ case on lines 7-8. This change reduces the computation time from $O(k^2)$ to $O(k)$.

Although the computation time is reduced with the Lanczos iteration, since the outputs are matrices of the same size as with Arnoldi, the memory required is basically the same as what was given in Equation 1. Some savings is possible if H is stored as a sparse matrix, since H is now tridiagonal rather than dense, but since typically $k \ll n$, this is savings is small.

We propose a new modification to the Lanczos iteration that can save significant memory, when it is used for the verification problem. Since we eventually project the result of $V_k e^{H_k t} \mathbf{e}_1$ onto the output space matrix C (or the transpose of the initial space matrix E^T), we propose to embed this projection inside the loop in the Lanczos algorithm, at each iteration. The output of the iteration is then the much smaller $o \times k$ matrix CV_k (or the $i \times k$ matrix $E^T V_k$). This eliminates the need to store V_k, a potentially large $n \times k$ matrix, reducing the memory required for the algorithm. The modified Lanczos iteration, which includes both this projection and the a posteriori error bound for selecting k, is shown in Algorithm 4. As before, changes compared with the original Lanczos algorithm are in red. With this improvement, the memory needed to perform the computation is reduced to:

$$(3k + (n \times \min(i, o)) + 3n) \times \texttt{sizeof(double)} \qquad (5)$$

Importantly, compared with Equation 1, there is no term where k is multiplied by n. This makes it possible to analyze high-dimensional systems with a much larger number of Krylov iterations, which may be needed for accuracy. This will be needed in our evaluation when we evaluate a billion-dimensional system where we need $k = 5932$ iterations.

5 EVALUATION

We evaluate the proposed approach on several high dimensional benchmarks[1]. Measurements were performed using Amazon Web Services Elastic Computing Cloud (EC2), on a powerful `m4.10xlarge` instance with 40 cores and a large 160 GB of memory that can be rented by the hour. Note that we perform analysis in discrete time, whereas SpaceEx [26] and the decomposition method [13] we compare against do dense time analysis. Although this requires a few more operations, specifically a bloating at the initial step using

an approximation model, we do not expect qualitative runtime differences.

5.1 Modified Nodal Analysis (MNA5)

We first verify a benchmark model based on a system from the field of electrical circuit analysis, where the state variables relate to the node voltage and currents inside a circuit [14, 48]. Originally a DAE system, the dynamics matrix has been adapted to create a benchmark for verification using ODE reachability methods. As far as we are aware, this benchmark is the largest linear system ever verified [9], where full analysis of the safe version previously took a little over 24 hours. This model has also been investigated using a decomposition approach that uses a series of two-dimensional projections to enable much faster analysis with modest overapproximation error [13]. Here, we apply the proposed algorithm which does not have overapproximation error and can provide counter-examples when property violations are detected.

In this system, the number of dimensions $n = 10923$, the number of output space dimensions $o = 2$, the number of initial space dimensions $i = 10$, and the number of steps is 20000. Our implementation selected a Krylov subspace dimension $k = 63$ using the a posteriori error bound approach, and verified the safe version of this system in 3.8 *seconds*. The unsafe version of the benchmark was checked in 1.1 seconds, generating a counter-example at the same time in the analysis as the earlier approach, at exactly step 1919.

The initial state from the counter-example was then used to compute an external, high accuracy simulation of the system. By comparing the final value in the external simulation versus the output variables assigned by the LP solver when the counter-example was found, we can gauge our method's accuracy. In this case, the relative error between the two was 6.17×10^{-9}, demonstrating the accuracy of the proposed approach.

5.2 Replicated Helicopter

A tunable benchmark is created based on a 28-dimensional helicopter model and controller originally released as an example system with the SpaceEx tool[2]. The helicopter is copied multiple times within the same model, in order to create a verification problem that can scale to an arbitrary number of dimensions.

In the replicated helicopter benchmark, the 28-dimensional helicopter model is copied h times, so that the number of dimensions $n = 28h$. We take initial conditions from the `x8_over_time_large` configuration, where eight of the variables for each helicopter are initially intervals, making the dimension of initial space $i = 8h$. The error condition checks if the average of the x_8 variables is greater than 0.45, $o = 1$. Finally, the problem calls to verify up to time 30 with a step of 0.1, so that the number of steps is 300.

Figure 4 shows the scalability of the new approaches compared with the SpaceEx tool and the basic approach implemented in the Hylaa tool. We also tried to compare against the linear dynamics method in the Flow* tool [15], but could not analyze the $h = 1$ case due to the large uncertainty in the initial set. Using the `stc` scenario [25] of SpaceEx [26], the largest system successfully analyzed had $h = 20$ (560 dimensions) and took 17 minutes (larger
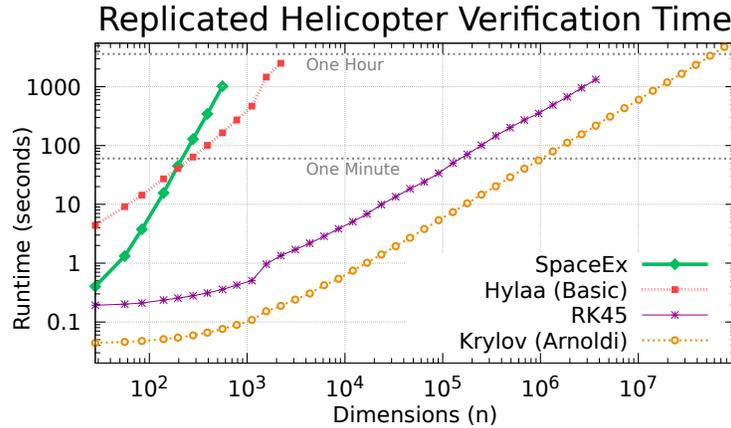

Figure 4: The `Krylov` method scales better than the `RK45` numerical simulation method with input/output spaces, the full space method using numerical simulation used by `Hylaa`, and the space-time clustering scenario of the `SpaceEx` tool.

systems crashed). The basic approach implemented in the `Hylaa` tool, which computes the full $n \times n$ basis matrix using numerical simulations, verified a system $h = 79$ (2212 dimensions) in 42 minutes (larger systems had a memory error). Using the proposed input / output spaces with Runge-Kutta numerical simulations, the RK45 method scaled up to $h = 131389$ (3.6 million dimensions) in about 22 minutes (larger systems had a memory error). Finally, combining both input / output spaces and Krylov subspace simulations, the `Krylov` approach analyzed the system with $h = 2714654$ (76 million dimensions) in 79 minutes, without memory errors. Since the initial space dimension i grows as the number of helicopters h increases, the LP solving step takes increasing amounts of time for this benchmark. This explains why the slopes for RK45 and `Krylov` are similar: the LP solving step has become the bottleneck.

5.3 Symmetric 3D Heat Diffusion

The third benchmark considered is a 3D Heat Diffusion system taken from the field of partial differential equations (PDEs). This benchmark is based on a 2D version that has previously been analyzed up to a 50×50 mesh (2500 dimensions) [34, 36]. The problem is to examine the temperature at the center point of a $1.0 \times 1.0 \times 1.0$ block, where one edge of the block is initially heated. As before, all of the sides of the block are insulated except the $x = 1.0$ edge, which allows for heat exchange with the ambient environment with a heat exchange constant of 0.5. A heated initial region is present in the region where $x \in [0.0, 0.4]$, $y \in [0.0, 0.2]$, and $z \in [0.0, 0.1]$. The heated region temperature is between 0.9 and 1.1, with the rest of material initially at temperature 0.0. The system dynamics is given by the heat equation PDE $u_t = \alpha^2(u_{xx} + u_{yy} + u_{zz})$, where $\alpha = 0.01$ is the diffusivity of the material, as in the previous work.

A linear state space model of the system is obtained using the semi-finite difference method [24], discretizing the block with an $m \times m \times m$ grid. This results in an m^3-dimensional linear system describing the evolution of the temperature at each mesh point.

Due to the initially heated region, we expect the temperature at the center of the block to first increase, and then decrease due to the heat loss along the $x = 1$ edge. Further, there may be error due
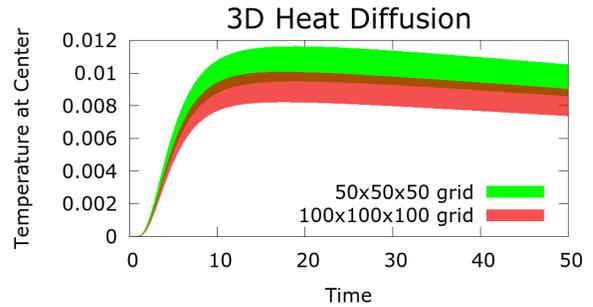

Figure 5: The maximum temperature at the center point occurs around time 15.

to the space discretization step, so if m is too small, the model does not accurately predict the behavior of the PDE. We can see both of these effects by computing and plotting the reachable states, as shown in Figure 5. Since the peak temperature happens at around time $t = 15$, we perform further analysis by running the system with max time $T = 20.0$ and step size $\delta = 0.02$, making the number of steps 1000. This system presents a particularly good case for our analysis method, since $i = 1$, $o = 1$, and the dynamics matrix is symmetric which allows us to use the more efficient Lanczos iteration.

The runtimes and temperatures reachable for various values of m are given in Table 1. Accurate analysis requires high dimensions, motivating the need for the types of analysis methods developed in this paper. The $1000 \times 1000 \times 1000$ version can be analyzed using our approach in about 30 hours of computation time. Over 95% of the runtime was spent in the Lanczos iteration, indicating that we optimized the correct operation. In this case, each of the billion rows of the A matrix generally has 7 entries, so that simply storing the elements of the matrix (8 bytes per double-precision number) consumes 56 GB of RAM (this could be reduced by about half by using data structures that take advantage of the symmetry). Further, since a 5932-dimensional Krylov subspace is needed for sufficient

Table 1: 3D Heat Diffusion with $n = m^3$ Dimensions

m	$T_{\max}$	k	Lanczos	Arnoldi
10	0.02934	57	0.6s	0.4s
20	0.01713	115	1.0s	1.2s
50	0.01161	277	4.4s	16.8s
100	0.01005	544	18.0s	7m13s
200	0.00933	1170	3m56s	9h14m
500	0.00890	2764	1h46m	-
1000	0.00877	5932	29h46m	-

numerical accuracy, the unmodified Lanczos iteration would be infeasible for this system, as it would require storing 5932 vectors for the V matrix, each of which contains a billion numbers (8 GB each), for a total memory requirement of 46 TB (recall Equation 1).

Lastly, we examine the computed error bound from Lemma 4.1 for the 100x100x100 version of this benchmark, as the dimension of the Krylov subspace k is increased. While performing the Arnoldi or Lanczos algorithm, our implementation periodically checks the current error. Once $k = 544$ iterations have been performed, the computed error bound is $5.8 * 10^{-7}$, which is below the desired error threshold of 10^{-6}. The plot is shown in Figure 6. The blue line is the error bound computed using Lemma 4.1 at each iteration, and the points indicate where the bound gets sampled. The thinner green line is the relative error of the projected simulation between iterations k and $k + 1$, which provides an error *estimate* that was used as a stopping criteria in previous work [7]. Notice that the old error bound can not be used when the number of iterations is low, as the projected simulation onto the output variables is zero when k is small. For this system, using the old bound might reach the 10^{-6} threshold earlier and terminate prematurely, although the number of iterations for both bounds is similar, within a factor of two. Recall from the discussion after Equation 3 that the a priori bound was unusable for this system, even with $k = 10^6$.

6 RELATED WORK

The proposed method uses convergent numerical schemes to compute simulations as part of a verification procedure. Convergent numerical schemes have been used before to approximate reachable sets of nonlinear hybrid systems, in particular, level-set methods that approximate solutions to Hamilton-Jacobi PDEs [12, 42, 47]. These methods compute reachable states with a grid over the state space, and in the limit at the number of grid points increases, the computed result approaches the true solution.

Other methods for this class of systems have used simulations for formal analysis, where individual executions are bloated according to model-specific discrepancy functions [22], as implemented in tools such as C2E2 [20, 23]. Another analysis approach for nonlinear systems uses Taylor models, such as those in Flow* [15], which can scale to around ten real variables [16]. For affine systems, as recently as 2011 the state-of-the-art for reachability computation was on the order of a hundred real variables [26].

Our work uses the Krylov subspace to simulate high-dimensional systems, which is often also used in model order reduction methods [5]. Notice that in our case, since each simulation has a different
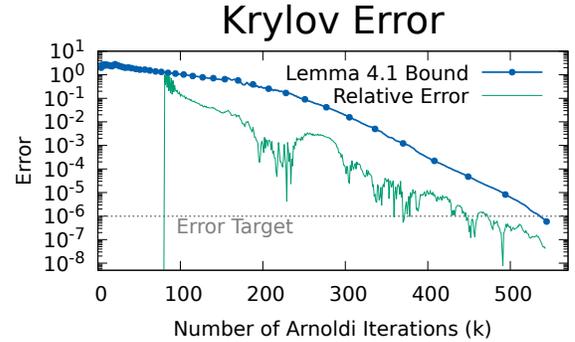

Figure 6: A 544-dimensional Krylov subspace exceeds the desired error target of 10^{-6} for the 100x100x100 (one million dimensional) 3D Heat diffusion system.

Krylov subspace, *there is no single reduced order model that can be constructed and analyzed* (we are not creating a low-dimensional abstraction of the system). Model-order reduction approaches verify a smaller dimensional model [17], and can sometimes use an error bound to compute a guaranteed overapproximation of the original full-order system [34, 35, 49]. Such approximation methods may be formalized as sound abstractions or developed in the context of approximate simulation and bisimulation relations [30, 31]. Model order reduction methods have verified linear systems with on the order of a thousand real variables.

Our approach builds on the basic verification approach used in the Hylaa tool [10], which has verified systems with up to ten thousand dimensions [7, 9, 11]. We scale to larger systems here by leveraging initial and output spaces and using Krylov subspace methods for numerical simulation. The basic approach here is also related to the symbolic orthogonal projection method [33], where the current-time variables in our approach could be considered the variables onto which we are computing the projection.

Recent work on reachablity with Krylov methods [1] has used better a priori bounds to reduce the number of required Arnoldi iterations, compared with earlier work [36]. Further, more efficient methods exist which perform Krylov simulations in multiple stages [44], rather than only from the initial time as in this work. Integrating these into our approach could further improve our result by reducing the required dimension of the Krylov subspace.

For large systems, the only analysis option we are aware of beyond simple simulation would be falsification methods [4, 19, 45], which run individual simulations trying to optimize towards an unsafe region. Unlike our approach, these methods do not exhaustively explore the state space.

7 CONCLUSION

The state-space explosion problem usually prevents analysis of high-dimensional affine systems. In order to achieve scalability, we exploited up to four types of problem structure: (i) a small dimension of initial states, (ii) a small dimension of the output space, (iii) the sparsity of the system A matrix, and (iv), optionally, the symmetry of the A matrix. When problems have this structure, we

have shown it is possible to efficiently perform verification or plot projections of the reachable states despite a large number of state variables. As the structure assumptions are violated, the proposed approach degrades gracefully, requiring more computation time and memory depending on the degree of the violation. We have evaluated our approach on several large benchmarks, including a 3D Heat Diffusion system with one billion continuous state variables. Prior to this work, no existing method for affine systems has demonstrated scalability beyond a few thousand variables.

REFERENCES

[1] Matthias Althoff. 2017. Reachability Analysis of Large Linear Systems with Uncertain Inputs in the Krylov Subspace. *arXiv preprint arXiv:1712.00369* (2017).
[2] Matthias Althoff, Olaf Stursberg, and Martin Buss. 2008. Reachability analysis of nonlinear systems with uncertain parameters using conservative linearization. In *Decision and Control, 2008. CDC 2008. 47th IEEE Conference on.* IEEE, 4042–4048.
[3] Rajeev Alur, Costas Courcoubetis, Thomas A Henzinger, and Pei-Hsin Ho. 1993. Hybrid automata: An algorithmic approach to the specification and verification of hybrid systems. In *Hybrid systems.* Springer, 209–229.
[4] Yashwanth Annpureddy, Che Liu, Georgios Fainekos, and Sriram Sankaranarayanan. 2011. S-taliro: A tool for temporal logic falsification for hybrid systems. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems.* Springer, 254–257.
[5] A. C. Antoulas, D. C. Sorensen, and S. Gugercin. 2001. A survey of model reduction methods for large-scale systems. *Contemp. Math.* 280 (2001), 193–219.
[6] Walter Edwin Arnoldi. 1951. The principle of minimized iterations in the solution of the matrix eigenvalue problem. *Quarterly of applied mathematics* 9, 1 (1951).
[7] Stanley Bak. 2018. Numerical Verification of 10000-dimensional Linear Systems 10000x Faster. In *ARCH18. 5th International Workshop on Applied Verification of Continuous and Hybrid Systems (EPiC Series in Computing)*, Goran Frehse (Ed.), Vol. 54. EasyChair, 135–144. https://doi.org/10.29007/gv5q
[8] Stanley Bak, Sergiy Bogomolov, Thomas A. Henzinger, Taylor T. Johnson, and Pradyot Prakash. 2016. Scalable Static Hybridization Methods for Analysis of Nonlinear Systems. In *International Conference on Hybrid Systems: Computation and Control.* 10. https://doi.org/10.1145/2883817.2883837
[9] Stanley Bak and Parasara Sridhar Duggirala. 2017. Direct Verification of Linear Systems with over 10000 Dimensions. In *4th International Workshop on Applied Verification of Continuous and Hybrid Systems (EPiC).* EasyChair.
[10] Stanley Bak and Parasara Sridhar Duggirala. 2017. Hylaa: A tool for computing simulation-equivalent reachability for linear systems. In *Proceedings of the 20th International Conference on Hybrid Systems: Computation and Control.* ACM.
[11] Stanley Bak and Parasara Sridhar Duggirala. 2017. Simulation-Equivalent Reachability of Large Linear Systems with Inputs. In *Proceedings of the 29th International Conference on Computer Aided Verification.* Springer.
[12] Somil Bansal, Mo Chen, Sylvia Herbert, and Claire J Tomlin. 2017. Hamilton-Jacobi Reachability: A Brief Overview and Recent Advances. *1709.07523* (2017).
[13] Sergiy Bogomolov, Marcelo Forets, Goran Frehse, Andreas Podelski, Christian Schilling, and Frédéric Viry. 2018. Reach Set Approximation through Decomposition with Low-dimensional Sets and High-dimensional Matrices. In *21st International Conference on Hybrid Systems: Computation and Control.* ACM.
[14] Younes Chahlaoui and Paul Van Dooren. 2002. A collection of benchmark examples for model reduction of linear time invariant dynamical systems. (2002).
[15] Xin Chen, Erika Abraham, and Sriram Sankaranarayanan. 2012. Taylor Model Flowpipe Construction for Non-linear Hybrid Systems. *Real-Time Systems Symposium* (2012). https://doi.org/10.1109/RTSS.2012.70
[16] Xin Chen, Stefan Schupp, Ibtissem Ben Makhlouf, Erika Ábrahám, Goran Frehse, and Stefan Kowalewski. 2015. A benchmark suite for hybrid systems reachability analysis. In *NASA Formal Methods Symposium.* Springer, 408–414.
[17] Yi Chou, Xin Chen, and Sriram Sankaranarayanan. 2017. A Study of Model-Order Reduction Techniques for Verification. In *Numerical Software Verification.*
[18] Thao Dang, Oded Maler, and Romain Testylier. 2010. Accurate hybridization of nonlinear systems. In *Hybrid systems: computation and control.* 10.
[19] Alexandre Donzé. 2010. Breach, a toolbox for verification and parameter synthesis of hybrid systems. In *Computer Aided Verification.* Springer.
[20] Parasara Sridhar Duggirala, Sayan Mitra, Mahesh Viswanathan, and Matthew Potok. 2015. C2E2: a verification tool for stateflow models. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems.*
[21] Parasara Sridhar Duggirala and Mahesh Viswanathan. 2016. Parsimonious, simulation based verification of linear systems. In *Computer Aided Verification.*
[22] Chuchu Fan and Sayan Mitra. 2015. Bounded verification with on-the-fly discrepancy computation. In *Automated Technology for Verification and Analysis.*
[23] Chuchu Fan, Bolun Qi, Sayan Mitra, Mahesh Viswanathan, and Parasara Sridhar Duggirala. 2016. Automatic reachability analysis for nonlinear hybrid models with C2E2. In *International Conference on Computer Aided Verification.*
[24] Stanley J Farlow. 1993. *Partial differential equations for scientists and engineers.*
[25] Goran Frehse, Rajat Kateja, and Colas Le Guernic. 2013. Flowpipe approximation and clustering in space-time. In *Hybrid systems: computation and control.*
[26] Goran Frehse, Colas Le Guernic, Alexandre Donzé, Scott Cotton, Rajarshi Ray, Olivier Lebeltel, Rodolfo Ripado, Antoine Girard, Thao Dang, and Oded Maler. 2011. SpaceEx: Scalable Verification of Hybrid Systems. In *International Conference on Computer Aided Verification.* Springer.
[27] Efstratios Gallopoulos and Yousef Saad. 1992. Efficient solution of parabolic equations by Krylov approximation methods. *SIAM J. Sci. Statist. Comput.* (1992).
[28] Antoine Girard. 2005. Reachability of uncertain linear systems using zonotopes. In *International Workshop on Hybrid Systems: Computation and Control.* Springer.
[29] Antoine Girard, Colas Le Guernic, and Oded Maler. 2006. Efficient computation of reachable sets of linear time-invariant systems with inputs. In *International Workshop on Hybrid Systems: Computation and Control.* Springer, 257–271.
[30] Antoine Girard and George J Pappas. 2007. Approximation metrics for discrete and continuous systems. *Automatic Control, IEEE Transactions on* 52, 5 (2007).
[31] Antoine Girard and George J Pappas. 2011. Approximate bisimulation: A bridge between computer science and control theory. *European Journal of Control* (2011).
[32] Amit Gurung and Rajarshi Ray. 2016. An Efficient Algorithm for Vertex Enumeration of Two-Dimensional Projection of Polytopes. *CoRR* abs/1611.10059 (2016). http://arxiv.org/abs/1611.10059
[33] Willem Hagemann. 2014. Reachability analysis of hybrid systems using symbolic orthogonal projections. In *Computer Aided Verification.*
[34] Zhi Han. 2005. *Formal verification of hybrid systems using model order reduction and decomposition.* Ph.D. Dissertation. Dept. of ECE, Carnegie Mellon University.
[35] Zhi Han and Bruce Krogh. 2004. Reachability analysis of hybrid control systems using reduced-order models. In *American Control Conference.*
[36] Zhi Han and Bruce H Krogh. 2006. Reachability analysis of large-scale affine systems using low-dimensional polytopes. In *HSCC*, Vol. 6. Springer, 287–301.
[37] Cornelius Lanczos. 1950. *An iteration method for the solution of the eigenvalue problem of linear differential and integral operators.* USA Press Office.
[38] Colas Le Guernic. 2009. *Reachability analysis of hybrid systems with linear continuous dynamics.* Ph.D. Dissertation. Université Joseph-Fourier-Grenoble I.
[39] Colas Le Guernic and Antoine Girard. 2010. Reachability analysis of linear systems using support functions. *Nonlinear Analysis: Hybrid Systems* 4, 2 (2010).
[40] Alexander V Lotov, Vladimir A Bushenkov, and Georgy K Kamenev. 2013. *Interactive decision maps: Approximation and visualization of Pareto frontier.*
[41] Oded Maler, Zohar Manna, and Amir Pnueli. 1991. From timed to hybrid systems. In *Workshop/School/Symposium of the REX Project.*
[42] Ian Mitchell and Claire Tomlin. 2000. Level set methods for computation in hybrid systems. In *HSCC*, Vol. 1790. Springer, 310–323.
[43] Cleve Moler and Charles Van Loan. 2003. Nineteen dubious ways to compute the exponential of a matrix, twenty-five years later. *SIAM review* 45, 1 (2003), 3–49.
[44] Jitse Niesen and Will M. Wright. 2012. A Krylov Subspace Algorithm for Evaluating the ϕ-Functions Appearing in Exponential Integrators. *ACM Trans. Math. Softw.* (2012). http://doi.acm.org/10.1145/2168773.2168781
[45] Aurélien Rizk, Grégory Batt, François Fages, and Sylvain Soliman. 2008. On a continuous degree of satisfaction of temporal logic formulae with applications to systems biology. In *Computational Methods in Systems Biology.*
[46] O. Stauning. 1997. *Automatic validation of numerical solutions.* Ph.D. Dissertation. IMU-DTU, Richard Petersens Plads, Building 321, DK-2800 Kgs. Lyngby.
[47] C.J. Tomlin, I. Mitchell, A.M. Bayen, and M. Oishi. 2003. Computational techniques for the verification of hybrid systems. *Proc. IEEE* (2003).
[48] Hoang-Dung Tran, Luan Viet Nguyen, and Taylor T Johnson. 2016. Large-scale linear systems from order-reduction (benchmark proposal). In *3rd Applied Verification for Continuous and Hybrid Systems Workshop (ARCH), Vienna, Austria.*
[49] Hoang-Dung Tran, Luan Viet Nguyen, Weiming Xiang, and Taylor T Johnson. 2017. Order-Reduction Abstractions for Safety Verification of High-Dimensional Linear Systems. *Discrete Event Dynamic Systems* (2017).
[50] Lloyd N Trefethen and David Bau III. 1997. *Numerical linear algebra.* SIAM.
[51] Hao Wang and Qiang Ye. 2017. Error bounds for the Krylov subspace methods for computations of matrix exponentials. *SIAM J. Matrix Anal. Appl.* 38, 1 (2017).