

MECH 6327 Project Report: Examining Discrete-Time Polytopic Linear Parameter-Varying Systems under threat of malicious actuator and sensor manipulation

Jonas Wagner

2021 May 14

Abstract

In this project the dynamics of Discrete-Time Polytopic Linear Parameter-Varying (LPV) Systems will be examined. Specifically, various methods for the dual state and parameter estimation will be reproduced with the intent of analyzing effectiveness of these observers against various attacks. Each method performs optimization to minimize the estimation error in various ways while remaining stable and achieving certain performance criteria. Potentially the reachability of the system may be determined for various fault and attack scenarios through the minimization of an ellipsoidal bound.

Contents

1	Polytopic Systems Background	3
1.1	General Continuous Time Polytopic Model	3
1.2	Discrete Time Polytopic Model	3
2	Joint State-Parameter Estimation Problem	4
2.1	Problem Simplifications/Assumptions [1]	4
2.2	Potential Methods	4
3	Kalmen Filter Derived Methods	5
3.1	Extended Kalmen Filter (EKF) Method [2]	5
4	DE Method Optimization Problem Definition and Solution [1]	6
4.1	State Estimation Problem and Solution	6
4.1.1	State Estimation Error:	6
4.1.2	Estimator ISS Lyapanov Function:	6
4.1.3	LMI Feasibility Problem for ISS	7
4.2	Parameter Estimation Problem and Solution	7
4.2.1	Parameter Estimate Optimization Problem[1]	7
4.2.2	Step 1: Determine Polynomial Coefficients	8
4.2.3	Step 2: Determine Scheduling Parameter	8
5	Numerical Experimentation	9
5.1	Simulation Implementation	9
5.2	Example Polytopic System	9
5.3	DE Implementation	9
5.4	EKF Implementation	9
5.5	Recreated Results	9
6	Attack Implementation	9
7	Results and Discussion	13
.1	MATLAB	14
.2	Complete Set of Results	26

1 Polytopic Systems Background

Polytopic LPV system models are essentially a smooth interpolation of a set of LTI submodels constructed using a specified weighting function. This can be looked at as decomposing a system into multiple operating spaces that operate as linear submodels. It is possible for a Polytopic model to take a complex nonlinear model and redefine it as a time-varying interpolation of multiple linear submodels.

Section references:¹ [1] [3] [4]

1.1 General Continuous Time Polytopic Model

The simple polytopic LPV structure can be described by the following weighted linear combination of LTI submodels:

$$\begin{cases} \dot{x}(t) = \sum_{i=1}^r \mu_i(\xi(t)) \{A_i x(t) + B_i u(t)\} \\ y(t) = \sum_{i=1}^r \mu_i(\xi(t)) C_i x(t) \end{cases} \quad (1)$$

with state variable $x \in \mathbb{R}^n$ common to all r submodels, control input $u \in \mathbb{R}^p$, output $y \in \mathbb{R}^q$, weighting function $\mu_i(\cdot)$ and premise variable $\xi(t) \in \mathbb{R}^w$.

Additionally, the weighting functions $\mu_i(\cdot)$ for each subsystem must satisfy the convex sum constraints:

$$0 \leq \mu_i(\xi), \forall i = 1, \dots, r \text{ and } \sum_{i=1}^r \mu_i(\xi) = 1 \quad (2)$$

One notable downside, for our application, is the requirement for $\xi(t)$ to be explicitly known in real-time for the model to function. This requirement is the primary driving factor in investigating this system as when $\xi(t)$ is not explicitly known additional uncertainties now exist in a system that are open for exploitation by an attacker.

1.2 Discrete Time Polytopic Model

In the DT-Polytopic Model the CT-Polytopic Model, (1), is extended into the discrete time equivalence (either through sampling and zero-order holds or by definition) by the following parameter-varying system:

$$\begin{cases} x_{k+1} &= \sum_{i=1}^m \alpha^i (A_i x_k + B_i u_k) \\ y &= C x_k \end{cases} \quad (3)$$

with state variable $x \in \mathbb{R}^n$, control input $u \in \mathbb{R}^p$, and output $y \in \mathbb{R}^q$ common to all of the m submodels. Each submodel is also associated with state matrices A_i and B_i while the output is calculated from the actual state by matrix C .

The scheduling parameter, $\alpha \in \mathcal{A}$ is unknown and time-varying, with \mathcal{A} defined as:

$$\mathcal{A} = \{\alpha \in \mathbb{R}^m \mid \sum_{i=1}^m \alpha^i = 1, \alpha^i \geq 0 \forall i \in \{1, 2, \dots, m\}\} \quad (4)$$

In the discrete time case, the unknown scheduling parameter, α , is problematic for when developing a state-estimator, thus a Joint State-Parameter estimator must be used. The discrete nature of the measurements may also prove to be even more problematic if an attack is injected in any discrete measurement.

¹Each subsection is mostly a summary of sections from these sources but with elaboration and consistent notation.

2 Joint State-Parameter Estimation Problem

The problem of developing joint state and parameter estimator is defined as finding recursive update rules to ensure that state and parameter estimates approach the actual states and parameters. This can be described as finding f_x and f_α such that

$$\begin{cases} \hat{x}_{k+1} &= f_x(\hat{x}_k, \hat{\alpha}_k, \{u_l, y_l\}_{l=\bar{k}_x}) \\ \hat{\alpha}_{k+1} &= f_\alpha(\hat{x}_k, \hat{\alpha}_k, \{u_l, y_l\}_{l=\bar{k}_\alpha}) \end{cases} \quad (5)$$

with $\bar{k}_x, \bar{k}_\alpha < k$ result in $\|x_k - \hat{x}_k\| \rightarrow 0$ and $\|\alpha - \hat{\alpha}_k\| \rightarrow 0$ as $k \rightarrow \infty$.

2.1 Problem Simplifications/Assumptions [1]

For simplicity, and probably for feasibility reasons, the following assumptions will also be made:

1. $\alpha \in \mathcal{A}$ is constant (or at least slowly time-varying)
2. The estimated system is observable $\forall \hat{\alpha} \in \mathcal{A}$, (i.e. $(\sum_{i=1}^m \hat{\alpha} A_i, C)$ is an observable pair)
3. A unique solution exists to the joint-estimation problem

$$\begin{aligned} & C(qI - \sum_{i=1}^m \alpha^i A_i x_k)^{-1} \sum_{i=1}^m \alpha^i B_i \\ &= C(qI - \sum_{i=1}^m \hat{\alpha}^i A_i x_k)^{-1} \sum_{i=1}^m \hat{\alpha}^i B_i \implies \alpha = \hat{\alpha} \end{aligned}$$

2.2 Potential Methods

Three joint state and parameter estimator methods for LPV systems will be developed and tested for the ability of each to react to malicious input and measurement interference. The three estimation methods of interest include:

1. Dual-Estimation (DE) approach is a method that first solves a two step optimization problem for parameters-estimation and then uses a "traditional" robust polytopic observer design for state estimation. [1]
2. Extended Kalman Filter (EKF) using prediction and update steps for the system estimates, but this version does require the assumption of Gaussian noise. [2]
3. Interacting Multiple Model estimation (IMM) method which uses a different kalmen filter for multiple modes and the probability that the system will be a certain mode.[5]

3 Kalmen Filter Derived Methods

3.1 Extended Kalmen Filter (EKF) Method [2]

The Extended Kalmen Filter (EKF) method relies on a prediction and update steps for both state and parameter estimates by augmenting the state with the parameters creating a nonlinear system.

$$\begin{bmatrix} x_{k+1} \\ \alpha_{k+1} \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^m \alpha^i (A_i x_k + B_i u_k) \\ \alpha_k \end{bmatrix} := f(x_k, \alpha_k, u_k) \quad (6)$$

This is then solved through a two-step prediction and estimation process. The prediction step is defined as

$$\hat{x}_{k|k-1} = \sum_{k=1}^m \hat{\alpha}_{k-1|k-1}^i (A_i \hat{x}_{k-1|k-1} + B_i u_{k-1}) \quad (7a)$$

$$\hat{\alpha}_{k|k-1} = \hat{\alpha}_{k|k} \quad (7b)$$

$$P_{k|k-1} = \hat{A}_{k-1} P_{k-1|k-1} \hat{A}_{k-1}^\top + Q \quad (7c)$$

with the update steps defined by

$$\begin{bmatrix} \hat{x}_{k|k} \\ \hat{\alpha}_{k|k} \end{bmatrix} = \begin{bmatrix} \hat{x}_{k|k-1} \\ \hat{\alpha}_{k|k-1} \end{bmatrix} + L_k (y_k - C \hat{x}_{k|k-1}) \quad (8a)$$

$$P_{k|k} = (I - L_k \hat{C}) P_{k|k-1} \quad (8b)$$

and then to restrict $\hat{\alpha}_{k|k} \in \mathcal{A}$, it is determined by

$$\hat{\alpha}_{k|k} = \arg \min_{\alpha \in \mathcal{A}} \|\alpha - \bar{\alpha}_{k|k}\| \quad (8c)$$

with the augmented estimated parameters defined by

$$L_k = P_{k|k-1} \hat{C}^\top (R + \hat{C} P_{k|k-1} \hat{C}^\top)^{-1} \quad (9a)$$

$$\begin{aligned} \hat{A}_k &= \left. \frac{\partial f(x_k, \alpha_k, u_k)}{\partial [x \quad \alpha]^\top} \right|_{(\hat{x}_k, \hat{\alpha}_k, u_k)} \\ &= \begin{bmatrix} \sum_{i=1}^m \hat{\alpha}_{k|k}^i A_i & [A_1 x_{k|k} + B_1 u_k \cdots A_m x_{k|k} + B_m u_k] \\ 0 & I \end{bmatrix} \end{aligned} \quad (9b)$$

$$\hat{C} = [C \quad 0] \quad (9c)$$

In this estimator, Q and R are Positive Semi-definite covariance matrices, traditionally corresponding to process and measurement noise estimates, and can be used as tuning parameters. However, this does require the assumption of Gaussian noise which is not always valid within the actual system.

This method is widely used and will converge when the state and parameter estimates are close enough and any nonlinearities are continuous, but no specific guarantees can be made about the robustness.

4 DE Method Optimization Problem Definition and Solution [1]

The primary optimization problems to be analyzed are the state and parameter estimation problems within the DE method. This method separates the joint-estimation problem into a state and then a parameter-estimation problem, each of which are determined using convex optimization techniques.

The parameter estimation is done independently using a recursive parameter-estimation method relying solely on the input-output data $\{u_l, y_l\}_{l=1}^k$ without any reliance on the state-estimates. This itself is an optimization problem that aims to minimize the prediction error given the parameter estimates.

4.1 State Estimation Problem and Solution

The system state is estimated using a modified Luenberger observer defined by

$$\hat{x}_{k+1} = \sum_{i=1}^m \hat{\alpha}_k^i (A_i \hat{x}_k + B_i u_k + L_i (C \hat{x}_k - y_k)) \quad (10)$$

with $\hat{\alpha}_k \in \mathcal{A}$ being the estimated parameters and L_i being selected to ensure the estimation error, $e_k = x_k - \hat{x}_k$, is stable.

4.1.1 State Estimation Error:

The estimation error is defined as

$$e_{k+1} = \sum_{i=1}^m (\hat{\alpha}_k^i (A_i + L_i C) e_k + (\alpha^i - \hat{\alpha}_k^i) (A_i x_k + B_i u_k)) \quad (11)$$

A solution that ensures e_k is stable is that when the disturbance caused by the deviation of $\hat{\alpha}$, v_k decays to zero (a requirement for the parameter estimation problem), e_k also decays to zero. This is equivalent to ensuring Input-to-State Stability (ISS) of the estimator.

4.1.2 Estimator ISS Lyapunov Function:

The estimator can be rewritten as

$$e_{k+1} = \sum_{i=1}^m \hat{\alpha}_k^i (A_i + L_i C) e_k + v_k \quad (12)$$

and ISS with respect to the disturbance term, v_k , defined by

$$v_k = \sum_{i=1}^m (\alpha^i - \hat{\alpha}_k^i) (A_i x_k + B_i u_k) \quad (13)$$

can be guaranteed by using an ISS Lyapunov function to solve for an appropriate Luenberger gain.

The ISS Lyapunov function is a function $V : \mathbb{R}^{n+m} \rightarrow \mathbb{R}$, such that

$$V(e, \hat{\alpha}) > 0 \forall e \neq 0, \quad V(0, \hat{\alpha}) = 0,$$

and

$$V(e_{k+1}, \hat{\alpha}_{k+1}) - V(e_k, \hat{\alpha}_k) < -\|e_k\|^2 + \zeta \|v_k\|^2 \quad (14)$$

for all $k \in \{0, \dots, N\}$.

4.1.3 LMI Feasibility Problem for ISS

A theorem provided in [6] states that if there exists matrices P_i , F_i , G_i , $i \in \{1, \dots, m\}$ and a scalar ζ such that

$$\begin{bmatrix} G_i + G_i^\top - P_j & 0 & G_i A_i + F_i C & G_i \\ 0 & I & I & 0 \\ A_i^\top G_i^\top + C^\top F_i^\top & I & P_i & 0 \\ G_i^\top & 0 & 0 & \zeta I \end{bmatrix} \succ 0 \quad (15)$$

for all $i, j \in \{1, \dots, m\}$, then for (12) with the Luenberger gains calculated with

$$L_i = G_i^{-1} F_i \quad (16)$$

guarantees ISS with respect to v .² This essentially means that the $e_k \rightarrow 0$ whenever $v_k \rightarrow 0$, or identically, $\hat{x}_k \rightarrow x_k$ and $\hat{\alpha}_k \rightarrow \alpha_k$, over a finite number of steps.

4.2 Parameter Estimation Problem and Solution

The DE method estimates the parameters by using only the input-output data $\{u_l, y_l\}_{l=0}^k$ and completely ignores the estimated states. This is done through the minimization of the prediction error directly through an optimization based on a modified least-squares type algorithm.

4.2.1 Parameter Estimate Optimization Problem[1]

The parameter-estimation problem is solved by minimizing the prediction error defined as

$$\hat{\alpha}_k = \arg \min_{\alpha \in \mathcal{A}} \sum_{l=0}^k \gamma^{k-l} \|y_k - \varphi_k \nu(\alpha)\|^2 \quad (17a)$$

with a the forgetting factor $0 < \gamma \leq 1$ and

$$\varphi_k = [y_{k-1} \ \dots \ y_{k-n_x} \ u_{k-1} \ \dots \ u_{k-n_x}] \quad (17b)$$

$$\nu(\alpha) = [a_1(\alpha) \ \dots \ a_{n_x}(\alpha) \ b_1(\alpha) \ \dots \ b_{n_x}(\alpha)]^\top \quad (17c)$$

with variables a_i and b_i being the coefficients from the polynomials

$$\sum_{i=1}^{n_x} a_i(\alpha) q^i = \det \left(qI - \sum_{i=1}^m \alpha^i A_i \right) \quad (18a)$$

$$\sum_{i=1}^{n_x} b_i(\alpha) q^i = C \text{adj} \left(qI - \sum_{i=1}^m \alpha^i A_i \right) \sum_{i=1}^m \alpha^i B_i \quad (18b)$$

which are in fact the transfer function coefficients for the system itself, (3).

It is also noteworthy that the optimization problem (17) is a nonlinear (and possibly nonconvex) optimization problem, so instead of solving this directly, a two step approach can be taken to first solve for the coefficients in (18) and then determining the scheduling parameter corresponding to those transfer function coefficients.

²The derivation explained in [6] is interesting (but long and more Optimal Estimation focused in nature) and essentially proves that the ISS Lyapunov function always exists if the solution is feasible.

4.2.2 Step 1: Determine Polynomial Coefficients

Determining the polynomial coefficients is a simple optimization problem consisting of an unconstrained linear least-squares optimization problem given by

$$\hat{\nu}_k = \arg \min_{\nu} \sum_{l=0}^k \gamma^{k-l} \|y_k - \varphi_k \nu\|^2 \quad (19)$$

The solution to this optimization problem can be computed recursively with

$$P_k = \frac{1}{\gamma} P_{k-1} - \frac{1}{\gamma} \varphi_k^\top (\gamma I + \varphi P_{k-1} \varphi_k^\top)^{-1} \varphi_k P_{k-1} \quad (20a)$$

$$\hat{\nu}_k = \hat{\nu}_{k-1} + P_k \varphi_k^\top (y_k - \varphi_k \hat{\nu}_{k-1}) \quad (20b)$$

with initial estimates of $\hat{\nu}_0$ and some matrix $P \succ 0$.

4.2.3 Step 2: Determine Scheduling Parameter

The second step is to calculate parameter estimates as the solution to the nonlinear optimization problem

$$\hat{\alpha}_k = \arg \min_{\alpha \in \mathcal{A}} (\hat{\nu}_k - \nu(\alpha))^\top P_k^{-1} (\hat{\nu}_k - \nu(\alpha)) \quad (21)$$

The optimal solution and resultant parameter estimates are equivalent to the optimal solution to the original optimization problem (17). The proof for this is shown in [1] and consists of taking the solution to (21) and going in reverse through the definition and solution of (19) to equate the optimal solution to that of the original optimization problem (17).

5 Numerical Experimentation

5.1 Simulation Implementation

A generalized Method in Simulink was attempted to be created that would be able to simulate an arbitrary Polytopic System, each of the estimation methods, and introduce attack vectors into the system. This is the future goal to be worked on this summer, but after spending a considerable amount of time learning how to do this, it was abandoned for the short term testing for this project and instead the "toy"/numerical DT model given in [1] was simulated alongside each estimator in a simple MATLAB script for the entirely DT model. These MATLAB scripts can be seen in Appendix .1.

5.2 Example Polytopic System

A simple DT Polytopic Model taken from [1] is defined by (3) with the following state-matrices:

$$\begin{aligned} A_1 &= \begin{bmatrix} -0.80 & 0.25 \\ 0.25 & -0.30 \end{bmatrix}, B_1 = \begin{bmatrix} 1.9 \\ 0 \end{bmatrix} \\ A_2 &= \begin{bmatrix} 0.30 & 0.70 \\ 0.70 & 0 \end{bmatrix}, B_2 = \begin{bmatrix} -1 \\ 1.50 \end{bmatrix} \\ A_3 &= \begin{bmatrix} -0.30 & 0.65 \\ 0.55 & 0.10 \end{bmatrix}, B_3 = \begin{bmatrix} 0.30 \\ -2 \end{bmatrix} \\ A_4 &= \begin{bmatrix} 0.55 & 0.20 \\ -0.40 & -0.30 \end{bmatrix}, B_4 = \begin{bmatrix} -0.60 \\ 0 \end{bmatrix} \end{aligned} \tag{22}$$

with the output matrix $C = [0 \ 1]$, making the dimensions

5.3 DE Implementation

The Dual Estimate method was implemented as explained in Section 4 was then simulated. Untimely it was very difficult to find a solver that was consistent at solving and being robust for a wider array of starting positions for the second step. Eventually it was determined that using Yalmip directly to solve the parameter problem was more effective at converging. (This primarily occurred with the first step as the update for P could result in a negative semi-definite matrix) The 'forgetting factor' from the paper was used:

$$\gamma = 0.9$$

5.4 EKF Implementation

The Extended Kalmen Filter was implemented as well using the parameters from the paper:

$$R = 0.01 \text{ and } Q = \begin{bmatrix} 0_{n \times n} & \\ & 100 * I_{m \times m} \end{bmatrix}$$

5.5 Recreated Results

The results from the original paper were recreated in MATLAB (Appendix.1). The input was selected as essentially a random PWM signal of strength $u_0 = 0.5$. Specifically, the actual system response is can be seen in Figure 1. Additionally, the response for the DE and EKF methods can be seen in Figure 2 and Figure 3 respectively.

6 Attack Implementation

The attacks for this system were essentially Gaussian Noise. In the future a more general model (preferably in Simulink) will be developed to test this on more complicated systems, but for now a simple Gaussian noise signal was generated and applied to the measurement each time-step.

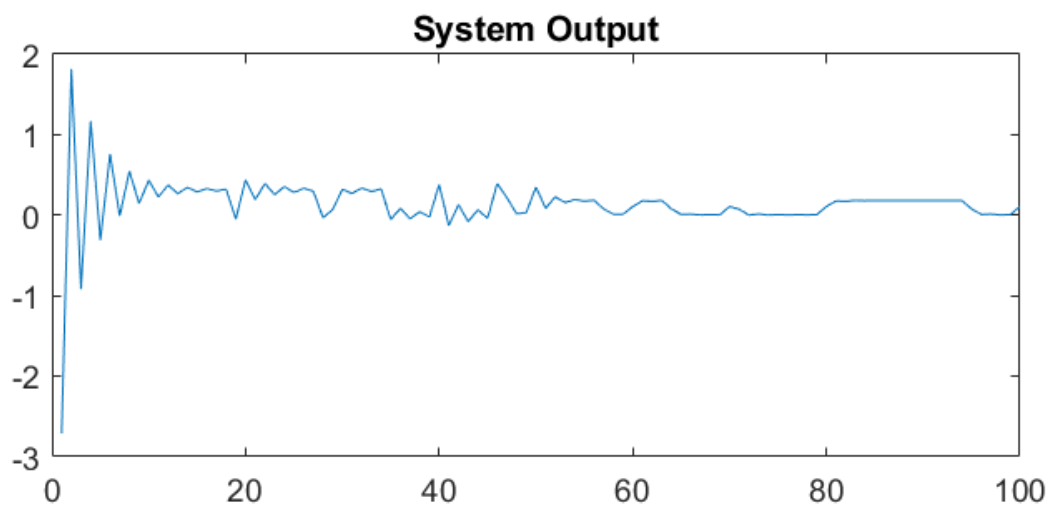
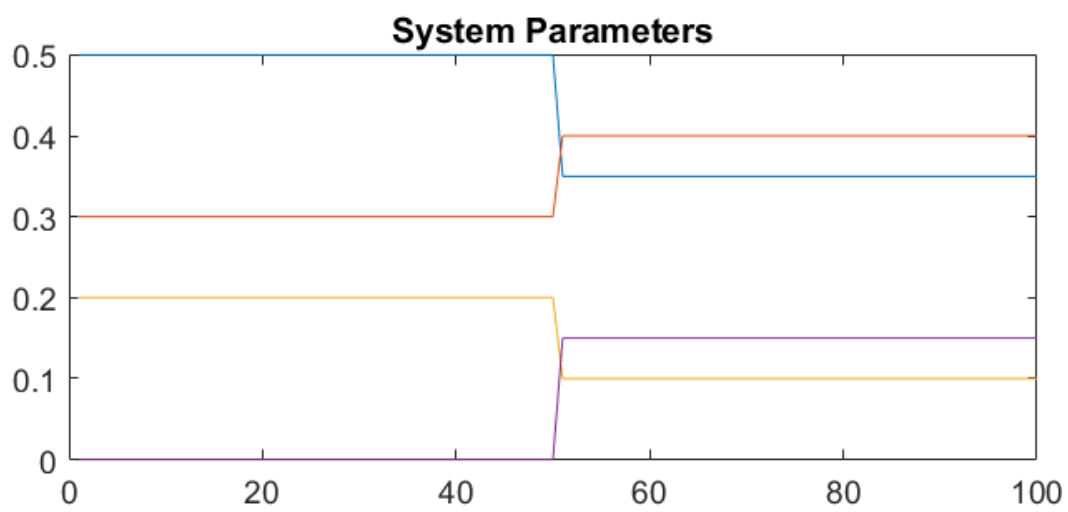
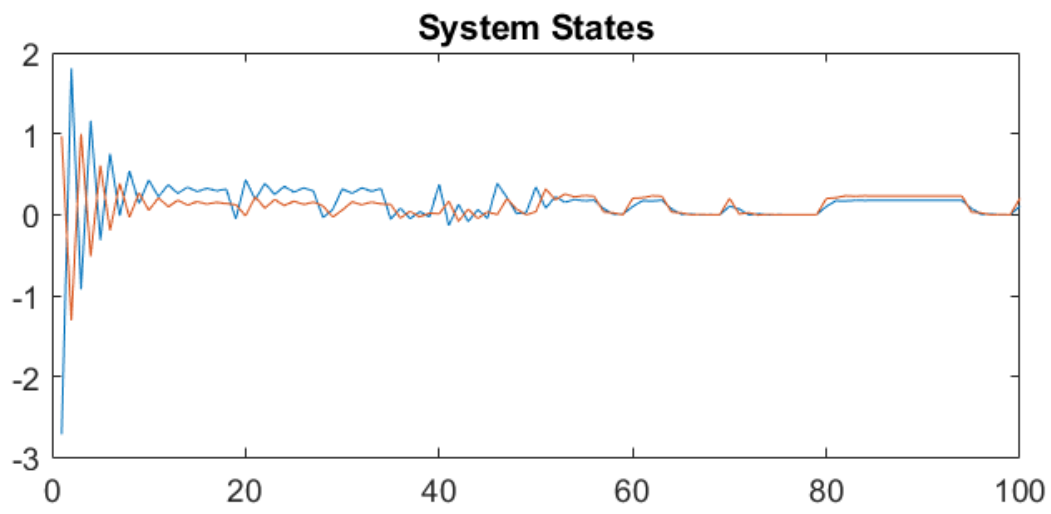


Figure 1: Repeated System Results

DE

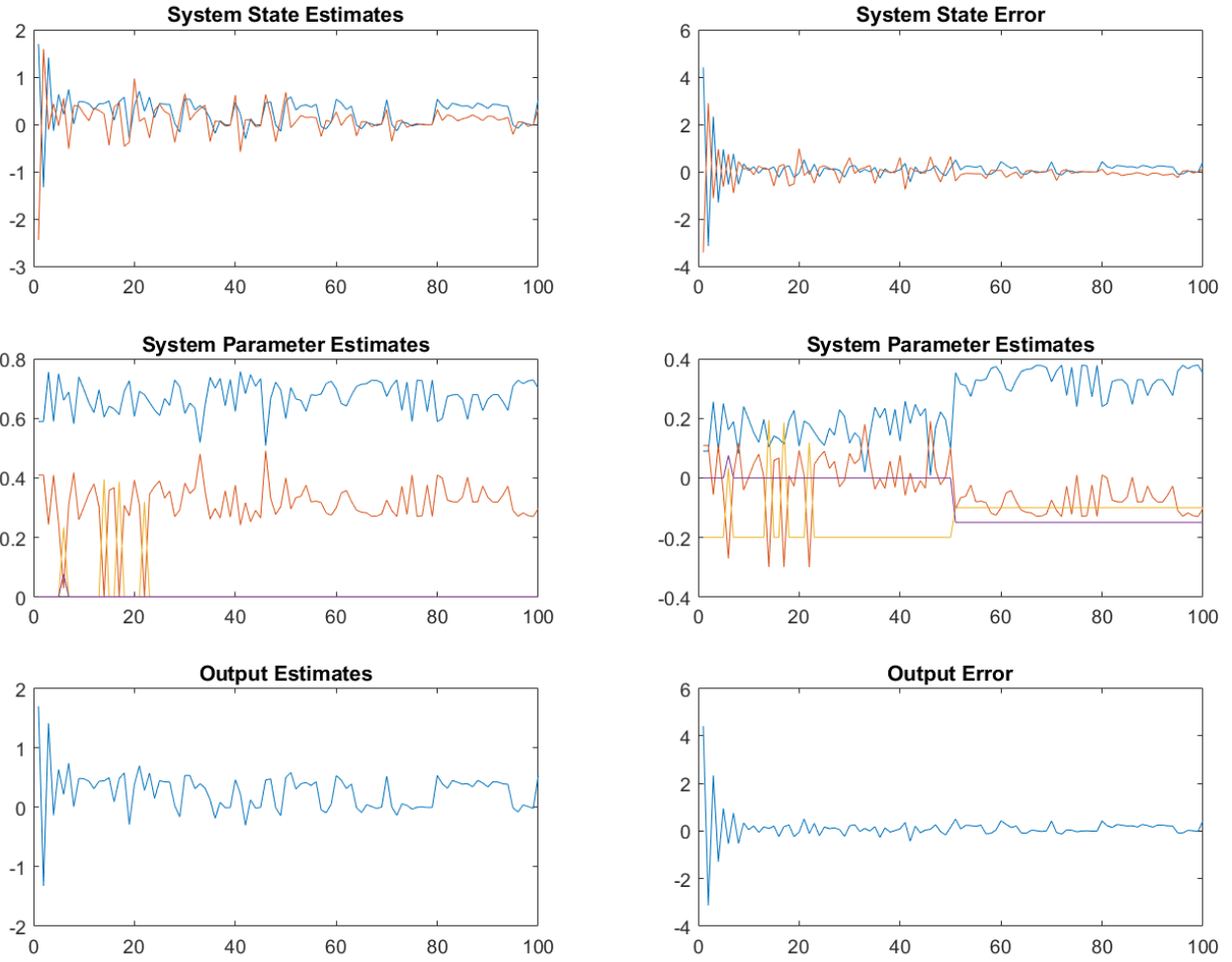


Figure 2: Recreated DE Simulated Results

EKF

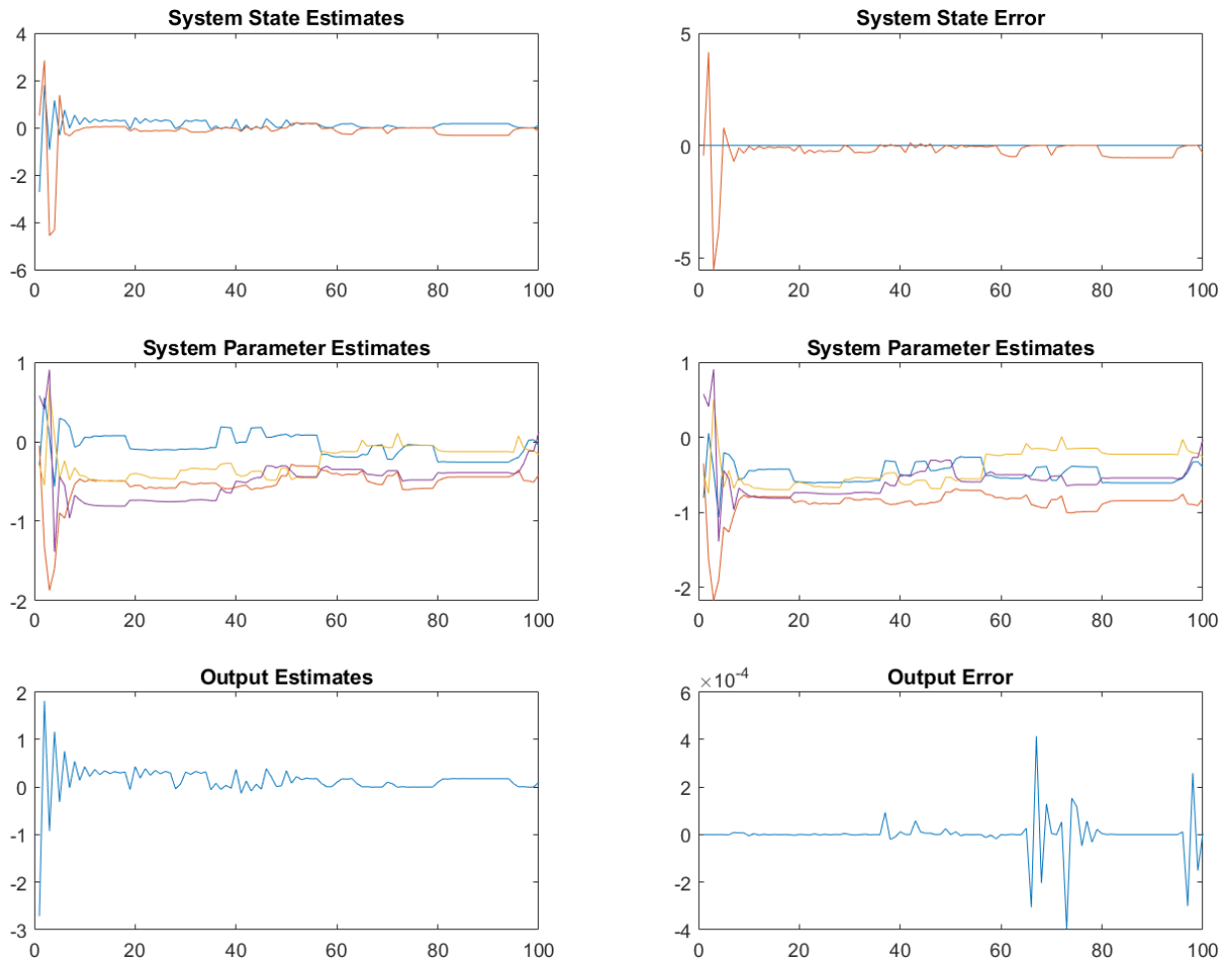


Figure 3: Recreated EKF Simulated Results

7 Results and Discussion

The results from the original paper were recreated in MATLAB (Appendix.1) and a complete collection of the plots can be seen attached in Appendix.2. The input was selected as essentially a random PWM signal of strength $u_0 = 0.5$ and multiple power levels of gaussian (attack) noise were tested.

A few important examples to look at are (as expected) the noise increases, the error increases for both of the methods of estimation. One big takaway can be seen in the ability for the EKF method to work better for really small deviations, but untimely, the state estimates are not stable and the error goes to infinity, while the DE method actually guarantees feasibility and convergence.

References

- [1] H. Beelen and M. Donkers, “Joint state and parameter estimation for discrete-time polytopic linear parameter-varying systems,” *IFAC-PapersOnLine*, vol. 50, no. 1, pp. 9778–9783, 2017.
- [2] L. Ljung, “Asymptotic behavior of the extended kalman filter as a parameter estimator for linear systems,” *IEEE Transactions on Automatic Control*, vol. 24, no. 1, pp. 36–50, 1979.
- [3] R. Orjuela, D. Ichalal, B. Marx, D. Maquin, and J. Ragot, “Chapter 9 - polytopic models for observer and fault-tolerant control designs,” in *New Trends in Observer-Based Control* (O. Boubaker, Q. Zhu, M. S. Mahmoud, J. Ragot, H. R. Karimi, and J. Dávila, eds.), Emerging Methodologies and Applications in Modelling, pp. 295–335, Academic Press, 2019.
- [4] R. Orjuela, B. Marx, J. Ragot, and D. Maquin, “Nonlinear system identification using heterogeneous multiple models,” *International Journal of Applied Mathematics and Computer Science*, vol. 23, no. 1, pp. 103–115, 2013.
- [5] Y. Bar-Shalom, X. R. Li, and T. Kirubarajan, *Estimation with applications to tracking and navigation: theory algorithms and software*. John Wiley & Sons, 2004.
- [6] W. P. M. H. Heemels, J. Daafouz, and G. Millerioux, “Observer-based control of discrete-time lpv systems with uncertain parameters ,” *IEEE Transactions on Automatic Control*, vol. 55, no. 9, pp. 2130–2135, 2010.

.1 MATLAB

All code I wrote for this project can be found on my GitHub repository:
https://github.com/jonaswagner2826/DT_LPV_attack_analysis

Script 1: DT_LPV_sim_script

```
1 % simulink Generation Script
2 % Jonas Wagner
3 % 2021-04-08
4 %
5 % Important References:
6 % -----
7 % https://www.mathworks.com/help/simulink/slref/add\_block.html
8 % https://www.mathworks.com/help/simulink/programmatic-modeling.html
9 % https://www.mathworks.com/help/simulink/referencelist.html
10 % https://www.mathworks.com/help/simulink/slref/common-block-parameters.html
11 % https://www.mathworks.com/help/simulink/slref/block-specific-parameters.html
12
13 clear
14 close all
15
16 %% Settings
17 generateModel = false;
18 openModel = true;
19 simulateModel = false;
20 plotResults = false;
21
22 % Name of the simulink model
23 [cfolder,~,~] = fileparts(mfilename('fullpath'));
24 subfolder = ''; %include / at end of subfolder
25 fname = 'DT_LPV_sim';
26
27
28 %% System Definitions
29
30 %-----
31 % DT_LPV Model Definition
32 % (https://www.sciencedirect.com/science/article/pii/S2405896317313459)
33
34 % DT Settings
35 k_max = 100;
36 K = 1:k_max; % Simulation Time
37
38 % System Size
39 n = 2; % States
40 m = 4; % Subsystems
41 p = 1; % Inputs
42 q = 1; % Outputs
43
44 % Polytopic Matrix definitions
45 A(:,:,1) = [-0.80, 0.25; 0.25,-0.30]; B(:,:,1) = [ 1.90; 0.00];
46 A(:,:,2) = [ 0.30, 0.70; 0.70, 0.00]; B(:,:,2) = [-1.00; 1.50];
47 A(:,:,3) = [-0.30, 0.65; 0.55, 0.10]; B(:,:,3) = [ 0.30;-2.00];
48 A(:,:,4) = [ 0.55,-0.20;-0.40,-0.30]; B(:,:,4) = [-0.60; 0.00];
```

```

49
50 A0 = [0,0;0,0]; B0 = [0;0];
51 A1 = [-0.80, 0.25; 0.25,-0.30]; B1 = [ 1.90; 0.00];
52 A2 = [ 0.30, 0.70; 0.70, 0.00]; B2 = [-1.00; 1.50];
53 A3 = [-0.30, 0.65; 0.55, 0.10]; B3 = [ 0.30;-2.00];
54 A4 = [ 0.55,-0.20;-0.40,-0.30]; B4 = [-0.60; 0.00];
55
56 % Output Eq
57 C = [1, 0]; D = 0;
58
59 % Input
60 u0 = 0.5;
61 T = 10;
62 U = zeros(1,k_max);
63 i = 1;
64 duty = 100;
65 for k = 1:k_max
66     if i == T
67         duty = 100*rand();
68         i = 1;
69     end
70     U(:,k) = square((2*pi*k)/T, duty);
71     i = i+1;
72 end
73 U = u0 * (U+1)/2;
74
75 % Attack (essentially just noise for this small system)
76 v0 = 0.5;
77 V = v0 * randn(q,k_max);
78
79
80
81 % Initial Conditions
82 x_0 = [0.25; -6.4];
83 x_hat_0 = [-1.25; 3.4];
84 alpha_hat_0 = [0.25; 0.25; 0.25; 0.25];
85
86
87
88 %% Rough Simulation of the System
89 % Sim Data
90 X = zeros(n,k_max);
91 Y = zeros(q, k_max);
92 Alpha = zeros(m,k_max);
93
94 % DE Data
95 X_hat_DE = zeros(n,k_max);
96 Alpha_hat_DE = zeros(m,k_max);
97 P_data_DE = zeros(2*n,2*n,k_max);
98 Phi = zeros(1,2*n,k_max);
99 Nu_hat = zeros(2*n,1,k_max);
100
101 % EKF Data
102 X_hat_EKF = zeros(n,k_max);

```

```

103 Alpha_hat_EKF = zeros(m,k_max);
104 P_data_EKF = zeros(n+m,n+m,k_max);
105
106
107 % Simulation Initialization
108 x = x_0;
109
110 % DE Initialization
111 x_hat_DE = x_hat_0;
112 alpha_hat_DE = alpha_hat_0;
113 P_DE = randn(2*n); P_DE = P_DE*P_DE.'; %random ??? or this rand one:
114 P_DE = [1.7552 1.7244 0.7521 0.4716;
115         1.7244 4.3679 -0.4500 -0.7000;
116         0.7521 -0.4500 3.5146 2.5992;
117         0.4716 -0.7000 2.5992 1.9602];
118 % nu_hat = randn(2*n,1); %random ??? or this rand one:
119 nu_hat = [0.3; -0.2; 0.1; -0.4];
120 % phi_hat = randn(2*n,1); %random ??? or this rand one: %[1.2424; -1.0667; 0.9337;
121     0.3503];
121 phi = zeros(1,m);
122
123 gamma = 0.9;
124
125 % EKF Initialization
126 x_hat_EKF = x_hat_0;
127 alpha_hat_EKF = alpha_hat_0;
128 P_EKF = 1e5 * eye(n+m); % random large P...
129 Q_EKF = diag([zeros(1,n), 100*ones(1,m)]);
130 R_EKF = 0.01;
131
132 for k = K(1:100)
133     % Plant Simulation
134     alpha = alpha_traj(k);
135     u = U(k);
136     x_old = x;
137     x = 0;
138     for i = 1:m
139         x = x + alpha(i) * (A(:, :, i) * x_old + B(:, i) * u);
140     end
141     y = C*x + D*u + V(:,k);
142
143
144     % Sim Data
145     X(:,k) = x;
146     Y(:,k) = y;
147     Alpha(:,k) = alpha;
148
149     % DE Method
150     if k >= 3
151         disp(['DE Method: k = ', num2str(k)])
152         P_DE_old = P_DE;
153         P_DE = (1/gamma)*P_data_DE(:, :, k-1) - (1/gamma)*(phi' ...
154             * (gamma + phi * P_data_DE(:, :, k-1) * phi')^(-1)*(phi * P_data_DE(:, :, k-1)));
155         if any(eig(inv(P_DE)) < 0)

```



```

156         warning('INV(P_DE) not PSD')
157
158     end
159
160     end
161 % phi = [fliplr(Y(:,(k-2):(k-1))), fliplr(U((k-2):(k-1)))];
162     phi_old = phi;
163     phi(2:n) = phi_old(1:n-1);
164     phi(1) = y;
165     phi(n+2:2*n) = phi_old(n+1:2*n-1);
166     phi(n+1) = u;
167     Phi(:, :, k) = phi;
168
169
170     [x_hat_DE] = est_DE(x_hat_DE, alpha_hat_DE, y, u,...
171                        P_DE, phi, nu_hat,...
172                        gamma, A, B, C);
173
174
175     % Param. Opt. Problem
176     disp('DE Parameter Estimation Started')
177     % attempting yalmip
178     yalmip('clear')
179     alpha_yalmip = sdpvar(m,1);
180     nu_alpha = sdpvar(2*n,1);
181     param_opt_sum = sdpvar(k,1);
182
183     Constraints = [sum(alpha_yalmip) == 1];
184     Constraints = [Constraints, nu_alpha == nuVector(alpha_yalmip)];
185
186     Constraints = [Constraints, nu_alpha == [
187     1;
188     -((3*alpha_yalmip(2))/10 - (11*alpha_yalmip(1))/10 - alpha_yalmip(3)/5 + alpha_yalmip
189         (4)/4 + 1);
190     0;
191     (alpha_yalmip(2) - (19*alpha_yalmip(1))/10 - (3*alpha_yalmip(3))/10 + (3*alpha_yalmip
192         (4))/5 + 1)
193     ]];
194
195     for i = 1:m
196         Constraints = [Constraints, alpha_yalmip(i) >= 0];
197     end
198     Constraints = [Constraints, param_opt_sum(1) == norm(Y(:,1) - Phi(1) * nu_alpha)^2];
199
200     for i = 2:k
201         Constraints = [Constraints, param_opt_sum(i) ...
202             == gamma * param_opt_sum(k-1) + norm(Y(:,k) - Phi(k) * nu_alpha)^2];
203     end
204
205     Objective = sum(param_opt_sum);
206
207     options = sdpsettings();

```

```

208     sol = optimize(Constraints,Objective,options);
209
210     if sol.problem == 0
211         % Extract and display value
212         alpha_hat_DE = value(alpha_yalmip);
213         nu_hat = value(nu_alpha);
214     else
215         disp('Hmm, something went wrong!');
216         sol.info
217         yalmiperror(sol.problem)
218         error('issue here.....')
219     end
220
221     % DE Data
222     X_hat_DE(:,k) = x_hat_DE;
223     Alpha_hat_DE(:,k) = alpha_hat_DE;
224     P_data_DE(:, :,k) = P_DE;
225     Nu_hat(:, :,k) = nu_hat;
226
227     % EKF Method
228     disp(['EKF Method: k = ', num2str(k)])
229     [x_hat_EKF, alpha_hat_EKF, P_EKF] = est_EKF(x_hat_EKF,...
230                                             alpha_hat_EKF,...
231                                             P_EKF, y, u,...
232                                             A, B, C, Q_EKF, R_EKF);
233
234     % EKF Data
235     X_hat_EKF(:,k) = x_hat_EKF;
236     Alpha_hat_EKF(:,k) = alpha_hat_EKF;
237     P_data_EKF(:, :,k) = P_EKF;
238 end
239
240
241 X = X(:,1:k);
242 Alpha = Alpha(:,1:k);
243 Y = Y(:,1:k);
244 X_hat_DE = X_hat_DE(:,1:k);
245 Alpha_hat_DE = Alpha_hat_DE(:,1:k);
246 X_hat_EKF = X_hat_EKF(:,1:k);
247 Alpha_hat_EKF = Alpha_hat_EKF(:,1:k);
248
249 figure('position', [0,0,500,750])
250 sgtitle(['Attack Power: ',num2str(v0)])
251 subplot(3,1,1)
252 plot(X')
253 title('System States')
254 subplot(3,1,2)
255 plot(Alpha')
256 title('System Parameters')
257 subplot(3,1,3)
258 plot(Y')
259 title('System Output')
260
261

```

```

262 figure('position', [0,0,1000,750])
263 sgtitle(['DE: Attack Power = ', num2str(v0)])
264 ax1 = subplot(3,2,1);
265 plot(X_hat_DE')
266 title('System State Estimates')
267 ax2 = subplot(3,2,3);
268 plot(Alpha_hat_DE')
269 title('System Parameter Estimates')
270 ax3 = subplot(3,2,5);
271 plot((C*X_hat_DE)')
272 title('Output Estimates')
273 ax4 = subplot(3,2,2);
274 plot(X_hat_DE'-X')
275 title('System State Error')
276 ax5 = subplot(3,2,4);
277 plot(Alpha_hat_DE'- Alpha')
278 title('System Parameter Estimates')
279 ax6 = subplot(3,2,6);
280 plot((C*X_hat_DE) '-Y')
281 title('Output Error')
282
283 figure('position', [0,0,1000,750])
284 sgtitle(['EKF: ', 'Attack Power = ', num2str(v0)])
285 ax1 = subplot(3,2,1);
286 plot(X_hat_EKF')
287 title('System State Estimates')
288 ax2 = subplot(3,2,3);
289 plot(Alpha_hat_EKF')
290 title('System Parameter Estimates')
291 ax3 = subplot(3,2,5);
292 plot((C*X_hat_EKF)')
293 title('Output Estimates')
294 ax4 = subplot(3,2,2);
295 plot(X_hat_EKF'-X')
296 title('System State Error')
297 ax5 = subplot(3,2,4);
298 plot(Alpha_hat_EKF'- Alpha')
299 title('System Parameter Estimates')
300 ax6 = subplot(3,2,6);
301 plot((C*X_hat_EKF) '-Y')
302 title('Output Error')

```

Script 2: alpha_traj

```
1 function [p] = alpha_traj(k)
2     %alpha_traj Function returns the parameter as a function of k
3     k_limit = 50;
4     p1 = [0.50; 0.30; 0.20; 0.00];
5     p2 = [0.35; 0.40; 0.10; 0.15];
6     if (k > k_limit)
7         p = p2;
8     else
9         p = p1;
10    end
11 end
```

```

1 function [x_hat, alpha_hat, nu_hat] ...
2                                     = est_DE(x_hat, alpha_hat, y, u,...
3                                           P, phi, nu_hat,...
4                                           gamma, A, B, C)
5
6 %est_JSPE Function performs a single iteration of the joint state and
7 %parameter estimator
8
9 arguments
10 x_hat %x_hat_{k-1} (n,1)
11 alpha_hat %\alpha_hat_{k-1} (m,1)
12 y %y_k (1,1)
13 u %u_k (1,1)
14 P %P_k (2n,2n)
15 phi %phi_k (2n,1)
16 nu_hat %nu_hat_k (1,2n)
17 gamma = 0.9
18 A = string(-1)
19 B = string(-1)
20 C = [1, 0]
21 % nu_alpha = -1
22 end
23
24 %% System Definition
25 if string(A) == string(-1)
26     clear A
27     A(:,:,1) = [-0.80, 0.25; 0.25,-0.30];
28     A(:,:,2) = [ 0.30, 0.70; 0.70, 0.00];
29     A(:,:,3) = [-0.30, 0.65; 0.55, 0.10];
30     A(:,:,4) = [ 0.55,-0.20;-0.40,-0.30];
31 end
32 if string(B) == string(-1)
33     clear B
34     B(:,:,1) = [ 1.90; 0.00];
35     B(:,:,2) = [-1.00; 1.50];
36     B(:,:,3) = [ 0.30;-2.00];
37     B(:,:,4) = [-0.60; 0.00];
38 end
39
40 % System Dimenstions
41 n = size(A,1);
42 m = size(A,3);
43 p = size(B,2);
44 q = size(C,1);
45
46 %% State Estimation
47 % CVX Feasability Problem on LMI
48 disp('DE State Estimation Started')
49 tol = 1e-6;
50 cvx_begin sdp quiet
51     variable P_cvx(n,n,m) symmetric
52     variable F(n,q,m)
53     variable G(n,n,m)

```

```

54     variable zeta
55     subject to
56         for i = 1:m
57             for j = 1:m
58                 [G(:,:,i) + G(:,:,i)' - P_cvx(:,:,j), zeros(n), G(:,:,i)*A(:,:,i) + F
                    (:,:,i)*C, G(:,:,i);
59                 zeros(n), eye(n), eye(n), zeros(n);
60                 A(:,:,i)'*G(:,:,i)' + C'*F(:,:,i)', eye(n), P_cvx(:,:,i), zeros(n);
61                 G(:,:,i)', zeros(n), zeros(n), zeta*eye(n)] >= tol*eye(4*n);
62             end
63         end
64         zeta >= 1;
65     cvx_end
66
67     % Gain Result Calc
68     for i = 1:m
69         L(:,:,i) = inv(G(:,:,i))*F(:,:,i);
70     end
71
72     % State Estimate Update
73     x_hat_old = x_hat; % x_hat_{k-1}
74     x_hat = 0;
75     for i = 1:m
76         x_hat = x_hat + alpha_hat(i)*(A(:,:,i)*x_hat_old + B(:,i)*u)...
77             + L(:,:,i)*(C*x_hat_old - y);
78     end
79     %Note: x_hat = x_hat k
80
81     %% Parameter Estimation
82
83     % % Phi Matrix Definitions
84     % if string(nu_alpha) == string(-1)
85     % Alpha_sym = sym('alpha',[m,1]);
86     % A_bar = zeros(n);
87     % B_bar = zeros(n,p);
88     % for i = 1:m
89     % A_bar = A_bar + Alpha_sym(i) * A(:,:,i);
90     % B_bar = B_bar + Alpha_sym(i) * B(:,:,i);
91     % end
92     % a_coeff = charpoly(A_bar);
93     % temp = sym('temp');
94     % b_coeff = charpoly(C*adjoint(temp*eye(n) - A_bar)*B_bar);
95     % if size(a_coeff,2) <= n
96     % a_coeff = [zeros(1,n+1-size(a_coeff,2)),a_coeff];
97     % end
98     % if size(b_coeff,2) <= n
99     % b_coeff = [zeros(1,n+1-size(b_coeff,2)),b_coeff];
100    % end
101    % nu_alpha = matlabFunction([fliplr(a_coeff(1:n)), fliplr(b_coeff(1:n))]);
102    % end
103
104
105    % % Est. Opt. Problem
106    % disp('DE Parameter Estimation Started')

```

```

107 % % attempting yalmip
108 % yalmip('clear')
109 % Alpha = sdpvar(m,1);
110 % % nu_alpha = sdpvar(2*n,1);
111 %
112 % Constraints = [sum(Alpha) == 1];
113 % % Constraints = [Constraints, nu_alpha == nuVector(Alpha)];
114 % for i = 1:m
115 % Constraints = [Constraints, Alpha(i) >= 0];
116 % end
117 %
118 % Objective = (nu_hat - nu_alpha)' * inv(P) * (nu_hat - nu_alpha);
119 %
120 % options = sdpsettings();%'debug',1,'verbose',1)%,'solver','quadprog','quadprog.maxiter
    ',100);
121 %
122 % sol = optimize(Constraints,Objective,options)
123 %
124 %
125 % % cvx_begin quiet
126 % % variable alpha_cvx(m,1)
127 % % nu_alpha = nu_alpha(alpha_cvx);
128 % %
129 % % % Minimize This
130 % % minimize((nu_hat - nu_alpha)' * inv(P) * (nu_hat - nu_alpha))
131 % % subject to
132 % % sum = 0;
133 % % for i = 1:m
134 % % alpha_cvx(i) >= 0;
135 % % sum = sum + alpha_cvx(i);
136 % % end
137 % % sum == 1;
138 % % cvx_end
139 % if sol.problem == 0
140 % % Extract and display value
141 % alpha_hat = value(Alpha) %alpha_cvx;
142 % nu_hat = value(nu_alpha)
143 % else
144 % disp('Hmm, something went wrong!');
145 % sol.info
146 % yalmiperror(sol.problem)
147 % error('issue here.....')
148 % end
149
150
151
152 % % k step
153 % P_old = P;
154 % phi_old = phi;
155 % nu_hat_old = nu_hat;
156 %
157 %
158 % % % k+1 step
159 % % phi(2:n) = phi_old(1:n-1);

```

```
160 % % phi(1) = y;
161 % % phi(n+2:2*n) = phi_old(n+1:2*n-1);
162 % % phi(n+1) = u;
163 %
164 % % Recursive Update
165 % P = (1/gamma)*P_old - (1/gamma)*(phi'...
166 % * inv(gamma + phi * P_old * phi') * phi * P_old);
167 % nu_hat = nu_hat_old + P * phi' * (y - phi*nu_hat_old);
168
169
170
171
172
173 end
```


Script 4: est_EKF

```

1 function [x_hat, alpha_hat, P] = est_EKF(x_hat, alpha_hat, P, y, u,...
2     A, B, C, Q, R)
3     %est_EKF Function performs a single iteration of the parameter and
4     %state estimation using an EKF
5
6     arguments
7         x_hat
8         alpha_hat
9         P
10        y
11        u
12        A = string(-1)
13        B = string(-1)
14        C = [1, 0]
15        Q = string(-1)
16        R = string(-1)
17    end
18
19    % System Matrices
20    if string(A) == string(-1)
21        clear A
22        A(:,:,1) = [-0.80, 0.25; 0.25,-0.30];
23        A(:,:,2) = [ 0.30, 0.70; 0.70, 0.00];
24        A(:,:,3) = [-0.30, 0.65; 0.55, 0.10];
25        A(:,:,4) = [ 0.55,-0.20;-0.40,-0.30];
26    end
27    if string(B) == string(-1)
28        clear B
29        B(:,:,1) = [ 1.90; 0.00];
30        B(:,:,2) = [-1.00; 1.50];
31        B(:,:,3) = [ 0.30;-2.00];
32        B(:,:,4) = [-0.60; 0.00];
33    end
34
35    % System Dimenstion Matrices
36    n = size(A,1);
37    m = size(A,3);
38    p = size(B,1);
39    q = size(C,1);
40
41
42    % EKF Covariance Matrices
43    if string(Q) == string(-1)
44        clear Q
45        Q = diag([zeros(1,n), 100*ones(1,m)]);
46    end
47    if string(R) == string(-1)
48        clear R
49        R = 0.01;
50    end
51
52    % A_hat (k-1) calc
53    A_hat = diag([zeros(1,n), ones(1,m)]);

```

```

54     for i = 1:m
55         A_hat(1:n,1:n) = A_hat(1:n,1:n) + alpha_hat(i) * A(:, :, i);
56         A_hat(1:n,n+i) = A(:, :, i) * x_hat + B(:, :, i) * u;
57     end
58
59
60     % Prediction Step
61     x_hat_pre = 0;
62     for i = 1:m
63         x_hat_pre = x_hat_pre + alpha_hat(i) * (A(:, :, i) * x_hat + B(:, :, i) * u);
64     end
65     alpha_hat_pre = alpha_hat;
66     P_pre = A_hat * P * A_hat' + Q;
67
68     % L (k) calc
69     C_hat = [C, zeros(1,m)];
70     L = P_pre * C_hat' * inv(R + C_hat * P_pre * C_hat');
71
72     % Update Step
73     x_alpha_pre = [x_hat_pre; alpha_hat_pre];
74     x_alpha_post = x_alpha_pre + L * (y - C * x_hat_pre);
75     P_post = (eye(n+m) - L * C_hat) * P_pre;
76
77     % Estimates
78     x_hat = x_alpha_post(1:n);
79     alpha_hat = x_alpha_post(n+1:n+m);
80     P = P_post;
81 end

```

.2 Complete Set of Results

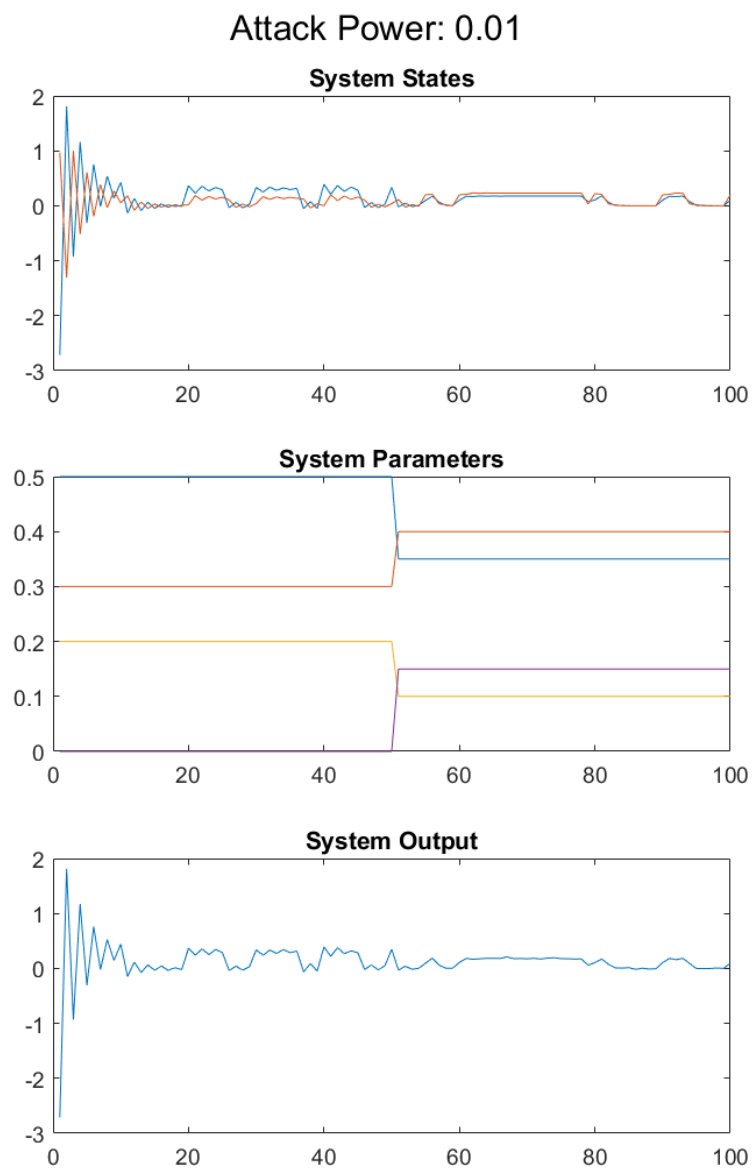


Figure 4: $v_0 = 0.01$ System Results

DE: Attack Power = 0.01

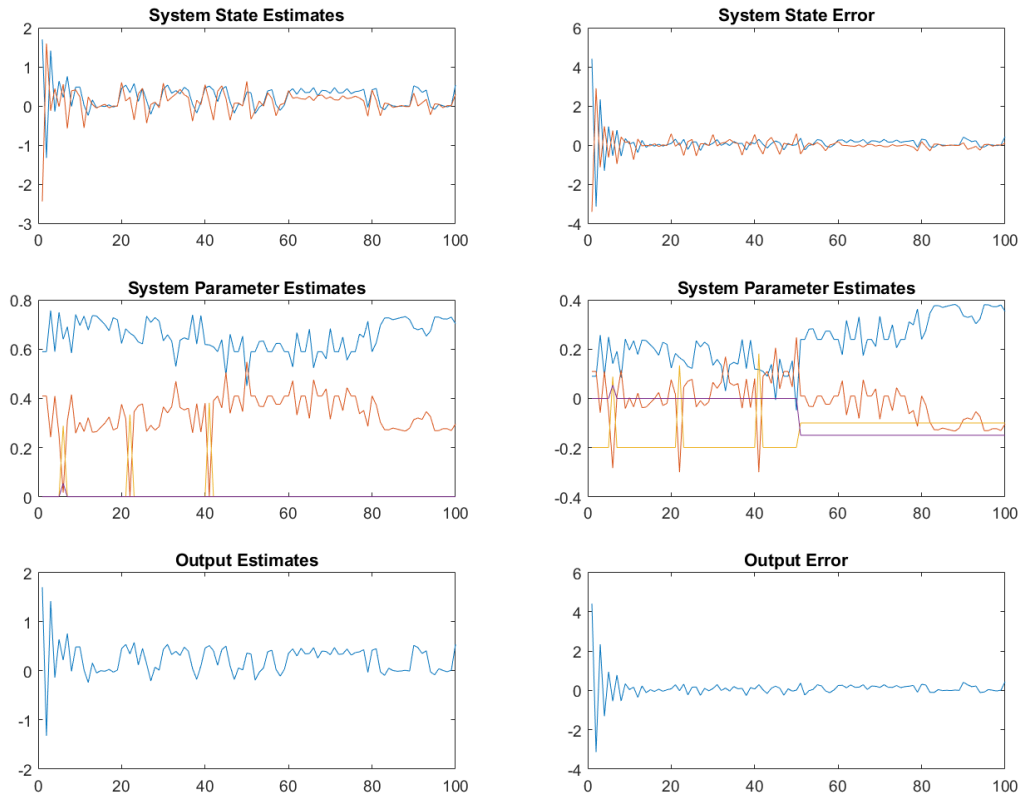


Figure 5: DE with $v_0 = 0.01$ Simulated Results

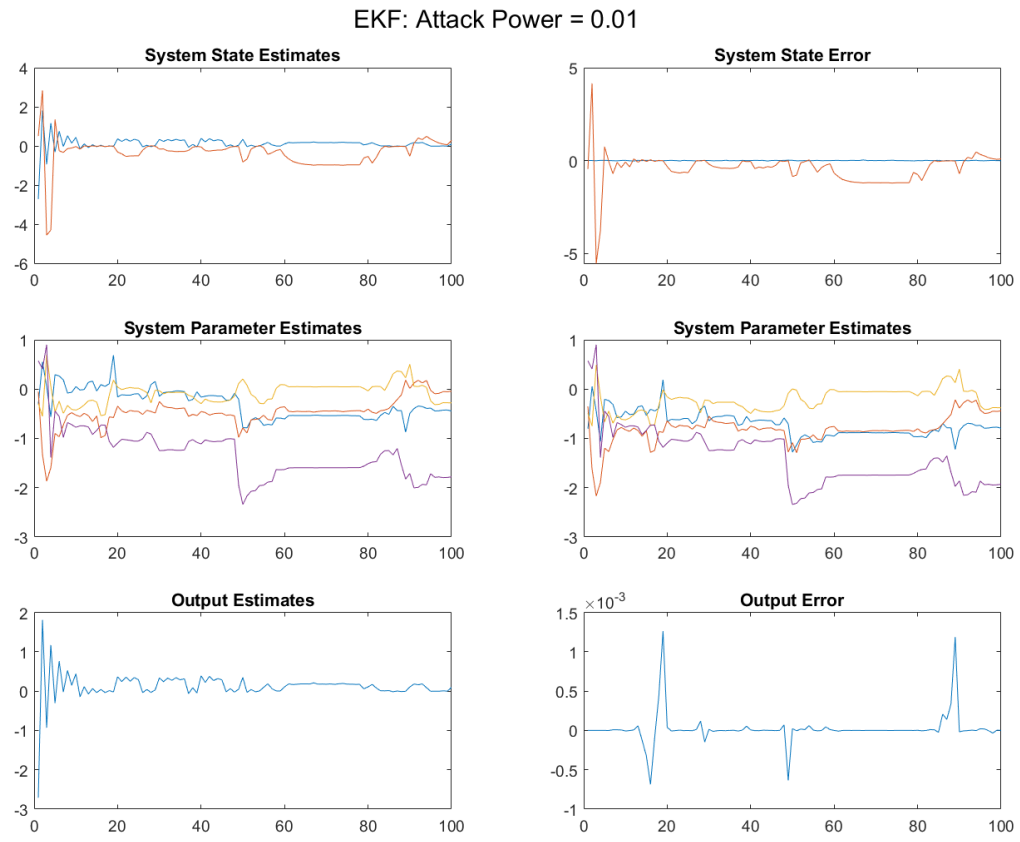


Figure 6: EKF with $v_0 = 0.01$ Simulated Results

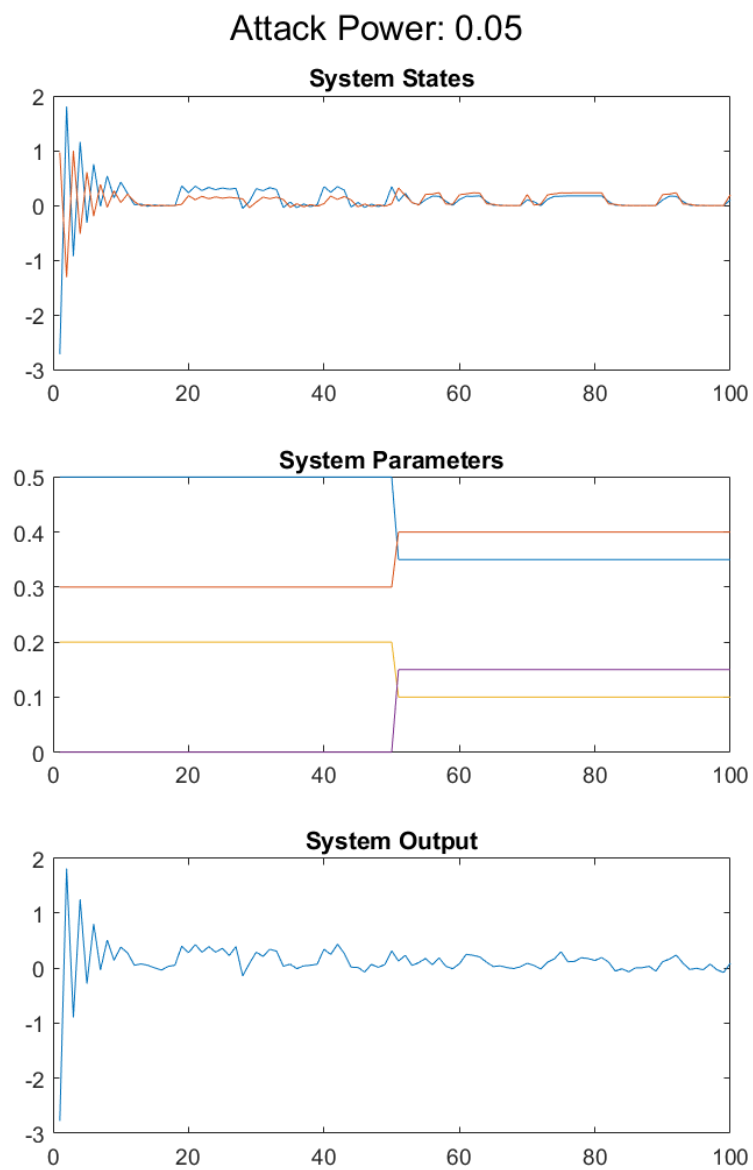


Figure 7: $v_0 = 0.05$ System Results

DE: Attack Power = 0.05

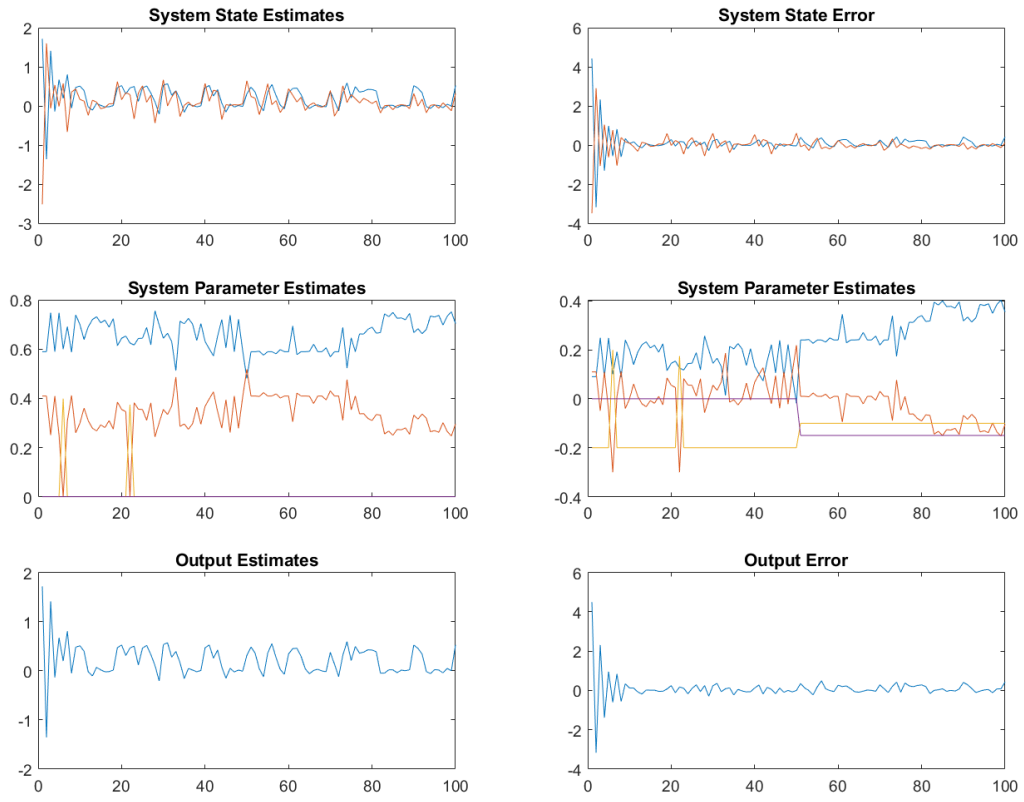


Figure 8: DE with $v_0 = 0.05$ Simulated Results

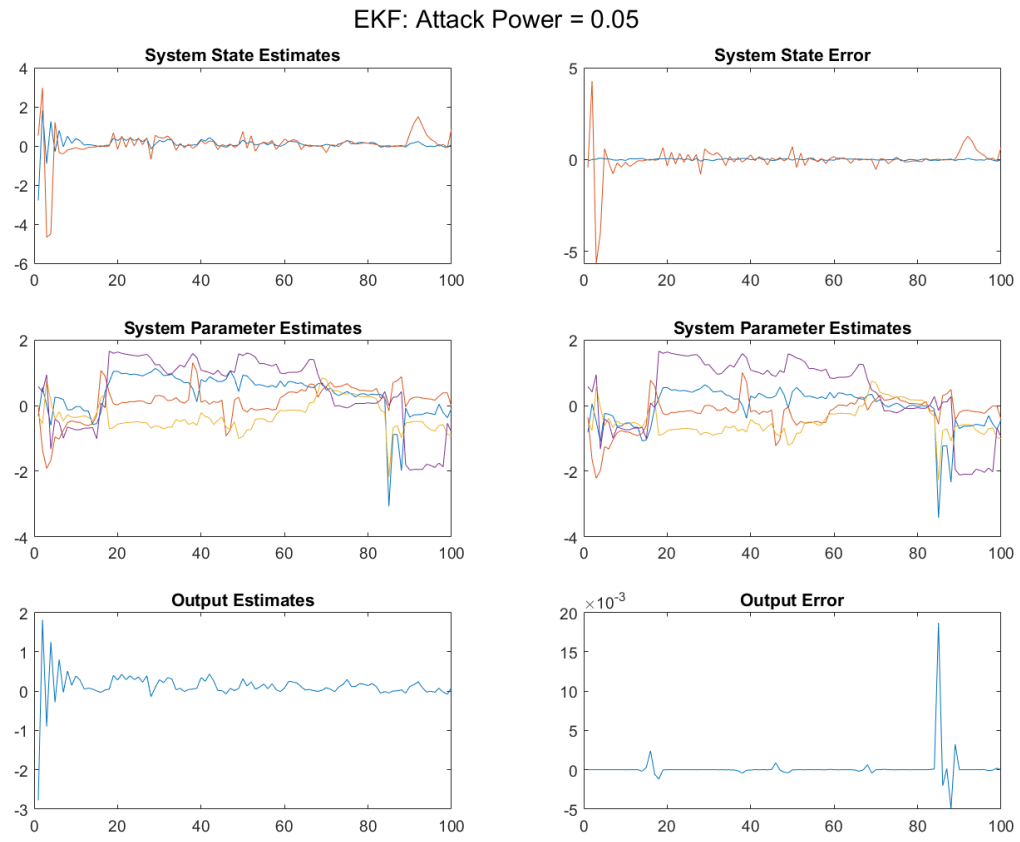


Figure 9: EKF with $v_0 = 0.05$ Simulated Results

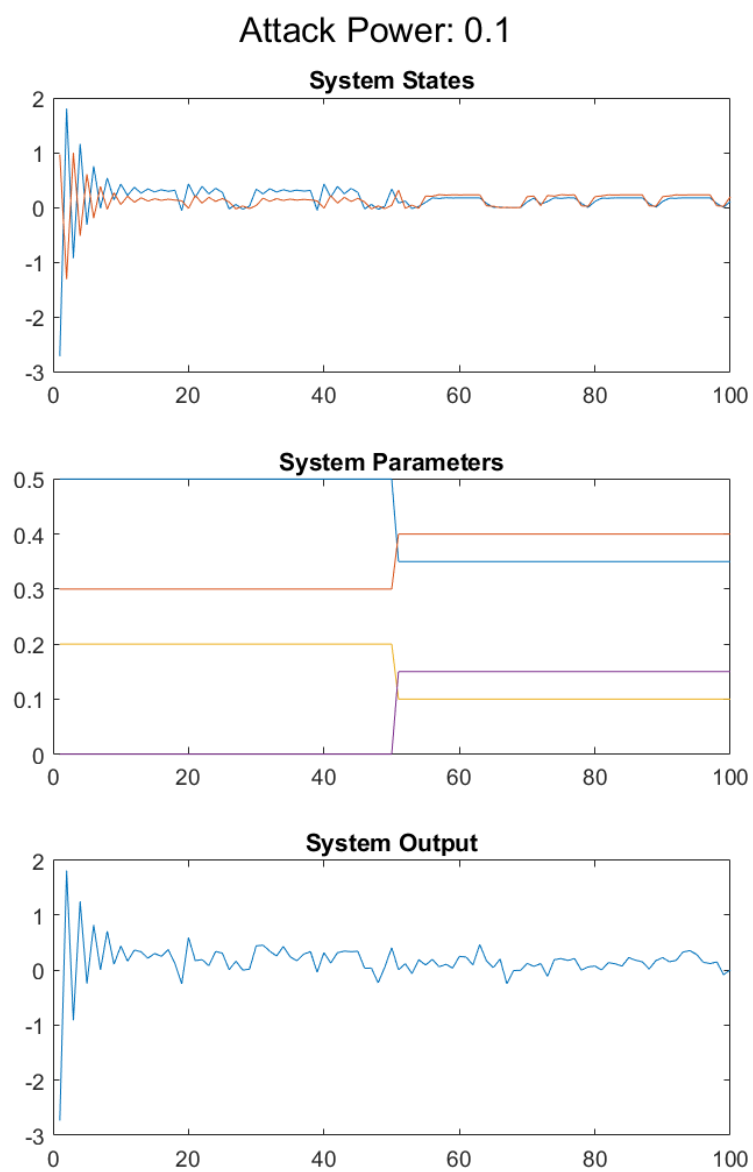


Figure 10: $v_0 = 0.1$ System Results

DE: Attack Power = 0.1

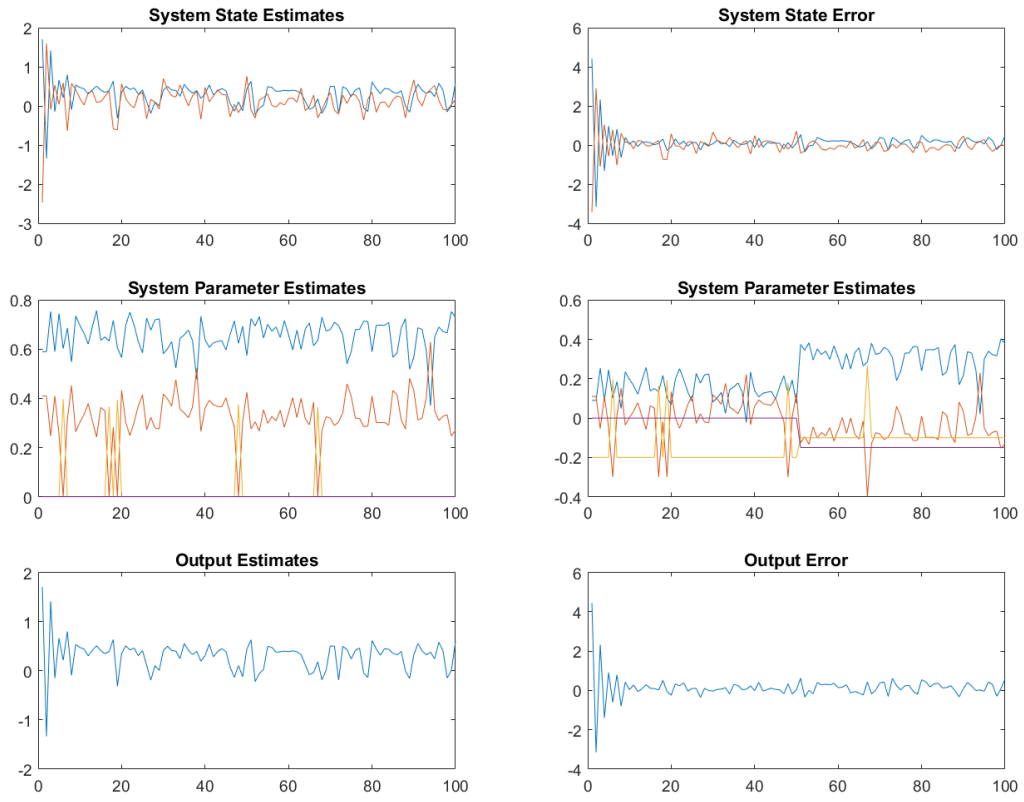


Figure 11: DE with $v_0 = 0.1$ Simulated Results

EKF: Attack Power = 0.1

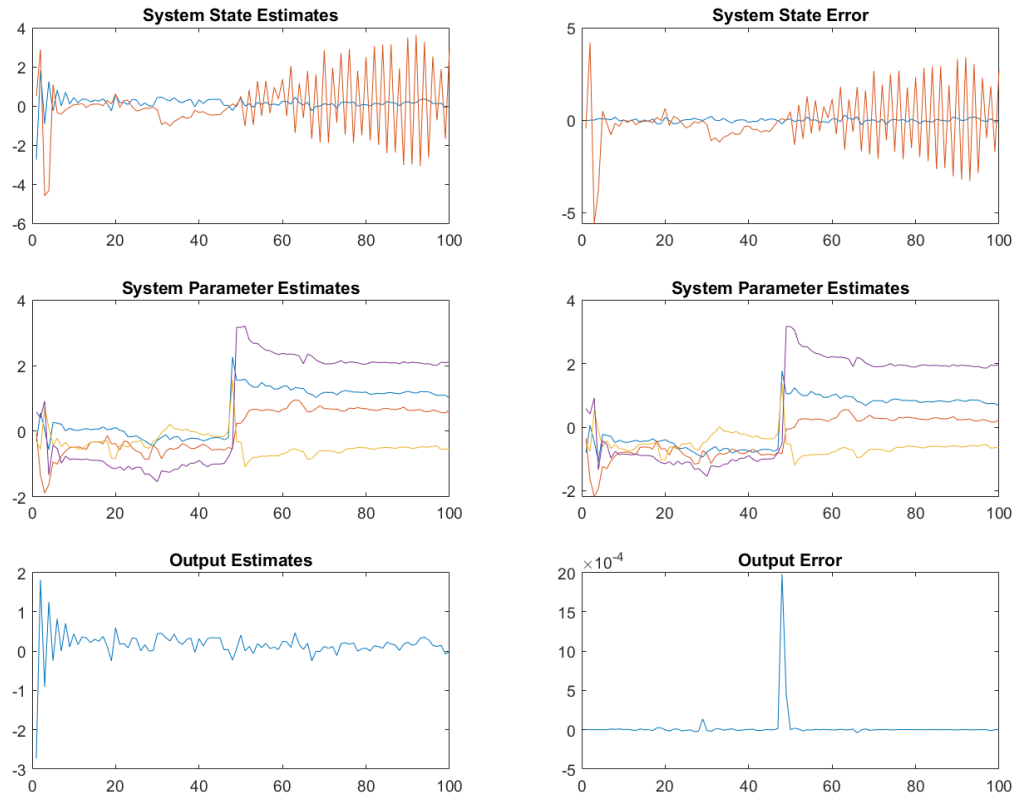


Figure 12: EKF with $v_0 = 0.1$ Simulated Results

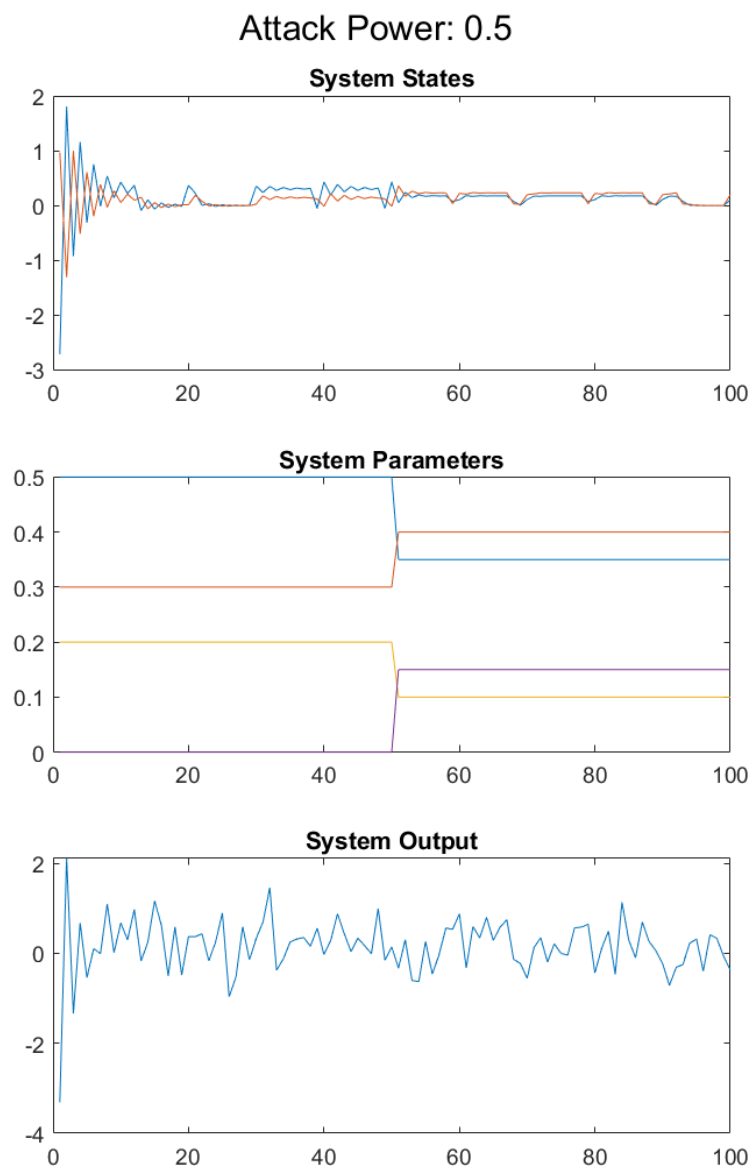


Figure 13: $v_0 = 0.5$ System Results

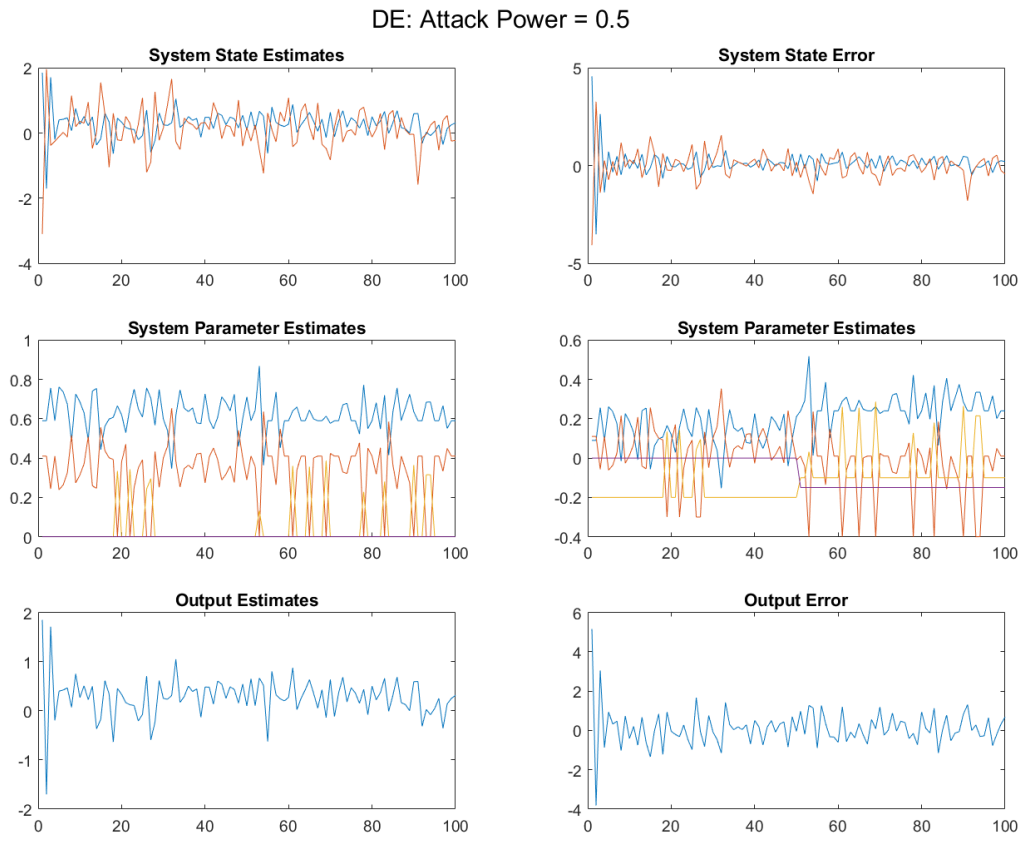


Figure 14: DE with $v_0 = 0.5$ Simulated Results

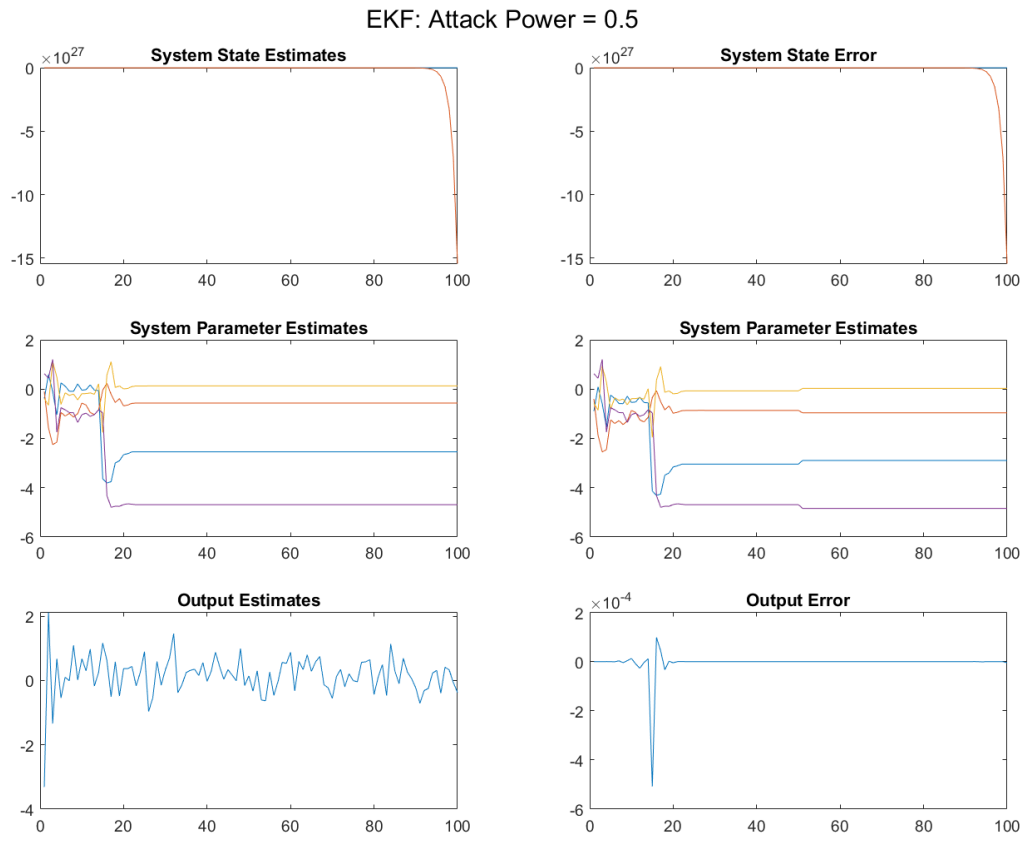


Figure 15: EKF with $v_0 = 0.5$ Simulated Results