ARL-MR-1128 ● JUNE 2025

Pontryagin Neural Network Solutions for Hamilton–Jacobi–Bellman Optimality of High-Order Systems

by Bradley T. Burchett

NOTICES

Disclaimers

ARL-MR-1128 ● June 2025

Pontryagin Neural Network Solutions for Hamilton–Jacobi–Bellman Optimality of High-Order Systems

Bradley T. Burchett
DEVCOM Army Research Laboratory

REPORT DOCUMENTATION PAGE

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED START DATE	END DATE
June 2025	Memorandum Report	10/01/2024	12/19/2024

4. TITLE AND SUBTITLE
Pontryagin Neural Network Solutions for Hamilton–Jacobi–Bellman Optimality of High-Order Systems

5a. CONTRACT NUMBER	5b. GRANT NUMBER	5c. PROGRAM ELEMENT NUMBER

5d. PROJECT NUMBER	5e. TASK NUMBER	5f. WORK UNIT NUMBER

6. AUTHOR(S)
Bradley T. Burchett

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)	8. PERFORMING ORGANIZATION REPORT NUMBER
DEVCOM Army Research Laboratory ATTN: FCDD-RLA-WD Aberdeen Proving Ground, MD 21005	ARL-MR-1128

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
DISTRIBUTION STATEMENT A. Approved for public release: distribution unlimited.

13. SUPPLEMENTARY NOTES
ORCID: Bradley T. Burchett, 0000-0002-1934-0537

14. ABSTRACT
Recently, Physics Informed Neural Networks (PINNs) have come to the forefront as a novel means to solve partial differential equations (PDEs). Nonlinear optimal control problems often require the solution of a nonlinear PDE to satisfy necessary conditions for optimality. PINNs can be trained to solve such a PDE; however, for high-dimensional systems, discretizing the operating space leads to very large training datasets unless current space-filling methods are used. The use of the rank one lattice method is shown to reduce the training set size. In this report, analytic gradients are derived and used to streamline the gradient-based neural network training required to solve the Hamilton–Jacobi–Bellman (HJB) PDE. These two methods provide a means of offline training to render a PINN-based solution to HJB optimal control. The method is demonstrated on a four-state nonlinear chaotic system.

15. SUBJECT TERMS
neural network, machine learning, optimal control, design of experiments, Weapons Sciences

16. SECURITY CLASSIFICATION OF: a. REPORT	b. ABSTRACT	C. THIS PAGE	17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UU	28

19a. NAME OF RESPONSIBLE PERSON	19b. PHONE NUMBER *(Include area code)*
Bradley T. Burchett	(520) 691-5110

STANDARD FORM 298 (REV. 5/2020)
Prescribed by ANSI Std. Z39.18

Contents

List of Figures

1. Introduction

Artificial neural networks, specifically Physics Informed Neural Networks (PINNs),[1,2] show promise in solving nonlinear optimal control problems through direct solution of the Hamilton–Jacobi–Bellman (HJB) partial differential equation (PDE). In previous work,[3] we demonstrated such an approach for a two-state system with limited operational envelope. That solution relied on the low dimensionality of the system in leveraging 1) a full-factorial discretization of the operational space for training and 2) a fine enough discretization to allow finite differencing to render accurate gradients required in the solution.

This work improves on the previous method by 1) finding analytic second derivatives through forward- and back-propagation, 2) employing the method of successive approximation to simplify nonlinear least-squares iterations, and 3) starting the iteration with a stable control law.[4]

Nonlinear System Dynamics

In this report, we consider a system described by the nonlinear ordinary differential equation (ODE) 1[2,4]

$$\dot{\boldsymbol{x}} = \boldsymbol{f}(\mathbf{x}) + \boldsymbol{g}(\mathbf{x})\boldsymbol{u} \tag{1}$$

Where $\boldsymbol{f}(\mathbf{x})$ and $\boldsymbol{g}(\mathbf{x})$ are vector valued nonlinear functions of the system state vector. Note that the dynamics are *affine* in the control, $\boldsymbol{u}$.

Next, consider the infinite horizon optimal control problem

$$J(\boldsymbol{x}) = \int_{\mathbf{0}}^{\infty} (Q(\boldsymbol{x}) + \boldsymbol{u}^T \boldsymbol{R} \boldsymbol{u})\, dt \tag{2}$$

Where $Q(\boldsymbol{x}) \geq 0$ is positive semi-definite, and $\boldsymbol{R} = \boldsymbol{R}^T > 0$ penalizes the control effort. The optimal control can be determined by solution of the HJB Equation (3)[4]:

$$\begin{gathered} \sup_{\mathbf{u} \in U} \{-V_x^T \dot{\mathbf{x}} - \mathcal{L}(t, \mathbf{x}, \mathbf{u})\} = 0 \\ \text{subject to} \\ V(\mathbf{0}) = 0 \end{gathered} \tag{3}$$

The optimal control $\mathbf{u}^*$ is derived with respect to the value function V; that is,

$$\mathbf{u}^* = \underset{\mathbf{u} \in U}{\operatorname{argsup}} \{-V_x^T \dot{\mathbf{x}} - \mathcal{L}(t, \mathbf{x}, \mathbf{u})\} = -\frac{1}{2} \mathbf{R}^{-1} \mathbf{g}(\mathbf{x}) V_x \tag{4}$$

Once $\mathbf{u}^*$ is derived, its closed analytical form is substituted into Equation (3). Then, Equation (3) reduces to the nonlinear PDE Equation (5), which is solved for V.

$$\mathcal{R}(\mathbf{x}, V, V_x) = \mathbf{Q}(\mathbf{x}) + V_x^T \mathbf{f}(\mathbf{x}) - \frac{1}{4} V_x^T \mathbf{g}(\mathbf{x}) \mathbf{R}^{-1} \mathbf{g}^T(\mathbf{x}) V_x = 0$$

$$\text{subject to} \tag{5}$$

$$V(\mathbf{0}) = 0$$

PINNs have recently become prominent in the machine learning literature.[1,2,4] These networks incorporate constitutive physical relationships into network training to limit solutions to ones that are physically meaningful. A particularly useful application of a PINN is to solve a nonlinear PDE. We seek to leverage a PINN to solve the HJB PDE. Section 2 presents two improvements on our previous solution.

2. Improving the Previous Method

2.1 Analytic Second Derivatives

Consider Equation (5) and suppose we want to solve it using a neural network. The objective V can be modeled by a PINN with the state vector x as network input. Thus, the term V_x is the sensitivity of the network output with respect to its inputs. Neural network training focuses on sensitivities of the network output with respect to network weights. Back-propagation algorithms are thus tailored to finding sensitivities with respect to the network weights exclusively. The application of a PINN to fitting a PDE presents the need to find sensitivities with respect to network inputs as well as the weights, and derivatives of V_x wrt the weights (which are second derivatives of V). More directly, if we take the gradient of Equation (5) with respect to the network weights, we get

$$\boldsymbol{J} = \frac{\partial \mathcal{R}}{\partial \boldsymbol{w}} = \frac{\partial V_x^T}{\partial \boldsymbol{w}} \mathbf{f}(\mathbf{x}) - \frac{1}{2} \frac{\partial V_x^T}{\partial \boldsymbol{w}} \mathbf{g}(\mathbf{x}) \mathbf{R}^{-1} \mathbf{g}^T(\mathbf{x}) V_x \tag{6}$$

Thus, making it necessary to find the term $\frac{\partial V_x^T}{\partial \boldsymbol{w}}$. In our previous work,[3] we found V_x^T for a problem with inputs gridded over the input domain; then, found $\frac{\partial V_x^T}{\partial \boldsymbol{w}}$ simply by finite differencing $\frac{\partial V^T}{\partial \boldsymbol{w}}$ and $\boldsymbol{x}$ over the input grid. However, when it is impossible or impractical to sample the input space over a factorial grid and finite difference, an analytic algorithm for $\frac{\partial V_x^T}{\partial \boldsymbol{w}}$ is necessary. We can derive such an algorithm by differentiating the steps of forward- and back-propagation with respect to the inputs.

Consider the static feed-forward network shown in Figure 1.[3]

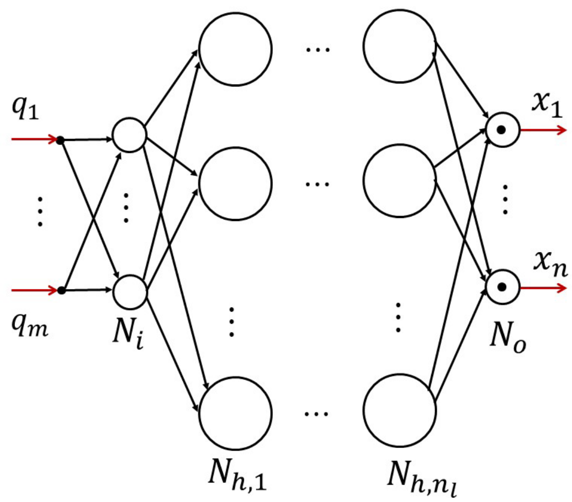


Figure 1. Multilayer feed-forward network with n_l hidden layers.

Assume the following constraints on the network architecture:

- The input layer has N_i units with tanh σ activation.
- There are n_l hidden layers that all have equal numbers of neurons N_h.
- There are N_o output nodes with identity activation $f_o(\sigma_0) = \sigma_o$.
- All wires (synapses) are adjustable connecting weights.
- All hidden nodes have bias b_k.
- Output nodes have no bias.
- Input nodes may or may not have bias.
- All nodes in layer $l - 1$ are connected to all nodes in layer l.

Note that a synapse output is determined by its input from the layer on its left times its synaptic weight.

$$s_k = w_{kj} x_j \tag{7}$$

Equation (7) relates the three quantities of input, gain, and output. The output of layer k is determined by $x_k = f(\sigma_k)$ where σ_k can be written as a sum over the synapse values fed from the previous layer,

$$\sigma_k = \sum_{N_j} w_{kj} x_j + b_k \tag{8}$$

Equation (8) can be written in matrix-vector form,

$$\boldsymbol{\sigma}_l = \begin{bmatrix} \mathbf{w}_{l,l-1}^T & \mathbf{b}_l \end{bmatrix} \cdot \begin{Bmatrix} \mathbf{x}_{l-1} \\ 1 \end{Bmatrix} \tag{9}$$

and each node (neuron) in the input and hidden layers has its output determined by the activation function,

$$x_k = f_k(\sigma_k) \text{ where } f_k(\sigma_k) = \tanh \sigma_k \tag{10}$$

for the applications in this report.

Thus, for multilayer feed-forward networks, Equations (9) and (10) are applied repeatedly.

$$\hat{\mathbf{y}} = \mathbf{x}_o = \mathbf{f}_o(\mathbf{w}_o \cdot \mathbf{f}_{o-1}(\mathbf{w}_{o-1}^T \cdot \mathbf{f}_{\boldsymbol{i}}(\cdots) + \mathbf{b}_{o-1}))$$

For one hidden layer we have

$$\hat{\mathbf{y}} = \mathbf{x}_o = \mathbf{f}_o(\mathbf{w}_o \cdot \mathbf{f}_h(\mathbf{w}_h^T \cdot \mathbf{f}_{\boldsymbol{i}}(\mathbf{w}_h^T \cdot \mathbf{x}_{\boldsymbol{i}} + \mathbf{b}_{\boldsymbol{i}}) + \mathbf{b}_h)) \tag{11}$$

Differentiating Equation (9), we can write

$$\frac{\partial \boldsymbol{\sigma}_l}{\partial \mathbf{x}_{l-1}} = \mathbf{w}_{l,l-1}^T \tag{12}$$

We observe further that for hyperbolic tangent (tanh) activation, the sensitivity of a single-layer output with respect to its inputs is found via the chain rule

$$\frac{\partial \mathbf{y}_l}{\partial \mathbf{x}_{l-1}} = \frac{\partial \mathbf{y}_l}{\partial \boldsymbol{\sigma}_l} \frac{\partial \boldsymbol{\sigma}_l}{\partial \mathbf{x}_{l-1}} = \mathbf{w}_{l,l-1}^T \odot (1 - \mathbf{y}_l^2)$$

Thus, we can propagate the sensitivity with respect to the inputs and render V_x^T as

$$\frac{\partial \mathbf{y}_{l+1}}{\partial \mathbf{x}_{l-1}} = \mathbf{w}_{l+1,l}^T \frac{\partial \mathbf{y}_l}{\partial \mathbf{x}_{l-1}} \odot (1 - \mathbf{y}_{l+1}^2) \tag{13}$$

Equation (13) may be applied recursively for any number of hidden layers. It is also applied to the output layer; however, for identity output activation function $\frac{\partial \mathbf{y}_o}{\partial \boldsymbol{\sigma}_o} = 1$, the last factor in Equation (13) vanishes. The sensitivity of $\frac{\partial \mathbf{y}_o}{\partial \mathbf{x}_i}$ with respect to all weights in the network may be found by back propagation. Due to applying the chain rule to Equations (9) and (10), terms from standard back-propagation will appear as well. Thus, we define the generalized delta for unit *j* in terms of the deltas for unit *k* in the subsequent layer and weights w_{kj} connecting unit *j* to unit *k* in the next layer forward.

$$\delta_j = f'(\sigma_j^p) \sum_{Nk} \delta_k^p w_{kj} \tag{14}$$

$$\text{or, } \delta_j = f'(\sigma_j^p) \mathbf{w}_{kj} \boldsymbol{\delta}_k^p = \partial E_p / \partial \sigma_j$$

Then, applying the chain rule to Equation (10), we get the generalized delta rule for Equation (13) at the penultimate layer as

$$\boldsymbol{\partial\delta}_{o-1} = -2\mathbf{w}_o \odot \frac{\boldsymbol{\partial}\mathbf{y}_{o-1}}{\partial\mathbf{x}_i} \odot \mathbf{y}_{o-1} \odot (1-\mathbf{y}_{o-1}^{\mathbf{2}}) \tag{15}$$

Hidden layers prior to *o*-1 require additional terms due to the chain rule

$$\begin{aligned}\boldsymbol{\partial\delta}_{l-1} = -2\mathbf{w}_{l,l-1}\delta_l \odot \frac{\boldsymbol{\partial}\mathbf{y}_{l-1}}{\partial\mathbf{x}_i} \odot \mathbf{y}_{l-1} \odot \left(1-\mathbf{y}_{l-1}^{\mathbf{2}}\right) \\ + \mathbf{w}_{l,l-1}\boldsymbol{\partial\delta}_l\left(1-\mathbf{y}_{l-1}^{\mathbf{2}}\right)\end{aligned} \tag{16}$$

Likewise, the sensitivity of delta with respect to network inputs for the input layer is an algorithm that proceeds as follows[5]:

$$\begin{aligned}\boldsymbol{\partial\delta}_i = \frac{\partial(\boldsymbol{\delta}_i)}{\partial\mathbf{x}_i} = -2\mathbf{w}_{i,h}\delta_h \odot \frac{\boldsymbol{\partial}\mathbf{y}_i}{\partial\mathbf{x}_i} \odot \mathbf{y}_{l-1} \odot \left(1-\mathbf{y}_{l-1}^{\mathbf{2}}\right) \\ + 2\mathbf{w}_{i,h}\boldsymbol{\partial\delta_h}\left(1-\mathbf{y}_i^{\mathbf{2}}\right)\end{aligned} \tag{17}$$

Note that $\odot$ in the expressions above is the Hadamard product, and bias terms are ignored in Equations (13)–(17).

To render sensitivities for the network output with respect to the network weights $\mathbf{w}_{l,l-1}$, one simply multiplies $\boldsymbol{\delta}_l$ with $\boldsymbol{y}_{l-1}$. Generally, using an outer product in this instance will render $\partial y_{\mathrm{o}}/\partial\mathbf{w}_{l,l-1}$ in the same dimensions as $\mathbf{w}_{l,l-1}$. However, to find $\partial V_{\mathrm{x}}/\partial\mathbf{w}_{l,l-1}$ is more involved.

Starting at the input layer, for $\partial y_{\mathrm{o}}/\partial\mathbf{w}_{l,l-1}$, we have an outer product.

$$\partial y_{\mathrm{o}}/\partial\mathbf{w}_{l,l-1} = \boldsymbol{\delta}_i\boldsymbol{x}^{\boldsymbol{T}}$$

For $\partial V_{\mathrm{x}}/\partial\mathbf{w}_{l,l-1}$, a unique term will be rendered for each element of $\boldsymbol{x}$ since V_{x} is a vector of sensitivities with respect to $\boldsymbol{x}$.

$$\partial V_{xj}/\partial\mathbf{w}_i = \boldsymbol{\partial\delta}_i^{\mathbf{j}}\boldsymbol{x}^{\boldsymbol{T}} + [\mathbf{0}_{\boldsymbol{N_h}\times\boldsymbol{j}-\mathbf{1}} \quad \boldsymbol{\delta_i} \quad \mathbf{0}_{\boldsymbol{N_h}\times\boldsymbol{N_x}-\mathbf{1}}] \tag{18}$$

Such that only the *j*th column of the second term is populated by the customary input layer delta. Propagating forward, each hidden layer sensitivity can be written as

$$\partial V_{xj}/\partial\mathbf{w}_{l-1,l} = \boldsymbol{\partial\delta}_l\boldsymbol{y}_{\boldsymbol{l}-\mathbf{1}}^{\boldsymbol{T}} + \boldsymbol{\delta_l}\frac{\boldsymbol{\partial}\mathbf{y}_{l-1}}{\partial\mathbf{x}_i} \tag{19}$$

And sensitivity of the output layer weights is simply $\partial V_{xj}/\partial\mathbf{w}_{o,l} = \frac{\boldsymbol{\partial}\mathbf{y}_l}{\boldsymbol{\partial}\mathbf{x}_i}$ for identity activation and $\partial V_{xj}/\partial\mathbf{w}_{o,l} = \frac{\boldsymbol{\partial}\mathbf{y}_l}{\boldsymbol{\partial}\mathbf{x}_i}(1-\mathbf{y}_o^{\mathbf{2}})$ for tanh activation.

2.2 Method of Successive Approximation

If we attempt to fit a neural network to Equation (5) starting with random weights, the solution will be very difficult. Instead, we use the method of successive approximation (MSA) to solve a simpler, linear version that converges to Equation (5) in the limit. Consider the following linear approximation to Equation (5):

$$\mathcal{R} = V_x^{(\boldsymbol{k})T}\left(\boldsymbol{f}(\mathbf{x}) + \boldsymbol{g}(\mathbf{x})\boldsymbol{u}_{(\boldsymbol{k})}\right) + \mathbf{Q}(\mathbf{x}) + \boldsymbol{u}_{(\boldsymbol{k})}^{\boldsymbol{T}}\boldsymbol{R}\boldsymbol{u}_{(\boldsymbol{k})} = 0 \tag{20}$$

$$\boldsymbol{u}_{(\boldsymbol{k+1})} = -\frac{1}{2}\boldsymbol{R}^{-\mathbf{1}}\boldsymbol{g}(\mathbf{x})^T V_x^{(\boldsymbol{k})} \tag{21}$$

Such that $\boldsymbol{u}_{(\boldsymbol{k})}$ is considered constant in Equation (20) and is updated at each iteration by Equation (21). In Section 3, we show how a deep neural network can be used to solve Equation (5) by repeatedly solving Equation (20), updating the control $\boldsymbol{u}_{(\boldsymbol{k+1})}$ using Equation (21), and back-substituting into Equation (20). Note that when Equation (20) is used as the residual, the gradient reduces to

$$\boldsymbol{J} = \frac{\partial \mathcal{R}}{\partial \boldsymbol{w}} = \frac{\partial V_x^T}{\partial \boldsymbol{w}}\boldsymbol{f}(\mathbf{x})$$

3. Example: Training a Pontryagin Neural Network (PoNN) Offline Using Successive Approximation

3.1 System Dynamics and Stabilizing Controller

To demonstrate the deep PoNN approach, we use a system from Scorsoglio et al.[5] The following set of ODEs can be used to describe the attitude dynamics of an axisymmetric spacecraft treated as a rigid body:

$$\dot{x}_1 = amx_2 + u_1 \tag{22}$$

$$\dot{x}_2 = -amx_1 + u_2 \tag{23}$$

$$\dot{x}_3 = mx_4 + x_2x_3x_4 + (x_1/2)(1 + x_3^2 - x_4^2) \tag{24}$$

$$\dot{x}_3 = -mx_3 + x_1x_3x_4 + (x_2/2)(1 + x_4^2 - x_3^2) \tag{25}$$

For a certain flight condition with a certain set of physical properties, we set $m = -0.5$ and $a = 0.5$. Since we need a stabilizing controller to initiate the MSA iteration, consider that Equations (22)–(25) have the following linearized dynamics at the $\boldsymbol{x} = \mathbf{0}$, $\boldsymbol{u} = \mathbf{0}$ equilibrium point:

$$\dot{\boldsymbol{x}} = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{B}\boldsymbol{u} \tag{26}$$

Where

$$\boldsymbol{A} = \begin{bmatrix} \mathbf{0} & -\mathbf{1/4} & \mathbf{0} & \mathbf{0} \\ \mathbf{1/4} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{1/2} & \mathbf{0} & \mathbf{0} & -\mathbf{1/2} \\ \mathbf{0} & \mathbf{1/2} & \mathbf{1/2} & \mathbf{0} \end{bmatrix}, \quad \boldsymbol{B} = \begin{bmatrix} \mathbf{1} & \mathbf{0} \\ \mathbf{0} & \mathbf{1} \\ \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix}$$

Equation (25) has open-loop poles (eigenvalues) at $\lambda_i \in \{\pm 0.5i, \pm 0.25i\}$; thus, its open-loop dynamics are a limit cycle about the origin with amplitude and phase determined by a linear combination of the right eigenvectors. To determine a stabilizing controller, we specify poles with ω_d unchanged from the open-loop locations, and set the damping ratio, ζ to $\sqrt{2}/2$, or $\lambda_{iCL} \in \{-0.5 \pm 0.5i, -0.25 \pm 0.25i\}$. Using the **place** function in MATLAB, the control law is

$$\boldsymbol{u} = -\boldsymbol{K}\boldsymbol{x} \tag{27}$$

Where

$$\boldsymbol{K} = \begin{bmatrix} \mathbf{0.752} & -\mathbf{1.00} & -\mathbf{0.251} & -\mathbf{0.753} \\ \mathbf{0.996} & \mathbf{0.748} & \mathbf{0.7473} & -\mathbf{0.249} \end{bmatrix}$$

Due to the nonlinearity of Equations (22)–(25), its stability properties vary with input commands. We explore these stability properties in the Appendix.* Since the dynamics are well characterized, and physically meaningful near the origin, we restrict training and initial conditions to $\boldsymbol{x} \in [-0.5, 0.5]$.

3.2 Network Architecture and Training Data

We designed a PoNN such as the one in Figure 1 to solve Equations (20) and (5) by MSA. The network input is the system state vector of length 4. Its output is the value function *V*, a scalar. The network has two hidden layers with 25 units each with tanh activation. The input layer also has tanh activation, and the output layer has linear (identity) activation. All network weights were set initially to $\boldsymbol{w}_i \in [-0.5, 0.5]$.

Since the input data is 4-D, we use the rank-one lattice design of experiments method to choose inputs for the training set.[6,7] Rank-one lattice is a space-filling design that renders a uniform sampling of the input space similar to Latin hypercube design. We show one such dataset for a 4-D input space with $N = 625$ in Figure 2.

Note that using a full factorial input data grid would result in $x_i = 0.25i - 0.75, i = 1,2, \dots, 5$, which is only 5 points in each direction since $5^4 = 625$. We show such a design in Figure 3. Note that the smallest distance between samples in

*The underlying physics rely on direction cosines; therefore, values outside of $\boldsymbol{x} \in [-1, 1]$ are not physically meaningful. For a comprehensive treatment see Scorsoglio et al.[5]

this space is 0.25. This is too large a spacing to render accurate gradients by finite differencing. Previous attempts to train a PoNN using this grid and the finite differencing approach of Burchett[3] failed because of inaccurate gradient estimates. Also, note that any finite differencing scheme will render only p-1 independent derivatives where p is the number of points in each direction. For a full factorial grid with gradients determined by finite differencing, that limitation results in a maximum number of independent rows of the Jacobian equal to $(p-1)^n$ where n is the state vector length. Clearly, the points in Figure 2 have much closer spacing than those in Figure 3, with the largest difference being about 0.09 between the bands in $\{x_2, x_3\}$ space. Thus, the space-filling design is superior in exploring the space with an equal number of points.

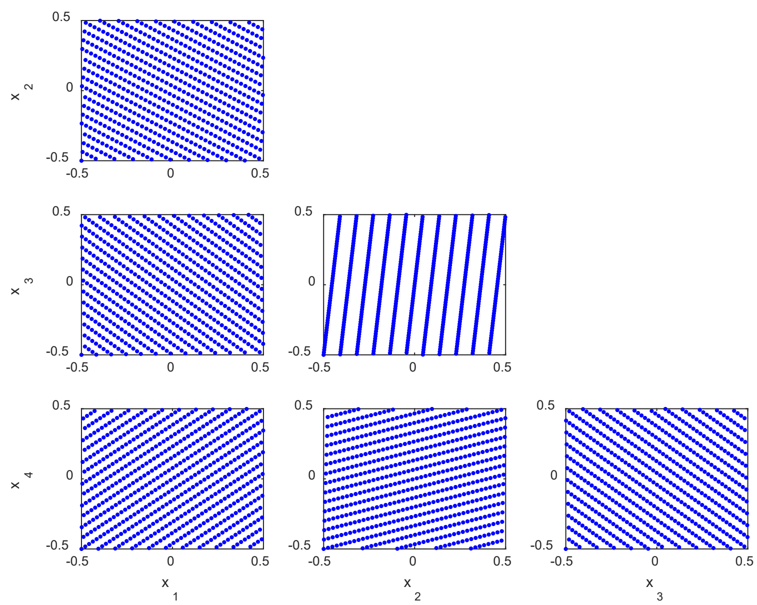


Figure 2. Rank-one lattice design of experiments for four inputs with N = 625 and $\boldsymbol{x} \in [-\mathbf{0.5}, \mathbf{0.5}]$.

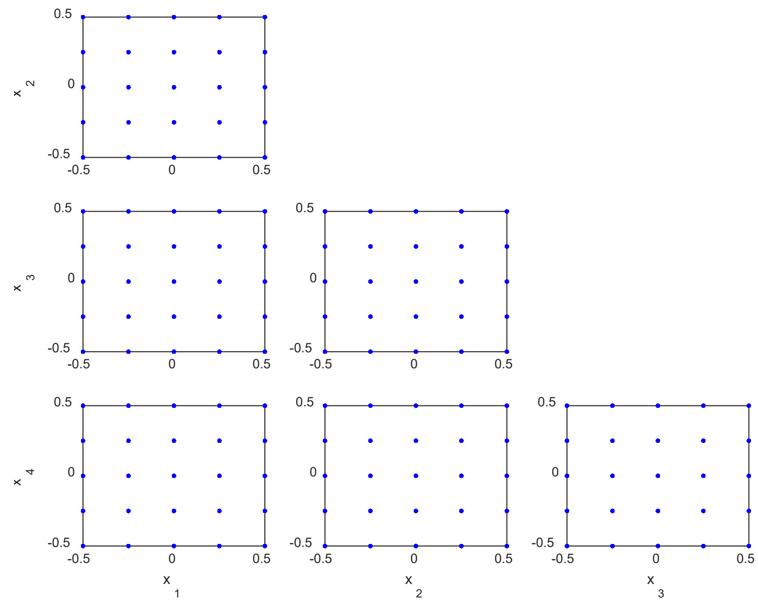


Figure 3. Full factorial design of experiments for four inputs with N = 625 and $\boldsymbol{x} \in [-\mathbf{0.5}, \mathbf{0.5}]$.

Using a space-filling design has the disadvantage of varying inputs along all dimensions from sample to sample; thus, finite differencing for derivatives is impossible since effects of variance in individual states cannot be isolated.

3.3 Training Convergence and Closed-Loop Performance

We trained the network described in Section 3.2 for 12 MSA iterations. At each MSA iteration, the Levenberg–Marquardt (LM) algorithm[8] was used to modify the network weights to best fit Equation (20) with the last update to the control signal vectors (Equation 21). LM was allowed to use up to 20 iterations unless the mean square error fell below 10^{-5} at which point LM halted and the result was used to compute the next control vector for use in MSA. Figure 4 shows the convergence of the nonlinear HJB and the linear generalized HJB during network training. Note that the nonlinear residual from Equation (5) approaches that of Equation (20) as the solution converges. After MSA iteration 6, the two residuals differ by less than 10^{-7}.

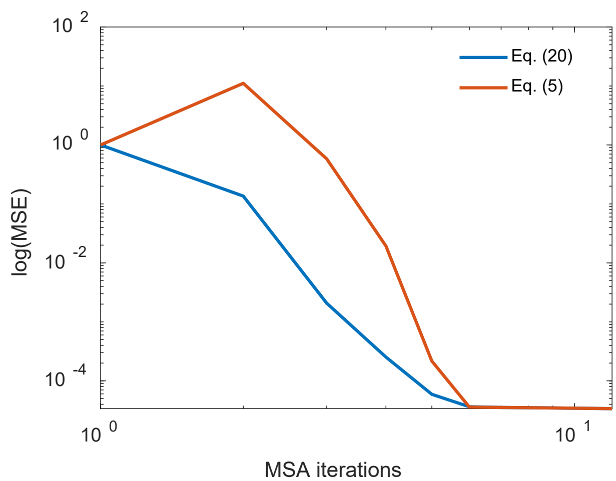


Figure 4. Convergence of training using MSA.

Figure 5 compares the closed-loop performance of the HJB optimal controller with that of the ad hoc pole placement controller for an initial condition chosen randomly from the training set. Note that in the optimal case, all states except x_4 approach the ordinate sooner and track it more closely as the system settles. This shows that the quadratic penalty on the state is reduced by the optimal controller as specified. The settling times and overshoots are comparable, which is not surprising since quadratically optimal controllers (i.e., LQR) drive the closed-loop system to a damping ratio of $\sqrt{2}/2$ as was specified in pole placement. The settling times are similar since the optimal controller achieves this damping ratio with minimal control effort, so it does not push the closed-loop poles any further left in the complex plane than the pole placement design.

Figure 6 compares the control effort for the optimal controller with that of the initially stabilizing controller. Note that u_2 is much greater at $t = 0$ for the pole placement case. The pole placement u_1 control starts near zero but rises above the maximum magnitude exerted by the optimal controller in either channel.

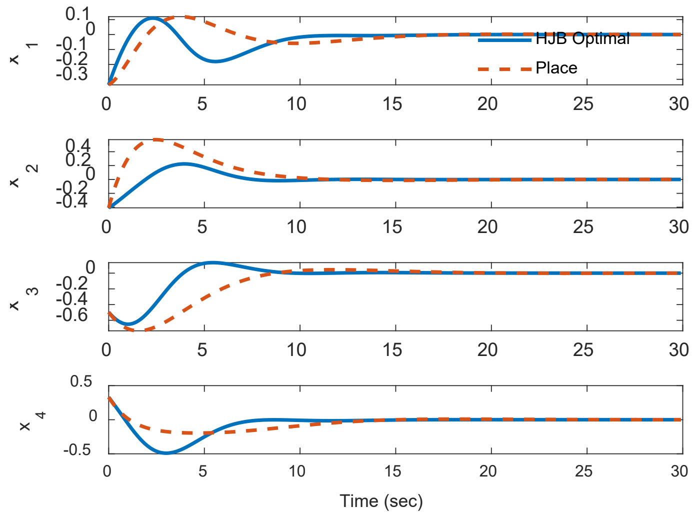


Figure 5. Comparing system response with the HJB optimal controller and ad hoc pole-placement controller.

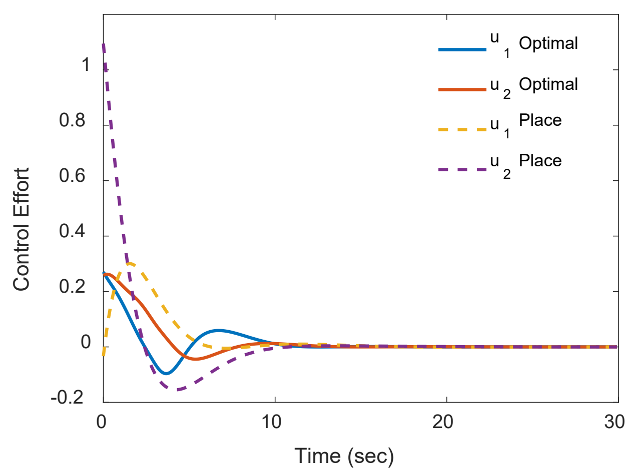
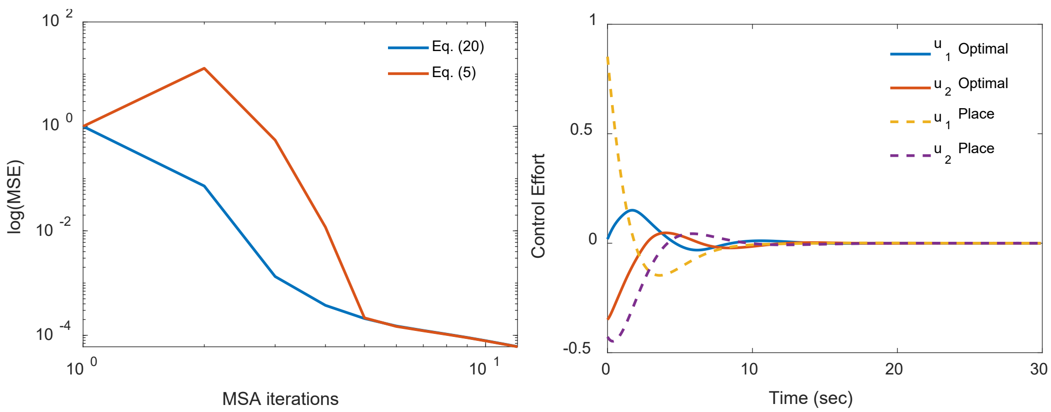


Figure 6. Comparing control effort for HJB optimal and ad hoc pole-placement controllers.

These large excursions from zero indicate that the optimal controller has significantly reduced the quadratic penalty on control effort from that of the initial stabilizing controller. Again, equal settling times are evident in the convergence of all control signals to the ordinate around 10 s.

3.4 Convergence with a Smaller Training Grid

Figures 7 and 8 show the results of training with the training grid reduced to 437 points. The rank-one lattice method is used to choose the training points in 4-D space. In Figure 7a, the convergence to optimal over 21 MSA iterations is comparable to that shown in Figure 4 for the full training grid of 625. Since the linear approximation (blue) approaches the nonlinear (red), and the final MSE is similar to that with the larger training grid, the new controller should be as close to HJB optimal as the previous one. Figure 7b shows the control effort for a sample response from an initial condition that is part of the reduced training set. Note that there is no direct comparison between Figure 7b and Figure 6 as the two training sets should be unique—i.e., not sharing any points.



a) Convergence over 20 MSA iterations **a) Control effort after training**

Figure 7. Convergence and control effort when using training grid of $N = 437$.

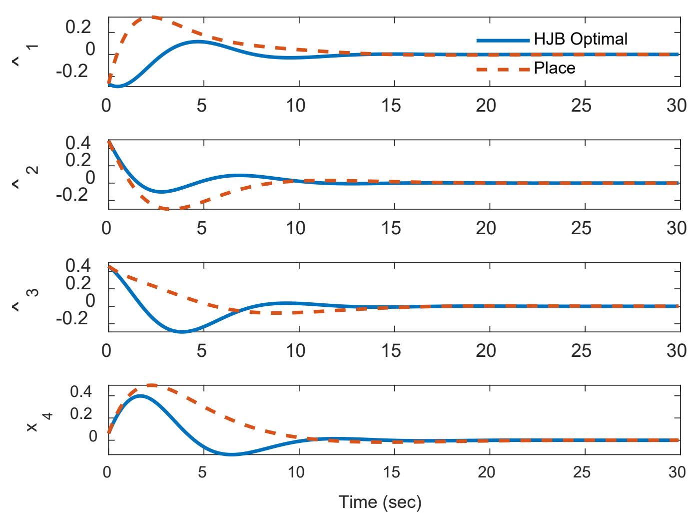
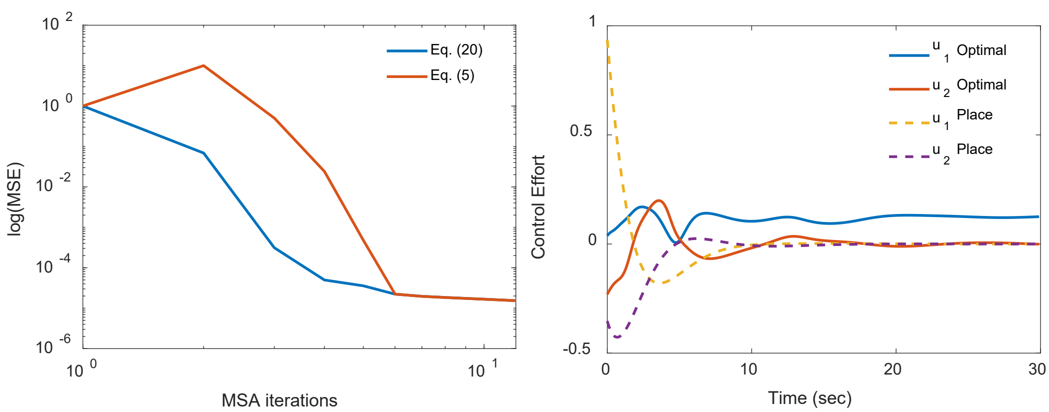


Figure 8. Sample closed-loop response compared to initial stabilizing controller when using training grid of $N = 437$.

Figure 8 shows the closed-loop response for the initial controller, and the HJB optimal for initial conditions chosen randomly from the smaller training set. Again, there is no direct comparison with Figure 5, since the new training set shares no points with the previous one. However, it is apparent that the response characteristics for the HJB optimal are consistent after training with the reduced set. That is, the frequency and number of oscillations prior to settling, and the settling time are very similar between the two controllers.

Reducing the training set even further to a total of 252 samples, we observed the convergence to optimal shown in Figure 9a. Again, the convergence to optimal resembles that for the large- and medium-size training sets; thus, we would expect the rendered controller to have similar response characteristics. However, in Figures 9b and 10, we observe erratic controls and much more oscillation in the state response than in the previous cases.



a) Convergence over 20 MSA iterations **b) Control effort after training**

Figure 9. Convergence and control effort when using training grid of $N = 252$.

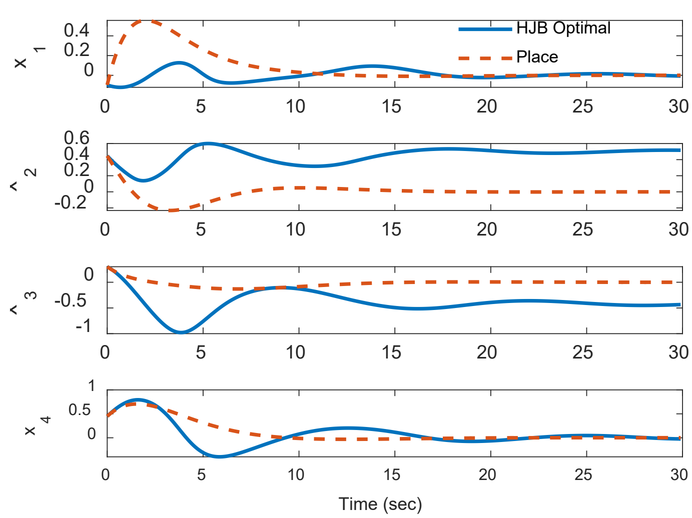


Figure 10. Sample closed-loop response compared to initial stabilizing controller when using training grid of $N = 252$.

4. Conclusion

The combination of rank-one lattice sampling and analytic derivatives has allowed us to train a neural network to solve the HJB equation for a four-state system with minimal training data. The full factorial grid of $5^4 = 625$ training points was too coarse for a finite difference scheme to render accurate derivatives.

Furthermore, finite difference schemes reduce the row rank of the system Jacobian matrix as only $(p-1)^n$ independent derivatives are rendered for a full factorial grid of p^n points. Thus, as the system dimension n grows, finite differencing becomes less capable of providing adequate sets of derivatives. Using analytic derivatives with a rank-one lattice space filling grid of 625, training was successful. The two methods led to successful training with grid size reduced to 437, but not with a grid size of 252.

5. References

1. Michek N, Mehta P, Huebsch W. Methodology development of a free-flight parameter estimation technique using physics-informed neural networks. 2023 IEEE Aerospace Conference; 2023; Big Sky, MT, USA. p. 1–18. https://doi.org.10.1109/AERO55745.2023.10115728

2. Furfaro R, D'Ambrosio A, Schiassi E, Scorsoglio A. Physics-informed neural networks for closed-loop guidance and control in aerospace systems. AIAA SCITECH 2022 Forum; 2022 Jan. AIAA 2022-0361.

3. Burchett BT. Toward training physics-informed neural networks without automatic differentiation. DEVCOM Army Research Laboratory (US); 2025 Mar. Report No.: ARL-MR-1121.

4. Furfaro R, D'Ambrosio A. Increasing autonomy of aerospace systems via PINN-based solutions of HJB equation. AIAA SCITECH 2024 Forum; 2024 Jan 8–12; Orlando, FL. https://doi.org.10.1109/10.2514/6.2024-1786

5. Scorsoglio A, D'Ambrosio A, Furfaro R. Stability certification of reinforcement learning-based control for aerospace applications. 2025 AAS/AIAA Space Flight Mechanics Meeting; 2025 Jan 19–23; Kaua'I, Hawaii, USA.

6. Lyu Y, Yuan Y, Tsang I. Subgroup-based rank-1 lattice quasi-Monte Carlo. 34th Conference on Neural Information Processing Systems (NeurIPS 2020); 2020; Vancouver, Canada.

7. Huang H, Giacobello M. Uncertainty quantification in store separation analysis using Kestrel, design of experiments, and surrogate modelling. AIAA SciTech Forum; 2022 Jan 3–7; San Diego, CA.

8. Hagan MT, Menhaj MB. Training feedforward networks with the Marquardt algorithm. IEEE Trans Neural Netw. 1994 Nov;5(6):989–993. https://doi.org.10.1109/72.329697

Appendix. Nonlinear Dynamics

A.1 Nonlinear Dynamics of the Open-Loop System

Looking more closely at Equations (22)–(25), there is only one equilibrium point for the open-loop system, which is $\boldsymbol{x} = \mathbf{0}$. This can be shown by setting the control to zero, and solving Equations (22) and (23) for x_2 and x_1 respectively, resulting in zeros. Substituting $x_1 = x_2 = 0$ into Equations (24) and (25), all but the first terms on the right-hand side disappear, such that $x_3 = x_4 = 0$ is the only solution.

In the top left pane of Figure A-1, we simulate the nonlinear dynamics of Equations (22)–(25) for 100 s, starting with initial conditions set to the average of the right eigenvectors associated with the fast oscillatory mode of $\boldsymbol{A}$ ($\lambda_i = \pm 0.5j$). That is, $x_4(0) = 2x_2(0)$, all others zero. The response is a limit cycle in the $\{x_1, x_4\}$ plane that oscillates above and below the line $x_4 = 2x_2$ defined by the eigenvector. The small departures from expected linear behavior (i.e., oscillation colinear with eigenvector) are due to internal nonlinear dynamics. The system remains marginally stable within this domain.

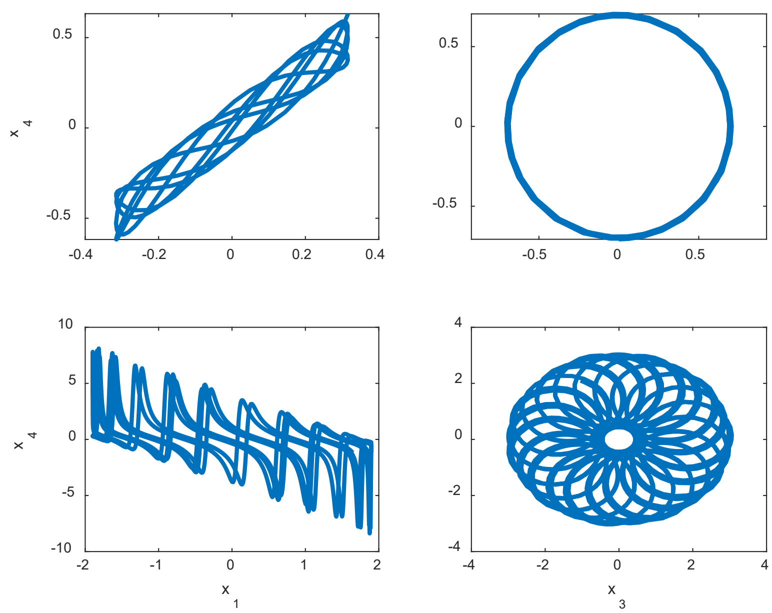


Figure A-1. Open-loop simulation of nonlinear system with initial conditions set to illustrate limit cycle behavior.

In the top right pane, we set the initial condition equal to the average of the right eigenvectors associated with the slow mode ($\lambda_i = \pm 0.25j$). That is, $x_4(0) = -\sqrt{2}$, all others zero. The response is a limit cycle that is circular (however, plot scaling skews it into an ellipse).

The bottom two panes are scenarios starting from outside the physically meaningful domain of the dynamics ($x_i \in [-1, 1]$) to see how the stability properties change for the nonlinear system. Surprisingly, both cases reach a limit cycle with apparently chaotic behavior caused by shifting the relative phase of the excited modes. Initial conditions for the bottom left pane were

$$\boldsymbol{x}(0) = [1.644, 0.9497, -0.9356, -1.177\mathbf{5} \qquad \text{(A-1)}$$

Initial conditions for the bottom right pane were

$$\boldsymbol{x}(\mathbf{0}) = [-1.418, -2.202, -1.041, 2.093] \qquad \text{(A-2)}$$

A.2 Chaotic Behavior of the Closed-Loop System

The equilibrium point for the closed-loop system may be set by choosing the steady state control signals $(u_1(\infty), u_2(\infty))$. For instance, if we set $u_1(\infty) = -1/2$, $u_2(\infty) = 1/2$, the equilibrium values of x_1 and x_2 move to –2 and –2, respectively. The corresponding values of x_3 and x_4 are 0.5931 and –0.5931, respectively. Note that x_1 and x_2 move outside the physically meaningful space $x_i \in [-1, 1]$; however, we want to see how the dynamics change. First, let us see what the linearized model predicts for these conditions. Note that we will need the full linearization of $\boldsymbol{A}$, which is

$$A = \begin{bmatrix} 0 & am & 0 & 0 \\ -am & 0 & 0 & 0 \\ (1 + x_3^2 - x_4^2) & x_3 x_4 & x_1 x_3 + x_2 x_4 & x_2 x_3 - x_1 x_4 \\ x_3 x_4 & (1 + x_4^2 - x_3^2) & -x_2 x_3 + x_1 x_4 & x_1 x_3 + x_2 x_4 \end{bmatrix} \text{(A-3)}$$

The eigenvalues of $\boldsymbol{A}$ for $\boldsymbol{x} = [-2, -2, 0.5931, -0.5931]$ are $\lambda_{1,2} = \pm 0.25j$ and $\lambda_{3,4} = 4(10^{-11}) \pm 2.3723j$. Thus, the second oscillatory mode is much faster and very slightly unstable—although, the real part of $\lambda_{3,4}$ is so small that oscillatory growth will not be detectable for reasonable time horizons.

We simulated the nonlinear response of the system for two sets of initial conditions, for 100 s, holding u_1 and u_2 constant at –1/2 and 1/2, respectively. The responses are shown as phase plane plots in Figure A-2. In the top left and right panes, we show the response from $x_4(0) = -\sqrt{2}/2$, all others zero. Note that x_1 and x_2 follow a limit cycle centered at the new equilibrium point. Note that x_3 and x_4 make large departures from equilibrium but return to the tiny red dot shown in each plot.

In the bottom left and right panes, we show the response from a random initial condition $\boldsymbol{x}(0) = [0.0518, 2.221, 1.155, -2.4734]$. Once again, x_1 and x_2 trace a limit cycle centered at $\{-2, -2\}$. Again, x_3 and x_4 make large excursions, but always return to a point near the equilibrium as shown by the red dot depicting the final value.

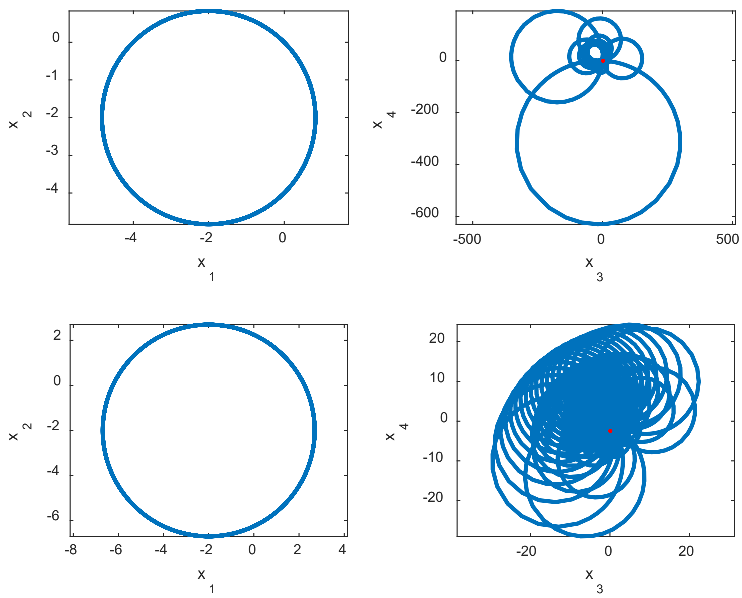


Figure A-2. System response for large steady state control signals.

We can thus state that the system is chaotic as control commands sustained at a moderate level can move the system into a region of drastically different behavior. Thus, our control laws need to limit control signals to a small region near the origin for safe control.

List of Symbols, Abbreviations, and Acronyms

4-D	four-dimensional
$\boldsymbol{A}$	linear system dynamics matrix
$\boldsymbol{B}$	linear system controls matrix
b	bias
δ_j	generalized delta for unit j
$f(.)$	activation function
HJB	Hamilton–Jacobi–Bellman
$\mathbf{J}$	training Jacobian matrix
LM	Levenberg–Marquardt
LQR	linear-quadradic regulator
MSA	method of successive approximation
n_l	number of total layers
N	number of units in a layer
ODE	ordinary differential equation
PDE	partial differential equation
PINN	Physics Informed Neural Network
PoNN	Pontryagin Neural Network
q	network input vector
$\mathbf{R}$	training residual vector
tanh	hyperbolic tangent
$\boldsymbol{w}$	synaptic weights
x	output of neuron
$\hat{y}$	network output
σ	sum of input signals from synapses

Subscript

h	hidden layer
i	input layer
$k\,j$	from unit j to unit k
o	output layer
p	training pair p
$train$	training pairs

Superscript

T	matrix transpose

1 (PDF)	DEFENSE TECHNICAL INFORMATION CTR DTIC OCA
1 (PDF)	DEVCOM ARL FCDD RLB CI TECH LIB