ACTIVE FLOW CONTROL WITH ADAPTIVE DESIGN TECHNIQUES FOR IMPROVED AIRCRAFT SAFETY

Software Users Manual (SUM)

December 31, 2009

CONTRACT # NND08AA58C

Prepared by:

Jason O. Burkholder
BARRON ASSOCIATES, INC.
1410 Sachem Place
Suite 202
Charlottesville, VA 22901
434-973-1215, Ext. 121
burkholder@barron-associates.com

Gang Tao, Ph.D.
UNIVERSITY OF VIRGINIA
Elec. & Computer Engineering
351 McCormick Road
Charlottesville, VA 22904
434-924-4586
gt9s@virginia.edu

Douglas R. Smith, Ph.D.
UNIVERSITY OF WYOMING
Mechanical Engineering
1000 E. University Avenue
Laramie, WY 82071
307-766-3647
drsmith@uwyo.edu

Distribution:

Timothy H. Cox
Christopher J. Miller
Mail Stop 4840D / P.O. Box 273
NASA Dryden
Edwards, CA 93523

NASA

Mail Stop 1422D / P.O. Box 273
NASA Dryden
Edwards, CA 93523

Contents

List of Figures

1 Scope

1.1 Identification

This Software Users Manual (SUM) documents AIFAC — the Barron Associates, Inc. Adaptive Inverse For Actuator Compensation toolbox for MATLAB®, Version 1.0. MATLAB is a product of The MathWorks™.

1.2 System Overview

The overall objective of this STTR effort is to evaluate and demonstrate the potential for well-designed, strategically-located synthetic jet actuators to provide improved commercial transport aircraft safety by: (1) delaying wing stall and improving aircraft controllability at high angles of attack and (2) providing low-cost actuation redundancy to improve controllability in the event of a mechanical control surface failure. Delaying flow separation (i.e., wing stall) and providing "back-up" control power could allow an aircraft to recover from adverse conditions (due to a control surface failure, pilot/autopilot error, etc.) that would otherwise result in a loss of control.

Flow control studies have shown that synthetic jet actuators are efficient devices for controlling separated internal and external flows [1, 2]. Smith et al. [2] and Amitay et al. [1] have used synthetic jet control near the leading edge of an airfoil to prevent boundary layer separation and hence prevent stall at high angles of attack. However, an obstacle to the widespread application of synthetic jet actuators for practical flight control is that modulated input signals to achieve closed-loop flow control objectives have been shown to be complex. The input signal variables that can be manipulated for control include the number of active jets in the array, and the signal amplitude, waveform, and frequency delivered to each jet. The control effect from manipulating each of these variables is typically a complex nonlinear function. Developing a comprehensive, wind tunnel-validated model of synthetic jet actuators is technically challenging and expensive.

Control techniques, based on the *adaptive inverse* methodology, overcome prior limitations and set the stage for practical flight control using synthetic jet actuators to achieve virtual shaping of aerodynamic surfaces. The adaptive controller design is dependent upon the actuation method (e.g., amplitude or frequency) and the approximation method used to design the adaptive inverse function. The research team designed several adaptive controllers and developed the Adaptive Inverse for Actuator Compensation (AIFAC) Toolbox for Simulink®.[1] The AIFAC Toolbox features adaptive inverse controllers based on previous designs and new approximation-based adaptive controllers, which are widely applicable to systems, such as active flow control systems, that suffer from complex unknown actuator nonlinearities.

1.3 Document Overview

This SUM shall document installation and usage of the AIFAC computer software configuration item (CSCI).

[1] *www.mathworks.com*

2 Referenced Documents

References

[1] M. Amitay, D. Pitt, V. Kibens, D. E. Parekh, and A. Glezer, "Control of internal flow separation using synthetic jet actuators," in *Aerospace Sciences Meeting and Exhibit, 38th*, no. AIAA-2000-0903, 2000.

[2] D. Smith, M. Amitay, V. Kibens, D. E. Parekh, and A. Glezer, "Modification of the lifting body aerodynamics by synthetic jet actuators," in *Aerospace Sciences Meeting and Exhibit, 36th*, no. AIAA-98-0209, 1998.

[3] G. Tao, *Adaptive Control Design and Analysis*. John Wiley & Sons, 2003.

3 Software Summary

3.1 Software Application

The AIFAC toolbox provides advanced adaptive inverse compensation algorithms in the form of Simulink block components. These components may be used to develop control applications with the Simulink modeling tool. The applications can be tested and refined within the high-fidelity Simulink modeling environment. After verification, the Real-Time Workshop code generation tool may be used to generate deployable C language code for integration in online real-time control systems.

The AIFAC algorithms provide out-of-the-box diagnostic capabilities that are applicable to a wide range of linear and non-linear control problems.

3.2 Software Inventory

The AIFAC software is distributed in either hard or soft format. The hard format distribution is a single CD with the AIFAC toolbox in the root directory. The soft format distribution is a single compressed archive with the AIFAC toolbox in the root directory.

The root directory of the distribution contains this user's manual, the installation script and toolbox implementation. Subdirectories contain the support files, including documentation, examples and Windows runtime.

The root directory contains the following installation files:

AifacSoftwareUsersManual.pdf	This user's guide.
aifacFileList.m	List of files in the distribution.
autorun.inf	The distribution CD "autorun" file.
BarronAssociatesIncLogo.ico	The Barron Associates corporate logo.
barron associates, inc.url	URL link to the Barron Associates website.
htmlHelpHelper.m	Integrated HTML help script.
installAifacToolbox.m	The toolbox installation script.
libAifac.mdl	The AIFAC library.
slblocks.m	Simulink library toolbox integration script.

The root directory contains the following implementation files:

ann3layerLogisticSigmoid_callback.m	backlash_callback.m
CubicCardinalBSplineApproxsfunc.cpp	CubicCardinalBSplineApproxsfunc.mexw32
deadzone_callback.m	disturbanceRejection_callback.m
dOmega_a_dv.c	dOmega_a_dv.mexw32
gaDetectBacklash.m	gaDetectDeadzone.m
GaussianRBFsfunc.cpp	GaussianRBFsfunc.mexw32
grbf_callback.m	omega_c.c
omega_c.mexw32	

The "Doc" subdirectory contains the integrated help files:

ann3niHelp.html	backlashCharacteristic.png
BacklashHelp.html	BarronLogo.png
CubicCardinalBSplineHelp.html	deadzoneCharacteristic.png
DeadzoneHelp.html	demoAnn3ni.PNG
demoAnn3ni_e.PNG	demoBacklash.png
demoCubicCardBSpline.png	demoDeadzone.png
demoDisturbanceRejection.PNG	demoDisturbanceRejectionPerformance.png
demoDisturbanceRejectionSystem.png	demoRbf.png
DisturbanceRejectionControlSystem.png	DisturbanceRejectionHelp.html
gaBacklashProgressPlot.png	gaDeadzoneProgressPlot.png
gaussianRbfCurve.png	GaussianRbfHelp.html
referenceSystem.png	TaylorSeriesHelp.html

The "Microsoft.VC80.CRT" subdirectory contains Microsoft C/C++ runtime files with which the toolbox was built. These files are required to run the s-functions:

Microsoft.VC80.CRT.manifest	msvcp80.dll
msvcm80.dll	msvcr80.dll

3.3 Software Environment

AIFAC is designed for the R2007b release of The MathWorks™ products:

Product	Version
MATLAB®	7.5
Simulink®	7.0
Real-Time Workshop®	7.0
xPC Target™	3.3

AIFAC is integrated with Simulink so that it is available in the Library Browser tool.

3.4 Software Organization and Overview of Operation

The AIFAC toolbox is a collection of reusable Simlink blocks in Simulink library form. After installation, AIFAC appears as a folder within the Simulink library browser. The blocks may be used like any other Simulink component. Each block has an integrated help page.

3.5 Contingencies, Alternate States and Modes of Operation

Each AIFAC block is designed to be simulated under the Simulink system, as well as allowing code generation for deployment in real-time applications.

3.6 Security and Privacy

The AIFAC toolbox has no security or privacy issues.

3.7 Assistance and Problem Reporting

Direct all problems and inquiries to the report preparer:

Barron Associates, Inc
1410 Sachem Place, Suite 202
Charlottesville, VA 22901
(P) 434.973.1215

4 Access To The Software

4.1 First-Time User of the Software

4.1.1 Equipment Familiarization

AIFAC is targeted for a standard Windows XP computer.

4.1.2 Access Control

AIFAC has no access control features beyond the operating system login.

4.1.3 Installation and Setup

The AIFAC toolbox is distributed as both a simple compressed (e.g. zip) file or a CD, both of which contain all required resources. The distribution contains a MATLAB script that installs and cleanly uninstalls the software. This script executes within the MATLAB environment.

The AIFAC distribution includes the script file "installAifacToolbox.m". This script installs and uninstalls the toolbox. The base installation directory is the Windows "Program Files" path, which is defined by the environment variable "ProgramFiles". This path is the default installation location for

all programs, including MATLAB itself. The script will install the toolbox under the folder "BarronAssociates", in the subfolder "AIFAC". The AIFAC folder will be added to the end of the MATLAB path, so that the toolbox is accessible by MATLAB and Simulink.

To install the AIFAC toolbox:

1. Extract the toolbox distribution's compressed contents.
2. Start MATLAB R2007b.
3. Navigate to the directory containing the distribution's uncompressed contents.
4. At the MATLAB command line, execute the command: installAifacToolbox

The install script will respond with the following status messages:

```
>> installAifacToolbox
Installing AIFAC Toolbox
   ...creating directory C:\Program Files\BarronAssociates
   ...creating directory C:\Program Files\BarronAssociates\AIFAC
   ...creating directory C:\Program Files\BarronAssociates\AIFAC\Doc
   ...creating directory C:\Program Files\BarronAssociates\AIFAC\ToolboxExamples
   ...creating directory C:\Program Files\BarronAssociates\AIFAC\Microsoft.VC80.CRT
   ...copying files to C:\Program Files\BarronAssociates\AIFAC
   ...copying files to C:\Program Files\BarronAssociates\AIFAC\Doc
   ...copying files to C:\Program Files\BarronAssociates\AIFAC\ToolboxExamples
   ...copying files to C:\Program Files\BarronAssociates\AIFAC\Microsoft.VC80.CRT
AIFAC Toolbox installed; path is 'C:\Program Files\BarronAssociates\AIFAC'
```

The base path "C:\Program Files" above will be replaced by the system's "program files" path.

After installation, the install program will open this user's manual.

Installation Errors. Either directory creation or file copy could fail if the installer does not have necessary permissions.

The install script could fail to permanently add the toolbox to the MATLAB path if the installer does not have necessary permissions. In this case, the install script will display the following error message:

```
** Permanent path modification failed.
** Use File->Set Path or the pathtool command to permanently add
   the AIFAC toolbox to the MATLAB path.
```

4.1.4 Removal

The AIFAC installation includes the script file "installAifacToolbox.m". This script installs and uninstalls the toolbox.

To uninstall the AIFAC toolbox:

1. Start MATLAB R2007b.
2. At the MATLAB command line, execute the command: installAifacToolbox -uninstall

The install script will respond with the following status messages:

```
>> installAifacToolbox -uninstall
Uninstalling AIFAC Toolbox
   ...deleting directory C:\Program Files\BarronAssociates\AIFAC\Microsoft.VC80.CRT
   ...deleting directory C:\Program Files\BarronAssociates\AIFAC\ToolboxExamples
   ...deleting directory C:\Program Files\BarronAssociates\AIFAC\Doc
   ...deleting directory C:\Program Files\BarronAssociates\AIFAC
AIFAC Toolbox uninstalled
```

4.2 Initiating a Session

The AIFAC components appear in the Simulink library browser like any other Simulink block. Open the library browser by executing "simulink" at the MATLAB command line, or from an open model's toolbar as shown in Figure 1.

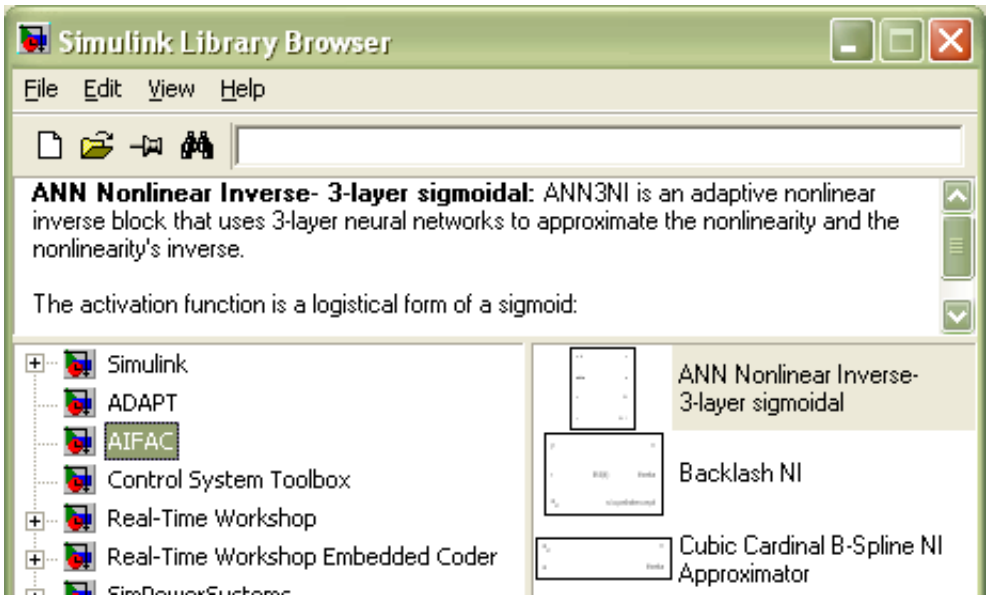


Figure 1: AIFAC in Simulink Library Browser

4.3 Stopping and Suspending Work

This section does not apply.

5 Processing Reference Guide

5.1 Capabilities

The AIFAC Simulink blocks may be used like any other Simulink block, and are compatible with Real-Time Workshop code generation.

5.2 Conventions

All AIFAC blocks are masked, providing a dialog for entering block parameters, including prompts and help text.

Each toolbox component has an integrated help page. This page is accessible by selecting the "Help" menu item on the component's right-click context menu. The help pages contain the following topics:

Overview	Provides background information on the block, including its purpose, restrictions and applicability.
Input Signals	Defines each input signal, possibly including dimensionality and restrictions.
Output Signals	Defines each output signal, possibly including dimensionality and restrictions.
Parameters	Defines each block parameter, possibly including dimensionality, restrictions and requirements.
Usage	Gives guidelines for the block's usage, including any restrictions.
Example	Shows an example usage of the component, possibly including simulation results.

5.3 Processing Procedures

The following subsections briefly describe each AIFAC block and its applicability.

The generalized feedback control system with adaptive nonlinear inverse is presented in Figure 2: C_1 is the control law, $\widehat{NI}$ is the Nonlinear Inverse, N is the actuator nonlinearity, G is the system with known dynamics, and C_2 is the output feedback.

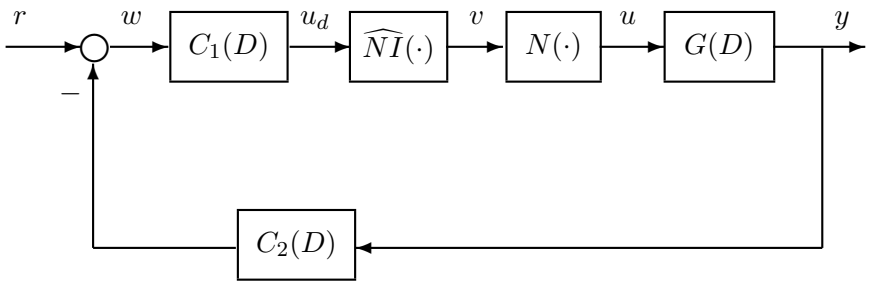


Figure 2: Actuator Compensation Feedback Control System

5.3.1 Adaptive Neural Network Inverse

Description

The Adaptive Neural Network (3-Layer) Nonlinear Inverse (*ANN3NI*) block provides an adaptive inverse approximation to an actuator that has a nonlinear control effect. This block allows the controller to treat the actuator's effect as though it were linear.

The *ANN3NI* block is applicable to linear (or linearizable) systems with measurable system states and a single nonlinear actuator. The system may have any combination of single or multiple inputs and outputs. The actuator's nonlinearity (N) must be a scalar function of two variables: $N(u\_d, alpha)$.

Algorithm The *ANN3NI* block consists of two neural network (NN) approximators: one approximator for the nonlinearity (N) and one for the nonlinear inverse (NI). Each NN has 3-layers. The input layer consists of two neurons, since the nonlinearity is a

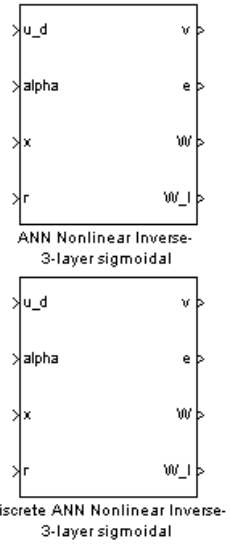


Figure 3: Adaptive Neural Network Simulink Block

function of two variables. The output layer consists of one neuron, since the NN is approximating a scalar function (N or NI). The number of hidden neurons is defined by the dimensionality of the block parameters as defined below. More neurons may increase accuracy at the expense of runtime cost. Each NN activation function is the logistical form of the sigmoid:

$$f(v) \;=\; \frac{1}{(1+e^{-av})}, \text{ for } a>0 \tag{1}$$

Input Signals

u_d	The desired system input signal, as if the actuator provided linear control.
$alpha$	The scalar, second independent actuator nonlinearity input signal.
x	The measured system state.
r	The controller input (reference) signal.

Output Signals

v	The output signal to drive the actuator to achieve the response desired by u_d. The signal dimensions are the same as the input signal u_d.
e	The error between the desired and actual system response, $e = x - x_m$. The signal dimensions are the same as the system state x.
W	The adapted weights for the output layer of the neural network that approximates the nonlinearity (N). These weights may be used in successive simulations to provide more accurate initial values in the block's W parameter. The W dimensions are $[n1]$, where n is the number of hidden neurons defined by the parameters.
W_I	The adapted weights for the output layer of the neural network that approximates the nonlinear inverse (NI). These weights may be used in successive simulations to provide more accurate initial values in the block's W_I parameter. The W_I dimensions are $[m1]$, where m is the number of hidden neurons defined by the parameters.

Parameters and Dialog Box

Advanced settings	Advanced settings exposes a configuration option. When this option is not selected (i.e., not checked), the parameter P is disabled and the block automatically calculates the P matrix as a solution to the Lyapunov equation $PAm + Am'P + Q = 0$, where Q is the identity matrix I and Am' is the transpose of Am. When this option is selected (i.e., checked), the parameter P is editable and the user may enter his choice for P. In most cases, the user will want to use the default solution and leave this option unselected. HOWEVER, the default solution requires the Control System Toolbox, as the lyap function is needed.
nonlinear actuator index	The (one-based) scalar index of the nonlinear actuator's input signal within the system input vector u(t).
A_m	The system reference model matrix is an estimated system matrix, and is chosen to have all its eigenvalues in $Re[s] < 0$.
P	The P matrix is described above. This field is only editable when the "Advanced Settings" box is checked.
$x0$	The reference system's initial states.
B	The system input matrix.

a	The scalar activation function parameter for the NN that approximates the nonlinearity (N). This value must be greater than 0.
V	The constant hidden layer weights for the NN that approximates the nonlinearity (N). This matrix has dimensions $[2n]$, where n is the number of hidden neurons. Each element v_{ij} is the weight of the link between input node i and hidden node j. These weights represent the stiffness of the activation functions and should be randomly selected $-1 \leq v_{ij} \leq 1$.
v_0	The constant hidden layer bias in the activation function for the NN that approximates the nonlinearity (N). This vector has the dimensions $[n1]$, where n is the number of hidden neurons. Each element v_0_j is the bias for the activation function of hidden node j. These values should have a range large enough to cover the nonlinearity width.
$gamma1$	The gain matrix for adapting the output neuron weights W. $gamma1$ must be symmetric and greater than zero; $gamma1$ inverse must also be greater than zero. The gain matrix dimensions are $[nn]$, where n is the number of hidden neurons.
W_0	The initial conditions for the output neuron weights W. These values must be non-zero and should uniformly span the actuator's input range. The W_0 dimensions are $[n1]$, where n is the number of hidden neurons.
a_I	The scalar activation function parameter for the NN that approximates the nonlinear inverse (NI). This value must be greater than 0.
V_I	The constant hidden layer weights for the NN that approximates the nonlinear inverse (NI). This matrix has dimensions $[2m]$, where m is the number of hidden neurons. Each element v_{ij} is the weight of the link between input node i and hidden node j. These weights represent the stiffness of the activation functions and should be randomly selected $-1 \leq v_{ij} \leq 1$.
v_0_I	The constant hidden layer bias in the activation function for the NN that approximates the nonlinear inverse (NI). This vector has the dimensions $[m1]$, where m is the number of hidden neurons. Each element v_0_j is the bias for the activation function of hidden node j. These values should have a range large enough to cover the nonlinearity width.
$gamma2$	The gain matrix for adapting the output neuron weights W_I. $gamma2$ must be symmetric and greater than zero; $gamma2$ inverse must also be greater than zero. The gain matrix dimensions are $[mm]$, where m is the number of hidden neurons.
W_0_I	The initial conditions for the output neuron weights W_I. These values must be non-zero and should uniformly span the actuator's input range. The W_0_I dimensions are $[m1]$, where m is the number of hidden neurons.

Usage

To use the *ANN3NI* component, the system must be linear (or linearizable) with measurable system states and a single nonlinear actuator. The system may have any combination of single or multiple inputs and outputs. The actuator's nonlinearity (N) must be a function of two variables: $N(u\_d, alpha)$.

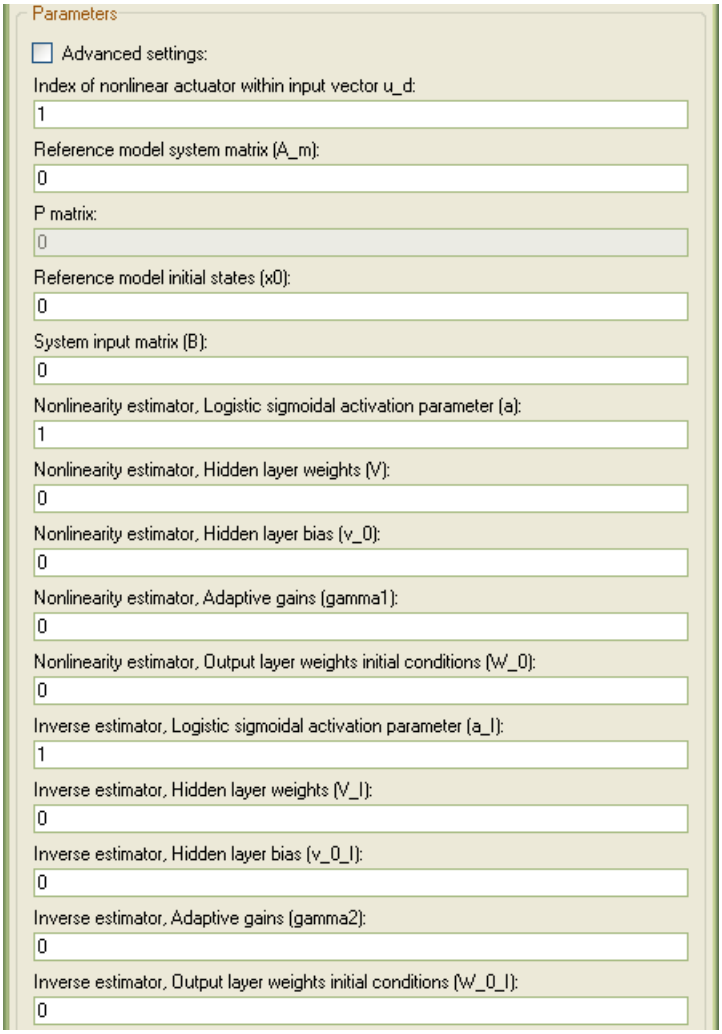


Figure 4: Adaptive Neural Network Simulink Block Parameters

The engineer will decide upon some number of hidden nodes and perform simulations to evaluate performance. This number may be increased or decreased to achieve a balance between accuracy and speed. The adaptive output layer weights (W and W_I) may be output to the workspace or a file, and serve as initial conditions (W_0 and W_0_I) to subsequent runs in order to start off with more accuracy.

Example

Consider an aircraft model with the states $x = [\alpha \;\; p \;\; \beta \;\; q \;\; r]^T$ where α and β are the angle of attack and sideslip angles, and p, q, and r are the roll, pitch, and yaw rates, respectively. The control inputs are $u = [\delta_e \;\; \delta_a \;\; \delta_r]^T$, which are the elevator, aileron, and rudder deflection angles, respectively. The elevator deflection angle δ_e is implemented through a synthetic jet actuator.

Since the width of the nonlinearity is -20.076 to 25, the bias vectors v_0 and v_0_I can be initialized between -30 to $+30$. The output weight initial conditions W_0 and W_0_I are chosen to be uniformly randomly distributed between -50 to $+50$. The simulation model is shown in Figure 5.

Using system matrices representative of a commercial transport aircraft, we conducted a set of simulations. The system is driven by sinusoidal inputs applied to all three control surfaces. The number of neurons was steadily increased until acceptable performance was attained at $n = m = 20$ hidden neurons for both approximators. The state errors are shown in Figure 6.

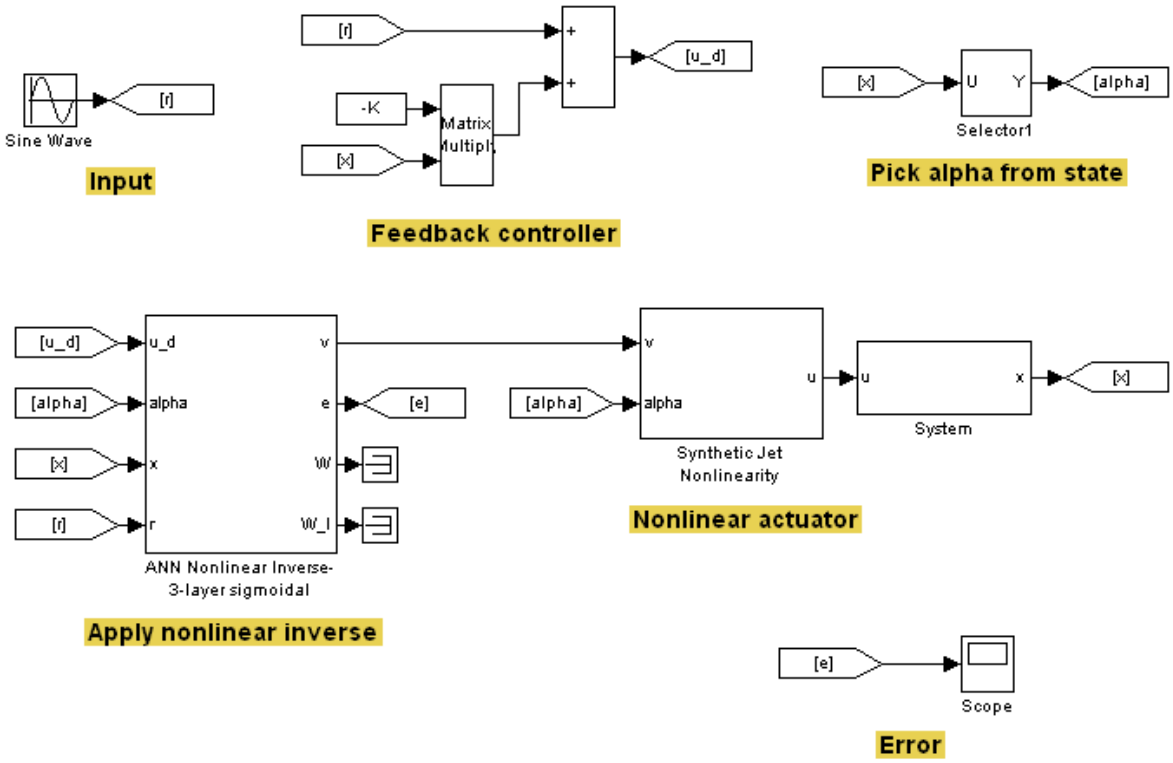


Figure 5: ANN3NI Model

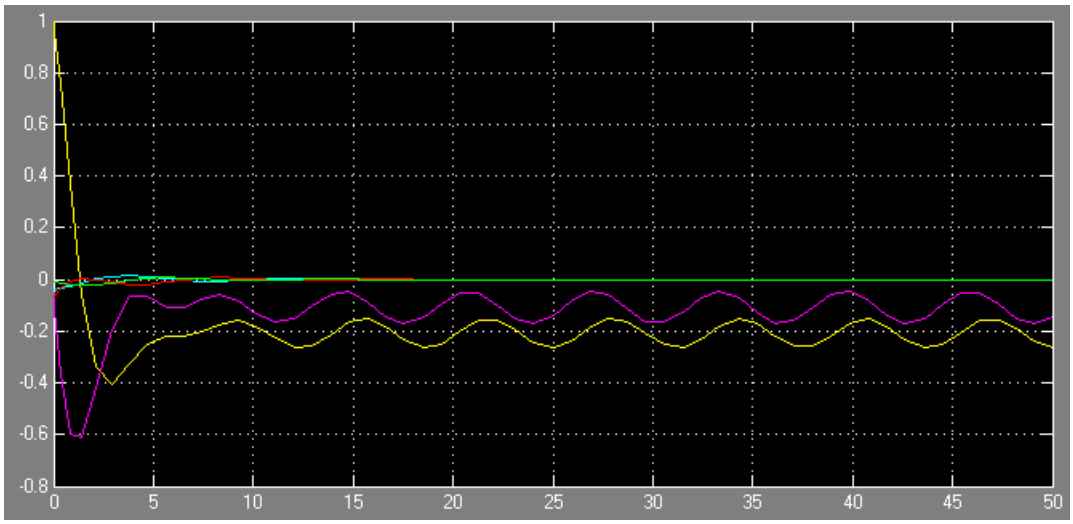


Figure 6: ANN3NI State Errors

5.3.2 Backlash Inverse

Description

The Simulink backlash inverse (*BI*) blocks (continuous and discrete) are shown in Figure 7. The block inputs and outputs are those defined in the generalized system of Figure 2. In addition, the adaptive parameters are provided as outputs in the form of $theta$ (16) and in the slope-intercept form of the backlash characteristics (6), which are useful for providing initial conditions to the *BI* block prior to deployment. The design engineer may run simulations to determine the $theta$ values to which the *BI* adapts when producing a suitably small error. Those $theta$ values may then be used as initial conditions (i.e. $theta(0)$) prior to code generation and deployment, ensuring acceptable control from the start.

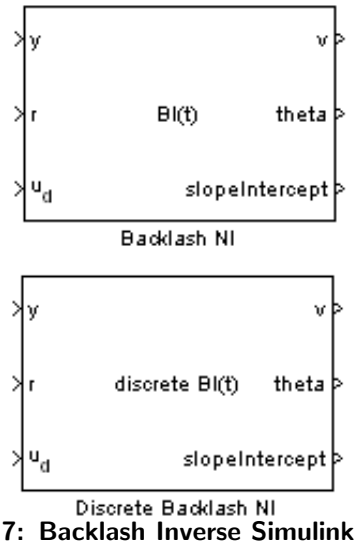


Figure 7: Backlash Inverse Simulink Blocks

The *BI* blocks apply to continuous- and discrete-time single-input / single-output (SISO) systems that are being driven by actuators containing backlash. The blocks allow the engineer to specify the system equations in the most convenient of state-space, pole-zero-gain, or polynomial transfer function form. They have no dependency upon any toolbox outside of the standard Simulink blocks. These blocks use feedback to adaptively estimate the backlash parameters, and compensate the actuator input command signal so that the system input u is as desired (u_d) to achieve the required system response, y.

Algorithm

Backlash characteristic. A backlash characteristic (Figure 8) $u(t) = B(v(t)) = B(m, c_r, c_l; v(t))$, which is a dynamic nonlinearity, is described by two parallel straight lines connected with horizontal line segments. The upward side is active when both $v(t)$ and $u(t)$ increase:

$$u(t) = m(v(t) - c_r),\ \dot{v}(t) > 0,\ \dot{u}(t) > 0. \tag{2}$$

The downward side is active when both $v(t)$ and $u(t)$ decrease:

$$u(t) = m(v(t) - c_l),\ \dot{v}(t) < 0,\ \dot{u}(t) < 0, \tag{3}$$

where $m > 0$, $c_l < c_r$ are constant parameters. The motion on any inner segment is characterized by $\dot{u}(t) = 0$ even if $v(t)$ increases or decreases.

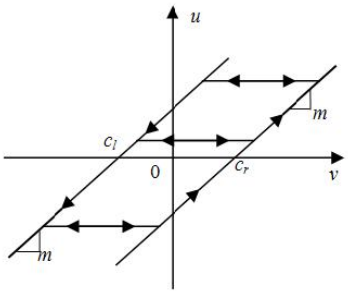


Figure 8: Backlash Characteristic

A compact description of the backlash characteristic $B(\cdot)$ is

$$\dot{u}(t) = \begin{cases} m\dot{v}(t) & \text{if } \dot{v}(t) > 0 \text{ and } u(t) = m(v(t) - c_r) \text{ or} \\ & \text{if } \dot{v}(t) < 0 \text{ and } u(t) = m(v(t) - c_l), \\ 0 & \text{otherwise.} \end{cases} \tag{4}$$

A backlash characteristic in *discrete time* can be described as

$$u(t) = B(v(t)) = \begin{cases} m(v(t) - c_l) & \text{if } v(t) \le v_l, \\ m(v(t) - c_r) & \text{if } v(t) \ge v_r, \\ u(t-1) & \text{if } v_l < v(t) < v_r, \end{cases} \tag{5}$$

where the v-axis projections v_l and v_r of the intersections of the two parallel lines of slope m with the horizontal inner segment containing $u(t-1)$ are

$$v_l = \frac{u(t-1)}{m} + c_l, \; v_r = \frac{u(t-1)}{m} + c_r. \tag{6}$$

For a parametrized nonlinearity $N(\cdot)$, we will develop an adaptive inverse as a compensator for canceling $N(\cdot)$ with unknown parameters.

Backlash inverse. Let the estimates of the backlash parameters m, c_r, c_l be $\widehat{m}$, $\widehat{c_l}$, $\widehat{c_r}$. A backlash inverse $v(t) = \widehat{BI}(u_d(t))$ is described by

$$\dot{v}(t) = \begin{cases} \frac{1}{\widehat{m}}\dot{u}_d(t) & \text{if } \dot{u}_d(t) > 0 \text{ and } v(t) = \frac{u_d(t)}{\widehat{m}} + \widehat{c_r} \text{ or} \\ & \text{if } \dot{u}_d(t) < 0 \text{ and } v(t) = \frac{u_d(t)}{\widehat{m}} + \widehat{c_l}, \\ 0 & \text{if } \dot{u}_d(t) = 0, \\ \widehat{g}(t,t) & \text{if } \dot{u}_d(t) > 0 \text{ and } v(t) = \frac{u_d(t)}{\widehat{m}} + \widehat{c_l}, \\ -\widehat{g}(t,t) & \text{if } \dot{u}_d(t) < 0 \text{ and } v(t) = \frac{u_d(t)}{\widehat{m}} + \widehat{c_r}, \end{cases} \tag{7}$$

where, with the Dirac δ-function $\delta(t)$,

$$\widehat{g}(\tau, t) = \delta(\tau - t)(\widehat{c_r} - \widehat{c_l}). \tag{8}$$

The signal motion of $v(t)$ and $u_d(t)$ of the backlash inverse is on two straight lines and vertical jumps between the lines, where the downward side line is

$$v(t) = \frac{u_d(t)}{\widehat{m}} + \widehat{c_l}, \; \dot{u}_d(t) < 0, \tag{9}$$

and the upward side line is

$$v(t) = \frac{u_d(t)}{\widehat{m}} + \widehat{c_r}, \; \dot{u}_d(t) > 0. \tag{10}$$

Vertical jumps of $v(t)$ occur whenever $\dot{u}_d(t)$ changes its sign: from the downward line to the upward line if from $\dot{u}_d(t) < 0$ to $\dot{u}_d(t) > 0$ or from the upward line to the downward line if from $\dot{u}_d(t) > 0$ to $\dot{u}_d(t) < 0$.

The *discrete-time* backlash inverse $v(t) = \widehat{BI}(u_d(t))$ is

$$v(t) = \begin{cases} v(t-1) & \text{if } u_d(t) = u_d(t-1), \\ \frac{u_d(t)}{\widehat{m}} + \widehat{c_l} & \text{if } u_d(t) < u_d(t-1), \\ \frac{u_d(t)}{\widehat{m}} + \widehat{c_r} & \text{if } u_d(t) > u_d(t-1). \end{cases} \tag{11}$$

The backlash inverse has the desired property

$$u_d(t_0) = B(BI(u_d(t_0))) \Rightarrow B(BI(u_d(t))) = u_d(t),\ \forall t \geq t_0, \tag{12}$$

where $BI(\cdot) = \widehat{BI}(\cdot)|_{\widehat{m}=m,\widehat{c_l}=c_l,\widehat{c_r}=c_r}$, the exact backlash inverse.

To parametrize the above backlash inverse, using the indicator function χ, we introduce the backlash inverse indicator functions

$$\widehat{\chi_r}(t) = \chi[v(t) = \tfrac{u_d(t)}{\widehat{m}} + \widehat{c_r}], \tag{13}$$

$$\widehat{\chi_l}(t) = \chi[v(t) = \tfrac{u_d(t)}{\widehat{m}} + \widehat{c_l}] \tag{14}$$

and the backlash indicator functions

$$\chi_r(t) = \chi[\dot{u}(t) > 0],\ \chi_l(t) = \chi[\dot{u}(t) < 0],\ \chi_s(t) = \chi[\dot{u}(t) = 0]. \tag{15}$$

With $\widehat{mc_r} = \widehat{m}\widehat{c_r}$, $\widehat{mc_l} = \widehat{m}\widehat{c_l}$, we also introduce

$$\theta = [\widehat{mc_r}, \widehat{m}, \widehat{mc_l}]^T, \tag{16}$$

$$\theta^* = [mc_r, m, mc_l]^T, \tag{17}$$

$$\omega(t) = [\widehat{\chi_r}(t), -v(t), \widehat{\chi_l}(t)]^T. \tag{18}$$

It can be verified that

$$u_d(t) = -\theta^T \omega(t), \tag{19}$$

$$u(t) - u_d(t) = (\theta - \theta^*)^T \omega(t) + d_N(t), \tag{20}$$

where, with u_s being the value of $u(t)$ when $\dot{u}(t) = 0$,

$$\begin{aligned} d_N(t) = &(\chi_r(t) - \widehat{\chi_r}(t))(m(v(t) - c_r)) \\ &+ (\chi_l(t) - \widehat{\chi_l}(t))(m(v(t) - c_l)) + \chi_s(t)u_s. \end{aligned} \tag{21}$$

Input Signals

- r The input signal to the actuator's controller (e.g. pilot stick input).
- u_d The desired signal input to an ideal (i.e. no backlash) actuator that would achieve the response dictated by the reference signal, r.
- y The measured system response to the actuator control.

Output Signals

v	The compensated actuator driving signal that will achieve the desired system response.
$theta$	The adapting nonlinear inverse parameters, in an internal form.
$slopeIntercept$	The adapting nonlinear inverse parameters, $[m, c_r, c_l]$.

Parameters and Dialog Box

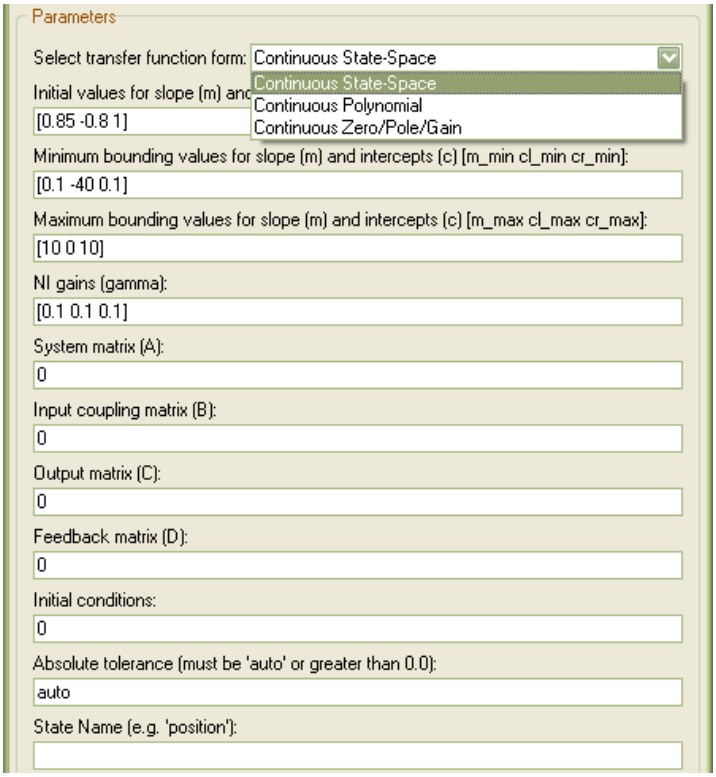


Figure 9: Backlash Inverse Simulink Block Parameters

Transfer Function Type	Selects the form for expressing the system dynamics. The parameters dialog will adjust to show all parameters required by the chosen transfer function type, which is one of state-space, polynomial, or zero/pole/gain.
Initial Values	A vector of the initial values for the backlash parameters, $[m0, c_r0, c_l0]$.
Minimum Bounding Values	A 3-element vector of the minimum bounds for the backlash parameters, $[m_{min}, c_{lmin}, c_{rmin}]$.
Maximum Bounding Values	A 3-element vector of the maximum bounds for the backlash parameters, $[m_{max}, c_{lmax}, c_{rmax}]$.
NI Gains	A 3-element vector of the gains (Γ) controlling the backlash parameter adaptation, $[\Gamma_m, \Gamma_{cl}, \Gamma_{cr}]$.

Usage

The *BI* components are applicable to continuous- and discrete-time single-input / single-output (SISO) systems driven by actuators containing backlash, and having measurable system output, y. The generalized feedback control system is shown in Figure 2.

The engineer will design a controller (C_1) to generate the idealized actuator control signal (u_d) as though the actuator did not exhibit backlash. This control signal will feed into the *BI* block, be compensated and produce the actuator driving signal (v) to achieve the required response.

5.3.3 Backlash Parameter ID

Description

The AIFAC toolbox's genetic algorithm (GA)-based backlash (gaDetectBacklash) detection tool is designed to help the engineer identify the backlash nonlinearity and its parameters from data, whether it be measurement data or simulation output. In Figure 2, the block $N(\cdot)$ represents the nonlinear actuator. The tool accepts as input the actuator driving signal v and the actuator response signal u (Figure 2). It produces figures showing the actuator's input-versus-output characteristics, estimates of the nonlinearity's parameters, and a measure of the actuator's response fitted to the nonlinearity.

A most compelling capability of the GA technique is its accuracy in the presence of random measurement noise. The engineer need not measure or remove the noise from the data. Unlike calculus-based search techniques, GA methods generally do not get "fooled" by local optima. Whereas traditional search methods parse the data to identify parameters, a GA estimates parameters and then compares the solution against the data. Thus, the GA tolerates the noise because the GA is evaluating an ideal nonlinearity with the candidate parameters against the measured system data, and searches for the "best" fit, not a perfect fit.

The gaDetectBacklash tool has the following MATLAB syntax:

```
[ mb ] = gaDetectBacklash( v, u, maxGens, minFitness, visualize )
```

Algorithm A genetic algorithm (GA) is a global search heuristic technique that seeks to find an exact or approximate solution by using stochastic processes to "evolve" to the solution. One application of GA is curve fitting, and the AIFAC toolbox uses GA to fit ideal dead-zone (DZ) and backlash (BL) characteristics to measured actuator data.

The most fundamental element of a GA is a "gene", which is a representation of part of a solution. One or more genes are combined to form a "chromosome", which represents a solution. In this toolbox, a gene is a parameter of the ideal DZ or BL characteristic. The GA begins by randomly generating an initial population (i.e. set) of candidate solutions, which comprise generation 0. This generation is evaluated and sorted based on each chromosome's "fitness". Fitness is a measure of the optimality of the solution. In this application, fitness is the RMS error between the ideal curve, represented by the candidate solution, and the measured actuator response. The GA then enters the select-mutate-evaluate loop.

The loop first randomly selects candidates from generation i, where the fittest candidates are more likely to be selected than the less fit candidates, and adds the selected candidates to generation $i+1$. The selected candidates each mate with a partner from the candidate pool, and through an operation known as crossover, swap their genes at a certain point and create two offspring, each of which contain parts of their parents chromosomes. The next step is to randomly mutate the genes of each generation $i+1$ candidate, adding the mutated chromosome back into generation $i+1$. Finally, generation $i+1$ is evaluated, sorted, and trimmed to a fixed population size. The loop termination criteria are evaluated, and the loop exits or operates on $i+1$. The loop termination criteria are a maximum number of loops, or a solution whose RMS error is less than or equal to a given value.

Input Parameters

v	The actuator input signal.
u	The actuator response signal.
maxGens	The maximum number of generations to evaluate (must be greater than 0.0).
minFitness	The minimum fitness measure required of an acceptable solution (must be positive).
visualize	The visualization flag. If the flag is non-zero, the tool will plot and display intermediate generations.

The *visualize* parameter enables and disables the algorithm's progress display. When visualization is enabled, the tool outputs its progress to a MATLAB figure and the console every $1/20$ of *maxGens*. The figure will plot the BL characteristic of the current best solution. The figure will update as the

generations are finished, providing an animated progress indication. The console will show the current best solution and its fitness.

The *maxGens* parameter limits the number of generations that the GA uses to develop a solution. The more generations, the longer the runtime and the more accurate the BL parameters up to a limit imposed by the noise in the signals. This parameter is the primary means of limiting the tool's runtime. In general, the engineer should start with a value of 1000. Adjust the parameter upwards if the resulting fitness is not acceptable. Visualization may indicate that RMS error does not decrease below a certain value due to the noise.

The *minFitness* parameter is another terminating condition that allows the engineer to quantify the maximum RMS error (e.g. minimum fitness) of an acceptable solution. The RMS error will never be zero in the presence of noise.

Output Parameters

gaDetectBacklash returns a MATLAB structure contains the best fit backlash parameters and their fitness measure. The parameters are defined in Section 5.3.2. The structure fields are:

m	The slope of the backlash legs.
cr	The intercept of the rightmost backlash leg, c_r.
cl	The intercept of the leftmost backlash leg, c_l.
rms	The RMS error between these backlash parameters and the input data.

If $mb.cl > mb.cr$, then no backlash was detected, and the script displays the message "No backlash detected."

Usage

To use the gaDetectBacklash tool, the engineer must first obtain data for the actuator driving signal v and the actuator response signal u, whether by measurement or simulation. These data are passed to the MATLAB function, and the maximum generations and minimum fitness levels are varied to obtain the required accuracy. The scripts results may then be used as initial values to the Backlash Inverse block, Section 5.3.2.

Example

The system shown in Figure 10 models a unity-gain actuator with a backlash nonlinearity. The backlash is driven by a sum of sinusoids, with normally distributed white noise added to both the input and output measurements. While the system is very simple, it is important to notice that the noise is not added to the input of the actuator itself, but is only added to the actuator input and output signal measurements, as is often the case in practice. The BL has the true backlash parameters:

m	0.25
c_l	-1.2
c_r	1.0

The system generates v and u as shown in Figure 11(a). The BL characteristic can be seen in the output u, which is constant when the slope of v changes from positive to negative and vice versa. The actuator input versus response is shown in Figure 11(b). This plot shows the classic parallelogram-shaped BL characteristic. Graphical analysis of Figure 11(b) would allow the engineer to determine the initial value for the m BL parameter.

The example model logs the signals v and u to the MATLAB workspace in a structure with the default name $logsout$. Each signal is a structure field, and is itself a structure containing the timestamps and signal values in the fields $Time$ and $Data$ respectively. To numerically analyze the data, issue the MATLAB command:

```
>> [ mb ] = gaDetectBacklash( logsout.v.Data, logsout.u.Data, 200, 0.0005, 1 )
```

This command constrains the GA search to a limit of 200 generations, and a maximum acceptable RMS error of 0.0005. The GA will terminate when either of these two conditions is met. The final 1 parameter specifies that visualization is enabled. The MATLAB command prompt shows the GA progress:

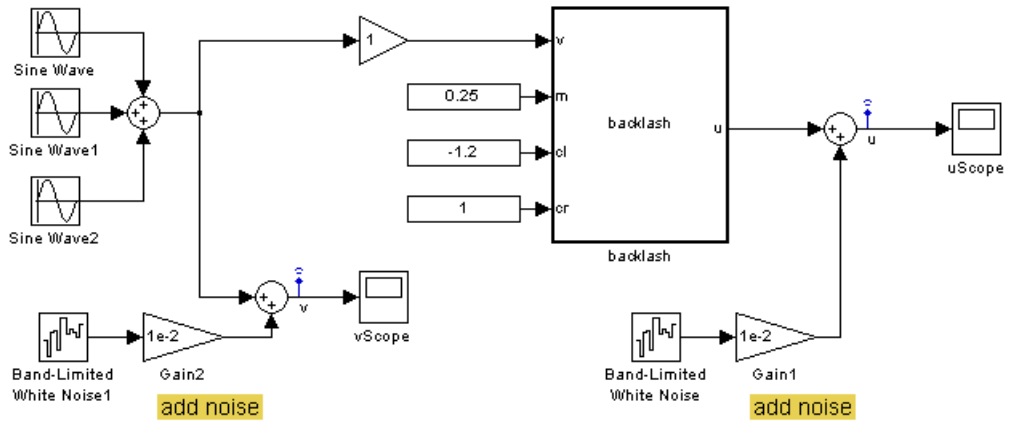
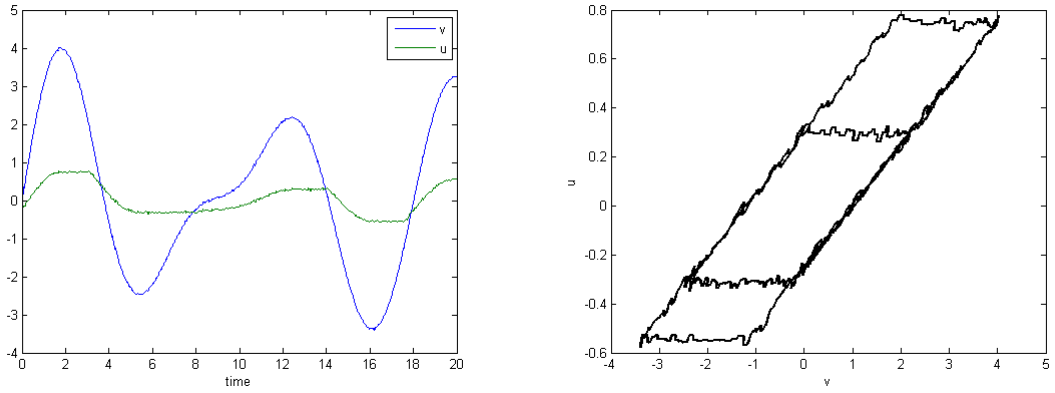


Figure 10: Backlash Model Example



(a) **Actuator input v and output u** (b) **Actuator response v vs u**

Figure 11: Backlash Input-Output

Gen	RMS	m	cl	cr
1	0.0721	0.2591	-0.9667	0.7716
11	0.0484	0.2373	-0.9667	0.8172
21	0.0408	0.2387	-1.0027	0.8662
31	0.0337	0.2354	-1.0706	0.8925
41	0.0279	0.2387	-1.0979	0.9557
51	0.0236	0.2419	-1.1604	0.9756
61	0.0212	0.2483	-1.1945	0.9863
71	0.0208	0.2483	-1.2031	1.0058
81	0.0207	0.2500	-1.2031	1.0058
91	0.0206	0.2500	-1.2038	1.0141
101	0.0206	0.2500	-1.2089	1.0121
111	0.0206	0.2500	-1.2079	1.0121
121	0.0206	0.2500	-1.2079	1.0121
131	0.0206	0.2500	-1.2079	1.0121
141	0.0206	0.2500	-1.2079	1.0121
151	0.0206	0.2500	-1.2082	1.0121

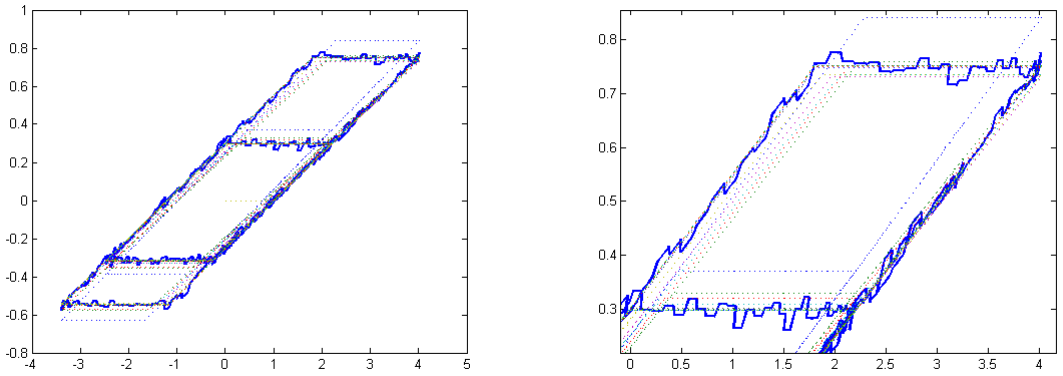
```
   161     0.0206     0.2500    -1.2082     1.0121
   171     0.0206     0.2500    -1.2082     1.0121
   181     0.0206     0.2500    -1.2082     1.0125
   191     0.0206     0.2500    -1.2082     1.0125

mb =

     m: 0.2500
    cl: -1.2082
    cr: 1.0123
   rms: 0.0206
```

The results show the evolution of the parameters and the final results.

In addition to the text output, the gaDetectBacklash tool will generate plots showing the evolution of the BL parameters. Figure 12(a) shows the final result of the evolutionary progress of the GA in determining the BL parameters. Figure 12(b) shows the detail at a backlash crossover.



(a) **BL evolutionary progress** (b) **BL evolutionary progress detail**

Figure 12: Backlash Parameter Evolution

5.3.4 Cardinal B-Splines Inverse

Description

The Cardinal B-Spline (*Spline*) Nonlinear Adaptive Inverse block compensates for a continuous-function actuator nonlinearity. The block uses a sum of Cardinal B-splines to approximate the nonlinearity.

The *Spline* approximator requires a mea-

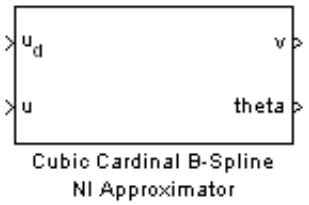


Figure 13: Cardinal B-Splines Simulink Block

surable the actuator response (u in Figure 2).

Input Signals

u_d	A signal proportional to the desired actuator response.
u	The measured actuator response signal.

Output Signals

v	The compensated actuator driving signal that will achieve the desired system response.
$theta$	The adapting nonlinear inverse parameters, in an internal form. The number of adaptive parameters ($theta$) is given by $n + k - 1$, where n is the number of knots and k is the spline order.

Parameters and Dialog Box

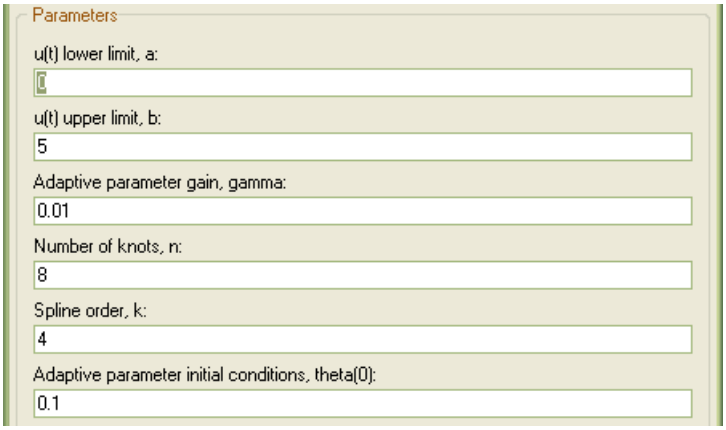


Figure 14: Cardinal B-Splines Simulink Block Parameters

$u(t)$ Lower Limit (a)	The lower bound of the closed interval describing the nonlinear function's range. This value must be less than b.
$u(t)$ Upper Limit (b)	The upper bound of the closed interval describing the nonlinear function's range. This value must be greater than a.
Adaptive Parameter Gain	The gain controlling the parameter adaptation rate.
Number of Spline Knots (N)	The number of spline knots. This value must be an integer greater than 1.
Spline Order (k)	The spline order is the spline curve's degree plus one. (Thus a second degree curve has order 3.) This value must be an integer greater than 0.
$theta(0)$	$theta(0)$ are the adaptive parameter initial conditions. It may be a scalar, which will be the initial value for all $theta_i$. $theta(0)$ may be a vector, in which case $theta_i = theta(0)_i$. If $size(theta(0)) < N$, then the last $theta(0)$ value will be duplicated for the remaining $theta_i$ values.

Usage

To use the *Spline* component, the system the actuator response must be measurable. During simulation and development, the theta outputs may be connected to a display or workspace block and the results subsequently used as the parameter initial values ($theta(0)$). This will start the next iteration with a more accurate approximation.

Example

Figure 15 shows the usage of the *Spline* block. The "Actuator" block models the actuator exhibiting a nonlinearity (e.g. ship rudder hydraulics). The actuator's response is measurable (e.g. rudder position). The "System" block models the system response to the actuator (e.g. ship's heading). The "Feedback Controller" block models the system controller (e.g. autopilot). The "Command Input" block models the controller input signal (e.g. helm wheel).

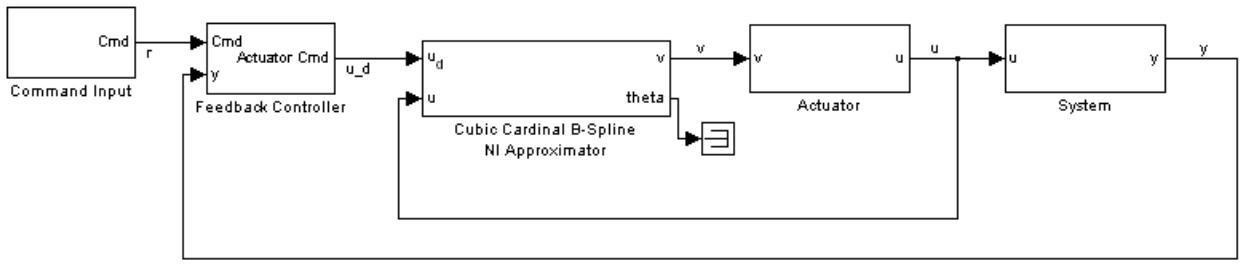


Figure 15: Cardinal B-Splines Block Simulink Example

5.3.5 Dead-zone Inverse

Description

The Simulink© dead-zone inverse (*DI*) blocks (continuous and discrete) are shown in Figure 16. The block inputs and outputs are those defined in the generalized system of Figure 2. In addition, the adaptive parameters are provided as outputs in the form of $theta$ (32) and in the slope-intercept form of the dead-zone characteristics (22), which are useful for providing initial conditions to the *DI* block prior to deployment. The design engineer may run simulations to determine the $theta$ values to which the *DI* adapts when producing a suitably small error. Those $theta$ values may then be used as initial conditions (i.e. $theta(0)$) prior to code generation and deployment, ensuring acceptable control from the start.

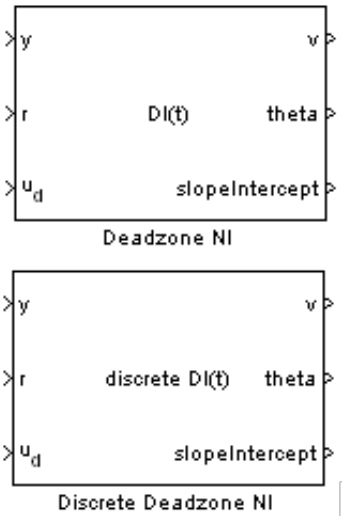


Figure 16: Dead-zone Inverse Simulink Blocks

The *DI* blocks apply to continuous- and discrete-time single-input / single-output (SISO) systems that are being driven by actuators containing dead-zones. The blocks allow the engineer to specify the system equations in the most convenient of state-space, pole-zero-gain, or polynomial transfer function form. They have no dependency upon any toolbox outside of the standard Simulink blocks. These blocks use feedback to adaptively estimate the dead-zone parameters, and compensate the actuator input command signal so that the system input u is as desired (u_d) to achieve the required system

response, y.

Algorithm The dead-zone characteristic $DZ(\cdot)$ is described as

$$u(t) = N(v(t)) = DZ(v(t)) = \begin{cases} m_r(v(t) - b_r) & \text{if } v(t) \geq b_r \\ 0 & \text{if } b_l < v(t) < b_r \\ m_l(v(t) - b_l) & \text{if } v(t) \leq b_l, \end{cases} \tag{22}$$

where $m_r > 0$, $m_l > 0$, $b_r > 0$, and $b_l < 0$ (see Figure 17).

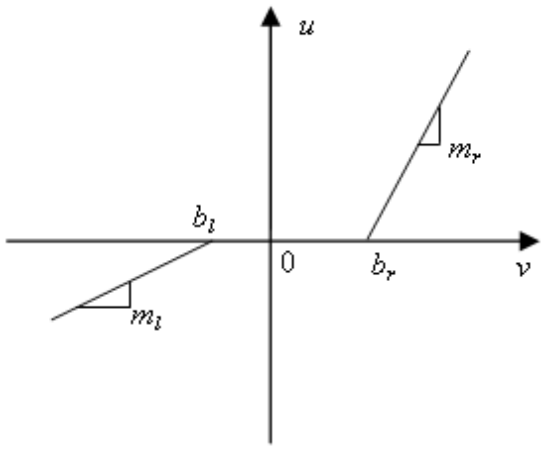


Figure 17: Dead-zone Characteristic

The parametrized model of the dead-zone nonlinearity can be unified as

$$u(t) = N(v(t)) = N(\theta^*; v(t)) = -\theta^{*T}\omega^*(t) + a_s^*(t) \tag{23}$$

Introducing the indicator function $\chi[X]$ of the event X as

$$\chi[X] = \begin{cases} 1 & \text{if } X \text{ is true,} \\ 0 & \text{otherwise,} \end{cases} \tag{24}$$

we define the dead-zone indicator functions

$$\chi_r(t) = \chi[u(t) > 0], \tag{25}$$

$$\chi_l(t) = \chi[u(t) < 0]. \tag{26}$$

Then, introducing the dead-zone parameter vector and its regressor,

$$\theta^* = [m_r, m_r b_r, m_l, m_l b_l]^T, \tag{27}$$

$$\omega^*(t) = [-\chi_r(t)v(t), \chi_r(t), -\chi_l(t)v(t), \chi_l(t)]^T, \tag{28}$$

we obtain the desired form (23) with $a_s^*(t) = 0$ for the dead-zone characteristic (22).

Dead-zone inverse. Let the estimates of the dead-zone parameters $m_r b_r$, m_r, $m_l b_l$, m_l be $\widehat{m_r b_r}$, $\widehat{m_r}$, $\widehat{m_l b_l}$, $\widehat{m_l}$, respectively. Then the inverse for the dead-zone characteristic is described by

$$v(t) = \widehat{NI}(u_d(t)) = \widehat{DI}(u_d(t)) = \begin{cases} \frac{u_d(t)+\widehat{m_r b_r}}{\widehat{m_r}} & \text{if } u_d(t) > 0, \\ 0 & \text{if } u_d(t) = 0, \\ \frac{u_d(t)+\widehat{m_l b_l}}{\widehat{m_l}} & \text{if } u_d(t) < 0. \end{cases} \tag{29}$$

For the dead-zone inverse to arrive at the desired form, we introduce the inverse indicator functions

$$\widehat{\chi_r}(t) = \chi[v(t) > 0], \tag{30}$$

$$\widehat{\chi_l}(t) = \chi[v(t) < 0] \tag{31}$$

and the inverse parameter vector and regressor

$$\theta = [\widehat{m_r}, \widehat{m_r b_r}, \widehat{m_l}, \widehat{m_l b_l}]^T, \tag{32}$$

$$\omega(t) = [-\widehat{\chi_r}(t)v(t), \widehat{\chi_r}(t), -\widehat{\chi_l}(t)v(t), \widehat{\chi_l}(t)]^T. \tag{33}$$

Then, the dead-zone inverse is

$$\begin{aligned} u_d(t) &= \widehat{m_r}\widehat{\chi_r}(t)v(t) - \widehat{m_r b_r}\widehat{\chi_r}(t) + \widehat{m_l}\widehat{\chi_l}(t)v(t) - \widehat{m_l b_l}\widehat{\chi_l}(t) \\ &= -\theta^T\omega(t). \end{aligned} \tag{34}$$

Input Signals

r	The input signal to the actuator's controller (e.g. pilot stick input).
u_d	The desired signal input to an ideal (i.e. no dead-zone) actuator that would achieve the response dictated by the reference signal, r.
y	The measured system response to the actuator control.

Output Signals

v	The compensated actuator driving signal that will achieve the desired system response.
$theta$	The adapting nonlinear inverse parameters, in an internal form.
$slopeIntercept$	The adapting nonlinear inverse parameters, $[m_l, b_l, m_r, c_r]$.

Parameters and Dialog Box

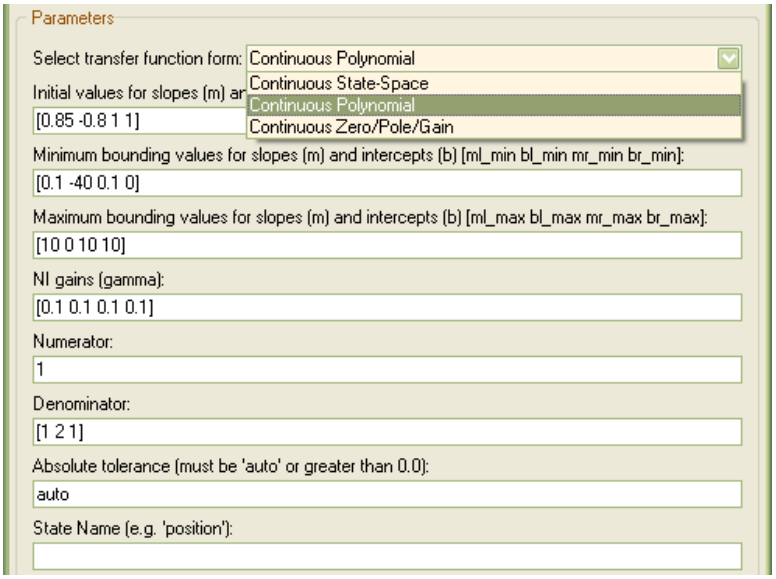


Figure 18: Dead-zone Inverse Simulink Block Parameters

Transfer Function Type	Selects the form for expressing the system dynamics. The parameters dialog will adjust to show all parameters required by the chosen transfer function type, which is one of state-space, polynomial, or zero/pole/gain.
Initial Values	A vector of the initial values for the dead-zone parameters, $[m_l 0, b_l 0, m_r 0, b_l 0]$.
Minimum Bounding Values	A 4-element vector of the minimum bounds for the dead-zone parameters, $[m_{lmin}, b_{lmin}, m_{rmin}, b_{rmin}]$.
Maximum Bounding Values	A 4-element vector of the maximum bounds for the dead-zone parameters, $[m_{lmax}, b_{lmax}, m_{rmax}, b_{rmax}]$.
NI Gains	A 4-element vector of the gains (Γ) controlling the dead-zone parameter adaptation, $[\Gamma_{ml}, \Gamma_{bl}, \Gamma_{mr}, \Gamma_{br}]$.

Usage

The *DI* components are applicable to continuous- and discrete-time single-input / single-output (SISO) systems driven by actuators containing dead-zone, and having measurable system output, y. The generalized feedback control system is shown in Figure 2.

The engineer will design a controller (C_1) to generate the idealized actuator control signal (u_d) as though the actuator did not exhibit dead-zone. This control signal will feed into the *DI* block, be compensated and produce the actuator driving signal (v) to achieve the required response.

5.3.6 Dead-zone Parameter ID

Description

The AIFAC toolbox's genetic algorithm (GA)-based dead-zone (gaDetectDeadzone) detection tool is designed to help the engineer identify the dead-zone nonlinearity and its parameters from data, whether it be measurement data or simulation output. In Figure 2, the block $N(\cdot)$ represents the nonlinear

actuator. The tool accepts as input the actuator driving signal v and the actuator response signal u (Figure 2). It produces figures showing the actuator's input-versus-output characteristics, estimates of the nonlinearity's parameters, and a measure of the actuator's response fitted to the nonlinearity.

A most compelling capability of the GA technique is its accuracy in the presence of random measurement noise. The engineer need not measure or remove the noise from the data. Unlike calculus-based search techniques, GA methods generally do not get "fooled" by local optima. Whereas traditional search methods parse the data to identify parameters, a GA estimates parameters and then compares the solution against the data. Thus, the GA tolerates the noise because the GA is evaluating an ideal nonlinearity with the candidate parameters against the measured system data, and searches for the "best" fit, not a perfect fit.

The gaDetectDeadzone tool has the following MATLAB syntax:

```
[ mb ] = gaDetectDeadzone( v, u, maxGens, minFitness, visualize )
```

Algorithm A genetic algorithm (GA) is a global search heuristic technique that seeks to find an exact or approximate solution by using stochastic processes to "evolve" to the solution. One application of GA is curve fitting, and the AIFAC toolbox uses GA to fit ideal dead-zone (DZ) and backlash (BL) characteristics to measured actuator data.

The most fundamental element of a GA is a "gene", which is a representation of part of a solution. One or more genes are combined to form a "chromosome", which represents a solution. In this toolbox, a gene is a parameter of the ideal DZ or BL characteristic. The GA begins by randomly generating an initial population (i.e. set) of candidate solutions, which comprise generation 0. This generation is evaluated and sorted based on each chromosome's "fitness". Fitness is a measure of the optimality of the solution. In this application, fitness is the RMS error between the ideal curve, represented by the candidate solution, and the measured actuator response. The GA then enters the select-mutate-evaluate loop.

The loop first randomly selects candidates from generation i, where the fittest candidates are more likely to be selected than the less fit candidates, and adds the selected candidates to generation $i+1$. The selected candidates each mate with a partner from the candidate pool, and through an operation known as crossover, swap their genes at a certain point and create two offspring, each of which contain parts of their parents chromosomes. The next step is to randomly mutate the genes of each generation $i+1$ candidate, adding the mutated chromosome back into generation $i+1$. Finally, generation $i+1$ is evaluated, sorted, and trimmed to a fixed population size. The loop termination criteria are evaluated, and the loop exits or operates on $i+1$. The loop termination criteria are a maximum number of loops, or a solution whose RMS error is less than or equal to a given value.

Input Parameters

v	The actuator input signal.
u	The actuator response signal.
maxGens	The maximum number of generations to evaluate (must be greater than 0.0).
minFitness	The minimum fitness measure required of an acceptable solution (must be positive).
visualize	The visualization flag. If the flag is non-zero, the tool will plot and display intermediate generations.

The *visualize* parameter enables and disables the algorithm's progress display. When visualization is enabled, the tool outputs its progress to a MATLAB figure and the console every $1/20$ of *maxGens*. The figure will plot the DZ characteristic of the current best solution. The figure will update as the generations are finished, providing an animated progress indication. The console will show the current best solution and its fitness.

The *maxGens* parameter limits the number of generations that the GA uses to develop a solution. The more generations, the longer the runtime and the more accurate the DZ parameters up to a limit imposed by the noise in the signals. This parameter is the primary means of limiting the tool's runtime. In general, the engineer should start with a value of 1000. Adjust the parameter upwards if the resulting

fitness is not acceptable. Visualization may indicate that RMS error does not decrease below a certain value due to the noise.

The *minFitness* parameter is another terminating condition that allows the engineer to quantify the maximum RMS error (e.g. minimum fitness) of an acceptable solution. The RMS error will never be zero in the presence of noise.

Output Parameters

gaDetectDeadzone returns a MATLAB structure contains the best fit dead-zone parameters and their fitness measure. The parameters are defined in Section 5.3.5. The structure fields are:

ml	The slope of the leftmost dead-zone characteristic, m_l.
bl	The intercept of the leftmost dead-zone characteristic, b_l.
mr	The slope of the rightmost dead-zone characteristic, m_r.
cr	The intercept of the rightmost dead-zone characteristic, b_r.
rms	The RMS error between these dead-zone parameters and the input data.

Usage

To use the gaDetectDeadzone tool, the engineer must first obtain data for the actuator driving signal v and the actuator response signal u, whether by measurement or simulation. These data are passed to the MATLAB function, and the maximum generations and minimum fitness levels are varied to obtain the required accuracy. The scripts results may then be used as initial values to the Deadzone Inverse block, Section 5.3.5.

Example

The system shown in Figure 5.3.6 models a unity-gain actuator with a dead-zone nonlinearity. The dead-zone is driven by a sum of sinusoids, with normally distributed white noise added to both the input and output measurements. While the system is very simple, it is important to notice that the noise is not added to the input of the actuator itself, but is only added to the actuator input and output signal measurements, as is often the case in practice. The DZ has the true dead-zone parameters:

m_l	6.5
b_l	-1.1
m_r	0.7
b_r	0.2

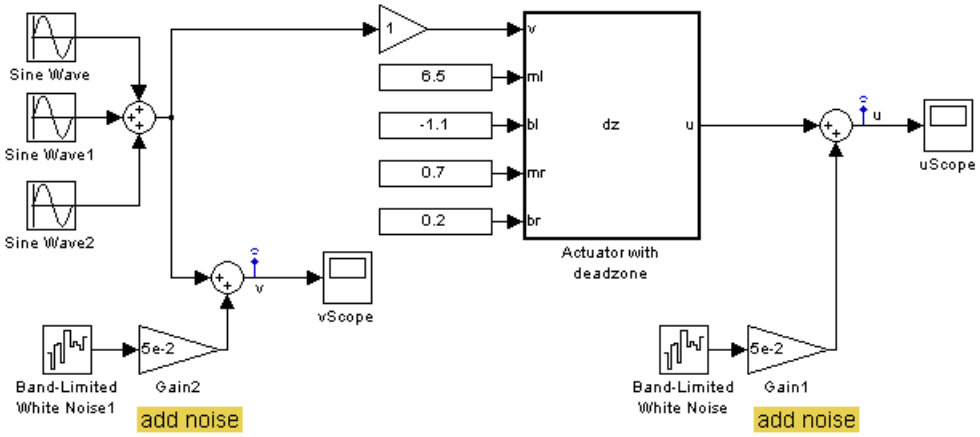
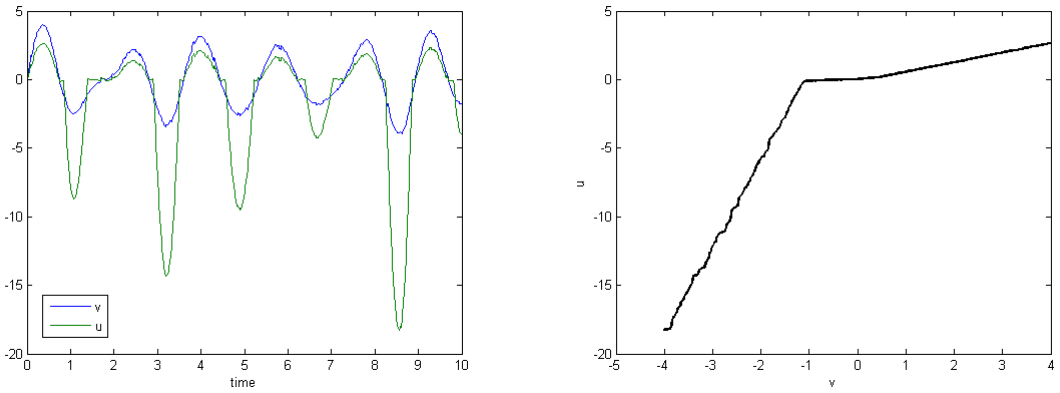


Figure 19: Dead-zone Model Example

The system generates v and u as shown in Figure 20(a). The DZ characteristic can be seen in the output u, which is zero plus some noise, when $b_l < v < b_r$. The actuator input versus response is shown in Figure 20(b). This plot shows the classic DZ characteristic. Graphical analysis of these plots would allow the engineer to determine initial values for the DZ parameters.



(a) **Actuator input v and output u** (b) **Actuator response v vs u**

Figure 20: Dead-zone Model Input-Output

The example model logs the signals v and u to the MATLAB workspace in a structure with the default name $logsout$. Each signal is a structure field, and is itself a structure containing the timestamps and signal values in the fields $Time$ and $Data$ respectively. To numerically analyze the data, issue the MATLAB command:

```
>> [ mb ] = gaDetectDeadzone( logsout.v.Data, logsout.u.Data, 1000, 0.05, 1 )
```

This command constrains the GA search to a limit of 1000 generations, and a maximum acceptable RMS error of 0.05. The GA will terminate when the either of these two conditions is met. The final 1 parameter specifies that visualization is enabled. The MATLAB command prompt shows the GA progress:

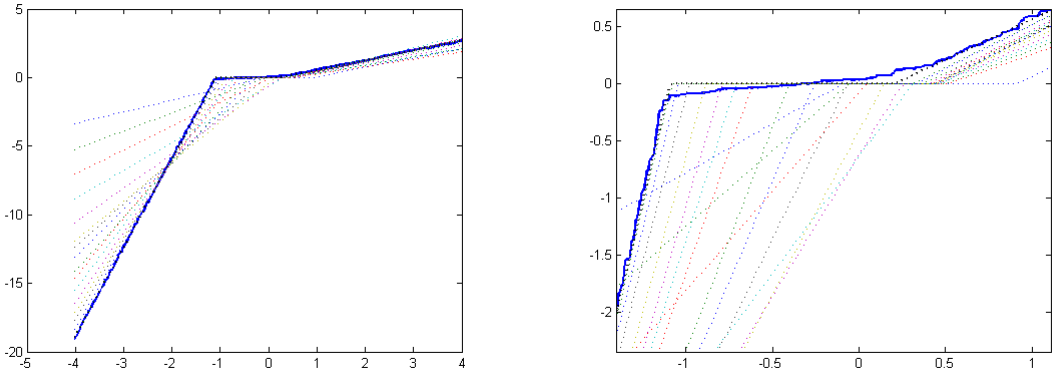
```
Gen      RMS       ml        bl        mr        br
  1    3.8083    0.8578   -0.0890    0.6888    0.9113
 51    3.2499    1.3226   -0.0542    0.5993    0.5099
101    2.7452    1.7366    0.0432    0.5380    0.5246
151    2.3082    2.0488    0.3109    0.6007    0.4605
201    1.9937    2.4798    0.2507    0.6582    0.4607
251    1.7893    2.8661    0.1432    0.7127    0.4518
301    1.6195    3.1350   -0.0669    0.7655    0.4447
351    1.4233    3.4925   -0.2549    0.7579    0.4457
401    1.2148    3.9167   -0.4020    0.6990    0.3678
451    1.0064    4.2972   -0.6125    0.7873    0.4280
501    0.8241    4.6888   -0.7074    0.8694    0.4845
551    0.6344    5.1007   -0.7902    0.7891    0.4037
601    0.4645    5.4775   -0.8834    0.7604    0.4507
651    0.3052    5.8246   -0.9808    0.7701    0.4378
701    0.1613    6.1635   -1.0418    0.7701    0.3347
751    0.0676    6.4348   -1.0807    0.7492    0.3127

mb =
```

```
 ml: 6.4659
 bl: -1.0888
 mr: 0.7127
 br: 0.2117
rms: 0.0496
```

The results show the evolution of the parameters, and the final results. Notice that the b_l starts out negative, evolves to a positive value, and the reverses direction to converge to -1.0888, which is very close to the true -1.1. This result demonstrates that the GA may identify false solutions in the short term, but will converge to a correct solution if correctly designed and left to evolve. It also cautions the user against using too few generations. As shown, if the maximum number of generations had been 150, the b_l parameter would have been about 0.3109, which is the worst parameter estimate. Again, the DI block would eventually adapt and converge to the true parameters, but would take longer given the poor initial estimate.

In addition to the text output, the gaDetectDeadzone tool will generate plots showing the evolution of the DZ parameters. Figure 21(a) shows the final result of the evolutionary progress of the GA in determining the DZ parameters. Figure 21(b) shows the detail at the dead-zone.



(a) **DZ evolutionary progress** (b) **DZ evolutionary progress detail**

Figure 21: Dead-zone Parameter Evolution

5.3.7 Disturbance Rejection

Description

The Disturbance Rejection (SISO, omega4) (*DR*) block calculates an adaptive system input signal (u_0) that cancels the output effect of a system state disturbance signal. This block allows the controller to operate without compensating for the disturbance.

The *DR* block is applicable to single-input, single-output (SISO) linear (or linearizable) systems with measurable system states, and an external

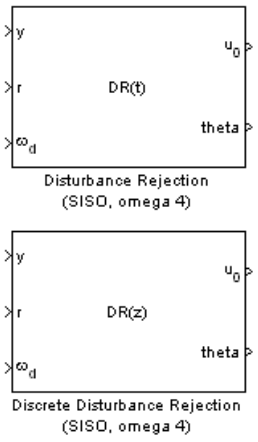


Figure 22: Disturbance Rejection Simulink Block

state disturbance signal.

Algorithm The control system model is shown in Figure 23. δ is the non-constant disturbance function modeled by the disturbance vector (B_d), and the disturbance function, $d(t)$. $d(t) = d0 + \Sigma d_j f j(t)$, where $d0$ and d_j are real constants and $f_j(t)$ are real continuous functions for $j = 1, 2, ..., q$.

The *DR* block approximates the disturbance using the input basis function vector ($omega\_d$) and adaptively calculated weights ($theta$) for the basis. It uses a reference model approximation [3] to calculate the disturbance rejection signal (u_0) which is added to the system input to reject the actuator disturbance.

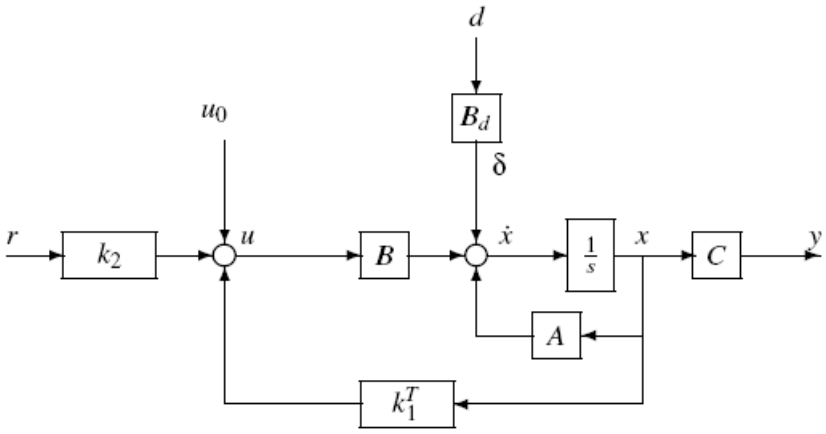


Figure 23: Disturbance Rejection Control System

Input Signals

y	The (reference) system input signal before the controller gain.
r	A vector of basis functions that are used to approximate the nonlinear part of the system.
$omega\_d$	The basis function vector that spans the actuator disturbance. The $omega\_d(1)$ signal must be a constant 1. The vector length must be 4.

Output Signals

u_0	The disturbance rejection signal that compensates for the state disturbance. This signal must be added to the feedback and controller output to form the system input signal, $u(t)$.
$theta$	These signals are the adaptive weights that are applied to the input $omega\_d$ signals to form the disturbance rejection signal, u_0.

Parameters and Dialog Box

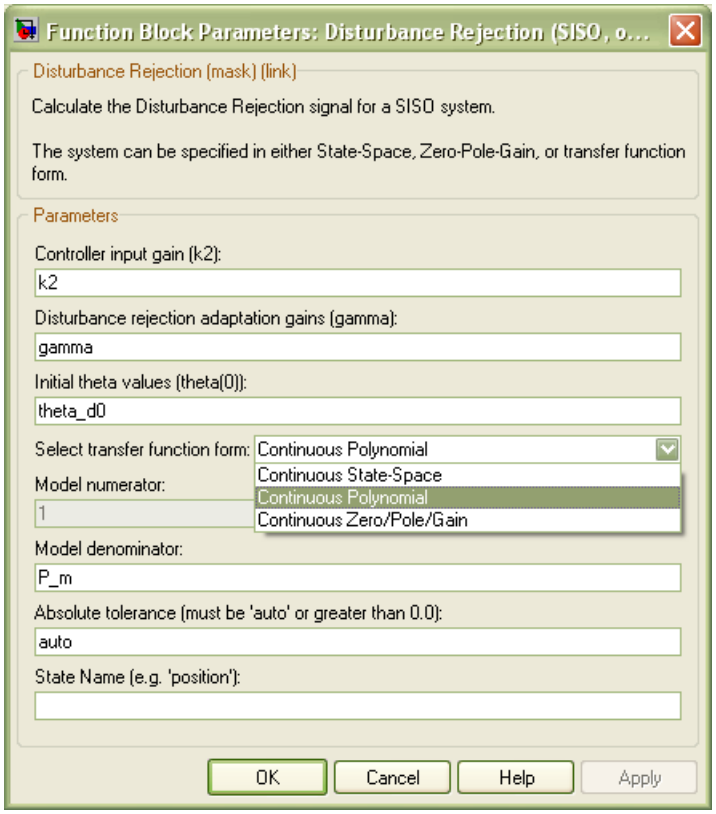


Figure 24: Disturbance Rejection Simulink Block Parameters

$k2$	The input gain for the nominal system controller.
$gamma1$	The gain vector for adapting the disturbance rejection parameters. This column vector has the same number of elements as the $omega\_d$ input signal.
$theta(0)$	The initial values for the adaptive disturbance rejection parameters, $theta$. This column vector has the same number of elements as the theta output signal.
Transfer Function Type	Selects the form for expressing the system reference model. The reference model transfer function must have no zeros (i.e. numerator $=$ 1). The denominator must be a monic polynomial of the same relative degree as the system, and must have all roots in the left half-plane. The parameters dialog will adjust to show all parameters required by the chosen transfer function type, and the Numerator and Zeros parameters will be disabled and forced to the required value, 1.

Usage

To use the *DR* component, the system must be a SISO, linear (or linearizable) system with measurable system states. Create a disturbance model to develop the disturbance signal spanning functions ($omega\_d$). Add the disturbance rejection signal (u_0) to the system input signal (u) to drive the system.

Typically, the adaptive weights ($theta$) output signal will be saved to the MATLAB workspace during simulation, and the final results used as the initial condition parameter $theta(0)$ to ensure that the deployed system starts execution with an accurate initial state.

Example

In example shown in Figure 25 is an aircraft, where the state variables in $x(t)$ are the lateral velocity $x1(t)$, roll rate $x2(t)$, yaw rate $x3(t)$, and roll angle $x4(t)$. The input $u(t)$ is used to control a synthetic jet actuator to change the output $y(t) = x4(t)$, or the roll angle. The system has an actuation disturbance $d(t) = B_d d(t)$. The transfer function has relative degree 2, so the chosen reference model is $\frac{1}{s^2+2s+2}$. The disturbance signal is a sum of high-frequency sinusoids.

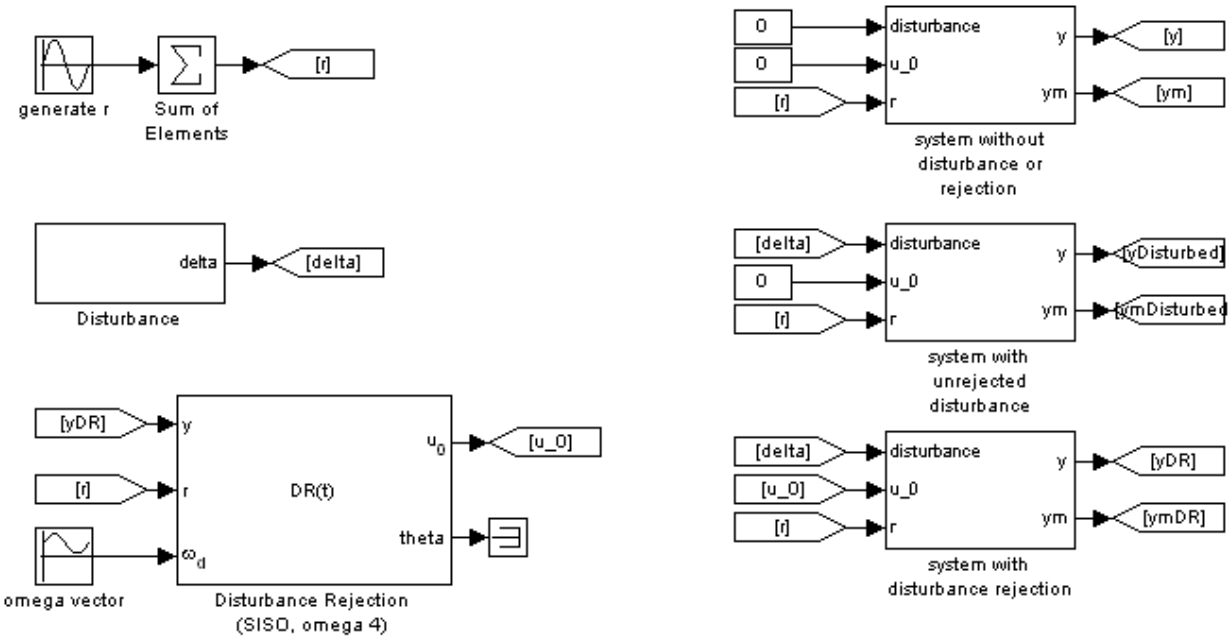


Figure 25: Disturbance Rejection Example Model

The disturbance and disturbance rejection signals are applied to the system states as shown in Figure 26.

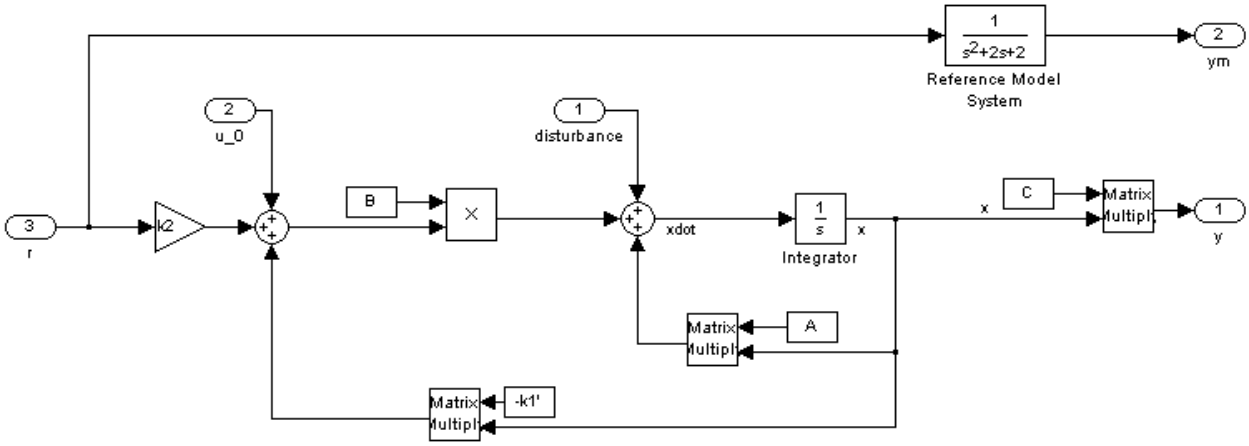


Figure 26: Disturbance Rejection Example System Model

Three system responses are plotted in Figure 27: no disturbance, unrejected disturbance, and rejected disturbance. The error signals are the differences in the system response and the reference model system response. The system with rejected disturbance shows a decaying error due to the *DR* component's adaptive rejection of the disturbance signal.

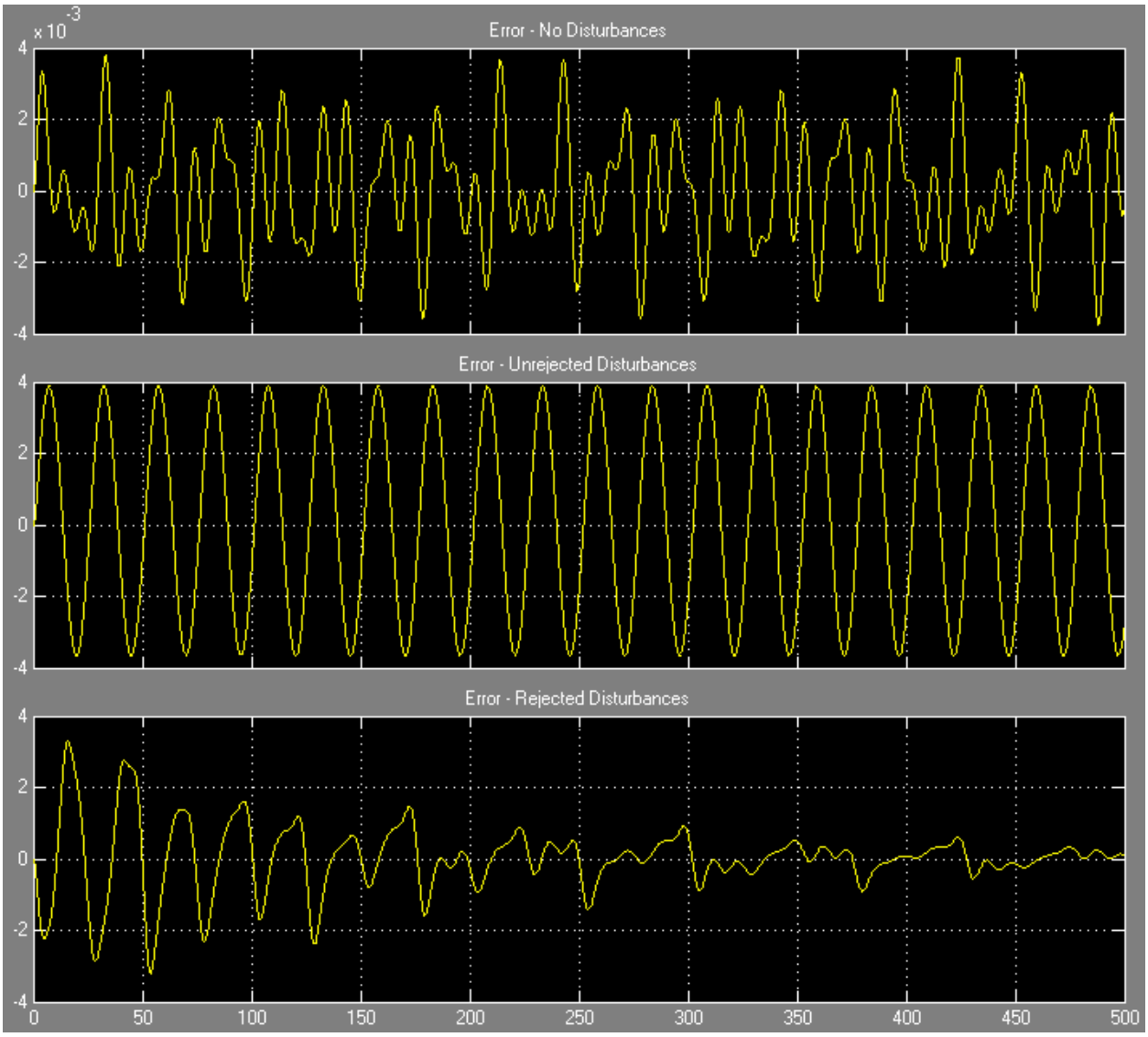


Figure 27: Disturbance Rejection Example Model Performance

5.3.8 Gaussian Radial Basis Functions Inverse

Description

The Gaussian Radial Basis Function (*GRBF*) block compensates for a continuous-function actuator nonlinearity.

The *GRBF* nonlinear inverse is applicable systems with measurable actuator response (u in Figure 2).

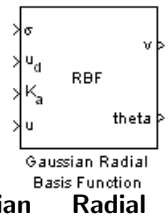


Figure 28: Gaussian Radial Basis Function Simulink Block

Algorithm The radial basis function (RBF) approx-

imator is defined as

$$\hat{f}(x) \;=\; \sum_{i=1}^{N} \theta_i g(\|x \;-\; c_i\|) \tag{35}$$

where $x \in \Re^n$, $\{c_i\}_{i=1}^N$ are a set of center locations, $\|x \;-\; c_i\|$ is the distance from the evaluation point to the i-th center, and $g(\cdot) : \Re^+ \rightarrow \Re^1$ is a radial function. One of the most common $g(\cdot)$ functions is a Gaussian:

$$g(x) \;=\; \sigma e^{-\pi\sigma^2(x-c_i)^2} \tag{36}$$

where σ is the standard deviation of the basis function.

The *GRBF* block approximates a nonlinearity as a sum of weighted Gaussian RBF. The RBF sum may be thought of as a Fourier series where the sine and cosine functions are replaced by Gaussian functions. Each curve in the sum is centered about a point in the nonlinearity's range. The figure shows a set of three Gaussian functions ($\sigma = 0.2$) that are evenly spaced at -3.0, 1.0, and 4.0. Both the number of basis functions, their standard deviations, and their centers are block parameters to be tuned for the application.

This block uses feedback to adaptively estimate the basis function weights, and compensates the actuator input command signal so that the actuator will achieve desired response.

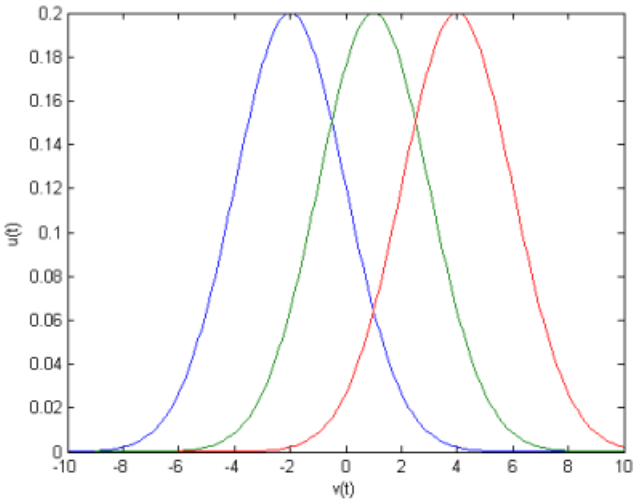


Figure 29: Gaussian Radial Basis Functions

Input Signals

σ	The standard deviation of the Gaussian basis functions.
u_d	A signal proportional to the desired actuator response.
K_a	The gain applied when adjusting the RBF weights.
u	The measured actuator response signal.

Output Signals

v	The compensated actuator driving signal that will achieve the desired system response.
$theta$	The adapting nonlinear inverse parameters, in an internal form.

Parameters and Dialog Box

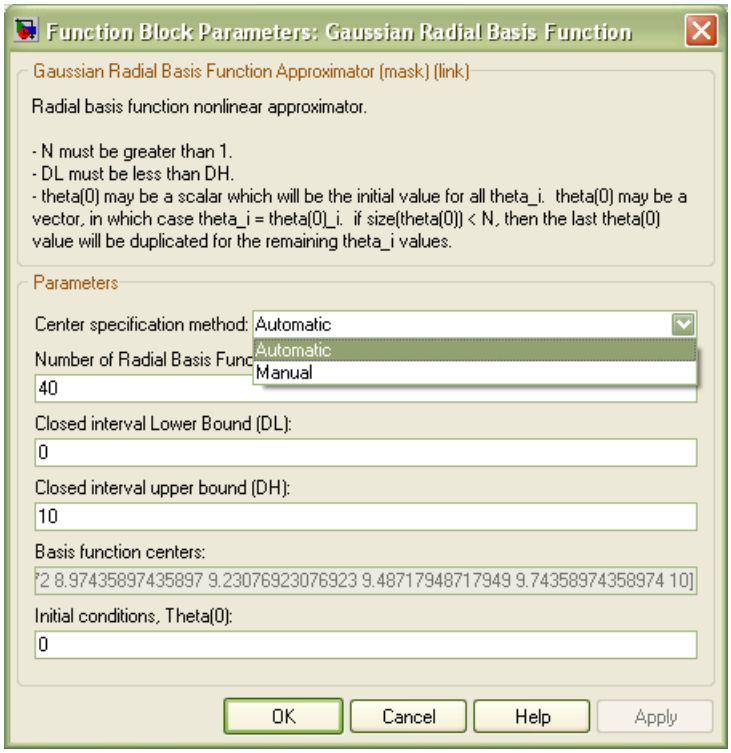


Figure 30: Gaussian Radial Basis Functions Simulink Block Parameters

Center Specification Method	This dropdown selects how RBF centers are specified. "Automatic" mode uses the number of RBFs and the bounds to evenly space the centers. "Manual" allows manual entry of centers.
Number Radial Basis Functions	The number of RBFs to employ in spanning the nonlinear function range. This value must be an integer greater than 1.
Lower Bound	The lower bound (DL) of the closed interval describing the nonlinear function's range. This value must be less than DH. The output signal v will be constrained to $DL \leq v \leq DH$.
Upper Bound	The upper bound (DH) of the closed interval describing the nonlinear function's range. This value must be greater than DL. The output signal v will be constrained to $DL \leq v \leq DH$.
Centers	The RBF function centers. When the Center Specification Method is "Automatic", this field is disabled and filled in automatically based upon the N, DL, and DH parameters. When the Center Specification Method is "Manual", this field is editable.
Initial Conditions	The initial RBF weights ($theta(0)$). $theta(0)$ may be a scalar, which will be the initial value for all $theta_i$. $theta(0)$ may be a vector, in which case $theta_i = theta(0)_i$. If $size(theta(0)) < N$, then the last $theta(0)$ value will be duplicated for the remaining $theta_i$ values.

Usage

To use the *GRBF* component, the system must be SISO where the actuator response (u) is measurable. The engineer must choose RBF parameters to adequately compensate for the nonlinearity while not overfitting the response. The runtime of the component increases as the number of RBF functions, which require the exponential, a relatively expensive function.

During simulation, the output signal $theta$ may be directed to a MATLAB Workspace block and the final results used as the initial conditions for the next run and/or deployment. This technique increases the adaptive weights' accuracy at $t = 0$.

Example

The $GRBF$ component was tested in the system shown in Figure 31. The input signal (u_d) is a sum of sinusoids. The nonlinear component is a high angle-of-attack model of the synthetic jet:

$$u(t) \;=\; 15.0 \;-\; \frac{10.0}{1 \;+\; v(t)}. \tag{37}$$

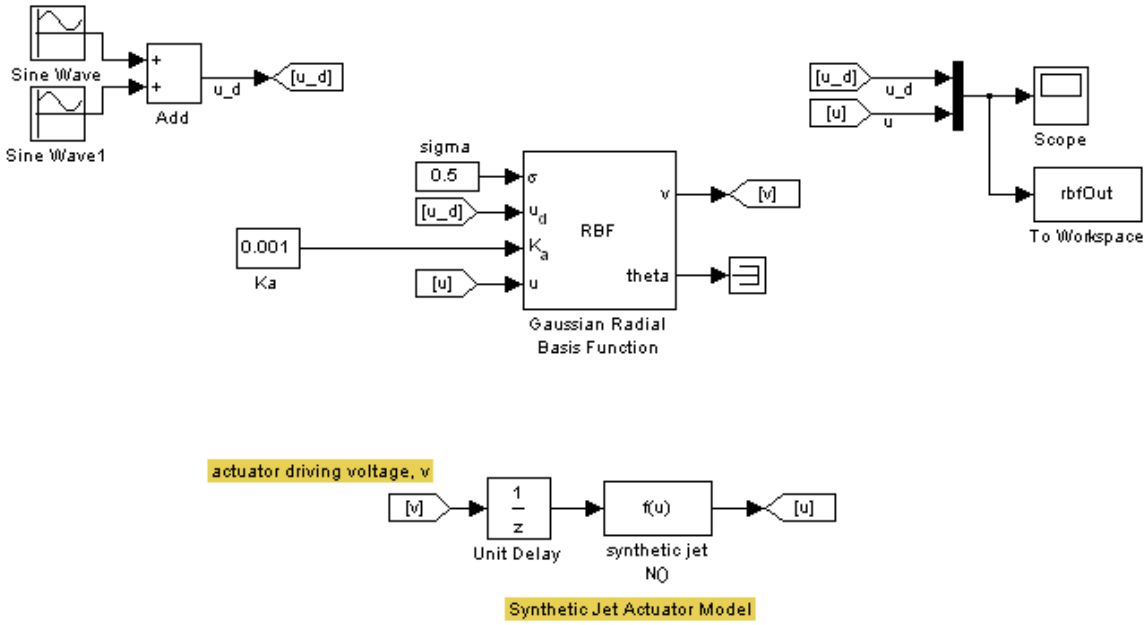
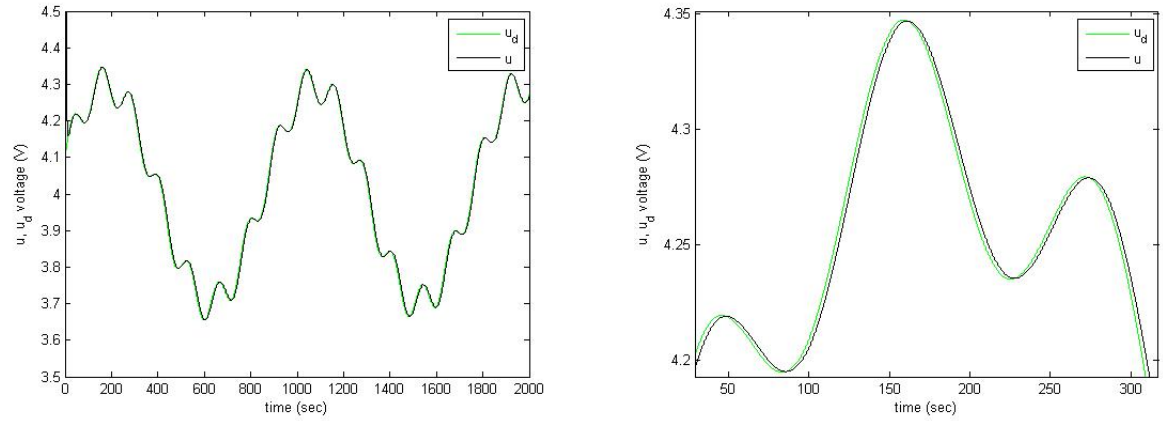


Figure 31: Gaussian RBF Test Harness

The relevant input (u_d) and outputs (u) are displayed in Simulink scopes for real-time analysis, and saved in the MATLAB workspace for offline analysis.

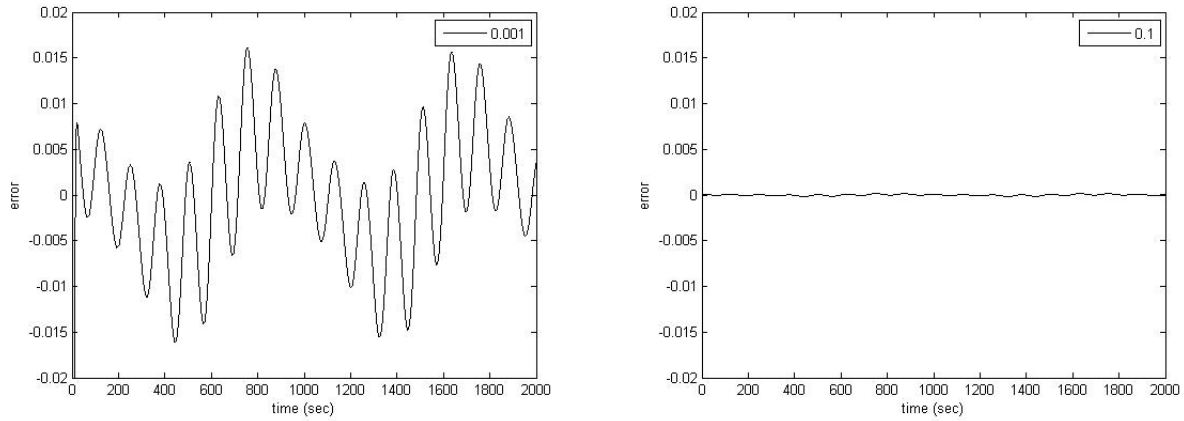
The performance is shown in Figure 32. Figure 32(a) shows the actuator's desired response (u_d) and actual response (u) for the adaptive gain $K_a = 0.001$. The error is small enough that Figure 32(b) is necessary to visualize it. The $GRBF$ adapts quickly, so a very small gain is necessary to see any appreciable error. Figure 33 shows that increasing K_a from 0.001 to 0.1 decreases the error appreciably.



(a) GRBF Test Results

(b) GRBF Test Results Detail

Figure 32: Gaussian RBF Test Results ($K_a = 0.001$)



(a) GRBF Error ($K_a = 0.001$)

(b) GRBF Error ($K_a = 0.1$)

Figure 33: Gaussian RBF Test Error

5.3.9 Taylor Series Inverse

Description

The Taylor Series (*TS*) Nonlinear Adaptive Inverse block compensates for a continuous-function actuator nonlinearity. The block approximates a nonlinearity using an n-order Taylor Series to approximate a nonlinearity which is a function of two variables, that is $N(u_d, alpha)$.

The TS approximator applies to a single-input, single-output (SISO) system with a measurable system response, y.

This block uses feedback to adaptively estimate the Taylor term coefficients, and compensates the actuator input command signal so that the actuator will achieve desired response without directly compensating for the nonlinearity in the control law.

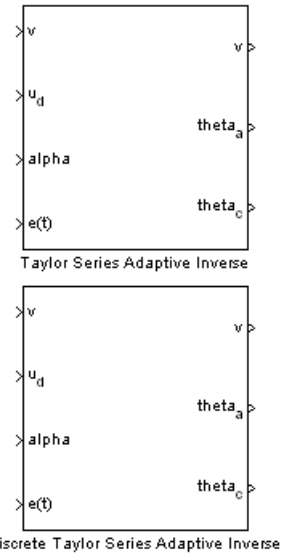


Figure 34: Taylor Series Simulink Block

Input Signals

v	The actuator driving signal.
u_d	A signal proportional to the desired actuator response.
$alpha$	The aircraft angle of attack.
$e(t)$	The system error signal defined as the difference between the system reference input and the system response.

Output Signals

v	The compensated actuator driving signal that will achieve the desired system response.
$theta_a$	The adapting nonlinear inverse parameters, in an internal form.
$theta_c$	The adapting nonlinear inverse parameters, in an internal form.

Parameters and Dialog Box

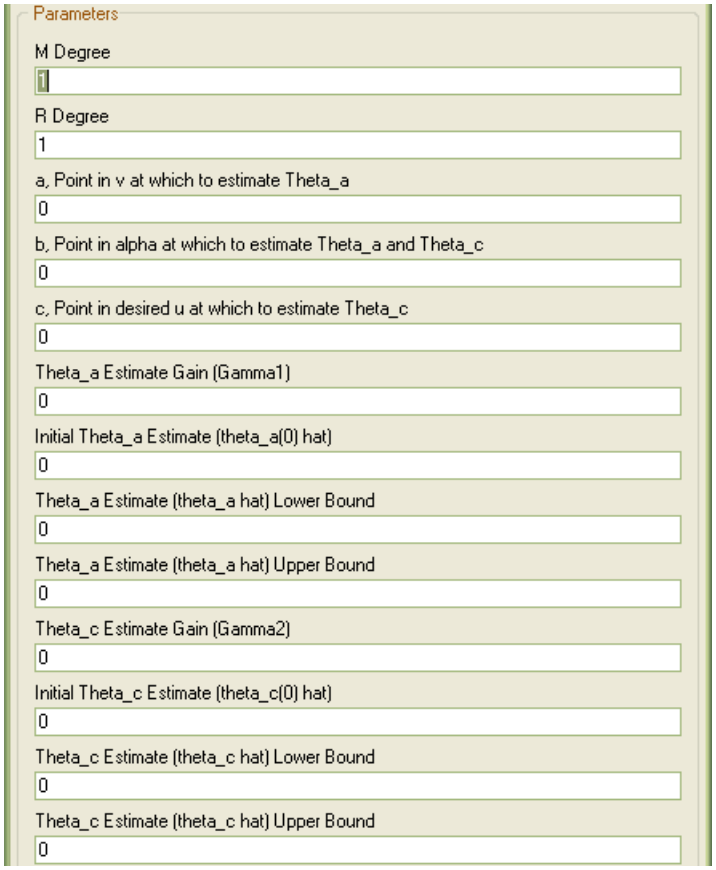


Figure 35: Taylor Series Simulink Block Parameters

M Degree	The Taylor Series degree. The number of *TS* terms is $N = (M+1)(R+1)$, where $0 \leq M \leq 9$.
R Degree	The Taylor Series degree. The number of *TS* terms is $N = (M+1)(R+1)$, where $0 \leq R \leq 9$.
a	The point in the range of v about which the *TS* is being estimated.
b	The point in the range of alpha about which the *TS* is being estimated.
c	The point in the range of u about which the *TS* is being estimated.
$Gamma1$	The adaptive estimate gain for $theta_a$. This value may be a scalar, or an $N \times N$ vector, where N is the number of *TS* terms.
$theta_a(0)$	The initial estimate for $theta_a$. This value must be an $N \times 1$ vector, where N is the number of *TS* terms.
$theta_a LowerBound$	The lower bound of the closed interval containing the $theta_a$ estimates. This value must be an $N \times 1$ vector, where N is the number of *TS* terms.
$theta_a UpperBound$	The upper bound of the closed interval containing the $theta_a$ estimates. This value must be an $N \times 1$ vector, where N is the number of *TS* terms.
$Gamma2$	The adaptive estimate gain for $theta_c$. This value may be a scalar, or an $N \times N$ vector, where N is the number of *TS* terms.

$theta_c(0)$	The initial estimate for $theta_c$. This value must be an $N \times 1$ vector, where N is the number of *TS* terms.
$theta_c LowerBound$	The lower bound of the closed interval containing the $theta_c$ estimates. This value must be an $N \times 1$ vector, where N is the number of *TS* terms.
$theta_c UpperBound$	The lower bound of the closed interval containing the $theta_c$ estimates. This value must be an $N \times 1$ vector, where N is the number of *TS* terms.

5.4 Related Processing

This section is not applicable to AIFAC.

5.5 Data Backup

This section is not applicable to AIFAC.

5.6 Recovery from Errors, Malfunctions, and Emergencies

This section is not applicable to AIFAC.

5.7 Messages

This section is not applicable to AIFAC.

5.8 Quick-Reference Guide

This section is not applicable to AIFAC.

6 Notes

6.1 Abbreviations and Acronyms

AIFAC	Adaptive Inverse For Actuator Compensation
ANN3NI	Adaptive Neural Network (3-Layer) Nonlinear Inverse
BI	Backlash Inverse
CSCI	Computer Software Configuration Item
DR	Disturbance Rejection
DZ	Dead-zone
GA	Genetic Algorithm
GRBF	Gaussian Radial Basis Function
N	Nonlinear
NI	Nonlinear Inverse
RBF	Radial Basis Function
RMS	Root Mean Square
SISO	Single-Input, Single-Output
SUM	Software Users Manual
TS	Taylor Series