recursiveARX

Create System object for online parameter estimation of ARX model

Use `recursiveARX` command for parameter estimation with real-time data. If all data necessary for estimation is available at once, and you are estimating a time-invariant model, use the offline estimation command, `arx`.

Syntax

```
obj = recursiveARX
obj = recursiveARX(Orders)
obj = recursiveARX(Orders,A0,B0)
obj = recursiveARX( ___ ,Name,Value)
```

Description

`obj = recursiveARX` creates a System object™ for online parameter estimation of a default ARX model structure. The default model structure has polynomials of order 1 and initial polynomial coefficient values `eps`.

After creating the object, use the `step` command to update model parameter estimates using recursive estimation algorithms and real-time data.

`obj = recursiveARX(Orders)` specifies the polynomial orders of the ARX model to be estimated.

`obj = recursiveARX(Orders,A0,B0)` specifies the polynomial orders and initial values of the polynomial coefficients.

`obj = recursiveARX( ___ ,Name,Value)` specifies additional attributes of the ARX model structure and recursive estimation algorithm using one or more `Name,Value` pair arguments.

Object Description

`recursiveARX` creates a System object for online parameter estimation of single-input single-output (SISO) or multiple-input single-output (MISO) ARX models using a recursive estimation algorithm.

A System object is a specialized MATLAB® object designed specifically for implementing and simulating dynamic systems with inputs that change over time. System objects use internal states to store past behavior, which is used in the next computational step.

After you create a System object, you use commands to process data or obtain information from or about the object. System objects use a minimum of two commands to process data — a constructor to create the object and the `step` command to update object parameters using real-time data. This separation of declaration from execution lets you create multiple, persistent, reusable objects, each with different settings.

You can use the following commands with the online estimation System objects in System Identification Toolbox™:

Command	Description

Command	Description
`step`	Update model parameter estimates using recursive estimation algorithms and real-time data. `step` puts the object into a locked state. In a locked state, you cannot change any nontunable properties or input specifications, such as model order, data type, or estimation algorithm. During execution, you can only change tunable properties.
`release`	Unlock the System object. Use this command to enable setting of nontunable parameters.
`reset`	Reset the internal states of a locked System object to the initial values, and leave the object locked.
`clone`	Create another System object with the same object property values. Do not create additional objects using syntax `obj2 = obj`. Any changes made to the properties of the new object created this way (`obj2`) also change the properties of the original object (`obj`).
`isLocked`	Query locked status for input attributes and nontunable properties of the System object.

Use the `recursiveARX` command to create an online estimation System object. Then estimate the ARX model parameters (`A` and `B`) and output using the `step` command with incoming input and output data, `u` and `y`.

```
[A,B,EstimatedOutput] = step(obj,y,u)
```

For `recursiveARX` object properties, see Properties.

Examples

collapse all

Estimate a SISO ARX Model Online

Create a System object for online parameter estimation of a SISO ARX model.

Try it in MATLAB

```
obj = recursiveARX;
```

The ARX model has a default structure with polynomials of order 1 and initial polynomial coefficient values, `eps`.

Load the estimation data. In this example, use a static data set for illustration.

```
load iddata1 z1;
output = z1.y;
```

```
input = z1.u;
```

Estimate ARX model parameters online using `step`.

```
for i = 1:numel(input)
[A,B,EstimatedOutput] = step(obj,output(i),input(i));
end
```

View the current estimated values of polynomial `B` coefficients.

```
obj.B
```

```
ans = 1×2

         0    0.7974
```

View the current covariance estimate of the parameters.

```
obj.ParameterCovariance
```

```
ans = 2×2

    0.0002    0.0001
    0.0001    0.0034
```

View the current estimated output.

```
EstimatedOutput
```

```
EstimatedOutput = -4.7766
```

Create System Object for SISO ARX Model With Known Initial Parameters

Specify ARX model orders and delays.

Try it in MATLAB

```
na = 1;
nb = 2;
nk = 1;
```

Create a System object for online estimation of SISO ARX model with known initial polynomial coefficients.

```
A0 = [1 0.5];
B0 = [0 1 1];
obj = recursiveARX([na nb nk],A0,B0);
```

Specify the initial parameter covariance.

```
obj.InitialParameterCovariance = 0.1;
```

`InitialParameterCovariance` represents the uncertainty in your guess for the initial parameters. Typically, the default `InitialParameterCovariance` (10000) is too large relative to the parameter values. This results in initial guesses being given less importance during

estimation. If you have confidence in the initial parameter guesses, specify a smaller initial parameter covariance.

Create System Object for MISO ARX Model With Known Initial Parameters

Specify orders and delays for ARX model with two inputs and one output.

Try it in MATLAB

```
na = 1;
nb = [2 1];
nk = [1 3];
```

`nb` and `nk` are specified as row vectors of length equal to number of inputs, *Nu*.

Specify initial polynomial coefficients.

```
A0 = [1 0.5];
B0 = [0 1 1 0; 0 0 0 0.8];
```

`B0` has *Nu* rows and `max(nb+nk)` columns. The *i*-th row corresponds to *i*-th input and is specified as having `nk(i)` zeros, followed by `nb(i)` initial values. Values after `nb(i)+nk(i)` are ignored.

Create a System object for online estimation of ARX model with known initial polynomial coefficients.

```
obj = recursiveARX([na nb nk],A0,B0);
```

Specify Estimation Method for Online Estimation of ARX Model

Create a System object that uses the normalized gradient algorithm for online parameter estimation of an ARX model.

Try it in MATLAB

```
obj = recursiveARX([1 2 1],'EstimationMethod','NormalizedGradient');
```

Input Arguments

collapse all

`Orders` — Model orders and delays

1-by-3 vector of integers | 1-by-3 vector of vectors

Model orders and delays of an ARX model, specified as a 1-by-3 vector of integers or vectors, `[na nb nk]`.

- `na` — Order of the polynomial $A(q)$, specified as a nonnegative integer.
- `nb` — Order of the polynomial $B(q)$ + 1, specified as 1–by-Nu vector of positive integers. Nu is the number of inputs.

For MISO models, there are as many $B(q)$ polynomials as the number of inputs. `nb(i)` is the order of ith polynomial $B_i(q)$+1 for the ith input.

- `nk` — Input-output delay, specified as a 1–by-Nu vector of nonnegative integers. Nu is the number of inputs.

  For MISO models, there are as many $B(q)$ polynomials as the number of inputs. `nk(i)` is the input-output delay time corresponding to the ith input.

`A0,B0` — Initial value of polynomial coefficients

row vector and matrix of real values | `[]`

Initial value of coefficients of $A(q)$ and $B(q)$ polynomials, specified as row vector and matrix or real values, respectively. Specify the elements in order of ascending powers of q^{-1}.

- `A0` — Initial value for the coefficients of the polynomial $A(q)$, specified as a 1-by-(`na+1`) row vector with 1 as the first element.
- `B0` — Initial value for the coefficients of the polynomial $B(q)$, specified as Nu-by-`max(nb+nk)` matrix. Nu is the number of inputs.

  For MISO models, there are as many $B(q)$ polynomials as the number of inputs. The ith row of `B0` corresponds to the ith input and must contain `nk(i)` leading zeros, followed by `nb(i)` initial parameter values. Entries beyond `nk(i)+nb(i)` are ignored.

`na`, `nb`, and `nk` are the `Orders` of the model.

Specifying as `[]`, uses the default value of `eps` for the polynomial coefficients.

If the initial parameter values are much smaller than `InitialParameterCovariance`, these initial values are given less importance during estimation. Specify a smaller initial parameter covariance if you have high confidence in the initial parameter values. This statement applies only for infinite-history estimation. Finite-history estimation does not use `InitialParameterCovariance`.

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside quotes. You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Use `Name,Value` arguments to specify writable properties of `recursiveARX` System object during object creation. For example, `obj = recursiveARX([2 2 1],'EstimationMethod','Gradient')` creates a System object to estimate an ARX model using the `'Gradient'` recursive estimation algorithm.

Properties

`recursiveARX` System object properties consist of read-only and writable properties. The writable properties are tunable and nontunable properties. The nontunable properties cannot be changed when the object is locked, that is, after you use the `step` command.

Use `Name,Value` arguments to specify writable properties of `recursiveARX` objects during object creation. After object creation, use dot notation to modify the tunable properties.

```
obj = recursiveARX;
obj.ForgettingFactor = 0.99;
```

`A`	Estimated coefficients of polynomial $A(q)$, returned as a row vector of real values specified in order of ascending powers of q^{-1}. `A` is a read-only property and is initially empty after you create the object. It is populated after you use the `step` command for online parameter estimation.
`B`	Estimated coefficients of polynomial $B(q)$, returned as a Nu-by-`max(nb+nk)` matrix of real values. Nu is the number of inputs. The ith row of `B` corresponds to the ith input and contains `nk(i)` leading zeros, followed by `nb(i)` estimated parameters, specified in order of ascending powers of q^{-1}. Ignore zero entries beyond `nk(i)+nb(i)`. `B` is a read-only property and is initially empty after you create the object. It is populated after you use the `step` command for online parameter estimation.
`InitialA`	Initial values for the coefficients of polynomial $A(q)$ of order `na`, specified as a row vector of length `na+1`, with 1 as the first element. Specify the coefficients in order of ascending powers of q^{-1}. If the initial guesses are much smaller than the default `InitialParameterCovariance`, 10000, the initial guesses are given less importance during estimation. In that case, specify a smaller initial parameter covariance. `InitialA` is a tunable property. You can change it when the object is in a locked state. **Default:** `[1 eps]`
`InitialB`	Initial values for the coefficients of polynomial $B(q)$, specified as an Nu-by-`max(nb+nk)` matrix. Nu is the number of inputs. For MISO models, there are as many $B(q)$ polynomials as the number of inputs. The ith row of `B0` corresponds to the ith input and must contain `nk(i)` zeros, followed by `nb(i)` initial parameter values. Entries beyond `nk(i)+nb(i)` are ignored. If the initial guesses are much smaller than the default `InitialParameterCovariance`, 10000, the initial guesses are given less importance during estimation. In that case, specify a smaller initial parameter covariance. `InitialB` is a tunable property. You can change it when the object is in a locked state. **Default:** `[0 eps]`

`InitialOutputs`	Initial values of the measured outputs buffer in finite-history estimation, specified as `0` or as a (W+na)-by-1 vector, where W is the window length and `na` is the order of the polynomial $A(q)$ that you specify when constructing the object. The `InitialOutputs` property provides a means of controlling the initial behavior of the algorithm. When `InitialOutputs` is set to `0`, the object populates the buffer with zeros. If the initial buffer is set to `0` or does not contain enough information, you see a warning message during the initial phase of your estimation. The warning should clear after a few cycles. The number of cycles it takes for sufficient information to be buffered depends upon the order of your polynomials and your input delays. If the warning persists, you should evaluate the content of your signals. Specify `InitialOutputs` only when `History` is `Finite`. `InitialOutputs` is a tunable property. You can change `InitialOutputs` when the object is in a locked state. **Default:** `0`
`InitialInputs`	Initial values of the inputs in the finite history window, specified as `0` or as a (W-1+max(nb)+max(nk))-by-nu matrix, where W is the window length and nu is the number of inputs. nb is the vector of $B(q)$ polynomial orders and nk is vector of input delays that you specify when constructing the `recursiveARX` object. The `InitialInputs` property provides a means of controlling the initial behavior of the algorithm. When the `InitialInputs` is set to `0`, the object populates the buffer with zeros. If the initial buffer is set to `0` or does not contain enough information, you see a warning message during the initial phase of your estimation. The warning should clear after a few cycles. The number of cycles it takes for sufficient information to be buffered depends upon the order of your polynomials and your input delays. If the warning persists, you should evaluate the content of your signals. Specify `InitialInputs` only when `History` is `Finite`. `InitialInputs` is a tunable property. You can change `InitialInputs` when the object is in a locked state. **Default:** `0`

`ParameterCovariance`	Estimated covariance `P` of the parameters, returned as an N-by-N symmetric positive-definite matrix. N is the number of parameters to be estimated. The software computes `P` assuming that the residuals (difference between estimated and measured outputs) are white noise, and the variance of these residuals is 1. `ParameterCovariance` is applicable only when `EstimationMethod` is `'ForgettingFactor'` or `'KalmanFilter'` or when `History` is `Finite`. The interpretation of `P` depends on your settings for the `History` and `EstimationMethod` properties. • If `History` is `Infinite`, then your `EstimationMethod` selection results in one of the following: - `'ForgettingFactor'` — $(R_2/2)$`P` is approximately equal to the covariance matrix of the estimated parameters, where R_2 is the true variance of the residuals. - `'KalmanFilter'` — R_2`P` is the covariance matrix of the estimated parameters, and R_1/R_2 is the covariance matrix of the parameter changes. Here, R_1 is the covariance matrix that you specify in `ProcessNoiseCovariance`. • If `History` is `Finite` (sliding-window estimation) — R_2`P` is the covariance of the estimated parameters. The sliding-window algorithm does not use this covariance in the parameter-estimation process. However, the algorithm does compute the covariance for output so that you can use it for statistical evaluation. `ParameterCovariance` is a read-only property and is initially empty after you create the object. It is populated after you use the `step` command for online parameter estimation.

`InitialParameterCovariance`	Covariance of the initial parameter estimates, specified as one of the following: • Real positive scalar, α — Covariance matrix is an N-by-N diagonal matrix, with α as the diagonal elements. N is the number of parameters to be estimated. • Vector of real positive scalars, $[\alpha_1,...,\alpha_N]$ — Covariance matrix is an N-by-N diagonal matrix, with $[\alpha_1,...,\alpha_N]$ as the diagonal elements. • N-by-N symmetric positive-definite matrix. `InitialParameterCovariance` represents the uncertainty in the initial parameter estimates. For large values of `InitialParameterCovariance`, less importance is placed on the initial parameter values and more on the measured data during beginning of estimation using `step`. Use only when `EstimationMethod` is `'ForgettingFactor'` or `'KalmanFilter'`. `InitialParameterCovariance` is a tunable property. You can change it when the object is in a locked state. **Default:** `10000`
`EstimationMethod`	Recursive estimation algorithm used for online estimation of model parameters, specified as one of the following values: • `'ForgettingFactor'` — Algorithm used for parameter estimation • `'KalmanFilter'` — Algorithm used for parameter estimation • `'NormalizedGradient'` — Algorithm used for parameter estimation • `'Gradient'` — Unnormalized gradient algorithm used for parameter estimation Forgetting factor and Kalman filter algorithms are more computationally intensive than gradient and unnormalized gradient methods. However, they have better convergence properties. For information about these algorithms, see Recursive Algorithms for Online Parameter Estimation. These methods all use an infinite data history, and are available only when `History` is `'Infinite'`. `EstimationMethod` is a nontunable property. You cannot change it during execution, that is, after the object is locked using the `step` command. **Default:** `Forgetting Factor`

`ForgettingFactor`	Forgetting factor, λ, relevant for parameter estimation, specified as a scalar in the range (0,1]. Suppose that the system remains approximately constant over T_0 samples. You can choose λ such that: $$T_0 = \frac{1}{1-\lambda}$$ • Setting $\lambda = 1$ corresponds to "no forgetting" and estimating constant coefficients. • Setting $\lambda < 1$ implies that past measurements are less significant for parameter estimation and can be "forgotten". Set $\lambda < 1$ to estimate time-varying coefficients. Typical choices of λ are in the range `[0.98 0.995]`. Use only when `EstimationMethod` is `'ForgettingFactor'`. `ForgettingFactor` is a tunable property. You can change it when the object is in a locked state. **Default:** `1`
`EnableAdapation`	Enable or disable parameter estimation, specified as one of the following: • `true` or `1`— The `step` command estimates the parameter values for that time step and updates the parameter values. • `false` or `0` — The `step` command does not update the parameters for that time step and instead outputs the last estimated value. You can use this option when your system enters a mode where the parameter values do not vary with time. **Note** If you set `EnableAdapation` to `false`, you must still execute the `step` command. Do not skip `step` to keep parameter values constant, because parameter estimation depends on current and past I/O measurements. `step` ensures past I/O data is stored, even when it does not update the parameters. `EnableAdapation` is a tunable property. You can change it when the object is in a locked state. **Default:** `true`

`DataType`	Floating point precision of parameters, specified as one of the following values: • `'double'` — Double-precision floating point • `'single'` — Single-precision floating point Setting `DataType` to `'single'` saves memory, but leads to loss of precision. Specify `DataType` based on the precision required by the target processor where you will deploy generated code. `DataType` is a nontunable property. It can only be set during object construction using `Name,Value` arguments and cannot be changed afterward. **Default:** `'double'`
`ProcessNoiseCovariance`	Covariance matrix of parameter variations, specified as one of the following: • Real nonnegative scalar, α — Covariance matrix is an N-by-N diagonal matrix, with α as the diagonal elements. • Vector of real nonnegative scalars, $[\alpha_1,...,\alpha_N]$ — Covariance matrix is an N-by-N diagonal matrix, with $[\alpha_1,...,\alpha_N]$ as the diagonal elements. • N-by-N symmetric positive semidefinite matrix. N is the number of parameters to be estimated. `ProcessNoiseCovariance` is applicable when `EstimationMethod` is `'KalmanFilter'`. Kalman filter algorithm treats the parameters as states of a dynamic system and estimates these parameters using a Kalman filter. `ProcessNoiseCovariance` is the covariance of the process noise acting on these parameters. Zero values in the noise covariance matrix correspond to estimating constant coefficients. Values larger than 0 correspond to time-varying parameters. Use large values for rapidly changing parameters. However, the larger values result in noisier parameter estimates. `ProcessNoiseCovariance` is a tunable property. You can change it when the object is in a locked state. **Default:** `0.1`
`AdaptationGain`	Adaptation gain, γ, used in gradient recursive estimation algorithms, specified as a positive scalar. `AdaptationGain` is applicable when `EstimationMethod` is `'Gradient'` or `'NormalizedGradient'`. Specify a large value for `AdaptationGain` when your measurements have a high signal-to-noise ratio. `AdaptationGain` is a tunable property. You can change it when the object is in a locked state. **Default:** `1`

`NormalizationBias`	Bias in adaptation gain scaling used in the `'NormalizedGradient'` method, specified as a nonnegative scalar. `NormalizationBias` is applicable when `EstimationMethod` is `'NormalizedGradient'`. The normalized gradient algorithm divides the adaptation gain at each step by the square of the two-norm of the gradient vector. If the gradient is close to zero, this can cause jumps in the estimated parameters. `NormalizationBias` is the term introduced in the denominator to prevent these jumps. Increase `NormalizationBias` if you observe jumps in estimated parameters. `NormalizationBias` is a tunable property. You can change it when the object is in a locked state. **Default:** `eps`
`History`	Data history type defining which type of recursive algorithm you use, specified as: • `'Infinite'` — Use an algorithm that aims to minimize the error between the observed and predicted outputs for all time steps from the beginning of the simulation. • `'Finite'` — Use an algorithm that aims to minimize the error between the observed and predicted outputs for a finite number of past time steps. Algorithms with infinite history aim to produce parameter estimates that explain all data since the start of the simulation. These algorithms still use a fixed amount of memory that does not grow over time. The object provides multiple algorithms of the `'Infinite'` `History` type. Specifying this option activates the `EstimationMethod` property with which you specify an algorithm. Algorithms with finite history aim to produce parameter estimates that explain only a finite number of past data samples. This method is also called *sliding-window* estimation. The object provides one algorithm of the `'Finite'` type. Specifying this option activates the `WindowLength` property that sizes the window. For more information on recursive estimation methods, see Recursive Algorithms for Online Parameter Estimation. `History` is a nontunable property. It can be set only during object construction using `Name,Value` arguments and cannot be changed afterward. **Default:** `'Infinite'`

`WindowLength`	Window size determining the number of time samples to use for the sliding-window estimation method, specified as a positive integer. Specify `WindowLength` only when `History` is `Finite`. Choose a window size that balances estimation performance with computational and memory burden. Sizing factors include the number and time variance of the parameters in your model. Always specify **Window Length** in samples, even if you are using frame-based input processing. `WindowLength` must be greater than or equal to the number of estimated parameters. Suitable window length is independent of whether you are using sample-based or frame-based input processing (see `InputProcessing`). However, when using frame-based processing, your window length must be greater than or equal to the number of samples (time steps) contained in the frame. `WindowLength` is a nontunable property. It can be set only during object construction using `Name,Value` arguments and cannot be changed afterward. **Default:** `200`
`InputProcessing`	Option for sample-based or frame-based input processing, specified as a character vector or string. • `Sample-based` processing operates on signals streamed one sample at a time. • `Frame-based` processing operates on signals containing samples from multiple time steps. Many machine sensor interfaces package multiple samples and transmit these samples together in frames. `Frame-based` processing allows you to input this data directly without having to first unpack it. Your `InputProcessing` specification impacts the dimensions for the input and output signals when using the `step` command: `[theta,EstimatedOutput] = step(obj,y,u)` • `Sample-based` - `y` and `EstimatedOutput` are scalars. - `u` is a 1-by-Nu vector, where Nu is the number of inputs. • - `Frame-based` with M samples per frame - `y` and `EstimatedOutput` are M-by-1 vectors. - `u` is an M-by-`Nu` matrix. `InputProcessing` is a nontunable property. It can be set only during object construction using `Name,Value` arguments and cannot be changed afterward. **Default:** `'Sample-based'`

Output Arguments

collapse all

`obj` — System object for online parameter estimation of ARX model

`recursiveARX` System object

System object for online parameter estimation of ARX model, returned as a `recursiveARX` System object. This object is created using the specified model orders and properties. Use `step` command to estimate the coefficients of the ARX model polynomials. You can then access the estimated coefficients and parameter covariance using dot notation. For example, type `obj.A` to view the estimated A polynomial coefficients.

More About

collapse all

ARX Model Structure

The ARX model structure is :

$$y(t) + a_1 y(t-1) + \ldots + a_{na} y(t-na) =$$
$$b_1 u(t-nk) + \ldots + b_{nb} u(t-nb-nk+1) + e(t)$$

The parameters `na` and `nb` are the orders of the ARX model, and `nk` is the delay.

- $y(t)$ — Output at time t.
- n_a — Number of poles.
- n_b — Number of zeroes plus 1.
- n_k — Number of input samples that occur before the input affects the output, also called the *dead time* in the system.
- $y(t-1) \ldots y(t-n_a)$ — Previous outputs on which the current output depends.
- $u(t-n_k) \ldots u(t-n_k-n_b+1)$ — Previous and delayed inputs on which the current output depends.
- $e(t)$ — White-noise disturbance value.

A more compact way to write the difference equation is

$$A(q)y(t) = B(q)u(t-n_k) + e(t)$$

q is the delay operator. Specifically,

$$A(q) = 1 + a_1 q^{-1} + \ldots + a_{n_a} q^{-n_a}$$

$$B(q) = b_1 + b_2 q^{-1} + \ldots + b_{n_b} q^{-n_b+1}$$

Tips

- Starting in R2016b, instead of using the `step` command to update model parameter estimates, you can call the System object with input arguments, as if it were a function. For example, `[A,B,EstimatedOutput] = step(obj,y,u)` and `[A,B,EstimatedOutput] = obj(y,u)` perform equivalent operations.

Extended Capabilities

› **C/C++ Code Generation**
Generate C and C++ code using MATLAB® Coder™.

See Also

Recursive Polynomial Model Estimator | `arx` | `clone` | `isLocked` | `recursiveAR` | `recursiveARMA` | `recursiveARMAX` | `recursiveBJ` | `recursiveLS` | `recursiveOE` | `release` | `reset` | `step`

Topics

Perform Online Parameter Estimation at the Command Line

Validate Online Parameter Estimation at the Command Line

Online ARX Parameter Estimation for Tracking Time-Varying System Dynamics

What Is Online Estimation?

Recursive Algorithms for Online Parameter Estimation

Introduced in R2015b