Package 'sysid'

January 7, 2016

Type Package

Title System Identification in R

Version 1.0

Date 2015-01-17

Author Suraj Yerramilli, Ashwin Raman, Narasimhan Balakrishnan, Arun Tangirala

Maintainer Suraj Yerramilli <surajyerramilli@outlook.com>

Description The sysid package provides functions for constructing mathematical models of dynamic systems from measured input-output data. The package contains functions for data visualization, data preprocessing, parametric and non-paramteric model estimation, and model predictions and validation.

License GPL-3

Imports signal,tframe,zoo, ggplot2, reshape2, polynom

RoxygenNote 5.0.1

NeedsCompilation no

R topics documented:

`armax` *Estimate ARMAX Models*

Description

Fit an ARMAX model of the specified order given the input-output data

Usage

```
armax(x, order = c(0, 1, 1, 0), options = optimOptions())
```

Arguments

`x`	an object of class `idframe`
`options`	Estimation Options, setup using `optimOptions`
`order:`	Specification of the orders: the four integer components (na,nb,nc,nk) are the order of polynolnomial A, order of polynomial B + 1, order of the polynomial C,and the input-output delay respectively

Details

SISO ARMAX models are of the form

$$y[k]+a_1y[k-1]+\ldots+a_{na}y[k-na] = b_{nk}u[k-nk]+\ldots+b_{nk+nb}u[k-nk-nb]+c_1e[k-1]+\ldots c_{nc}e[k-nc]+e[k]$$

The function estimates the coefficients using non-linear least squares (Levenberg-Marquardt Algorithm) \ The data is expected to have no offsets or trends. They can be removed using the `detrend` function.

Value

An object of class `estpoly` containing the following elements:

`sys`	an `idpoly` object containing the fitted ARMAX coefficients
`fitted.values`	the predicted response
`residuals`	the residuals
`input`	the input data used
`call`	the matched call

`stats`	A list containing the following fields: `vcov` - the covariance matrix of the fitted coefficients `sigma` - the standard deviation of the innovations
`options`	Option set used for estimation. If no custom options were configured, this is a set of default options
`termination`	Termination conditions for the iterative search used for prediction error minimization: `WhyStop` - Reason for termination `iter` - Number of Iterations `iter` - Number of Function Evaluations

References

Arun K. Tangirala (2015), *Principles of System Identification: Theory and Practice*, CRC Press, Boca Raton. Sections 14.4.1, 21.6.2

Examples

```
data(armaxsim)
z <- dataSlice(data,end=1533) # training set
mod_armax <- armax(z,c(1,2,1,2))
mod_armax
```

`arx` *Estimate ARX Models*

Description

Fit an ARX model of the specified order given the input-output data

Usage

```
arx(x, order = c(0, 1, 0))
```

Arguments

`x`	an object of class `idframe`
`order:`	Specification of the orders: the three integer components (na,nb,nk) are the order of polynolnomial A, (order of polynomial B + 1) and the input-output delay

Details

SISO ARX models are of the form

$$y[k] + a_1 y[k-1] + \ldots + a_{na} y[k-na] = b_{nk} u[k-nk] + \ldots + b_{nk+nb} u[k-nk-nb] + e[k]$$

The function estimates the coefficients using linear least squares (with no regularization). Future versions may include regularization parameters as well \ The data is expected to have no offsets or trends. They can be removed using the `detrend` function.

Value

An object of class `estpoly` containing the following elements:

`sys`	an `idpoly` object containing the fitted ARX coefficients
`fitted.values`	the predicted response
`residuals`	the residuals
`input`	the input data used
`call`	the matched call
`stats`	A list containing the following fields: `vcov` - the covariance matrix of the fitted coefficients `sigma` - the standard deviation of the innovations `df` - the residual degrees of freedom

References

Arun K. Tangirala (2015), *Principles of System Identification: Theory and Practice*, CRC Press, Boca Raton. Section 21.6.1

Lennart Ljung (1999), *System Identification: Theory for the User*, 2nd Edition, Prentice Hall, New York. Section 10.1

Examples

```
data(arxsim)
model <- arx(data,c(2,1,1))
model
plot(model) # plot the predicted and actual responses
```

`compare` *Compare the measured output and the predicted output(s)*

Description

Plots the output predictions of model(s) superimposed over validation data, data, for comparison.

Usage

```
compare(data, nahead = 1, ...)
```

Arguments

`data`	validation data in the form of an `idframe` object
`nahead`	number of steps ahead at which to predict (Default:1). For infinite- step ahead predictions, supply `Inf`.
`...`	models whose predictions are to be compared

See Also

`predict.estpoly` for obtaining model predictions

Examples

```
data(arxsim)
compare(data,nahead=Inf,mod1,mod2,mod3)
```

---

dataSlice *Subset or Resample idframe data*

Description

`dataSlice` is a subsetting method for objects of class `idframe`. It extracts the subset of the object `data` observed between indices `start` and `end`. If a frequency is specified, the series is then resampled at the new frequency.

Usage

```
dataSlice(data, start = NULL, end = NULL, freq = NULL)
```

Arguments

`data`	an object of class `idframe`
`start`	the start index
`end`	the end index
`freq`	fraction of the original frequency at which the series to be sampled.

Details

The dataSlice function extends the `window` function for idframe objects

Value

an idframe object

See Also

`window`

Examples

```
data(cstr)
cstrsub <- dataSlice(cstr,start=200,end=400) # extract between indices 200 and 400
cstrTrain <- dataSlice(cstr,end=4500) # extract upto index 4500
cstrTest <- dataSlice(cstr,start=6501) # extract from index 6501 till the end
cstr_new <- dataSlice(cstr,freq=0.5) # resample data at half the original frequency
```

detrend — *Remove offsets and linear trends*

Description

Removes the offsets or linear trends in each of the input and output matrices.

Usage

```
detrend(x, type = c("constant", "linear")[1])
```

Arguments

`x`	an object of class `idframe`
`type`	trend type - "constant" or "linear". (Default: `"constant"`)

Value

A list containing the following elements

`fitted.values`	`idframe` object with detrended variables
`output_trend`	`list` containing trend fits for each output variable
`input_trend`	`list` containing trend fits for each input variable

See Also

`predict.detrend`, `lm`

Examples

```
data(cstr)
fit <- detrend(cstr,type="linear") # remove linear trends
Zdetrend <- predict(fit) # get the detrended data

demean <- detrend(cstr) # remove offsets
Zcent <- predict(demean) # get the centered data
```

etfe — *Estimate empirical transfer function*

Description

Estimates the emperical transfer function from the data by taking the ratio of the fourier transforms of the output and the input variables

Usage

```
etfe(data)
```

Arguments

`data`	an object of class `idframe`

Value

an `idfrd` object containing the estimated frequency response

References

Arun K. Tangirala (2015), *Principles of System Identification: Theory and Practice*, CRC Press, Boca Raton. Sections 5.3 and 20.4.2

See Also

`fft`

Examples

```
data(frf)
frf <- etfe(data)
```

`fitch` *Fit Characteristics*

Description

Returns quantitative assessment of the estimated model as a list

Usage

```
fitch(x)
```

Arguments

`x`	the estimated model

Value

A list containing the following elements

`MSE`	Mean Square Error measure of how well the response of the model fits the estimation data
`FPE`	Final Prediction Error
`FitPer`	Normalized root mean squared error (NRMSE) measure of how well the response of the model fits the estimation data, expressed as a percentage.
`AIC`	Raw Akaike Information Citeria (AIC) measure of model quality
`AICc`	Small sample-size corrected AIC
`nAIC`	Normalized AIC
`BIC`	Bayesian Information Criteria (BIC)

getcov — *Parameter covariance of the identified model*

Description

Obtain the parameter covariance matrix of the linear, identified parametric model

Usage

```
getcov(sys)
```

Arguments

sys	a linear, identified parametric model

idframe — *S3 class for storing input-output data.*

Description

`idframe` is an S3 class for storing and manipulating input-ouput data. It supports discrete time and frequency domain data.

Usage

```
idframe(output = NULL, input = NULL, Ts = 1, start = 0, end = NULL,
  unit = c("seconds", "minutes", "hours", "days")[1])
```

Arguments

output	dataframe/matrix/vector containing the outputs
input	dataframe/matrix/vector containing the inputs
Ts	sampling interval (Default: 1)
start	Time of the first observation
end	Time of the last observation Optional Argument
unit	Time Unit (Default: "seconds")

Value

an idframe object

See Also

`plot.idframe`, the plot method for idframe objects, `summary.idframe`, the summary method for idrame objects

Examples

```
dataMatrix <- matrix(rnorm(1000),ncol=5)
data <- idframe(output=dataMatrix[,3:5],input=dataMatrix[,1:2],Ts=1)
```

idfrd *S3 class for storing frequency response data*

Description

S3 class for storing frequency response data

Usage

```
idfrd(response, freq, Ts)
```

Arguments

`response`	complex vector/matrix containing the response
`freq`	the frequencies at which the response is observed/estimated
`Ts`	sampling time of data

Value

an idfrd object

Note

The class can currently store only SISO Responses. Future versions will have support for multivariate data

See Also

`plot.idfrd` for generating bode plots; `spa` and `etfe` for estimating the frequency response given input/output data

idinput *function to generate input singals (rgs/rbs/prbs/sine)*

Description

`idinput` is a function for generating input signals (rgs/rbs/prbs/sine) for identification purposes

Usage

```
idinput(n, type = "rgs", band = c(0, 1), levels = c(-1, 1))
```

Arguments

`n`	integer length of the input singal to be generated
`type`	the type of input signal to be generated. 'rgs' - generates random gaussian signal 'rbs' - generates random binary signal 'prbs' - generates pseudorandom binary signal 'sine' - generates a signal that is a sum of sinusoids Default value is type='rgs'
`band`	determines the frequency content of the signal. For type='rbs'/'sine'/, band = [wlow,whigh] which specifies the lower and the upper bound of the passband frequencies(expressed as fractions of Nyquist frequency). Default is c(0,1) For type='prbs', band=[0,B] where B is such that the singal is constant over 1/B (clock period). Default is c(0,1)
`levels`	row vector defining the input level. It is of the form levels=c(minu, maxu) For 'rbs','prbs', 'sine', the generated signal always between minu and maxu. For 'rgs', minu=mean value of signal minus one standard deviation and maxu=mean value of signal plus one standard deviation Default value is levels=c(-1,1)

`idpoly` *Polynomial model with identifiable parameters*

Description

Creates a polynomial model with identifiable coefficients

Usage

```
idpoly(A = 1, B = 1, C = 1, D = 1, F1 = 1, ioDelay = 0, Ts = 1)
```

Arguments

`A`	Autoregressive coefficients
`B, F1`	Coefficients of the numerator and denominator respectively of the deterministic model between the input and output
`C, D`	Coefficients of the numerator and denominator respectively of the stochastic model
`ioDelay`	the delay in the input-output channel
`Ts`	sampling interval

Details

Discrete-time polynomials are of the form

$$A(q^{-1})y[k] = \frac{B(q^{-1})}{F1(q^{-1})}u[k] + \frac{C(q^{-1})}{D(q^{-1})}e[k]$$

Examples

```
# define output-error model
mod_oe <- idpoly(B=c(0.6,-0.2),F1=c(1,-0.5),ioDelay = 2,Ts=0.1)

# define box-jenkins model
B <- c(0.6,-0.2)
C <- c(1,-0.3)
D <- c(1,1.5,0.7)
F1 <- c(1,-0.5)
mod_bj <- idpoly(1,B,C,D,F1,ioDelay=1)
```

---

impulseest *Estimate Impulse Response Coefficients*

Description

`impulseest` is used to estimate impulse response coefficients from the data

Usage

```
impulseest(x, M = 30, K = NULL, regul = F, lambda = 1)
```

Arguments

`x`	an object of class `idframe`
`M`	Order of the FIR Model (Default:`30`)
`K`	Transport delay in the estimated impulse response (Default:NULL)
`regul`	Parameter indicating whether regularization should be used. (Default:`FALSE`)
`lambda`	The value of the regularization parameter. Valid only if `regul=TRUE`. (Default:`1`)

Details

The IR Coefficients are estimated using linear least squares. Future Versions will provide support for multivariate data.

References

Arun K. Tangirala (2015), *Principles of System Identification: Theory and Practice*, CRC Press, Boca Raton. Sections 17.4.11 and 20.2

See Also

`step`

Examples

```
uk <- rnorm(1000,1)
yk <- filter (uk,c(0.9,-0.4),method="recursive") + rnorm(1000,1)
data <- idframe(output=data.frame(yk),input=data.frame(uk))
fit <- impulseest(data)
plot(fit)
```

misdata *Replace Missing Data by Interpolation*

Description

Function for replacing missing values with interpolated ones. This is an extension of the `na.approx` function from the `zoo` package. The missing data is indicated using the value *NA*.

Usage

```
misdata(data)
```

Arguments

`data`	an object of class `idframe`

Value

data (an idframe object) with missing data replaced.

See Also

`na.approx`

Examples

```
data(cstr_mis)
summary(cstr_mis) # finding out the number of NAs
cstr <- misdata(cstr_mis)
```

oe *Estimate Output-Error Models*

Description

Fit an output-error model of the specified order given the input-output data

Usage

```
oe(x, order = c(1, 1, 0), options = optimOptions())
```

Arguments

`x`	an object of class `idframe`
`order`	Specification of the orders: the four integer components (nb,nf,nk) are order of polynomial B + 1, order of the polynomial F, and the input-output delay respectively
`options`	Estimation Options, setup using `optimOptions`

Details

SISO OE models are of the form

$$y[k]+f_1y[k-1]+\ldots+f_{nf}y[k-nf] = b_{nk}u[k-nk]+\ldots+b_{nk+nb}u[k-nk-nb]+f_1e[k-1]+\ldots f_{nf}e[k-nf]+e[k]$$

The function estimates the coefficients using non-linear least squares (Levenberg-Marquardt Algorithm) \ The data is expected to have no offsets or trends. They can be removed using the `detrend` function.

Value

An object of class `estpoly` containing the following elements:

`sys`	an `idpoly` object containing the fitted OE coefficients
`fitted.values`	the predicted response
`residuals`	the residuals
`input`	the input data used
`call`	the matched call
`stats`	A list containing the following fields: `vcov` - the covariance matrix of the fitted coefficients `sigma` - the standard deviation of the innovations
`options`	Option set used for estimation. If no custom options were configured, this is a set of default options
`termination`	Termination conditions for the iterative search used for prediction error minimization: `WhyStop` - Reason for termination `iter` - Number of Iterations `iter` - Number of Function Evaluations

References

Arun K. Tangirala (2015), *Principles of System Identification: Theory and Practice*, CRC Press, Boca Raton. Sections 14.4.1, 17.5.2, 21.6.3

Examples

```
data(oesim)
z <- dataSlice(data,end=1533) # training set
mod_oe <- oe(z,c(2,1,2))
mod_oe
plot(mod_oe) # plot the predicted and actual responses
```

`optimOptions` *Create optimization options*

Description

Specify optimization options that are to be passed to the numerical estimation routines

Usage

```
optimOptions(tol = 1e-05, maxIter = 20, LMinit = 100, LMstep = 8)
```

Arguments

`tol`	Minimum ratio of the improvement to the current loss function. Iterations stop if this ratio goes below the tolerance limit (Default: `1e-5`)
`maxIter`	Maximum number of iterations to be performed
`LMinit`	Starting value of search-direction length in the Levenberg-Marquardt method.
`LMstep`	Size of the Levenberg-Marquardt step

`plot.idframe` *Plotting idframe objects*

Description

Plotting method for objects inherting from class `idframe`

Usage

```
## S3 method for class 'idframe'
plot(x, col = "steelblue", lwd = 1, main = NULL)
```

Arguments

`x`	an `idframe` object
`col`	line color, to be passed to plot.(Default=`"steelblue"`)
`lwd`	line width, in millimeters(Default=1)
`main`	the plot title. (Default = `NULL`)

Examples

```
data(cstr)
plot(cstr,col="blue")
```

plot.idfrd *Plotting idfrd objects*

Description

Generates the bode plot of the given frequency response data. It uses the ggplot2 plotting engine

Usage

```
## S3 method for class 'idfrd'
plot(x)
```

Arguments

x	An object of class `idframe`

See Also

ggplot

Examples

```
data(frf)
frf <- spa(data) # Estimates the frequency response from data
plot(frf)
```

plot.impulseest *Impulse Response Plots*

Description

Plots the estimated IR coefficients along with the significance limits at each lag.

Usage

```
## S3 method for class 'impulseest'
plot(model, sig = 0.975)
```

Arguments

model	an object of class `impulseest`
sig	Significance Limits (Default: `0.975`)

See Also

impulseest,step

predict.detrend *Detrend data based on linear trend fits*

Description

Returns detrended `idframe` object based on linear trend fit

Usage

```
## S3 method for class 'detrend'
predict(model, newdata = NULL, ...)
```

Arguments

`model`	an object of class `detrend`
`newdata`	An optional idframe object in which to look for variables with which to predict. If ommited, the original detrended idframe object is used

Value

an `idframe` object

Examples

```
data(cstr)
train <- dataSlice(cstr,end=5000)
test <- dataSlice(cstr,start=6001)
fit <- detrend(train)
Ztrain <- predict(fit)
Ztest <- predict(fit,test)
```

predict.estpoly *Predictions of identified model*

Description

Predicts the output of an identified model (`estpoly`) object K steps ahead.

Usage

```
## S3 method for class 'estpoly'
predict(x, newdata = NULL, nahead = 1)
```

Arguments

`x`	`estpoly` object containing the identified model
`newdata`	optional dataset to be used for predictions. If not supplied, predictions are made on the training set.
`nahead`	number of steps ahead at which to predict (Default:1). For infinite- step ahead predictions or pure simulation, supply `Inf`.

Value

Time-series containing the predictions

References

Arun K. Tangirala (2015), *Principles of System Identification: Theory and Practice*, CRC Press, Boca Raton. Chapter 18

Examples

```
data(arxsim)
Yhat <- predict(mod1,data) #  1-step ahead predictions
Yhat_2 <- predict(mod1,data,nahead=2) # 2-step ahead predictions
Yhat_inf <- predict(mod1,data,nahead=Inf) # Infinite-step ahead predictions
```

`read.idframe` *Data input into a idframe object*

Description

Read the contents of a data.frame/matrix into a `idframe` object.

Usage

```
read.idframe(data, ninputs = NULL, Ts = 1, unit = c("seconds", "minutes",
  "hours", "days")[1])
```

Arguments

`data`	a `data.frame` object
`ninputs`	the number of input columns. (Default: 0)
`Ts`	sampling interval (Default: 1)
`unit`	Time Unit (Default: "seconds")

Value

an idframe object

Examples

```
data(cstr)
data <- read.idframe(cstrData,ninputs=1,Ts= 1,unit="minutes")
```

read.odf.idframe — *Reading from .odf documents*

Description

Read the contents of an a .odf document into a `idframe` object.

Usage

```
read.odf.idframe(file, sheetName, header = TRUE, ninputs = 0, Ts = 1,
  unit = c("seconds", "minutes", "hours", "days")[1], ...)
```

Arguments

`file`	the path to the file to read
`sheetName`	a character string with the sheet name
`header`	a logical value indicating whether the first row corresponding to the first element of the rowIndex vector contains the names of the variables.
`ninputs`	the number of input columns. (Default: 0)
`Ts`	sampling interval (Default: 1)
`unit`	Time Unit (Default: "seconds")
`...`	additional arguments to be passed to the `read.xlsx2` function

Details

The `read.odf.idframe` function uses the `read.gnumeric.sheet` function, provided by the **xlsx** package, to read data from a .odf file and then calls the `read.idframe` function to read the data into a idframe object

Value

an idframe object

See Also

`read.xlsx2`

read.table.idframe — *Read the contents of a table-formatted file*

Description

Read the contents of an file in table format into a `idframe` object.

Usage

```
read.table.idframe(file, header = TRUE, sep = ",", ninputs = 0, Ts = 1,
  unit = c("seconds", "minutes", "hours", "days")[1], ...)
```

Arguments

`file`	the path to the file to read
`header`	a logical value indicating whether the first row corresponding to the first element of the rowIndex vector contains the names of the variables. (Default: `TRUE`)
`sep`	the field separator character. Values on each line of the file are separated by this character. (Default: `","`)
`ninputs`	the number of input columns. (Default: 0)
`Ts`	sampling interval (Default: 1)
`unit`	Time Unit (Default: "seconds")
`...`	additional arguments to be passed to the `read.table` function

Details

The `read.table.idframe` function uses the `read.table` function, provided by the **utils** package, to read data from a table-formatted file and then calls the `read.idframe` function to read the data into a idframe object

Value

an idframe object

See Also

`read.table`

Examples

```
dataMatrix <- data.frame(matrix(rnorm(1000),ncol=5))
colnames(dataMatrix) <- c("u1","u2","y1","y2","y3")
write.csv(dataMatrix,file="test.csv",row.names=FALSE)

data <- read.table.idframe("test.csv",ninputs=2,unit="minutes")
```

`read.xls.idframe` *Read the contents of a worksheet into a idframe object*

Description

Read the contents of an excel worksheet into a `idframe` object.

Usage

```
read.xls.idframe(file, sheetName, header = TRUE, ninputs = 0, Ts = 1,
  unit = c("seconds", "minutes", "hours", "days")[1], ...)
```

Arguments

`file`	the path to the file to read
`sheetName`	a character string with the sheet name
`header`	a logical value indicating whether the first row corresponding to the first element of the rowIndex vector contains the names of the variables.
`ninputs`	the number of input columns. (Default: 0)
`Ts`	sampling interval (Default: 1)
`unit`	Time Unit (Default: "seconds")
`...`	additional arguments to be passed to the `read.xlsx2` function

Details

The `read.xlsx.idframe` function uses the `read.xlsx2` function, provided by the **xlsx** package, to read data from an excel file and then calls the `read.idframe` function to read the data into a idframe object

The function requires the java runtime to be installed on the system (Requirement of the **xlsx** package).

Value

an idframe object

See Also

`read.xlsx2`

Examples

```
library(xlsx)
dataMatrix <- data.frame(matrix(rnorm(1000),ncol=5))
colnames(dataMatrix) <- c("u1","u2","y1","y2","y3")
write.xlsx2(dataMatrix,file="test.xlsx",row.names=FALSE)

data <- read.xls.idframe("test.xlsx","Sheet1",ninputs=2,unit="minutes")
```

---

sim *Simulate dynamic system*

Description

Simulate the response of a system given the input

Usage

```
sim(model, input, sigma = 0, seed = NULL)
```

Arguments

`model`	the system model to simulate
`input`	a vector/matrix containing the input
`sigma`	standard deviation of the innovations (Default= `0`)
`seed`	integer indicating the seed value of the random number generator

Details

The routine is currently built only for SISO systems. Future Versions will include support for MIMO systems. Current support

Value

a vector containing the output

See Also

`sim.idpoly` for simulating polynomial models

`sim.idpoly` *Simulate from a Polynomial Model*

Description

Simulate the response of a system governed by a polynomial model , given the input

Usage

```
## S3 method for class 'idpoly'
sim(model, input, sigma = 0, seed = NULL)
```

Arguments

`model`	an object of class `idpoly` containing the coefficients
`input`	a vector/matrix containing the input
`sigma`	standard deviation of the innovations (Default= `0`)
`seed`	integer indicating the seed value of the random number generator

Details

The routine is currently built only for SISO systems. Future Versions will include support for MIMO systems

Value

a vector containing the output

See Also

`idpoly` for defining polynomial models

Examples

```
# ARX Model
u <- rnorm(200,sd=1)
model <- idpoly(A=c(1,-1.5,0.7),B=c(0.8,-0.25),ioDelay=1)
y <- sim(model,u,sigma=0.1)
```

---

spa *Estimate frequency response*

Description

Estimates Frequency Response with fixed frequency resolution using spectral analysis

Usage

```
spa(data, npad = 255)
```

Arguments

data	an `idframe` object
npad	an integer representing the total length of each time series to analyze after padding with zeros. This argument allows the user to control the spectral resolution of the SDF estimates: the normalized frequency interval is deltaf=1/npad. (Default: 255)

Details

The function calls the `SDF` function in the `sapa` package to compute the cross-spectral densities. The method used is **Welch's Overlapped Segment Averaging** with a normalized **Hanning** window. The overlap used is 50

Value

an `idfrd` object containing the estimated frequency response

References

Arun K. Tangirala (2015), *Principles of System Identification: Theory and Practice*, CRC Press, Boca Raton. Sections 16.5 and 20.4

See Also

`SDF`

Examples

```
data(frf)
frf <- spa(data)
```

step — *Step Response Plots*

Description

Plots the step response of a system, given the IR model

Usage

```
step(model)
```

Arguments

model	an object of class impulseest

See Also

impulseest

Examples

```
uk <- rnorm(1000,1)
yk <- filter (uk,c(0.9,-0.4),method="recursive") + rnorm(1000,1)
data <- idframe(output=data.frame(yk),input=data.frame(uk))
fit <- impulseest(data)
step(fit)
```

Index