Package: CollocInfer (via r-universe)

November 6, 2024

Version 1.0.5

Date 2024-11-04

Title Collocation Inference for Dynamic Systems

Maintainer Giles Hooker <ghooker@wharton.upenn.edu>

Depends R (>= 4.3.0), fda

Imports MASS, Matrix, spam, deSolve, methods

Suggests pomp, SparseM, subplex, trust, maxLik

Description These functions implement collocation-inference for continuous-time and discrete-time stochastic processes. They provide model-based smoothing, gradient-matching, generalized profiling and forwards prediction error methods.

License GPL (>= 2)

URL http://www.gileshooker.com

LazyData true

NeedsCompilation no

Author Giles Hooker [aut, cre], Luo Xiao [aut], James Ramsay [ctb]

Date/Publication 2024-11-05 08:50:01 UTC

Config/pak/sysreqs make

Repository https://gileshooker.r-universe.dev

RemoteUrl https://github.com/cran/CollocInfer

RemoteRef HEAD

RemoteSha dc7eb4be5d1621ed3a69308e8c084f5bfee4ab87

Contents

`CollocInfer-package` *Collocation Inference in R*

Description

Functions carry out collocation inference method for nonlinear continuous-time dynamic systems. These are based on basis-expansion representations for the state of the system. Gradient-matching, profiling and EM algorithms are supported.

Details

Package:	CollocInfer
Type:	Package
Version:	2.1.0
Date:	2009-08-19
License:	GPL-2
LazyLoad:	yes

Author(s)

Giles Hooker, Luo Xiao

Maintainer: Giles Hooker <giles.hooker@cornell.edu>

References

Ramsay, James O., Giles Hooker, Jiguo Cao and David Campbell (2007), "Parameter Estimation in Ordinary Differential Equations: A Generalized Smoothing Approach", *Journal of the Royal Statistical Society*, 69

Ramsay, James O., and Silverman, Bernard W. (2006), *Functional Data Analysis, 2nd ed.*, Springer, New York.

`ChemoData` *Chemostat Example Data*

Description

Five-species Chemostat Model

Usage

```
ChemoData
```

Format

ChemoData A 61 by 2 matrix of data observed in a chemostat.

ChemoTime A vector of 61 observation times corresponding to ChemoData.

ChemoPars Named parameter vector as a starting point for estimation `ChemoData`.

ChemoVarnames `c('N','C1','C2','B','S')`: the state variable names for the chemostat system.

ChemoParnames parameter names for the chemostat system.

Source

Yoshida, T., L. E. Jones, S. P. Ellner, G. F. Fussmann and N. G. Hairston, 2003, "Rapid evolution drives ecological dynamics in a predator-prey system", Nature, 424, pp. 303-306.

ChemoRMData *Rosenzweig-MacArthur Model Applied to Chemostat Data*

Description

Two-Species Rosenzweig-MacArthur Model

Usage

```
ChemoRMData
```

Format

ChemoRMData A 108 by 2 matrix of data observed in a chemostat.

ChemoRMPars Named parameter vector as a starting point for estimation in `ChemoRMData`.

ChemoRMTime A vector of 108 observation times corresponding to ChemoData.

RMparnames parameter names for the Rosenzweig-MacArthur system.

RMvarnames the state variable names for the Rosenzweig-MacArthur system.

Source

Becks, L., S. P. Ellner, L. E. Jones, and N. G. Hairston, 2010, "Reduction of adaptive genetic diversity radically alters eco-evolutionary community dynamics", Ecology Letters, 13, pp. 989-997.

CollocInferPlots *Diagnostic PLots for CollocInfer*

Description

Diagnostic Plots on the Results of CollocInfer

Usage

```
CollocInferPlots(coefs,pars,lik,proc,times=NULL,data=NULL,
      cols=NULL,datacols=NULL,datanames=NULL,ObsPlot=TRUE,DerivPlot=TRUE,
      cex.axis=1.5,cex.lab=1.5,cex=1.5,lwd=2)
```

Arguments

coefs	Vector giving the current estimate of the coefficients.
pars	Vector of estimated parameters.
lik	lik object defining the observation process.
proc	proc object defining the state process.
times	Vector observation times for the data.
data	Matrix of observed data values.
cols	Optional vector specifying a color for each state variable.
datacols	Optional vector specifying a color for each observation dimension.
datanames	Optional character vector specifying a glyph to plot the data. Taken from the column-names of data if not given.
ObsPlot	Should a plot of predictions and observations be given?
DerivPlot	Should derivative diagnostics be produced?
cex.axis	Axis font size.
cex.lab	Label font size.
cex	Plotting point font size
lwd	Plotting line width

Details

Timevec is taken to be the quadrature values. Three plots can be produced:

If ObsPlot=TRUE a plot is given of the predicted values of the observations along with the observations themselves (if given).

If DerivPlot=TRUE two plots are produced. The first gives the value of the derivative of the estimated trajectory (dashed) and the value of the right-hand-side of the ordinary differential equation in proc (hence the predicted derivative) (solid). The second plot gives their difference in the first panel as well as the estimated trajectory in the second panel.

Value

A list containing elements used in plotting:

timevec	Times at which the trajectories etc were evaluated.
traj	Estimated value of the trajectory.
dtraj	Derivative of the estimated trajectory.
ftraj	Value of the derivative of the trajectory predicted by proc
otraj	Predicted values of the observations from lik.

FhNdata *FitzHugh-Nagumo data*

Description

Data generated for FitzHugh-Nagumo Examples

Usage

```
FhNdata
```

Format

FhNdata A 41 by 2 matrix of data generated from the FitzHugh Nagumo equations.

FhNtimes A vector of 41 observation times corresponding to FhNdata.

FhNpars Named parameter vector used to generate `FhNdata`.

FhNvarnames `c('V','R')`: the state variable names for the FitzHugh Nagumo system.

FhNparnames `c('a','b','c')` parameter names for the FitzHugh Nagumo system.

Source

James Ramsay, Giles Hooker David Campbell and Jiguo Cao, 2007. "Parameter Estimation for Differential Equations: A Generalized Smoothing Approach". Journal of the Royal Statistical Society Vol 69 No 5.

FhNest *Estimated Parameters for FitzHugh-Nagumo data*

Description

Parameters Estimated for FhN Data – used to speed up examples

Usage

```
FhNestPars
```

Format

FhNestPars Estimated parameters for the FhN Data example.

FhNestCoefs Estimated coefficients for the FhN Data example.

Source

James Ramsay, Giles Hooker David Campbell and Jiguo Cao, 2007. "Parameter Estimation for Differential Equations: A Generalized Smoothing Approach". Journal of the Royal Statistical Society Vol 69 No 5.

FitMatch *Estimating Hidden States*

Description

Estimating hidden states to maximize agreement with the process.

Usage

```
FitMatchOpt(coefs,which,pars,proc,meth='nlminb',control=list())

FitMatchErr(coefs,allcoefs,which,pars,proc,sgn=1)

FitMatchDC(coefs,allcoefs,which,pars,proc,sgn=1)

FitMatchDC2(coefs,allcoefs,which,pars,proc,sgn=1)

FitMatchList(coefs,allcoefs,which,pars,proc,sgn=1)
```

Arguments

coefs	Vector giving the current estimate of the coefficients for the hidden states.
allcoefs	Matrix giving the coefficients of all the states including initial values for coefs.
which	Vector of indices of states to be estimated.
pars	Parameters to be used for the processes.
proc	proc object defining the state process.
sgn	Is the minimizing (1) or maximizing (0)?
meth	Optimization function currently one of 'nlminb', 'MaxNR', 'optim' or 'trust'.
control	Control object for optimization function.

Details

These routines allow the values of coefficients for some states to be optimized relative to the others. That is, the objective defined by proc is minimized over those states specified in which leaving the others constant. This would be typically done, for example, a smooth is taken to estimate some states non-parametrically, but data is not available on all of them.

A number of optimization routines have been implemented in FitMatchOpt, some experimentation is advised.

Value

FitMatchOpt A list containing

coefs The optimized coefficients for all states.

res The output of the optimization routine.

FitMatchErr	The value of the process likelihood at the current estimated states.
FitMatchDC	The derivative of FitMatchErr with respect to the elements coefs for the states being estimated.
FitMatchDC2	The second derivative of FitMatchErr with respect to the elements coefs for the states being estimated.
FitMatchList	Returns a list with elements value, gradient and hessian given by the output of FitMatchErr, FitMatchDC and FitMatchDC2.

See Also

ParsMatchErr, SplineCoefsErr, inneropt

Examples

```
################################
#### Some Data            #####
################################

data(FhNdata)

# And parameter estimates

data(FhNest)


################################
####  Basis Object      #######
################################

knots = seq(0,20,0.2)
norder = 3
nbasis = length(knots) + norder - 2
range = c(0,20)

bbasis = create.bspline.basis(range=range(FhNtimes),nbasis=nbasis,
 norder=norder,breaks=knots)


# Initial values for coefficients will be obtained by smoothing

fd.data = FhNdata[,1]

DEfd = smooth.basis(FhNtimes,fd.data,fdPar(bbasis,1,0.5))

coefs = cbind(DEfd$fd$coefs,rep(0,nbasis))
colnames(coefs) = FhNvarnames

#############################################################
### If We Only Observe One State, We Can Re-Smooth Others ###
#############################################################
```

```
profile.obj = LS.setup(pars=FhNpars,coefs=coefs,fn=make.fhn(),
                       basisvals=bbasis,lambda=1000,times=FhNtimes)
lik = profile.obj$lik
proc= profile.obj$proc

#  DD = Matrix(diag(1,200),sparse=TRUE)
#  tDD = t(DD)

fres = FitMatchOpt(coefs=coefs,which=2,pars=FhNpars,proc)

plot(fd(fres$coefs,bbasis))
```

`forward.prediction.error` *forward.prediction.error*

Description

Forward prediction error objective for choice of lambda in square error criteria.

Usage

```
forward.prediction.error(times,data,coefs,lik,proc,pars,whichtimes=NULL)
```

Arguments

`times`	Vector observation times for the data.
`data`	Matrix of observed data values.
`coefs`	Vector giving the current estimate of the coefficients in the spline.
`lik`	`lik` object defining the observation process.
`proc`	`proc` object defining the state process.
`pars`	Initial values of parameters to be estimated processes.
`whichtimes`	Specifies the start and end times for forward prediction, given by indeces of `times`. This can be one of **list** each element of the list is itself a list of length 2; the first element gives the starting time to use and the second is a vector giving the prediction times. **matrix** the first column giving the starting times and the second giving the ending times. If left NULL, `whichtimes` defaults to predicting one observation ahead from each observation.

Details

Forward prediction error can be used to choose values of `lambda` in the profiled estimation routines. The ordinary differential equation is solved starting from the starting times specified in `whichtimes` and measured at the corresponding measurement times. The error is then recorded. This should then be minimized by a grid search.

Value

The forwards prediction error from the estimates.

See Also

ProfileSSE, outeropt

`inneropt` *Inner Optimization Functions*

Description

Estmates coefficients given parameters.

Usage

```
inneropt(data,times,pars,coefs,lik,proc,in.meth='nlminb',control.in=list())
```

Arguments

`data`	Matrix of observed data values.
`times`	Vector observation times for the data.
`pars`	Initial values of parameters to be estimated processes.
`coefs`	Vector giving the current estimate of the coefficients in the spline.
`lik`	`lik` object defining the observation process.
`proc`	`proc` object defining the state process.
`in.meth`	Inner optimization function currently one of 'nlminb', 'maxNR', 'optim', 'trust' or 'SplineEst'. The last calls `SplineEst.NewtRaph`. This is fast but has poor convergence.
`control.in`	Control object for inner optimization function.

Details

This minimizes the objective function defined by the addition of the `lik` and `proc` objectives with respect to the coefficients. A number of generic optimization routines can be used and some experimentation is recommended.

Value

A list with elements

`coefs`	A matrix giving he optimized coefficients.
`res`	The results of the inner optimization function.

See Also

outeropt, Smooth.LS,LS.setup, multinorm.setup, SplineCoefsErr

Examples

```
## Not run:
# FitzHugh-Nagumo Equations

data(FhNdata)   # Some data
data(FhNest)    # with some parameter estimates

knots = seq(0,20,0.2)         # Create a basis
norder = 3
nbasis = length(knots) + norder - 2
range = c(0,20)

bbasis = create.bspline.basis(range=range(FhNtimes),nbasis=nbasis,
                                  norder=norder,breaks=knots)

lambda = 10000               # Penalty value

DEfd = smooth.basis(FhNtimes,FhNdata,fdPar(bbasis,1,0.5))   # Smooth to estimate
                                                            # coefficients first
coefs = DEfd$fd$coefs
colnames(coefs) = FhNvarnames

profile.obj = LS.setup(pars=FhNpars,coefs=coefs,fn=make.fhn(),
                       basisvals=bbasis,lambda=lambda,times=FhNtimes)

lik = profile.obj$lik
proc= profile.obj$proc

res = inneropt(FhNdata,times=FhNtimes,FhNpars,coefs,lik,proc,in.meth='nlminb')

plot(fd(res$coefs,bbasis))

## End(Not run)
```

IntegrateForward *IntegrateForward*

Description

Solves a differential equation going forward based on a proc object.

Usage

```
IntegrateForward(y0,ts,pars,proc,more)
```

Arguments

y0	Initial conditions to start from.
ts	Vector of time points at which to report values of the differential equation solution.
pars	Initial values of parameters to be estimated processes.
proc	Object defining the state process. This can either be a function evaluating the right hand side of the differential equation or a proc object. If a proc object is given, proc$more$fn is assumed to give the right hand side of the differential equation.
more	If proc is a function, this contains a list of additional inputs.

Value

Returns the output from solving the differential equation using the lsoda routines. Specifically, it returns a list with elements

times The output times.

states The output states.

See Also

Profile.LS, Profile.multinorm

Examples

```
proc = make.SSEproc()
proc$more = make.fhn()
proc$more$names = c('V','R')

y0 = c(-1,1)
names(y0) = c('V','R')

pars = c(0.2,0.2,3)
names(pars) = c('a','b','c')

ts = seq(0,20,0.5)

value = IntegrateForward(y0,ts,pars,proc)

matplot(value$times,value$states)
```

make.findif *Finite Difference Functions*

Description

Returns a list of functions that calculate finite difference derivatives.

Usage

```
make.findif.ode()

make.findif.loglik()

make.findif.var()
```

Details

All these functions require the sepcification of `more$eps` to give the size of the finite differencing step. They also require `more` to specify the original object (ODE right hand side functions, definitions of `lik` and `proc` objects).

Value

A list of functions that calculate the derivatives via finite differencing schemes.

`make.findif.ode`
: calculates finite differences of a transform.

`make.findif.loglik`
: returns the finite differences to a calculated log likelihood; used within `lik` objects, or as `more` arguments to `Cproc` or `Dproc`.

`make.findif.var`
: finite difference approximations to variances; mostly used in the `Multinorm` functions.

See Also

`LS.setup`, `multinorm.setup`

Examples

```
# Sum of squared errors with finite differencing to get right-hand-side derivatives

proc = make.SSEproc()
proc$more = make.findif.ode()


# Finite differencing for the log likelihood
```

```
lik = make.findif.loglik()
lik$more = make.SSElik()


# Multivariate normal transitions with finite differencing for mean and variance functions

lik = make.multinorm()
lik$more = c(make.findif.ode,make.findif.var)

# Finite differencing for transition density of a discrete time system

proc = make.Dproc()
proc$more = make.findif.loglik()
```

`make.lik` *Observation Process Distribution Function*

Description

Returns a list of functions that calculate the observation process distribution and its derivatives; designed to be used with the collocation inference functions.

Usage

```
make.SSElik()

make.multinorm()
```

Details

These functions require `more` to be a list with elements:

`fn` The transform function of the states to observations, or to their derivatives.

`dfdx` The derivative of `fn` with respect to states.

`dfdp` The derivative of `fn` with respect to parameters.

`d2fdx2` The second derivative of `fn` with respect to states.

`d2fdxdp` The cross derivative of `fn` with respect to states and parameters.

`make.Multinorm` further requires:

`var.fn` The variance given in terms of states and parameters.

`var.dfdx` The derivative of `var.fn` with respect to states.

`var.dfdp` The derivative of `var.fn` with respect to parameters.

`var.d2fdx2` The second derivative of `var.fn` with respect to states.

`var.d2fdxdp` The cross derivative of `var.fn` with respect to states and parameters.

`make.SSElik` further requres `weights` giving weights to each observation.

Value

A list of functions that calculate the log observation distribution and its derivatives.

`make.SSElik`	calculates weighted squared error between predictions (given by `fn` in `more`) and observations
`make.Multinorm`	calculates a multivariate normal distribution.

See Also

`LS.setup`, `multinorm.setup`

Examples

```
# Straightforward sum of squares:

lik = make.SSElik()
lik$more = make.id()

# Multivariate normal about an exponentiated state with constant variance

lik = make.multinorm()
lik$more = c(make.exp(),make.cvar())
```

---

make.logtrans *Log Transforms*

Description

Functions to modify liklihood, transform, `lik` and `proc` objects so that the operate with the state defined on a log scale.

Usage

```
make.logtrans()

make.exptrans()

make.logstate.lik()

make.exp.Cproc()

make.exp.Dproc()
```

Details

All functions require `more` to specify the original object (ODE right hand side functions, definitions of `lik` and `proc` objects).

Value

A list of functions that calculate log transforms and derivatives in various contexts.

`make.logtrans`	modifies the right hand side of a differential equation and its derivatives for a loged state vector.
`make.exptrans`	modfies a map from states to observations to a map from logged states to observations along with its derivatives.
`make.logstate.lik`	modifies a `lik` object for state vectors given on the log scale.
`make.exp.Cproc`	`Cproc` with the state given on the log scale.
`make.exp.Dproc`	`Dproc` with the state given on the log scale.

See Also

`LS.setup`, `make.Cproc`, `make.Dproc`

Examples

```
# Model the log of an SEIR process

proc = make.SSEproc()
proc$more = make.logtrans()
proc$more$more = make.SEIR()

# Observe a linear combination  of

lik = make.logstate.lik()
lik$more = make.SSElik()
lik$more$more = make.genlin()

# SEIR Model with multivariate transition densities

proc = make.exp.Cproc()
proc$more = make.multinorm()
proc$more$more = c(make.SEIR(),make.cvar())
```

`make.proc` *Process Distributions*

Description

Functions to define process distributions in the collocation inference package.

Usage

```
make.Dproc()

make.Cproc()

make.SSEproc()
```

Details

All functions require `more` to specify this distribution. This should be a list containing

`fn` The distribution specified.

`dfdx` The derivative of `fn` with respect to states.

`dfdp` The derivative of `fn` with respect to parameters.

`d2fdx2` The second derivative of `fn` with respect to states.

`d2fdxdp` The cross derivative of `fn` with respect to states and parameters.

For `Cproc` and `Dproc` this should specify the distribution; for `SSEproc` it should specify the right hand side of a differential equation.

Value

A list of functions that the process distribution

`make.Cproc`	creates functions to evaluate the distribution of the derivative of the state vector given the current state for continuous-time systems.
`make.Dproc`	creates functions to evaluate the distribution of the next time point of the state vector given the current state for discrete-state systems.
`make.SSEproc`	treats the distribution of the derivative as an independent gaussian and cacluates weighted sums of squared errors between derivatives and the prediction from the current state.

See Also

`LS.setup`, `multinorm.setup`

Examples

```
# FitzHugh-Nagumo Equations

proc = make.SSEproc()
proc$more = make.fhn()

# Henon Map

proc = make.Dproc()
proc$more = make.Henon
```

```
# SEIR with multivariate normal transitions

proc = make.Cproc()
proc$more = make.multinorm()
proc$more$more = c(make.SEIR(),make.var.SEIR())
```

`make.transfer` *Transfer Functions*

Description

Returns a list of functions that calculate the transform and its derivatives.

Usage

```
make.id()

make.exp()

make.genlin()

make.fhn()

make.Henon()

make.SEIR()

make.NS()

chemo.fun(times,y,p,more=NULL)
```

Arguments

All the functions created by `make...` functions, require the arguments needed by `chemo.fun`

`times`	Evaluation times
`y`	Values of the state at the evaluation times
`p`	Parameters to be used
`more`	A list of additional arguments, in this case `NULL`, for `pomp.sekelton` and `pomp.dmeasure`, `more` should be a list containing a `pomp` object in the element `pomp.obj`.

Details

`make.genlin` requires the specification of further elements in the list. In particular the element `more` should be a list containing

`mat` a matrix defining the linear transform before any parameters are added. This may be all zero, but it may also specify fixed elements, if desired.

`sub` a k-by-3 matrix indicating which parameters should be entered into which elements of `mat`. Each row is a triple giving the row and colum of `mat` to be specified and the element of the parameter vector that should be substituted. `sub` over-rides any values in `mat`.

`force` if input functions are given, these are given as a list.

`force.mat` specifying the influence of the elements of `force` on the state variables. Defined as in `mat`.

`force.sub` defined as in `sub`, over-rides the elements of `force.mat` with parameter values.

`make.diagnostics` estimates forcing-function diagnostics as in Hooker, 2009 for goodness-of-fit assessment. It requires

`psi` Values of a basis expansion for forcing functions at the quadrature points.

`which` Which states are to be forced?

`fn`, `dfdx`, `d2fdx2` Functions and derivatives as would be used to estimate parameters for the original equations.

pars Parameters to go into `more$fn`.

`make.SEIR` estimates parameters and a seasonal variation in the infection rate in an SEIR model. It requires the specification of the seasonal change rate in `more` by a list with objects

`beta.fun` A function to calculate beta, it should have arguments `t`, `p` and `betadef` and return a matrix giving the value of beta at times `t` with parameters `p`.

`beta.dfdp` Should calculate the derivative of `beta.fun` with respect to `p`, at times `t` returning a matrix. The matrix should be of size `length(t)` by `length(p)` where `p` is the entire parameter vector.

`betadef` Additional inputs (eg bases) to `beta.fun` and `beta.dfdp`.

`make.NS` provides functions for the North Shore example. This is a possibly time-varying forced linear system of one dimension. It requires `more` to specify `betabasis` to describe the autoregressive coefficient, and `alphabasis` to provide a contant in front of the functional data object `rainfd`.

`chemo.fun` Is a five-state predator-prey-resources model used as an example. It stands alone as a function and should be used with the `findif.ode` functions.

Value

A list of functions that calculate the transform and its derivatives, in a form compatible with the collocation inference functions.

`make.id`	returns the identity transform.
`make.exp`	returns the exponential transform.
`make.genlin`	returns a linear combination transform – see details section below.
`make.fhn`	returns the FitzHugh-Nagumo equations.
`make.Henon`	reutrns the Henon map.
`make.SEIR`	returns SEIR equations for estimating the shape of a seasonal forcing component.
`make.diagnostics`	functions to perform forcing function diagnostics.

See Also

LS.setup, multinorm.setup

Examples

```
# Observe the FitzHugh-Nagumo equations

proc = make.SSEproc()
proc$more = make.fhn()

lik = make.SSElik()
lik$more = make.id()

# Observe an unknown scalar transform of each component of a Henon map, given
# in the first two elements of the parameter vector:

proc = make.Dproc()
proc$more = make.multinorm()
proc$more$more = c(make.Henon,make.cvar)

lik = make.multinorm()
lik$more = c(make.genlin,make.cvar)
lik$more$more = list(mat = matrix(0,2,2),sub=matrix(c(1,1,1,2,2,2),2,3,byrow=TRUE))

# Model SEIR equations on the log scale and then exponentiate

lik = make.SSElik()
lik$more = make.exp()

proc = make.SSEproc()
proc$more = make.logtrans()
proc$more$more = make.SEIR()
```

---

make.variance *Variance Functions*

Description

Returns a list of functions that calculate a (possibly state and parameter dependent) variance.

Usage

```
make.cvar()

make.var.SEIR()
```

Details

make.cvar requires the specification of further elements in the list. In particular the element more should be a list containing

Value

A list of functions that calculate a variance function and its derivatives, in a form compatible with the collocation inference functions.

`make.cvar`	returns a variance that is constant but may depend on parameters
`make.var.SEIR`	returns a state-dependent transition covariance matrix calculated for the SEIR equations.

See Also

`make.multinorm`

Examples

```
# Multivariate normal observation of the state vector.

lik = make.multinorm()
lik$more = c(make.id(),make.cvar())
```

NSdata *North Shore data*

Description

Groundwater Data from Vancouver's North Shore

Usage

```
NSgroundwater
```

Format

NSgroundwater A 315 by 1 matrix of data on groundwater level collected in vancouver.

NStimes A vector of 315 observation times corresponding to NSgroundwater.

NSrainfall Rainfall as a covariate to NSgroundwater; this quantity is lagged by 3 days.

`outeropt` *Outer Optimization Functions*

Description

Outer optimization; performs profiled estimation.

Usage

```
outeropt(data,times,pars,coefs,lik,proc,
      in.meth='nlminb',out.meth='nlminb',
      control.in=list(),control.out=list(),active=1:length(pars))
```

Arguments

`data`	Matrix of observed data values.
`times`	Vector observation times for the data.
`pars`	Initial values of parameters to be estimated processes.
`coefs`	Vector giving the current estimate of the coefficients in the spline.
`lik`	`lik` object defining the observation process.
`proc`	`proc` object defining the state process.
`in.meth`	Inner optimization function currently one of 'nlminb', 'maxNR', 'optim' or 'SplineEst'. The last calls `SplineEst.NewtRaph`. This is fast but has poor convergence.
`out.meth`	Outer optimization function to be used, one of 'optim' (defaults to BFGS routine in `optim` unless `control.out$meth` specifies otherwise), 'nlminb', 'maxNR' #, 'trust' or 'subplex'. When squared error is being used, 'ProfileGN' and 'nls' can also be given. The former of these calls `Profile.GausNewt`, a fast but naive Gauss-Newton solver.
`control.in`	Control object for inner optimization function.
`control.out`	Control object for outer optimization function.
`active`	Indices indicating which parameters of `pars` should be estimated; defaults to all of them.

Details

The outer optimization for parameters looks only at the objective defined by the `lik` object. For every parameter value, `coefs` are optimized by `inneropt` and then the value of `lik` for these coefficients is computed.

A number of optimization routines can be used here, some experimentation is recommended. Libraries for these optimization routines are not pre-loaded. Where these functions take options as explicit arguments instead of a list, they should be listed in `control.out` and will be called by their names.

The routine creates temporary files 'curcoefs.tmp' and 'optcoefs.tmp' to update coefficients as `pars` evolves. These overwrite existing files of those names and are deleted before the function terminates.

Value

A list containing

pars	Optimized parameters
coefs	Optimized coefficients at pars
res	The result of the outer optimization.
counter	A set of parameters and objective values for each successful iteration.

See Also

inneropt, Profile.LS, ProfileSSE, ProfileErr, LS.setup, multinorm.setup

Examples

```
## Not run:
data(FhNdata)
data(FhNest)

knots = seq(0,20,0.2)         # Create a basis
norder = 3
nbasis = length(knots) + norder - 2
range = c(0,20)

bbasis = create.bspline.basis(range=range,nbasis=nbasis,norder=norder,breaks=knots)

lambda = 10000               # Penalty value

DEfd = smooth.basis(FhNtimes,FhNdata,fdPar(bbasis,1,0.5))   # Smooth to estimate
                                                            # coefficients first
coefs = DEfd$fd$coefs
colnames(coefs) = FhNvarnames

profile.obj = LS.setup(pars=FhNpars,coefs=coefs,fn=make.fhn(),basisvals=bbasis,
      lambda=lambda,times=FhNtimes)

lik = profile.obj$lik
proc= profile.obj$proc

res = outeropt(data=FhNdata,times=FhNtimes,pars=FhNpars,coefs=coefs,lik=lik,proc=proc,
    in.meth="nlminb",out.meth="nlminb",control.in=NULL,control.out=NULL)


plot(res$coefs,main='outeropt')
print(blah)

## End(Not run)
```

ParsMatch *Estimate of Parameters from Smooth*

Description

Objective function and derivatives to estimate parameters with a fixed smooth.

Usage

```
ParsMatchOpt(pars,coefs,proc,active=1:length(pars),meth='nlminb',control=list())

ParsMatchErr(pars,coefs,proc,active=1:length(pars),allpars,sgn=1)

ParsMatchDP(pars,coefs,proc,active=1:length(pars),allpars,sgn=1)

ParsMatchList(pars,coefs,proc,active=1:length(pars),allpars,sgn=1)
```

Arguments

pars	Initial values of parameters to be estimated processes.
coefs	Vector giving the current estimate of the coefficients in the spline.
proc	proc object defining the state process.
active	Incides indicating which parameters of allpar should be estimated; defaults to all of them.
allpars	Vector of all parameters, the assignment allpar[active]=pars is made initially.
sgn	Is the minimizing (1) or maximizing (0)?
meth	Optimization function currently one of 'nlminb', 'MaxNR', 'optim' or 'trust'.
control	Control object for optimization function.

Details

These routines fix the estimated states at the value given by coefs and estimate pars to maximize agreement between the fixed state and the objective given by the proc object.

A number of optimization routines have been implemented in FitMatchOpt, some experimentation is advised.

Value

ParsMatchOpt	A list containing: **pars** The entire parameter vector after optimization. **res** The output of the optimization routine.
ParsMatchErr	The value of the process likelihood at the current estimated states.
ParsMatchDP	The derivative fo ParsMatchErr with respect to pars[active].
ParsMatchList	A list with entries value and gradient given by the output of ParsMatchErr and ParsMatchDP respectively.

See Also

FitMatchErr, SplineCoefsErr, inneropt

Examples

```
data(FhNdata)

###############################
####  Basis Object      #######
###############################

knots = seq(0,20,0.2)
norder = 3
nbasis = length(knots) + norder - 2
range = c(0,20)

bbasis = create.bspline.basis(range=range(FhNtimes),nbasis=nbasis,
 norder=norder,breaks=knots)


# Initial values for coefficients will be obtained by smoothing

DEfd = smooth.basis(FhNtimes,FhNdata,fdPar(bbasis,1,0.5))   # Smooth to estimate
                                                            # coefficients first
coefs = DEfd$fd$coefs
colnames(coefs) = FhNvarnames



################################
### Initial Parameter Guesses ###
################################

profile.obj = LS.setup(pars=FhNpars,coefs=coefs,fn=make.fhn(),basisvals=bbasis,
    lambda=1000,times=FhNtimes)
lik = profile.obj$lik
proc= profile.obj$proc

pres = ParsMatchOpt(FhNpars,coefs,proc)

npars = pres$pars
```

Profile.covariance *Profile.covariance*

Description

Newey-West estimate of covariance of parameter estimates from profiling.

Usage

```
Profile.covariance(pars,active=NULL,times,data,coefs,lik,proc,
          in.meth='nlminb',control.in=NULL,eps=1e-6,GN=FALSE)
```

Arguments

pars	Initial values of parameters to be estimated processes.
active	Incides indicating which parameters of pars should be estimated; defaults to all of them.
times	Vector observation times for the data.
data	Matrix of observed data values.
coefs	Vector giving the current estimate of the coefficients in the spline.
lik	lik object defining the observation process.
proc	proc object defining the state process.
in.meth	Inner optimization function currently one of 'nlminb', 'MaxNR', 'optim' or 'house'. The last calls SplineEst.NewtRaph. This is fast but has poor convergence.
control.in	Control object for inner optimization functions.
eps	Step-size for finite difference estimate of second derivatives.
GN	Indicator of whether a Gauss-Newton approximation for the Hessian should be employed. Only valid for least-squares methods.

Details

Currently assumes a lag-5 auto-correlation among observation vectors.

Value

Returns a Newey-West estimate of the covariance matrix of the parameter estimates.

See Also

ProfileErr, ProfileSSE, Profile.LS, Profile.multinorm

Examples

```
# See example in Profile.LS
```

ProfileObjective *Profile Estimation with Collocation Inference*

Description

Profile estimation and objective functions for collocation estimation of parameters in continuous-time stochastic processes.

Usage

```
Profile.GausNewt(pars,times,data,coefs,lik,proc,in.meth="nlminb",
      control.in=NULL,active=1:length(pars),
      control=list(reltol=1e-6,maxit=50,maxtry=15,trace=1))

ProfileSSE(pars,allpars,times,data,coefs,lik,proc,in.meth='nlminb',
      control.in=NULL,active=1:length(pars),dcdp=NULL,oldpars=NULL,
      use.nls=TRUE,sgn=1)

ProfileErr(pars,allpars,times,data,coefs,lik,proc,in.meth = "house",
      control.in=NULL,sgn=1,active=1:length(allpars))

ProfileDP(pars,allpars,times,data,coefs,lik,proc,in.meth = "house",
      control.in=NULL,sgn=1,sumlik=1,active=1:length(allpars))

ProfileList(pars,allpars,times,data,coefs,lik,proc,in.meth = "house",
      control.in=NULL,sgn=1,active=1:length(allpars))
```

Arguments

pars	Initial values of parameters to be estimated processes.
allpars	Full parameter vector including pars. Assignment allpars[active] = pars is always made.
times	Vector observation times for the data.
data	Matrix of observed data values.
coefs	Vector giving the current estimate of the coefficients in the spline.
lik	lik object defining the observation process.
proc	proc object defining the state process.
in.meth	Inner optimization function currently one of 'nlminb', 'MaxNR', 'optim' or 'house'. The last calls SplineEst.NewtRaph. This is fast but has poor convergence.
control.in	Control object for inner optimization function.
sgn	Is the minimizing (1) or maximizing (0)?
active	Incides indicating which parameters of pars should be estimated; defaults to all of them.

`oldpars` Starting parameter values for the Q-function in the EM algorithm.

`dcdp` Estimate for the gradient of the optimized coefficients with respect to parameters; used internally.

`use.nls` In ProfileSSE, is 'nls' being used in the outer-optimization?

`sumlik` In ProfileDP and ProfileDP.AllPar; should the gradient be given for each observation, or summed over them?

`control` A list giving control parameters for Newton-Raphson optimization. It should contain

reltol Relative tollerance criterion for the gradient and improvement before termination.

maxit Maximum number of iterations.

maxtry Maximum number of halving-steps to try before declaring no improvement is possible.

trace How much iteration history to output; 0 surpresses all output, a positive value outputs parameters and improvement at each iteration.

Details

`Profile.GausNewt` provides a simple implementation of Gauss-Newton optimization and may not result in optimized values that are as good as other optimizers in `R`.

When `Profile.GausNewt` is not being used, `ProfileSEE` and `ProfileERR` create the following temporary files:

counter.tmp The number of halving-steps taken on the current optimization step.

curcoefs.tmp The current estimates of the coefficients.

optcoefs.tmp The optimal estimates of the coefficients in the iteration history.

These need to be removed manually when the optimization is finished. `optcoefs.tmp` will contain the optimal value of `coefs` for plotting the estimated trajectories.

Value

`Profile.GausNewt`

Output of a simple Gaus-Newton iteration to optimizing the objective function when the observation likelihood takes the form of a sum of squared errors. Returns a list with the following elements:

pars The optimized value of the parameters.

in.res The result of the inner optimization.

value The value of the optimized sum of squared errors.

`ProfileSSE` Output for the outer optimization when the observation likelihood is given by squared error. This is a list with the following elements

value The value of the outer optimization criterion.

gradient The derivative of `f` with respect to `pars`.

coefs The optimized value of `coefs` for the current value of `pars`.

dcdp The derivative of the optimized value of `coefs` at the current value of `pars`.

`ProfileErr`	The outer optimization criterion in the general case: the value of the observation likelihood at the profiled estimates of `coefs`.
`ProfileDP`	The derivative of `ProfileErr` with respect to `allpars[active]`.
`ProfileList`	Returns the results of ProfileErr and ProfileDP as a list with elements `value` and `gradient`

See Also

`outeropt`, `Profile.LS`, `Profile.multinorm`, `LS.setup`, `multinorm.setup`

`Profiling Routines` *Profile Estimation Functions*

Description

These functions are wrappers that create lik and proc functions and run generalized profiling.

Usage

```
Profile.LS(fn,data,times,pars,coefs=NULL,basisvals=NULL,lambda,
                        fd.obj=NULL,more=NULL,weights=NULL,quadrature=NULL,
                        likfn = make.id(), likmore = NULL,
                        in.meth='nlminb',out.meth='nls',
                      control.in,control.out,eps=1e-6,active=NULL,posproc=FALSE,
                        poslik=FALSE,discrete=FALSE,names=NULL,sparse=FALSE)

Profile.multinorm(fn,data,times,pars,coefs=NULL,basisvals=NULL,var=c(1,0.01),
                        fd.obj=NULL,more=NULL,quadrature=NULL,
                        in.meth='nlminb',out.meth='optim',
                        control.in,control.out,eps=1e-6,active=NULL,
                  posproc=FALSE,poslik=FALSE,discrete=FALSE,names=NULL,sparse=FALSE)
```

Arguments

`fn`	A function giving the right hand side of a differential/difference equation. The function should have arguments **times** The times at which the RHS is being evaluated. **x** The state values at those times. **p** Parameters to be entered in the system. **more** An object containing additional inputs to `fn` It should return a matrix of the same dimension of `x` giving the right hand side values. If `fn` is given as a single function, its derivatives are estimated by finite-differencing with stepsize `eps`. Alternatively, a list can be supplied with elements: **fn** Function to calculate the right hand side should accept a matrix of state values at .

dfdx Function to calculate the derivative with respect to x

dfdp Function to calculate the derivative with respect to p

d2fdx2 Function to calculate the second derivative with respect to x

d2fdxdp Function to calculate the second derivative with respect to x and p

These functions take the same arguments as fn and should output multidimensional arrays with the dimensions ordered according to time, state, deriv1, deriv2; here derivatives with respect to x always precede derivatives with respect to p.

data — Matrix of observed data values.

times — Vector observation times for the data.

pars — Initial values of parameters to be estimated processes.

coefs — Vector giving the current estimate of the coefficients in the spline.

basisvals — Values of the collocation basis to be used. This can either be a basis object from the fda package, or a list elements:

bvals.obs A matrix giving the values of the basis at the observation times

bvals A matrix giving the values of the basis at the quadrature times

dbvals A matrix giving the derivative of the basis at the quadrature times

lambda — (Profile.LS only) Penalty value trading off fidelity to data with fidelity to differential equations.

var — (profile.Cproc or profile.Dproc) A vector of length 2, giving

fd.obj — (Optional) A functional data object; if this is non-null, coefs and basisvals is extracted from here.

more — An object specifying additional arguments to fn.

weights — (Profile.LS only)

quadrature — Quadrature points, should contain two elements (if not NULL)

qpts Quadrature points; defaults to midpoints between knots

qwts Quadrature weights; defaults to normalizing by the length of qpts.

in.meth — Inner optimization function to be used, currently one of 'nlminb', 'MaxNR', 'optim' or 'house'. The last calls SplineEst.NewtRaph. This is fast but has poor convergence.

out.meth — Outer optimization function to be used, depending on the type of method

Profile.LS One of 'nls' or 'ProfileGN'; the latter calls Profile.GausNewt which is fast but may have poor convergence.

Profile.multinorm One of 'optim' (defaults to BFGS routine in optim unless control.out$meth specifies otherwise), 'nlminb', or 'maxNR'.

control.in — Control object for inner optimization function.

control.out — Control object for outer optimization function.

eps — Finite differencing step size, if needed.

active — Incides indicating which parameters of pars should be estimated; defaults to all of them.

posproc	Should the state vector be constrained to be positive? If this is the case, the state is represented by an exponentiated basis expansion in the proc object.
poslik	Should the state be exponentiated before being compared to the data? When the state is represented on the log scale (posproc=TRUE), this is an alternative to taking the log of the data.
discrete	Is this a discrete-time or a continuous-time system? If discrete, the derivative is instead taken to be the value at the next time point.
names	The names of the state variables if not given by the column names of coefs.
sparse	Should sparse matrices be used for basis values? This option can save memory when ProfileGausNewt and SplineEstNewtRaph are called. Otherwise sparse matrices will be converted to full matrices and this can slow the code down.
likfn	Defines a map from the trajectory to the observations. This should be in the same form as fn. If a function is given, derivatives are estimated by finite differencing, otherwise a list is expected to provide the same derivatives as fn. If poslik=TRUE, the states are exponentiated before the likfn is evaluated and the derivatives are updated to account for this. Defaults to the identity transform.
likmore	A list containing additional inputs to likfn if needed, otherwise set to NULL

Details

These functional all carry out the profiled optimization method of Ramsay et al 2007. Profile.LS uses a sum of squared errors criteria for both fit to data and the fit of the derivatives to a differential equation. Profile.multinorm uses multivariate normal approximations. discrete changes the system to a discrete-time difference equation with the right hand side function giving the transition function.

Note that these all call outeropt, which creates temporary files 'curcoefs.tmp' and 'optcoefs.tmp' to update coefficients as pars evolves. These overwrite existing files of those names and are deleted before the function terminates.

Value

A list with elements

pars	Optimized parameters
coefs	Optimized coefficients at pars
lik	The lik object generated
proc	The proc item generated
data	The data used in doing the fitting.
times	The vector of times at which the observations were made

See Also

outeropt, ProfileErr, ProfileSSE, LS.setup, multinorm.setup

Examples

```
##############################
####   Data            #######
##############################

data(FhNdata)

##############################
####  Basis Object     #######
##############################

knots = seq(0,20,0.2)
norder = 3
nbasis = length(knots) + norder - 2
range = c(0,20)

bbasis = create.bspline.basis(range=range(FhNtimes),nbasis=nbasis,
 norder=norder,breaks=knots)


#### Start from pre-estimated values to speed up optimization

data(FhNest)

spars = FhNestPars
coefs = FhNestCoefs

lambda = 10000

res1 = Profile.LS(make.fhn(),data=FhNdata,times=FhNtimes,pars=spars,coefs=coefs,
  basisvals=bbasis,lambda=lambda,in.meth='nlminb',out.meth='nls')

Covar = Profile.covariance(pars=res1$pars,times=FhNtimes,data=FhNdata,
  coefs=res1$coefs,lik=res1$lik,proc=res1$proc)


## Not run:
## Alternative, starting from perturbed coefficients -- takes too long for
# automatic checks in CRAN

# Initial values for coefficients will be obtained by smoothing

DEfd = smooth.basis(FhNtimes,FhNdata,fdPar(bbasis,1,0.5))   # Smooth to estimate
                                                            # coefficients first
coefs = DEfd$fd$coefs
colnames(coefs) = FhNvarnames


##############################
####     Optimization       ###
##############################
```

```
spars = c(0.25,0.15,2.5)          # Perturbed parameters
names(spars)=FhNparnames
lambda = 10000

res1 = Profile.LS(make.fhn(),data=FhNdata,times=FhNtimes,pars=spars,coefs=coefs,
  basisvals=bbasis,lambda=lambda,in.meth='nlminb',out.meth='nls')

par(mfrow=c(2,1))
plotfit.fd(FhNdata,FhNtimes,fd(res1$coefs,bbasis))

## End(Not run)




## Not run:
#####################################################
###  An Explicitly Multivariate Normal Formation ###
#####################################################

var = c(1,0.0001)

res2 = Profile.multinorm(make.fhn(),FhNdata,FhNtimes,pars=res1$pars,
          res1$coefs,bbasis,var=var,out.meth='nlminb', in.meth='nlminb')

## End(Not run)
```

SEIRdata *SEIR data*

Description

Data generated for SEIR Examples

Usage

```
SEIRdata
```

Format

SEIRdata A 261 by 1 matrix of data generated from the SEIR Gillespie process run over 5 years.

SEIRtimes A vector of 261 observation times corresponding to SEIRdata.

SEIRpars Named parameter vector used to generate `SEIRdata`.

SEIRvarnames `c('V','R')`: the state variable names for the SEIR system.

SEIRparnames parameter names for the SEIR system.

Source

Giles Hooker, Stephen P. Ellner, David Earn and Laura Roditi, 2010. "Parameterizing State-space Models for Infectious Disease Dynamics by Generalized Profiling: Measles in Ontario", Technical Report, Cornell University.

`setup` *Setup Functions for proc and lik objects*

Description

These functions set up lik and proc objects of squared error and multinormal processes.

Usage

```
LS.setup(pars,coefs=NULL,fn,basisvals=NULL,lambda,fd.obj=NULL,
        more=NULL,data=NULL,weights=NULL,times=NULL,quadrature=NULL,
        likfn = make.id(), likmore = NULL,eps=1e-6,
        posproc=FALSE,poslik=FALSE,discrete=FALSE,names=NULL,sparse=FALSE)

multinorm.setup(pars,coefs=NULL,fn,basisvals=NULL,var=c(1,0.01),fd.obj=NULL,
        more=NULL,data=NULL,times=NULL,quadrature=NULL,eps=1e-6,posproc=FALSE,
        poslik=FALSE,discrete=FALSE,names=NULL,sparse=FALSE)
```

Arguments

`pars`	Initial values of parameters to be estimated processes.
`coefs`	Vector giving the current estimate of the coefficients in the spline.
`fn`	A function giving the right hand side of a differential/difference equation. The function should have arguments **times** The times at which the RHS is being evaluated. **x** The state values at those times. **p** Parameters to be entered in the system. **more** An object containing additional inputs to `fn` It should return a matrix of the same dimension of `x` giving the right hand side values. If `fn` is given as a single function, its derivatives are estimated by finite-differencing with stepsize `eps`. Alternatively, a list can be supplied with elements: **fn** Function to calculate the right hand side should accept a matrix of state values at . **dfdx** Function to calculate the derivative with respect to `x` **dfdp** Function to calculate the derivative with respect to `p` **d2fdx2** Function to calculate the second derivative with respect to `x` **d2fdxdp** Function to calculate the second derivative with respect to `x` and `p`

These functions take the same arguments as fn and should output multidimensional arrays with the dimensions ordered according to time, state, deriv1, deriv2; here derivatives with respect to x always precede derivatives with respect to p.

fn can also be given as a pomp object (see the pomp package), in which case it is interfaced to CollocInfer through pomp.skeleton using a finite differencing.

basisvals — Values of the collocation basis to be used. This can either be a basis object from the fda package, or a list elements:

bvals.obs A matrix giving the values of the basis at the observation times

bvals A matrix giving the values of the basis at the quadrature times

dbvals A matrix giving the derivative of the basis at the quadrature times

For discrete systems, it may also be specified as a matrix, in which case bvals$bvals is obtained by deleting the last row and bvals$dbvals is obtained by deleting the first/

If left as NULL, it is taken from fd.obj for discrete=FALSE and defaults to an identity matrix of the same dimension as the number of observations for discrete=TRUE systems.

lambda — (LS.setup only) Penalty value trading off fidelity to data with fidelity to differential equations.

var — (profile.Cproc or profile.Dproc) A vector of length 2, giving

fd.obj — (Optional) A functional data object; if this is non-null, coefs and basisvals is extracted from here.

more — An object specifying additional arguments to fn.

data — The data to be used, this can be a matrix, or a three-dimensional array. If the latter, the middle dimension is taken to be replicates. The data are returned, if replicated they are returned in a concatenated form.

weights — (LS.setup only)

times — Vector observation times for the data. If the data are replicated, times are returned in a concatenated form.

quadrature — Quadrature points, should contain two elements (if not NULL)

qpts Quadrature points; defaults to midpoints between knots

qwts Quadrature weights; defaults to normalizing by the length of qpts.

eps — Finite differencing step size, if needed.

posproc — Should the state vector be constrained to be positive? If this is the case, the state is represented by an exponentiated basis expansion in the proc object.

poslik — Should the state be exponentiated before being compared to the data? When the state is represented on the log scale TRUE, this is an alternative to taking the log of the data.

discrete — Is this a discrete or continuous-time system?

names — The names of the state variables if not given by the column names of coefs.

sparse — Should sparse matrices be used for basis values? This option can save memory when ProfileGausNewt and SplineEstNewtRaph are called. Otherwise sparse matrices will be converted to full matrices and this can slow the code down.

likfn	Defines a map from the trajectory to the observations. This should be in the same form as fn. If a function is given, derivatives are estimated by finite differencing, otherwise a list is expected to provide the same derivatives as fn. If poslik=TRUE, the states are exponentiated before the likfn is evaluated and the derivatives are updated to account for this. Defaults to the identity transform.
likmore	A list containing additional inputs to likfn if needed, otherwise set to NULL

Details

These functions provide basic setup utilities for the collocation inference methods. They define lik and proc objects for sum of squared errors and multivariate normal log likelihoods with nonlinear transfer functions describing the evolution of the state vector.

LS.setup Creates sum of squares functions

multinorm.setup Creates multinormal log likelihoods for a continuous-time system.

Value

A list with elements

coefs	Starting values for coefs
lik	The lik object generated
proc	The proc item generated
data	The data matrix, concatenated if from a 3d array.
times	The vector of observation times, concatenated if data is a 3d array.

See Also

inneropt, outeropt, Profile.LS, Profile.multinorm, Smooth.LS, Smooth.multinorm

Examples

```
# FitzHugh-Nagumo

t = seq(0,20,0.05)            # Observation times

pars = c(0.2,0.2,3)           # Parameter vector
names(pars) = c('a','b','c')

knots = seq(0,20,0.2)         # Create a basis
norder = 3
nbasis = length(knots) + norder - 2
range = c(0,20)

bbasis = create.bspline.basis(range=range,nbasis=nbasis,norder=norder,breaks=knots)

lambda = 10000               # Penalty value

coefs = matrix(0,nbasis,2)   # Coefficient matrix
```

```
profile.obj = LS.setup(pars=pars,coefs=coefs,fn=make.fhn(),basisvals=bbasis,
                       lambda=lambda,times=t)


# Using multinorm

var = c(1,0.01)

profile.obj = multinorm.setup(pars=pars,coefs=coefs,fn=make.fhn(),
                                        basisvals=bbasis,var=var,times=t)


# Henon - discrete

hpars = c(1.4,0.3)
t = 1:200

coefs = matrix(0,200,2)
lambda = 10000

profile.obj = LS.setup(pars=hpars,coefs=coefs,fn=make.Henon(),basisvals=NULL,
                               lambda=lambda,times=t,discrete=TRUE)
```

Smooth.LS *Model-Based Smoothing Functions*

Description

Perform the inner optimization to estimate coefficients given parameters.

Usage

```
Smooth.LS(fn,data,times,pars,coefs=NULL,basisvals=NULL,lambda,fd.obj=NULL,
        more=NULL,weights=NULL,quadrature=NULL,likfn = make.id(),
        likmore = NULL,in.meth='nlminb',control.in,eps=1e-6,
        posproc=FALSE,poslik=FALSE,discrete=FALSE,names=NULL,
        sparse=FALSE)

Smooth.multinorm(fn,data,times,pars,coefs=NULL,basisvals=NULL,var=c(1,0.01),
        fd.obj=NULL,more=NULL,quadrature=NULL,in.meth='nlminb',
        control.in,eps=1e-6,posproc=FALSE,poslik=FALSE,discrete=FALSE,
        names=NULL,sparse=FALSE)
```

Arguments

fn	A function giving the right hand side of a differential/difference equation. The function should have arguments **times** The times at which the RHS is being evaluated. **x** The state values at those times.

p Parameters to be entered in the system.

more An object containing additional inputs to `fn`

It should return a matrix of the same dimension of `x` giving the right hand side values.

If `fn` is given as a single function, its derivatives are estimated by finite-differencing with stepsize `eps`. Alternatively, a list can be supplied with elements:

fn Function to calculate the right hand side should accept a matrix of state values at .

dfdx Function to calculate the derivative with respect to `x`

dfdp Function to calculate the derivative with respect to `p`

d2fdx2 Function to calculate the second derivative with respect to `x`

d2fdxdp Function to calculate the second derivative with respect to `x` and `p`

These functions take the same arguments as `fn` and should output multidimensional arrays with the dimensions ordered according to time, state, deriv1, deriv2; here derivatives with respect to `x` always precede derivatives with respect to `p`.

`data` Matrix of observed data values.

`times` Vector observation times for the data.

`pars` Initial values of parameters to be estimated processes.

`coefs` Vector giving the current estimate of the coefficients in the spline.

`basisvals` Values of the collocation basis to be used. This can either be a basis object from the `fda` package, or a list elements:

bvals.obs A matrix giving the values of the basis at the observation times

bvals A matrix giving the values of the basis at the quadrature times

dbvals A matrix giving the derivative of the basis at the quadrature times

`lambda` (`Smooth.LS` only) Penalty value trading off fidelity to data with fidelity to differential equations.

`var` (`Smooth.multinorm`) A vector of length 2, giving

`fd.obj` (Optional) A functional data object; if this is non-null, `coefs` and `basisvals` is extracted from here.

`more` An object specifying additional arguments to `fn`.

`weights` (`Smooth.LS` only)

`quadrature` Quadrature points, should contain two elements (if not NULL)

qpts Quadrature points; defaults to midpoints between knots

qwts Quadrature weights; defaults to normalizing by the length of `qpts`.

`in.meth` Inner optimization function to be used, currently one of 'nlminb', 'MaxNR', 'optim' or 'SplineEst'. The last calls `SplineEst.NewtRaph`. This is fast but has poor convergence.

`control.in` Control object for inner optimization function.

`eps` Finite differencing step size, if needed.

posproc	Should the state vector be constrained to be positive? If this is the case, the state is represented by an exponentiated basis expansion in the proc object.
poslik	Should the state be exponentiated before being compared to the data? When the state is represented on the log scale (posproc=TRUE), this is an alternative to taking the log of the data.
discrete	Is this a discrete or continuous-time system?
names	The names of the state variables if not given by the column names of coefs.
sparse	Should sparse matrices be used for basis values? This option can save memory when ProfileGausNewt and SplineEstNewtRaph are called. Otherwise sparse matrices will be converted to full matrices and this can slow the code down.
likfn	Defines a map from the trajectory to the observations. This should be in the same form as fn. If a function is given, derivatives are estimated by finite differencing, otherwise a list is expected to provide the same derivatives as fn. If poslik=TRUE, the states are exponentiated before the likfn is evaluated and the derivatives are updated to account for this. Defaults to the identity transform.
likmore	A list containing additional inputs to likfn if needed, otherwise set to NULL

Details

These routines create lik and proc objects and call inneropt.

Value

A list with elements

coefs	Optimized coefficients at pars
lik	The lik object generated
proc	The proc item generated
res	The result of the optimization method
data	The data used in doing the fitting.
times	The vector of times at which the observations were made

See Also

inneropt, LS.setup, multinorm.setup, SplineCoefsErr

Examples

```
##############################
####   Data            #######
##############################

data(FhNdata)

##############################
####  Basis Object     #######
##############################
```

```
knots = seq(0,20,0.2)
norder = 3
nbasis = length(knots) + norder - 2
range = c(0,20)

bbasis = create.bspline.basis(range=range(FhNtimes),nbasis=nbasis,
 norder=norder,breaks=knots)


#### Start from pre-estimated values to speed up optimization

data(FhNest)

spars = FhNestPars
coefs = FhNestCoefs

lambda = 10000

res1 = Smooth.LS(make.fhn(),data=FhNdata,times=FhNtimes,pars=spars,coefs=coefs,
  basisvals=bbasis,lambda=lambda,in.meth='nlminb')


## Not run:
# Henon system

hpars = c(1.4,0.3)              # Parameters
t = 1:200

x = c(-1,1)                     # Create some dataa
X = matrix(0,200+20,2)
X[1,] = x

for(i in 2:(200+20)){ X[i,] = make.Henon()$ode(i,X[i-1,],hpars,NULL) }

X = X[20+1:200,]

Y = X + 0.05*matrix(rnorm(200*2),200,2)

basisvals = diag(rep(1,200))    # Basis is just identiy
coefs = matrix(0,200,2)


# For sum of squared errors

lambda = 10000

res1 = Smooth.LS(make.Henon(),data=Y,times=t,pars=hpars,coefs,basisvals=basisvals,
  lambda=lambda,in.meth='nlminb',discrete=TRUE)

## End(Not run)
```

```
## Not run:
# For multinormal transitions

var = c(1,0.01)

res2 = Smooth.multinorm(make.Henon(),data=Y,t,pars=hpars,coefs,basisvals=NULL,
  var=var,in.meth='nlminb',discrete=TRUE)

## End(Not run)
```

SplineEst *Spline Estimation Functions*

Description

Model-based smoothing; estimation, objective criterion and derivatives.

Usage

```
SplineEst.NewtRaph(coefs,times,data,lik,proc,pars,
      control=list(reltol=1e-12,maxit=1000,maxtry=10,trace=0))

SplineCoefsList(coefs,times,data,lik,proc,pars,sgn=1)

SplineCoefsErr(coefs,times,data,lik,proc,pars,sgn=1)

SplineCoefsDC(coefs,times,data,lik,proc,pars,sgn=1)

SplineCoefsDP(coefs,times,data,lik,proc,pars,sgn=1)

SplineCoefsDC2(coefs,times,data,lik,proc,pars,sgn=1)

SplineCoefsDCDP(coefs,times,data,lik,proc,pars,sgn=1)
```

Arguments

coefs	Vector giving the current estimate of the coefficients in the spline.
times	Vector observation times for the data.
data	Matrix of observed data values.
lik	lik object defining the observation process.
proc	proc object defining the state process.
pars	Parameters to be used for the processes.
sgn	Is the minimizing (1) or maximizing (0)?
control	A list giving control parameters for Newton-Raphson optimization. It should contain

reltol Relative tollerance criterion for the gradient and improvement before termination.

maxit Maximum number of iterations.

maxtry Maximum number of halving-steps to try before declaring no improvement is possible.

trace How much iteration history to output; 0 surpresses all output, a positive value outputs parameters and improvement at each iteration.

Details

`SplineEst.NewtRaph` performs a simple Newton-Raphson estimate for the optimal value of the coefficients. This estimate lacks the convergence checks of other estimation packages, but may yeild a fast solution when needed.

Value

`SplineEst.NewtRaph`

Returns a list that is the result of the optimization with elements

value The final objective criterion.

coefs The optimizing value of the coefficients.

g The gradient at the optimizing value.

H The Hessian at the optimizing value.

`SplineCoefsList`

Collates the gradient calculations and returns a list with elements

value Output of `SplineCoefsErr`

gradient Output of `SplineCoefsDC`

Hessian Output of `SplineCoefsDC2`

`SplineCoefsErr` The complete data log likelihood for the smooth; the inner optimization objective.

`SplineCoefsDC` The derivative of `SplineCoefsErr` with respect to `coefs`.

`SplineCoefsDP` The derivative of `SplineCoefsErr` with respect to `pars`.

`SplineCoefsDC2` The second derivative of `SplineCoefsErr` with respect to `coefs`.

`SplineCoefsDCDP`

The second derivative of `SplineCoefsErr` with respect to `coefs` and `pars`.

The output of gradients is in terms of an array with dimensions corresponding to derivatives. Derivatives with with respect to coefficients are given in dimensions before those that give derivatives with respect to parameters.

See Also

`inneropt`, `Smooth.LS`

Index