gmx_sample

Syntax

```
r = gmx_sample ( Mi, Sig, N )
```

gpExactLINard

This function computes the predictive mean and variance at test input

Syntax

```
[m, S2] = gpExactLINard(logtheta, covfunc, invQ, input, target, test, SigmaX);
```

Description

This function computes the predictive mean and variance at test input If SigX: consider random input, with covariance SigX. Predictions computed using the exact equations. The form of the covariance function is

k(xˆp,xˆq) = xˆp'*inv(P)*xˆq

where the P matrix is diagonal with ARD parameters ell_1ˆ2,...,ell_Dˆ2, where D is the dimension of the input space. The hyperparameters are:

logtheta = [log(ell_1) log(ell_2) . log(ell_D) log(sqrt(s2))]]

Inputs:

- loghteta ... optimized hyperparameters
- covfunc ... dummy, used for (eventual) future compatibility
- invQ ... inverse of the data covariance matrix
- input ... input part of the training data, NxD matrix
- target ... output part of the training data (ie. target), Nx1 vector
- test ... D by 1 test input
- SigmaX ... covariance of the test input (OPTIONAL)

Outputs:

- m ... predicted mean
- S2 ... predicted variance (noise free)

Examples:

See also:

simulGPexactSE.m

Signature

gpExactSEard

gpExactSEard - This function computes the predictive mean and variance at test input

Syntax

```
[m, S2] = gpExactSEard(logtheta, covfunc, invQ, input, target, muX, SigX, lag);
```

Description

If SigX: consider random input, with covariance SigX. Predictions computed using the exact equations. The form of the covariance function is C(xˆp,xˆq) = v1*exp(-0.5 * sum_{d=1..D} w_d * (xˆp_d - xˆq_d)ˆ2) + v0\delta_{pq}

Input:

- loghteta ... optimized hyperparameters
- covfunc ... dummy, used for (eventual) future compatibility
- invQ ... inverse of the data covariance matrix
- input ... input part of the training data, NxD matrix
- target ... output part of the training data (ie. target), Nx1 vector
- muX ... D by 1 test input
- SigX ... covariance of the test input (OPTIONAL)
- lag ... the order of the model (number of used lagged outputs)

Output:

- m ...predicted mean
- S2 ... predicted variance (noise free)

Examples:

demo_example_gp_simulation.m

See Also:

simulGPexactSE.m

Signature

Based on the work of J. Quinonero-Candela and A. Girard.

gpTaylorSEard

Compute (marginal) predictions based on hyperparameters, training inputs and targets and test inputs from 2 to 4 outputs may be requested

Syntax

```
[m S2 deriv S2deriv] = gpTaylorSEard(logtheta, covfunc, invQ, input, target, test, SigmaX, I
```

Description

Input:

- loghteta ... optimized hyperparameters
- covfunc ... dummy, used for (eventual) future compatibility
- invQ ... inverse of the data covariance matrix
- input ... is a n by D matrix of training inputs
- target ... is a (column) vector (of size n) of targets
- test ... 1 by D test input
- SigmaX ... is the D by D input covariance matrix (optional)

Output:

- m ... is a (column) vector (of size nn) of prediced means
- S2 ... is a (column) vector (of size nn) of predicted variances
- deriv ... is a n by D matrix of mean partial derivatives
- S2deriv ... is a n by D by D matrix of (co-)variances on the partial derivatives (3 dimensional array (full covariance per test point)

Note that the reported variances in S2 are for the noise-free signal; to get the noisy variance, simply add the noise variance log(X(D+2)).

The form of the covariance function is

C(x^p,x^q) = v1 * exp(-0.5 * sum_{d=1..D} w_d * (x^p_d - x^q_d)^2)

Examples:

See also:

gp01lik

Signature

gpx

gpx - gp-extended is modified version of GP rutine from the basic 'gpml' toolbox.

Syntax

```
training: [nlZ dnlZ post       ] = gp(hyp, inf, mean, cov, lik, x, y);
prediction: [ymu ys2 fmu fs2 post] = gp(hyp, inf, mean, cov, lik, x, y, xs);
or: [ymu ys2 fmu fs2 post] = gp(hyp, inf, mean, cov, lik, x, y, xs, post);
```

Description

It was modified to calculate the model (inverse covariance matrix) only once during the simulation in order to speed-up. For this puprose an input and output parameter 'post' (posterior) was added, which is a struct that contains information about the model and is usualy returned by inference method (see infMethods.m). If the method is called without the 'post' input it will be calucalted and returned as the last output. On the other hand if this method is called with 'post' struct it will not be calculated again.

Gaussian Process inference and prediction. The gp function provides a flexible framework for Bayesian inference and prediction with Gaussian processes for scalar targets, i.e. both regression and binary classification. The prior is Gaussian process, defined through specification of its mean and covariance function. The likelihood function is also specified. Both the prior and the likelihood may have hyperparameters associated with them.

Two modes are possible: training or prediction: if no test cases are supplied, then the negative log marginal likelihood and its partial derivatives w.r.t. the hyperparameters is computed; this mode is used to fit the hyperparameters. If test cases are given, then the test set predictive probabilities are returned.

Input:

- hyp ... struct of hyperparameters
- inf ... function specifying the inference method
- cov ... prior covariance function (see below)
- mean ... prior mean function
- lik ... likelihood function
- x ... n by D matrix of training inputs
- y ... column vector of length n of training targets
- xs ... ns by D matrix of test inputs
- ys ... column vector of length nn of test targets

Output:

- nlZ ... returned value of the negative log marginal likelihood
- dnlZ ... column vector of partial derivatives of the negative log marginal likelihood w.r.t. each hyperparameter
- ymu ... column vector (of length ns) of predictive output means
- ys2 ... column vector (of length ns) of predictive output variances
- fmu ... column vector (of length ns) of predictive latent means
- fs2 ... column vector (of length ns) of predictive latent variances
- lp ... column vector (of length ns) of log predictive probabilities
- post ... struct representation of the (approximate) posterior 3rd output in training mode and 5th output in prediction mode

Examples:

simulGPnaive.m, simulGPmcmc.m

See also:

gp.m, covFunctions.m, infMethods.m, likFunctions.m, meanFunctions.m

Signature

- **Copyright(C)** by Carl Edward Rasmussen and Hannes Nickisch, 2011-02-18
- modified 2009 by Jus Kocijan
- modified 2010 by Jan Prikryl
- modified 2011 by Tomaž Šuštar

predNaive

% Syntax function [y, s2] = predARXnaive(logtheta, covfunc, input, target, xt, lag, kstep)

Description

"Naive" (i.e. without propagation of variance) simulation of the GP model. Uses routine gpr.

Input:

- hyp ... optimized hyperparameters (struct)
- inf ... function specifying the inference method
- meanfunc ... prior mean function
- covfunc ... specified covariance function, see help covFun for more info
- likfunc ... likelihood function
- input ... input part of the training data, NxD matrix
- target ... output part of the training data (ie. target), Nx1 vector
- xt ... matrix construct_ARsimul_input.m, kxD matrix,
- lag ... the order of the model (number of used lagged outputs)
- kstep ... number of steps to be predicted

Output:

- y ... mean predicted output
- s2 ... associated variances

Examples:

demo_example_gp_simulation.m

See Also:

spx.m, simulGPexact.m, simulGPmcmc.m

Signature

simulGPexactSE

Simulation of the GP model, where the output variance is propagated using analytical approximation

Syntax

```
[mu, sig2, m, s2] = simulGPexactSE(hyp, meanfunc, covfunc, likfunc, input, target, test, lag
```

Description

Simulation of the GP model, where the output variance is propagated using analytical approximation, see A. Girard, Approximate Methods for Propagation of Uncertainty with Gaussian Process Models, PhD thesis, 2004. Notes: Currently it can be used only with Gaussian covariance function and with white noise model (sum of covSEard and covNoise).

Uses routine gpExactSEard.

Inputs:

- hyp ... struct of optimized hyperparameters
- meanfunc ... prior mean function
- covfunc ... specified covariance function, see help covFun for more info
- likfunc ... likelihood function
- input ... input part of the training data, NxD matrix
- target ... output part of the training data (ie. target), Nx1 vector
- test ... input matrix for simulation, kxD vector, see construct_simul_input.m for more info
- lag ... the order of the model (number of used lagged outputs)

Outputs:

- mu ... predictive mean using "naive" approach (doesn't propagate the uncertainty)
- sig2 ... predictive variance using "naive" approach
- m ... predictive mean when propagating the uncertainty
- s2 ... predictive variance when propagating the uncertainty

The form of the covariance function is C(xˆp,xˆq) = v1*exp(-0.5 * sum_{d=1..D} w_d * (xˆp_d - xˆq_d)ˆ2) + v1 \delta_{pq} (computed using cov2.m)

Examples:

demo_example_gp_simulation.m

See Also:

simulGPnaive.m, gpExactSEard.m

Signature

Based on the work of J. Quinonero-Candela and A. Girard.

simulGPmcmc

simulGPmcmc - Simulation of the GP model, where the output variance is propagated using simple MCMC method

Syntax

```
[y, s2] = simulGPmcmc(hyp, inf, mean, cov, lik, input, target, test, lag, Nsamples);
```

Description

See A. Girard, Approximate Methods for Propagation of Uncertainty with Gaussian Process Models, PhD thesis, 2004. Idea: at every time step the output of GP model is approximated with Nsamples samples, which are used as the future inputs of the GP model. Samples are re-used if necessary (ie. y(k-1) for y(k-2) if lag=2 etc.) Uses routines gpx and gmx_sample.

Input:

- hyp ... struct of optimized hyperparameters
- inf ... function specifying the inference method
- mean ... prior mean function
- cov ... specified covariance function, see help covFun for more info
- lik ... likelihood function
- input ... input part of the training data, NxD matrix
- target ... output part of the training data (ie. target), Nx1 vector
- test ... input matrix for simulation, kxD vector, see construct_simul_input.m for more info
- lag ... the order of the model (number of used lagged outputs)
- Nsamples ... number of samples used in algorithm (ie. runs of simulation)

Output:

- mu ... mean predicted output
- s2 ... associated variances
- MU ... matrix of all predicted means, kxNsamples
- SIG2 ... associated predicted variances
- indexes_warning ... sampling problems, see mcmc_getsamplesgaussianmix.m

See also:

gpx, gmx_sample, simulGPnaive

Examples:

demo_example_gp_simulation

Signature

- Written by J. Prikryl, November 2010
- Based on the work of C.E. Rasmussen, A. Girard, K. Azman.

simulGPnaive

simulGPnaive - "Naive" (i.e. without propagation of variance) simulation of the GP ARX and AR model.

Syntax

```
[y, s2] = simulARXnaive(hyp, inf, mean, cov, lik, input, target, test, lag);
```

Description

Input:

- hyp ... column vector of hyperparameters
- inf ... function specifying the inference method
- cov ... prior covariance function (see below)
- mean ... prior mean function
- lik ... likelihood function
- input ... input part of the training data, NxD matrix
- target ... output part of the training data (ie. target), Nx1 vector
- test ... matrix construct_ARXsimul_input.m, kxD matrix,
- lag ... the order of the model (number of used lagged outputs)

Output:

- y ... mean predicted output
- s2 ... associated variances

See also:

gpx.m, simulGPmcmc, construct_ARXsimul_input.m

Examples:

demo_example_gp_simulation.m

Signature

simulGPtaylorSE

Simulation of the GP model, where the output variance is propagated using Taylor approximation.

Syntax

```
[mu, s2] = simulGPtaylorSE(logtheta, covfunc, input, target, test, lag);
```

Description

Simulation of the GP model, where the output variance is propagated using Taylor approximation, see A. Girard, Approximate Methods for Propagation of Uncertainty with Gaussian Process Models, PhD thesis, 2004. Notes: Currently it can be used only with Gaussian covariance function and with white noise model (sum of covSEard and covNoise).

Uses routine gpTaylorSEard.

Inputs:

- loghteta ... optimized hyperparameters
- covfunc ... specified covariance function, see help covFun for more info
- input ... input part of the training data, NxD matrix
- target ... output part of the training data (ie. target), Nx1 vector
- test ... input matrix for simulation, kxD vector, see construct_simul_input.m for more info
- lag ... the order of the model (number of used lagged outputs)

Outputs:

- mu ... predictive mean when propagating the uncertainty
- s2 ... predictive variance when propagating the uncertainty

The form of the covariance function is C(xˆp,xˆq) = v1*exp(-0.5 * sum_{d=1..D} w_d * (xˆp_d - xˆq_d)ˆ2) + v0\delta_{pq}

Examples:

demo_example_gp_simulation.m

See also:

simulGPnaive.m, gpTaylorSEard.m

Signature

Based on the work of J. Quinonero-Candela and A. Girard.

simulLINexact

Simulation of the GP model, where the output variance is propagated using analytical approximation

Syntax

```
[mu, sig2, m, s2] = simulLINexact(logtheta, covfunc, input, target, xt, lag);
```

Description

Simulation of the GP model, where the output variance is propagated using analytical approximation, see A. Girard, Approximate Methods for Propagation of Uncertainty with Gaussian Process Models, PhD thesis, 2004. Notes: Currently it can be used only with Gaussian linear function and with white noise model (sum of covLINard and covNoise).

Uses routine gpExactLINard.

Inputs:

- loghteta ... optimized hyperparameters
- covfunc ... specified covariance function, see help covFun for more info
- invQ ... inverse of the data covariance matrix
- input ... input part of the training data, NxD matrix
- target ... output part of the training data (ie. target), Nx1 vector
- xt ... input matrix for simulation, kxD vector, see construct_simul_input.m for more info
- lag ... the order of the model (number of used lagged outputs)

Outputs:

- mu ... predictive mean using "naive" approach (doesn't propagate the uncertainty)
- sig2 ... predictive variance using "naive" approach
- m ... predictive mean when propagating the uncertainty
- s2 ... predictive variance when propagating the uncertainty

The form of the covariance function is

k(x^p,x^q) = x^p'*inv(P)*x^q

where the P matrix is diagonal with ARD parameters ell_1^2,...,ell_D^2, where D is the dimension of the input space. The hyperparameters are:

logtheta = [log(ell_1) log(ell_2) . log(ell_D) log(sqrt(s2))]]

Examples:

demo_example_gp_simulation.m

See Also:

simulGPExactSE, gpExactLINard

Signature

gp_initial

Function for finding initial values of hyperparameters with random search.

Syntax

```
function [hyp, flogtheta0] = gpr_initial(inffunc, meanfunc, covfunc, likfunc, input, target,
```

Description

Returns best set of n random sets of hyperparameter values. As score it uses a log marginal likelihood.

Input:

- covfunc ... specified covariance function, see help covfun for more info
- input ... input part of the training data, NxD matrix
- target ... output part of the training data (ie. target), Nx1 vector
- bounds ... bounds values of hyperaparameters, vector [min, max] if all hyperparameters have the same bounds, otherwise vector of two hyperparameter structs [hyp_min, hyp_max][optional]
- islog ... indicates if random values should be in log-scale [optional, default = 1]
- npop ... number of population (sets of hyperparameters) [optional]
- meanfunc ... prior mean function - if not set meanZero is used
- likfunc ... likelihood function - if not set likGauss is used
- inf ... function specifying the inference method - if not set infExact is used

Output:

- hyp ... struct of initial hyperparameters
- flogtheta0 ... log marginal likelihood of initial hyperparameters

Examples:

demo_example_gp_training.m

See Also:

likelyhood, gp.m, minimize, covFunctions, likFunctions, meanFunctions, infMethod

Signature

- **Copyright(C)** 2010 by Dejan Petelin (2010-05-03).

trainDEgp

Minimize a multivariate function using differential evolution.

Syntax

```
[X, fX, i] = trainDEgp(X, f, itermax, P1, P2, P3, ... );
```

Description

Minimization of a user-supplied function using the differential with respect to X(1:dim), evolution (DE) algorithm. DE works best if [minbound, maxbound] covers the region where the global minimum is expected. DE is also somewhat sensitive to the choice of the stepsize F. A good initial guess is to choose F from interval [0, 2], e.g. 1.0. CR, the crossover probability constant from interval [0, 1] helps to maintain the diversity of the population. Only separable problems do better with CR close to 0. If the parameters are correlated, high values of CR work better. The reverse is true for no correlation.

The number of population members npop is also not very critical. A good initial guess is 10*dim. Depending on the difficulty of the problem npop can be lower than 10*dim or must be higher than 10*dim to achieve convergence.

minimize_de is a vectorized variant of DE which, however, has a property which differs from the original version of DE: the random selection of vectors is performed by shuffling the population array. Hence a certain vector can't be chosen twice in the same term of the perturbation expression. Due to the vectorized expressions minimize_de executes fairly fast in MATLAB's interpreter environment.

Input:

- X ... initial guess; may be of any type, including struct and cell array
- f ... the name or pointer to the function to be minimized. The function f must return the value of the function.
- itermax ... number of iterations.
- P1, P2, ... parameters are passed to the function f.

Output:

- X ... the returned solution
- fX ... vector of function values indicating progress made
- i ... number of iterations (generations) used at termination.

Examples:

See also:

Signature

- **Author**: Dejan Petelin, based on an algorithm by Kenneth Price and Rainer Storm

Note: This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 1, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details. A copy of the GNU General Public License can be obtained from the Free Software Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.

trainOEgp

Function for optimization (training) of the GP model hyperparameters

Syntax

```
function [logtheta, flogtheta] = trainOEgp(covfunc, input, target, logtheta0, lag, simf, mcm
```

Description

Function for optimization (training) of the GP model hyperparameters based on the training data via Maximum Likelihood (ML). Uses routines gpr and minimize. Based on the work of C.E.Rasmussen.

Inputs:

- covfunc ... specified covariance function, see help covFun for more info
- input ... input part of the training data, NxD matrix
- target ... output part of the training data (ie. target), Nx1 vector
- logtheta0 ... intial values of hyperparameters (optional)
- lag ... the order of the model (number of used lagged outputs)
- simf ... name of function for simulation (optional)
- mcmcn ... number of samples for MCMC simulation (optional)

Outputs:

- logtheta ... optimized hyperparameters
- flogtheta ... minus log likelihood for the different runs (init. to 0)
- i ... number of iterations needed for the last optimization

Examples:

demo_example_gp_training.m

See Also:

gp, minimize, covFunctions, trainlgmp

Signature

traingp

Function for optimization (training) of the GP model hyperparameters

Syntax

```
function [hyp, flogtheta, i] = traingp(hyp, inf, mean, cov, lik, x, y);
```

Description

Function for optimization (training) of the GP model hyperparameters based on the training data via Maximum Likelihood (ML). Uses routines gp and minimize. Based on the work of C.E.Rasmussen.

Inputs:

- hyp ... struct of initial hyperparameters
- inf ... function specifying the inference method
- cov ... prior covariance function (see below)
- mean ... prior mean function
- lik ... likelihood function
- x ... n by D matrix of training inputs
- y ... column vector of length n of training targets

Output:

- hyp ... optimized hyperparameters
- flogtheta ... minus log likelihood for the different runs (init. to 0)
- i ... number of iterations needed for the last optimization

Examples:

demo_example_gp_training

See Also:

gp, minimize, trainlgmp, covFunctions, infMethods, likFunctions, meanFunctions

Signature

add_noise_to_vector

add_noise_to_vector - Function adds white Gaussian noise to vector given

Syntax

```
[x_noise,x_no_noise] = add_noise_to_vector(x_no_noise, noise_std)
```

Description

as a parameter.

It is used to get noisy data from simulation noise-free data. Function uses randn (internal Matlab command) to obtain sampled Gaussian distribution.

Usage:

[x_noise,x_no_noise] = add_noise_to_vector(x_no_noise, noise_std)

where:

x_no_noise vector of noise-free data noise_std standard deviation of white Gaussian noise

x_noise noisy data x_no_noise original data

See also: randn

Examples: demo_example_gp_data.m

construct matrix input for simulation of ARX model

```
function xt = construct_ARXsimul_input(lag,y,u)
```

Syntax

```
function xt = construct_ARXsimul_input(lag,y,u);
```

Description

Function contructs matrix for GP model simulation from starting values of output of the system. Shape of the matrix: xt = [y(1)... y(lag); y(2)... 0 ;

```
     0   ... 0     ];
```

Inputs: lag .. order of the system y .. output necessary for starting values of system state (=output), x0 = [y(1) ... y(lag)] u .. system input signal u, u = [u(1) ... u(end)]' (ignored for AR models) Outputs: xt .. matrix used as the input to simulation rutines

See Also

SIMUL02NAIVE, SIMUL00EXACT, etc.

```
n=length(y);
x0=y(1:lag);

if nargin==2
    len = n - lag + 1;
    % matrix
    xt = zeros(len, lag);
    % y part
    for i=1:lag
        xt(i,1:lag-i+1) = x0(i:end);
    end
else
    if(size(u,1)<size(u,2))
        u = u';
    end

    len = length(u) - lag + 1;
    % matrix
    xt = zeros(len, 2*lag);
    % y part
    for i=1:lag
```

```
        xt(i,1:lag-i+1) = x0(i:end);
    end
    % u part
    for i = 1:lag
        xt(:,lag+i) = u(i:end-lag+i);
    end
end
return
```

construct matrix input for training

```
function [target,tinput] = construct_ARXtrain_input(lag,y,u)
```

Syntax

```
function [target,input] = construct_ARXtrain_input(lag,y,u);
```

Description

Function contructs matrix of regressors for GP model training from input and output signals of the system. Shape of the matrix: tinput = [y(1) ... y(lag) u(1) ... u(lag); y(2) ... y(lag+1) u(2) ... u(lag+1);

```
y(end-lag) ... y(end-1) u(end-lag) ... u(end-1)];
```

If there is no u input, the obtained output variables are for training AR model.

tinput = [y(lag+1) y(lag+2) ... y(end)]';

Inputs: lag .. order of the system u .. system input signal u, u = [u(1) ... u(end)]' y .. system output signal y, y = [y(1) ... y(end)]' Outputs: target.. target vector for the training routines tinput .. matrix of regressors for the training rutines

Examples

demo_example_gp_simulation.m

See Also

SIMUL02NAIVE, SIMUL00EXACT, etc.

```
if nargin==2
    if(size(y,1)<size(y,2))
        y = y';
    end

    len = length(y) - lag + 1;
    % input matrix

    % y part
    for i=1:lag
```

```
        tinput(:,i) = y(i:end-lag+i-1);
    end
else
    if(size(u,1)<size(u,2))
        u = u';
    end
    if(size(y,1)<size(y,2))
        y = y';
    end

    len = length(u) - lag + 1;
    % input matrix

    % y part
    for i=1:lag
        tinput(:,i) = y(i:end-lag+i-1);
    end
    % u part
    for i = 1:lag
        tinput(:,lag+i) = u(i:end-lag+i-1);
    end
end
% target

target=y(lag+1:end);
```

construct matrix input for simulation of FIR model

```
function xt = construct_FIRsimul_input(lag,u)
```

Syntax

```
function xt = construct_FIRsimul_input(lag,u);
```

Description

Function contructs matrix for GP model simulation from starting values of output of the system. Shape of the matrix: xt = [u(1)... u(lag); u(2)... 0 ;

```
    0   ... 0     ];
```

Inputs: lag .. order of the system u .. output necessary for starting values of system state (=output), x0 = [u(1) ... u(lag)] models)

Outputs: xt .. matrix used as the input to simulation rutines

See Also

SIMUL02NAIVE, SIMUL00EXACT, etc.

```
n=length(u);
x0=u(1:lag);

len = n - lag + 1;
% matrix
xt = zeros(len, lag);
for i=1:lag
    xt(i,1:lag-i+1) = x0(i:end);
end
```

construct matrix input for training

```
function [target,tinput] = construct_FIRtrain_input(lag,u)
```

Syntax

```
function [target,input] = construct_FIRtrain_input(lag,u);
```

Description

Function contructs matrix of regressors for GP model training from input and output signals of the system. Shape of the matrix: tinput = [u(1) ... u(lag); u(2) ... u(lag+1);

```
u(end-lag) ... u(end-1);]
```

Inputs: lag .. order of the system u .. system input signal u, u = [u(1) ... u(end)]' Outputs: target.. target vector for the training routines tinput .. matrix of regressors for the training rutines

Examples

demo_example_gp_simulation.m

See Also

SIMUL02NAIVE, SIMUL00EXACT, etc.

```
if(size(y,1)<size(u,2))
    u = u';
end

len = length(u) - lag + 1;

% input matrix
for i=1:lag
    tinput(:,i) = u(i:end-lag+i-1);
end

% target
target=u(lag+1:end);
```

eval_func

```
function [ varargout ] = eval_func( func, varargin )
```

Syntax

Description

```
Robust method to evaluate covariance, mean and likelyhood functions in
the context of gpml toolbox. Input can be a function handle, string or
cell array.
```

Inputs:

func .. Fuction handle, funtion name (string) or cell array, desribing a function
to be evaluated varargin .. Function parameters

```
if ischar(func) || isa(func, 'function_handle'), func = {func}; end  % make cell

varargout = {feval(func{:}, varargin)}; %evaluate function

end
```

likelihood

```
function [ml] = likelihood(hyp,covfunc,input,target, meanfunc, likfunc, inf)
```

Syntax

```
function ml = likelihood(logtheta,covfunc,input,target)
```

Description

Function for calculating the negative log marginal likelihood of given hyperparameters, covariance function and data. Based on the work of C.E.Rasmussen.

Inputs: hyp .. hyperparameter struct(s), row vector of structs covfunc .. specified covariance function, see help covFun for more info input .. input part of the training data, NxD matrix target .. output part of the training data (ie. target), Nx1 vector

Outputs: ml .. negative log marginal likelihood(s), vector

Examples

gp_initial.m

See Also

GPR, MINIMIZE, COVFUN

```
%allocate
ml=zeros(size(hyp));

for i = 1 : size(hyp,2)
  try

    mlt = gp(hyp(i), inf, meanfunc, covfunc, likfunc, input, target);
  catch
    mlt = +Inf;
  end

  % stability
  if (~isfinite(mlt) || isnan(mlt) || ~isreal(mlt))
    mlt = +Inf;
  end
  ml(i) = mlt;
```

```
end
```

lipschit

LIPSCHIT

Syntax

```
[OrderIndexMat]=lipschit(U,Y0,mvec,nvec)
```

Description

——— Determine the lag space.

Given a set of corresponding inputs and outputs the function calculates a matrix of indices, which can be helpful when trying to determine a proper lag space structure (m and n) before identifying a model of a dynamic system: y(t) = f(y(t-1),...,y(t-n), u(t-1),..., u(t-m))

An insufficient lag space structure leads to a large index. While increasing the lag space, the index will decrease until a sufficiently large lag space structure is reached. Increasing the lag space beyond this will not reduce the index significantly. In other words: look for theknee-point where the order index flattens out.

NB: So far, the function works for SISO systems only.

Please ignore the message "Divide by zero"

CALL: [OrderIndexMat]=lipschit(U,Y,m,n)

m is a vector specifying which input lag spaces to investigate and n is ditto for the output. If one is only interested in the order index for one particular choice of lag structure, n and m are specified as scalars, and only the order index is returned. In the more general case, where one or both are vectors, the function will also produce one or two plots.

Examples of some typical special cases:

o NNFIR model structure expected: m=[1:20]; n=0;

o Time series: U=[]; m=0;

o Check unly n=m: m=[1:5]; n=m;

INPUTS: U - Sequence of inputs (row vector) Y - Sequence of outputs (row vector) m - Vector specifying the input lag spaces to investigate n - Vector specifying the ouput lag spaces to investigate

OUPUTS: OrderIndexMat - A matrix containing the order indices for each combination of elements in the vectors m and n. The number of rows corresponds to the number of elements in m, while the number of columns corresponds to the number of elements in n

REFERENCE: X. He & H. Asada: "A New Method for Identifying Orders of Input-Output Models for Nonlinear Dynamic Systems" Proc. of the American Control Conf., S.F., California, 1993

SEE ALSO: Use function 'dscale' to scale the data.

Programmed by Magnus Norgaard, IAU/IMM, Technical Univ. of Denmark LastEdit* **Date:** July 16, 1996

———- Generate matrix of regressors, PHI ———

loss

```
function [ae, se, lpd, mrse] = loss(tt, mu, s2)
```

Syntax

```
function [ae, se, lpd, mrse] = loss2(tt, mu, s2);
```

Description

Function computes several frequently used perforamnce values: - mean absolute error AE - mean squared error SE - minus log-predicted density error LPD - mean relative squared error MRSE Notes: (1) It can be used with one-step-ahead prediction or simulation results. (2) Can be expanded with other performance values. Inputs: tt .. target output values (=system output) y .. predicted output values (=model output) s2 .. predicted output variance Outputs: ae .. mean absolute error se .. mean squared error lpd .. minus log-predicted density error mrse .. mean relative square error

```
if size(tt) ~= size(mu)
  mu = mu';
end


ae = mean(abs(tt-mu));
se = mean((tt-mu).^2);

% mean relative squared error MRSE
mrse = sqrt(sum((tt-mu).^2)/sum(tt.^2));

if(nargin==2)
    lpd = [];
    disp('loss: input s2 is undefined');
    printout(ae,se);
    printoutmrse(mrse);
    return;
elseif (isempty(s2))
    lpd = [];
    printout(ae,se);
    printoutmrse(mrse);
    return;
end

if size(tt) ~= size(s2)
  s2 = s2';
end
```

```
lpd = 0.5*mean(log(2*pi) + log(s2) + (tt-mu).^2./s2);
printout(ae,se);
printoutlpd(lpd);
printoutmrse(mrse);
return;

function []=printout(ae,se);
disp(strcat('AE = ',num2str(ae)));
disp(strcat('SE = ',num2str(se)));

function []=printoutlpd(lpd);
disp(strcat('LD = ',num2str(lpd)));

function []=printoutmrse(mrse);
disp(strcat('MRSE = ',num2str(mrse)));
```

mcmc_test_pdfs

```
function [] = mcmc_test_pdfs(MM,VV,desired_steps)
```

Syntax

```
function [] = mcmc_test_pdfs(MM,VV,desired_steps);
```

Description

Function test and plots the Gaussian mixture given my MM and VV in desired simulation steps Inputs: MM .. mean values of predicted distributions, matrix ksteps times Nsamples VV .. associated variances of predicted distributions desired_steps .. vector with the indices of steps, where we want to test the distributions Outputs: .. plots the figures of the distributions in the desired steps

See Also

SIMUL00EXACT

```
% XX,YY ... matrices for storing the GP model's pdfs
% XX(:,k) ... yvalues at time step k
% YY(:,k) ... density values at time step k and coresponding y values in XX(:,k)
% sum(YY(:,k)) = 1 for all k-s

% variables
MM = MM';
VV = VV';

Nsamples = size(MM,1);
mu = mean(MM);
s2 = mean(VV) + mean((MM-repmat(mu,Nsamples,1)).^2);
mu = mu';
s2 = s2';

% "calculation"
STD = sqrt(VV);
Xwide = 200; % resolution = no. of samples between xmin and xmax
Xmin = min(MM-4*STD);
Xmax = max(MM+4*STD);
dX = (Xmax-Xmin)/(Xwide-1);

XX = zeros(Xwide,size(VV,2));
YY = zeros(size(XX));
```

```
% for desired steps
for jj = 1:length(desired_steps)
    kk = desired_steps(jj);
    XX(:,kk) = Xmin(kk):dX(kk):Xmax(kk);
    % ... and for every realisation
    ytemp = zeros(size(YY,1),1);

    for ii=1:Nsamples
        ytemp = normpdf(XX(:,kk), MM(ii,kk), STD(ii,kk));
        YY(:,kk) = YY(:,kk) + ytemp;
    end
    YY(:,kk) = YY(:,kk)/Nsamples;

    yapprox = normpdf(XX(:,kk),mu(kk),sqrt(s2(kk)));

    % plot pdfs
    figure(10000+kk);

    plot(XX(:,kk),YY(:,kk),'Color',[0.6 0.6 0.6],'LineWidth',4);
    hold on;
    plot(XX(:,kk),yapprox,'LineStyle','--','Color',[0 0 1],'LineWidth',4);
     for ii=1:Nsamples
        ytemp = normpdf(XX(:,kk), MM(ii,kk), STD(ii,kk));
        plot(XX(:,kk),ytemp,'Color',[0.9 0.9 0.9]);
    end
    plot(XX(:,kk),YY(:,kk),'Color',[0.6  0.6 0.6],'LineWidth',4);
    plot(XX(:,kk),yapprox,'LineStyle','--','Color',[0 0 1],'LineWidth',4);
    hold off;
    grid;
    legend('true','GP approx','samples')


    title(strcat(['test mcmc pdfs, simul step=', num2str(kk)]));


    disp('press sth to continue');
    pause;


end
```

plotgp

```
function [i] = plotgp(i, t, sys, y, std)
```

Syntax

```
function [i] = plotgp(i, t, sys, y, std)
```

Description

Function plots the results of the simulation or one-step-ahead prediction. In the upper 2/3 of the figure the output together with 95% confidence band and target is plotted, in the lower third of the figure the error with corresponding 95% confidence band is plotted. Can be used to plot in new or currently active figure. Inputs: i .. figure handle, if i==0 function plots in current figure t .. time vector (x-axis) sys .. target output y .. predicted output std .. predicted standard deviation Output: i .. figure handle

Examples

demo_example_gp_simulation.m

See Also

PLOTGPY, PLOTGPE

```
% check sizes of vectors
sz = size(t);
if((size(t,1)~=sz(1) | size(sys,1)~=sz(1) |size(y,1)~=sz(1) | size(std,1)~=sz(1))...
        | (size(t,2)~=sz(2) | size(sys,2)~=sz(2) |size(y,2)~=sz(2) | size(std,2)~=sz(2)))
    warning(['figure ', num2str(i), ': vectors: t, tt, y, std must be same size']);
    disp(strcat(['t: ', num2str(size(t))]));
    disp(strcat(['sys: ', num2str(size(sys))]));
    disp(strcat(['y: ', num2str(size(y))]));
    disp(strcat(['std: ', num2str(size(std))]));
    out = -1;
    return;
end

ix_plot = 1:length(t);
% reduce vector if borders are wanted in figure,
% e.g. ixplot = 2:length(t)-1;

xfill = [t(ix_plot); flipdim(t(ix_plot),1)];
```

```
yfill = [y(ix_plot)+2*std(ix_plot);flipdim(y(ix_plot)-2*std(ix_plot),1)];


% if i==0 use current axis (figure)
if (i~=0)
figure(i);
end

% upper part of figure: y
subplot(3,1,1:2)
fill(xfill, yfill, [7 7 7]/8, 'EdgeColor', [0 0 0]/8);
hold on
plot(t,y, 'k-', 'LineWidth',1);
plot(t,sys, 'k--', 'LineWidth',2);
%plot(t,y-2*std, 'LineWidth',2, 'Color',[0.7 0.7 0.7]);
%plot(t,y+2*std, 'LineWidth',2, 'Color',[0.7 0.7 0.7]);
hold off
grid on
xlabel('t');
ylabel('y');
title('GP model simulation')
legend('\mu \pm 2\sigma', '\mu', 'system','Location','NorthEast');
AX=axis;
AX(1:2)=[t(1) t(end)];
axis(AX);

% lower part of figure: e
subplot(3,1,3)
xfill = [t(ix_plot); flipdim(t(ix_plot),1)];
yfill = [2*std(ix_plot);zeros(size(std(ix_plot)))];
fill(xfill, yfill, [7 7 7]/8, 'EdgeColor', [0 0 0]/8);
% plot(t,2*std, 'LineWidth',2, 'Color',[0.7 0.7 0.7]);
hold on
plot(t,abs(y-sys), 'k-', 'LineWidth',1);
hold off
legend('2\sigma', '|e|', 'Location','NorthEast');
grid
xlabel('t');
ylabel('e');
AX=axis;
AX(1:2)=[t(1) t(end)];
axis(AX);



return
```

plotgpe

```
function [i] = plotgpe(i, t, sys, y, std)
```

Syntax

```
function [i] = plotgpe(i, t, sys, y, std)
```

Description

Function plots the error with corresponding 95% confidence band. Can be used to plot in new or currently active figure. Inputs: i .. figure handle, if i==0 function plots in current figure t .. time vector (x-axis) sys .. target output y .. predicted output std .. predicted standard deviation Output: i .. figure handle

Examples

demo_example_gp_simulation.m

See Also

PLOTGP, PLOTGPY

```
% check sizes of vectors
sz = size(t);
if((size(t,1)~=sz(1) | size(sys,1)~=sz(1) |size(y,1)~=sz(1) | size(std,1)~=sz(1))...
        | (size(t,2)~=sz(2) | size(sys,2)~=sz(2) |size(y,2)~=sz(2) | size(std,2)~=sz(2)))
    warning(['figure ', num2str(i), ': vectors: t, tt, y, std must be same size']);
    disp(strcat(['t: ', num2str(size(t))]));
    disp(strcat(['sys: ', num2str(size(sys))]));
    disp(strcat(['y: ', num2str(size(y))]));
    disp(strcat(['std: ', num2str(size(std))]));
    out = -1;
    return;
end

ix_plot = 1:length(t);
% reduce vector if borders are wanted in figure,
% e.g. ixplot = 2:length(t)-1;

xfill = [t(ix_plot); flipdim(t(ix_plot),1)];
yfill = [y(ix_plot)+2*std(ix_plot);flipdim(y(ix_plot)-2*std(ix_plot),1)];
```

```
% if i==0 use current axis (figure)
if (i~=0)
figure(i);
end

xfill = [t(ix_plot); flipdim(t(ix_plot),1)];
yfill = [2*std(ix_plot);zeros(size(std(ix_plot)))];
fill(xfill, yfill, [7 7 7]/8, 'EdgeColor', [7 7 7]/8);
% plot(t,2*std, 'LineWidth',2, 'Color',[0.7 0.7 0.7]);
hold on
plot(t,abs(y-sys), 'k-', 'LineWidth',1);
hold off
legend('2\sigma', '|e|', 'Location','NorthEast');
grid
xlabel('t');
ylabel('y');
AX=axis;
AX(1:2)=[t(1) t(end)];
axis(AX);



return
```

plotgpy

```
function [i] = plotgpy(i, t, sys, y, std)
```

Syntax

```
function [i] = plotgpy(i, t, sys, y, std)
```

Description

Function plots the results of the simulation or one-step-ahead prediction: target and output together with 95% confidence band. Inputs: i .. figure handle, if i==0 function plots in current figure t .. time vector (x-axis) sys .. target output y .. predicted output std .. predicted standard deviation Output: i .. figure handle

Examples

demo_example_gp_simulation.m

See Also

PLOTGP, PLOTGPE

```
% check sizes of vectors
sz = size(t);
if((size(t,1)~=sz(1) | size(sys,1)~=sz(1) |size(y,1)~=sz(1) | size(std,1)~=sz(1))...
        | (size(t,2)~=sz(2) | size(sys,2)~=sz(2) |size(y,2)~=sz(2) | size(std,2)~=sz(2)))
    warning(['figure ', num2str(i), ': vectors: t, tt, y, std must be same size']);
    disp(strcat(['t: ', num2str(size(t))]));
    disp(strcat(['sys: ', num2str(size(sys))]));
    disp(strcat(['y: ', num2str(size(y))]));
    disp(strcat(['std: ', num2str(size(std))]));
    out = -1;
    return;
end

ix_plot = 1:length(t);
% reduce vector if borders are wanted in figure,
% e.g. ixplot = 2:length(t)-1;

xfill = [t(ix_plot); flipdim(t(ix_plot),1)];
yfill = [y(ix_plot)+2*std(ix_plot);flipdim(y(ix_plot)-2*std(ix_plot),1)];
```

```
% if i==0 use current axis (figure)
if (i~=0)
figure(i);
end

fill(xfill, yfill, [7 7 7]/8, 'EdgeColor', [7 7 7]/8);
hold on
plot(t,y, 'k-', 'LineWidth',1);
plot(t,sys, 'k--', 'LineWidth',2);
%plot(t,y-2*std, 'LineWidth',2, 'Color',[0.7 0.7 0.7]);
%plot(t,y+2*std, 'LineWidth',2, 'Color',[0.7 0.7 0.7]);
hold off
grid on
xlabel('t');
ylabel('y');
title('GP model simulation')
legend('\mu \pm 2\sigma', '\mu', 'system','Location','NorthEast');
AX=axis;
AX(1:2)=[t(1) t(end)];
axis(AX);




return
```

postmnmx

POSTMNMX Postprocesses data which has been preprocessed by PREMNMX.

Syntax

```
[p,t] = postmnmx(pn,minp,maxp,tn,mint,maxt) [p] = postmnmx(pn,minp,maxp);
```

Description

POSTMNMX postprocesses the network training set which was preprocessed by PREMNMX. It converts the data back into unnormalized units.

POSTMNMX takes these inputs, PN - RxQ matrix of normalized input vectors MINP- Rx1 vector containing minimums for each P MAXP- Rx1 vector containing maximums for each P TN - SxQ matrix of normalized target vectors MINT- Sx1 vector containing minimums for each T MAXT- Sx1 vector containing maximums for each T and returns, P - RxQ matrix of input (column) vectors. T - SxQ matrix of target vectors.

Examples

In this example we normalize a set of training data with PREMNMX, create and train a network using the normalized data, simulate the network, unnormalize the output of the network using POSTMNMX, and perform a linear regression between the network outputs (unnormalized) and the targets to check the quality of the network training.

p = [-0.92 0.73 -0.47 0.74 0.29; -0.08 0.86 -0.67 -0.52 0.93]; t = [-0.08 3.4 -0.82 0.69 3.1]; [pn,minp,maxp,tn,mint,maxt] = premnmx(p,t); net = newff(minmax(pn),[5 1],{'tansig' 'purelin'},'trainlm'); net = train(net,pn,tn); an = sim(net,pn); [a] = postmnmx(an,mint,maxt); [m,b,r] = postreg(a,t);

Algorithm

p = 0.5(pn+1)*(maxp-minp) + minp;

See also PREMNMX, PREPCA, POSTSTD.

- *Copyright *1992-2002 The MathWorks, Inc.

postmnmxvar

```
function s2 = postmnmxvar(s2n,min,max)
```

Syntax

```
function s2 = postmnmxvar(s2n,min,max)
```

Description

Postprocesses predicted variance for data which has been preprocessed by PREMNMX. Inputs: s2n .. normalized predicted variance min .. minimum of preprocessed data max .. maximum of preprocessed data Output: s2 .. post-processed predicted variance

See Also

POSTMNMX, PREMNMMX

```
stdn = sqrt(s2n);
std = stdn*(max-min)/2;
s2 = std.^2;

return
```

premnmx

PREMNMX Preprocesses data so that minimum is -1 and maximum is 1.

Syntax

```
[pn,minp,maxp,tn,mint,maxt] = premnmx(p,t) [pn,minp,maxp] = premnmx(p);
```

Description

PREMNMX preprocesses the network training set by normalizing the inputs and targets so that they fall in the interval [-1,1].

PREMNMX(P,T) takes these inputs, P - RxQ matrix of input (column) vectors. T - SxQ matrix of target vectors. and returns, PN - RxQ matrix of normalized input vectors MINP- Rx1 vector containing minimums for each P MAXP- Rx1 vector containing maximums for each P TN - SxQ matrix of normalized target vectors MINT- Sx1 vector containing minimums for each T MAXT- Sx1 vector containing maximums for each T

Examples

Here is how to normalize a given data set so that the inputs and targets will fall in the range [-1,1].

p = [-10 -7.5 -5 -2.5 0 2.5 5 7.5 10]; t = [0 7.07 -10 -7.07 0 7.07 10 7.07 0]; [pn,minp,maxp,tn,mint,maxt] = premnmx(p,t);

If you just want to normalize the input:

[pn,minp,maxp] = premnmx(p);

Algorithm

pn = 2*(p-minp)/(maxp-minp) - 1;

See also PRESTD, PREPCA, POSTMNMX, TRAMNMX.

- *Copyright *1992-2002 The MathWorks, Inc.

sig_prbs

```
function y = sig_prbs(n,m)
```

Syntax

```
function y = sig_prbs(n,m)
```

Description

Function generates pseudo-random binary signal (PRBS) with uniform amplitude, see D. Matko: Identifikacije, 1998, FE Ljubljana, p. 43-45 Inputs: n .. legth of the register m .. number of samples per tact of PRBS (the width of the narrowest impulse at sampling steps) Output: y .. signal values

```
ind1=[0 1 2 3 3 5 4 5 5 7 10];
ind2=[0 2 3 4 5 6 7 8 9 10 11 15 20 40 100];
z=ones(n,1);
for j=1:m
  y(j,1)=z(n);
end
for i=2:2^n-1
  ztemp=z(ind1(n))+z(ind2(n));
  for j=0:n-2
    z(n-j)=z(n-j-1);
  end
  z(1)=ztemp;
  if z(1)==2
     z(1)=0;
  end
  for j=(i-1)*m+1:m*i
    y(j,1)=2*z(n)-1;
  end
end
```

sig_prs_minmax

```
function y = sig_prs_minmax(n, ksw, min, max)
```

Syntax

```
function y = sig_prs_minmax(n, ksw, min, max)
```

Description

```
Function, which generates "random" signal with length n and uniform
distribution of amplitudes between min and max. ksw is the minimum width
(duration in steps) of the signal at particular amplitude.
Inputs:
n .. length of the signal
ksw .. tact (minimum duration of signal at constant value)
min .. minimum amplitude
max .. maximum amplitude
Output:
y .. signal values
```

Examples

demo_example_gp_data.m

```
k = ceil(n/ksw);

rand('state',sum(100*clock));
y = rand(1,k);

y = min + y*(max-min);

y = repmat(y, ksw, 1);
y = reshape(y, prod(size(y)),1);
```

test_hyp_format

```
function [ output, msg ] = test_hyp_format( hyp, D, covfunc, meanfunc, likfunc, throw_error)
```

Syntax

```
[ output ] = test_hyp_format( hyp, D, covfunc, likfunc, meanfunc);
```

Description

Function checks whether the hyperparameters' struct is in right format

Inputs: hyp .. struct of hyperparameters to be checked D .. size of the input space covfunc .. specified covariance function, see help covfun for more info meanfunc .. prior mean function - if not set meanZero is used likfunc .. likelihood function - if not set likGauss is used throw_error .. determines wether the error should be thrown (1) or passed forward (0). Default value is 1.

Outputs: output .. 1 if everything is ok, 0 on error mgs .. error message

```
output = 1;
msg = '';

% input validation
if nargin < 3
  error('Too few parameters are given.'); %
end
if nargin < 4
    %set default meanfunc
    meanfunc = @meanZero;
end
if nargin < 5
    %set default likfunc
    likfunc = @likGauss;
end
if nargin < 6
    % deafaoul throw_error
    throw_error = 1;
end

if isstruct(hyp) && isfield(hyp,'cov') && isfield(hyp,'lik') && isfield(hyp,'mean')
    hyp_cov_count = eval(eval_func(covfunc)); % number of covariance hyperparameters
    if length(hyp.cov) ~= hyp_cov_count
        output = 0;
        msg = ['hyp.cov (size: ',num2str(length(hyp.cov)),') should consist of ', num2str(hy
    end
```

```
    hyp_lik_count = eval(eval_func(likfunc)); % number of likelihood hyperparameters
    if length(hyp.lik) ~= hyp_lik_count
        output = 0;
        msg = [msg, ' hyp.lik (size: ',num2str(length(hyp.lik)),') should consist of ', num2
    end

    hyp_mean_count = eval(eval_func(meanfunc)); % number of mean hyperparameters
    if length(hyp.mean) ~= hyp_mean_count
        output = 0;
        msg = [msg,  ' hyp.mean (size: ',num2str(length(hyp.mean)),') should consist of ', n
    end

else
  output = 0;
  msg = 'hyp should be a struct with fields hyp.cov hyp.lik and hyp.mean.';
end

%throw error if needed
if throw_error && output == 0
    error(msg);
end

end
```