```
function [varargout] = MTGP(hyp, inf, mean, cov, lik, x, y, xs, ys)
% 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. Usage:
%
%   training: [nlZ dnlZ          ] = gp(hyp, inf, mean, cov, lik, x, y);
% prediction: [ymu ys2 fmu fs2   ] = gp(hyp, inf, mean, cov, lik, x, y, xs);
%         or: [ymu ys2 fmu fs2 lp] = gp(hyp, inf, mean, cov, lik, x, y, xs, ys);
%
% where:
%
%   hyp      column vector 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
%
%   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
%
```

\inf \mean, \cov \lik

posterior = prior x likelihood / marginal likelihood

marginal likelihood = \ prior x likelihood

$n_* \times 1$

```
%   post      struct representation of the (approximate) posterior
%             3rd output in training mode or 6th output in prediction mode
%             can be reused in prediction mode gp(.., cov, lik, x, post, xs,..)
%
% See also covFunctions.m, infMethods.m, likFunctions.m, meanFunctions.m.
%
% Copyright (c) by Carl Edward Rasmussen and Hannes Nickisch, 2013-01-21
if nargin<7 || nargin>9
  disp('Usage: [nlZ dnlZ          ] = gp(hyp, inf, mean, cov, lik, x, y);')
  disp('   or: [ymu ys2 fmu fs2   ] = gp(hyp, inf, mean, cov, lik, x, y, xs);')
  disp('   or: [ymu ys2 fmu fs2 lp] = gp(hyp, inf, mean, cov, lik, x, y, xs, ys);')
  return
end

if isempty(mean), mean = {@meanZero}; end                     % set default mean
if ischar(mean) || isa(mean, 'function_handle'), mean = {mean}; end  % make cell
if isempty(cov), error('Covariance function cannot be empty'); end  % no default
if ischar(cov)  || isa(cov,  'function_handle'), cov  = {cov};  end  % make cell
cov1 = cov{1}; if isa(cov1, 'function_handle'), cov1 = func2str(cov1); end
if isempty(inf)                                   % set default inference method
  if strcmp(cov1,'covFITC'), inf = @infFITC; else inf = @infExact; end
else
  if iscell(inf), inf = inf{1}; end                      % cell input is allowed
  if ischar(inf), inf = str2func(inf); end        % convert into function handle
end
if strcmp(cov1,'covFITC')                           % only infFITC* are possible
  if isempty(strfind(func2str(inf),'infFITC')==1)
    error('Only infFITC* are possible inference algorithms')
  end
end                            % only one possible class of inference algorithms
if isempty(lik),  lik = {@likGauss}; end                       % set default lik
if ischar(lik)  || isa(lik,  'function_handle'), lik  = {lik};  end  % make cell
if iscell(lik), likstr = lik{1}; else likstr = lik; end
if ~ischar(likstr), likstr = func2str(likstr); end

D = size(x,2);

%% these lines have to be added to be able to use Lab_covCC_chol_nD function
if size(x,2) > 1
    nL = max(x(:,end));
```

```
end

if ~isfield(hyp,'mean'), hyp.mean = []; end        % check the hyp specification
if eval(feval(mean{:})) ~= numel(hyp.mean)
  error('Number of mean function hyperparameters disagree with mean function')
end
if ~isfield(hyp,'cov'), hyp.cov = []; end
if eval(feval(cov{:})) ~= numel(hyp.cov)
  error('Number of cov function hyperparameters disagree with cov function')
end
if ~isfield(hyp,'lik'), hyp.lik = []; end
if eval(feval(lik{:})) ~= numel(hyp.lik)
  error('Number of lik function hyperparameters disagree with lik function')
end

try                                                   % call the inference method
  % issue a warning if a classification likelihood is used in conjunction with
  % labels different from +1 and -1
  if strcmp(likstr,'likErf') || strcmp(likstr,'likLogistic')
    if ~isstruct(y)
      uy = unique(y);
      if any( uy~=+1 & uy~=-1 )
        warning('You try classification with labels different from {+1,-1}')
      end
    end
  end
  if nargin>7   % compute marginal likelihood and its derivatives only if needed
    if isstruct(y)
      post = y;            % reuse a previously computed posterior approximation
    else
      post = inf(hyp, mean, cov, lik, x, y);
    end
  else
    if nargout==1
      [post nlZ] = inf(hyp, mean, cov, lik, x, y); dnlZ = {};
    else
      [post nlZ dnlZ] = inf(hyp, mean, cov, lik, x, y);
    end
  end
catch
```

```
  msgstr = lasterr;
  if nargin > 7, error('Inference method failed [%s]', msgstr); else
    warning('Inference method failed [%s] .. attempting to continue',msgstr)
    dnlZ = struct('cov',0*hyp.cov, 'mean',0*hyp.mean, 'lik',0*hyp.lik);
    varargout = {NaN, dnlZ}; return                    % continue with a warning
  end
end

if nargin==7                                     % if no test cases are provided
  varargout = {nlZ, dnlZ, post};     % report -log marg lik, derivatives and post
else
  alpha = post.alpha; L = post.L; sW = post.sW;
  if issparse(alpha)                        % handle things for sparse representations
    nz = alpha ~= 0;                                   % determine nonzero indices
    if issparse(L), L = full(L(nz,nz)); end       % convert L and sW if necessary
    if issparse(sW), sW = full(sW(nz)); end
  else nz = true(size(alpha,1),1); end                 % non-sparse representation
  if numel(L)==0                         % in case L is not provided, we compute it
    K = feval(cov{:}, hyp.cov, x(nz,:));
    L = chol(eye(sum(nz))+sW*sW'.*K);
  end
  Ltril = all(all(tril(L,-1)==0));               % is L an upper triangular matrix?
  ns = size(xs,1);                                        % number of data points
  nperbatch = 1000;                           % number of data points per mini batch
  nact = 0;                           % number of already processed test data points
  ymu = zeros(ns,1); ys2 = ymu; fmu = ymu; fs2 = ymu; lp = ymu;    % allocate mem
  while nact<ns                      % process minibatches of test cases to save memory
    id = (nact+1):min(nact+nperbatch,ns);               % data points to process
    kss = feval(cov{:}, hyp.cov, xs(id,:), 'diag');                % self-variance
    Ks  = feval(cov{:}, hyp.cov, x(nz,:), xs(id,:));           % cross-covariances
    ms = feval(mean{:}, hyp.mean, xs(id,:));
    N = size(alpha,2);  % number of alphas (usually 1; more in case of sampling)
    Fmu = repmat(ms,1,N) + Ks'*full(alpha(nz,:));          % conditional mean fs|f
    fmu(id) = sum(Fmu,2)/N;                                     % predictive means
    if Ltril           % L is triangular => use Cholesky parameters (alpha,sW,L)
      V  = L'\(repmat(sW,1,length(id)).*Ks);
      fs2(id) = kss - sum(V.*V,1)';                         % predictive variances
    else                % L is not triangular => use alternative parametrisation
      fs2(id) = kss + sum(Ks.*(L*Ks),1)';                   % predictive variances
    end
```

```
    fs2(id) = max(fs2(id),0);   % remove numerical noise i.e. negative variances
    Fs2 = repmat(fs2(id),1,N);     % we have multiple values in case of sampling
    if nargin<9
      [Lp, Ymu, Ys2] = feval(lik{:},hyp.lik,[],Fmu(:),Fs2(:));
    else
      [Lp, Ymu, Ys2] = feval(lik{:},hyp.lik,repmat(ys(id),1,N),Fmu(:),Fs2(:));
    end
    lp(id)  = sum(reshape(Lp, [],N),2)/N;    % log probability; sample averaging
    ymu(id) = sum(reshape(Ymu,[],N),2)/N;          % predictive mean ys|y and ..
    ys2(id) = sum(reshape(Ys2,[],N),2)/N;                          % .. variance
    nact = id(end);          % set counter to index of last processed data point
  end
  if nargin<9
    varargout = {ymu, ys2, fmu, fs2, [], post};        % assign output arguments
  else
    varargout = {ymu, ys2, fmu, fs2, lp, post};
  end
end
```