The GPML Toolbox version 3.2

Carl Edward Rasmussen & Hannes Nickisch

January 21, 2013

Abstract

The GPML toolbox is an Octave 3.2.x and Matlab 7.x implementation of inference and prediction in Gaussian process (GP) models. It implements algorithms discussed in Rasmussen & Williams: *Gaussian Processes for Machine Learning*, the MIT press, 2006 and Nickisch & Rasmussen: *Approximations for Binary Gaussian Process Classification*, JMLR, 2008.

The strength of the function lies in its flexibility, simplicity and extensibility. The function is flexible as firstly it allows specification of the properties of the GP through definition of mean function and covariance functions. Secondly, it allows specification of different inference procedures, such as e.g. exact inference and Expectation Propagation (EP). Thirdly it allows specification of likelihood functions e.g. Gaussian or Laplace (for regression) and e.g. cumulative Logistic (for classification). Simplicity is achieved through a single function and compact code. Extensibility is ensured by modular design allowing for easy addition of extension for the already fairly extensive libraries for inference methods, mean functions, covariance functions and likelihood functions.

This document is a technical manual for a developer containing many details. If you are not yet familiar with the GPML toolbox, the *user documentation* and examples therein are a better way to get started.

Contents

1 Gaussian Process Training and Prediction 3

2 The gp Function 4

3 Inference Methods 8
3.1 Exact Inference . . . 9
3.2 Laplace's Approximation . . . 10
3.3 Expectation Propagation . . . 10
3.4 Variational Bayes . . . 11
3.5 FITC Approximations . . . 11

4 Likelihood Functions 13
4.1 Prediction . . . 13
4.2 Interface . . . 14
4.3 Implemented Likelihood Functions . . . 15
4.4 Usage of Implemented Likelihood Functions . . . 16
4.5 Compatibility Between Likelihoods and Inference Methods . . . 18
4.6 Gaussian Likelihood . . . 18
4.6.1 Exact Inference . . . 19
4.6.2 Laplace's Approximation . . . 19
4.6.3 Expectation Propagation . . . 20
4.6.4 Variational Bayes . . . 20
4.7 Laplace Likelihood . . . 21
4.7.1 Laplace's Approximation . . . 21
4.7.2 Expectation Propagation . . . 21
4.7.3 Variational Bayes . . . 23
4.8 Student's t Likelihood . . . 24
4.8.1 Laplace's Approximation . . . 24
4.9 Cumulative Logistic Likelihood . . . 24
4.9.1 Laplace's Approximation . . . 25

5 Mean Functions 26
5.1 Interface . . . 26
5.2 Implemented Mean Functions . . . 27
5.3 Usage of Implemented Mean Functions . . . 27

6 Covariance Functions 29
6.1 Interface . . . 29
6.2 Implemented Covariance Functions . . . 31
6.3 Usage of Implemented Covariance Functions . . . 31

1 Gaussian Process Training and Prediction

The `gpml` toolbox contains a single user function `gp` described in section 2. In addition there are a number of supporting structures and functions which the user needs to know about, as well as an internal convention for representing the posterior distribution, which may not be of direct interest to the casual user.

Inference Methods An inference method is a function which computes the (approximate) posterior, the (approximate) negative log marginal likelihood and its partial derivatives w.r.t.. the hyperparameters, given a model specification (i.e., GP mean and covariance functions and a likelihood function) and a data set. Inference methods are discussed in section 3. New inference methods require a function providing the desired inference functionality and possibly extra functionality in the likelihood functions applicable.

Hyperparameters The hyperparameters is a struct controlling the properties of the model, i.e.. the GP mean and covariance function and the likelihood function. The hyperparameters is a struct with the three fields `mean`, `cov` and `lik`, each of which is a vector. The number of elements in each field must agree with number of hyperparameters in the specification of the three functions they control (below). If a field has no entries it can either be empty or non-existent.

Likelihood Functions The likelihood function specifies the form of the likelihood of the GP model and computes terms needed for prediction and inference. For inference, the required properties of the likelihood depend on the inference method, including properties necessary for hyperparameter learning, section 4.

Mean Functions The mean function is a cell array specifying the GP mean. It computes the mean and its derivatives w.r.t.. the part of the hyperparameters pertaining to the mean. The cell array allows flexible specification and composition of mean functions, discussed in section 5. The default is the zero function.

Covariance Functions The covariance function is a cell array specifying the GP covariance function. It computes the covariance and its derivatives w.r.t.. the part of the hyperparameters pertaining to the covariance function. The cell array allows flexible specification and composition of covariance functions, discussed in section 6.

Inference methods, see section 3, compute (among other things) an approximation to the posterior distribution of the latent variables f_i associated with the training cases, $i = 1, \ldots, n$. This approximate posterior is assumed to be Gaussian, and is communicated via a struct `post` with the fields `post.alpha`, `post.s` and `post.L`. Often, starting from the Gaussian prior $p(f) = \mathcal{N}(f|m, K)$ the approximate posterior admits the form

$$q(\mathbf{f}|\mathcal{D}) \;=\; \mathcal{N}\big(f|\mu = m + K\alpha,\; V = (K^{-1} + W)^{-1}\big), \;\text{ where }\; W \text{ diagonal with } \; W_{ii} = s_i^2. \qquad (1)$$

In such cases, the entire posterior can be computed from the two vectors `post.alpha` and `post.s`; the inference method may optionally also return $L = \text{chol}(\text{diag}(\mathbf{s})K\,\text{diag}(\mathbf{s}) + I)$.
If on the other hand the posterior doesn't admit the above form, then `post.L` returns the matrix $L = -(K+W^{-1})^{-1}$ (and `post.s` is unused). In addition, a sparse representation of the posterior may be used, in which case the non-zero elements of the `post.alpha` vector indicate the active entries.

2 The gp Function

The gp function is typically the only function the user would directly call.

4a ⟨*gp.m* 4a⟩≡

```
function [varargout] = gp(hyp, inf, mean, cov, lik, x, y, xs, ys)
⟨gp function help 4b⟩
⟨initializations 5b⟩
⟨inference 6b⟩
if nargin==7                                         % if no test cases are provided
  varargout = {nlZ, dnlZ, post};    % report -log marg lik, derivatives and post
else
  ⟨compute test predictions 7a⟩
end
```

It offers facilities for training the hyperparameters of a GP model as well as predictions at unseen inputs as detailed in the following help.

4b ⟨*gp function help* 4b⟩≡ (4a)

```
% 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
%
```

```
% 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.
%
⟨gpml copyright 5a⟩
```

5a ⟨*gpml copyright* 5a⟩≡ (4b 8 9 14 16 18 26 27 29 32)

```
% Copyright (c) by Carl Edward Rasmussen and Hannes Nickisch, 2013-01-21
```

Depending on the number of input parameters, gp knows whether it is operated in training or in prediction mode. The highlevel structure of the code is as follows. After some initialisations, we perform inference and decide whether test set predictions are needed or only the result of the inference is demanded.

5b ⟨*initializations* 5b⟩≡ (4a)

```
⟨minimalist usage 5c⟩
⟨process input arguments 5d⟩
⟨check hyperparameters 6a⟩
```

If the number of input arguments is incorrect, we echo a minimalist usage and return.

5c ⟨*minimalist usage* 5c⟩≡ (5b)

```
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
```

Set some useful default values for empty arguments, and convert `inf` and `lik` to function handles and `mean` and `cov` to cell arrays if necessary. Initialize variables.

5d ⟨*process input arguments* 5d⟩≡ (5b)

```
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);
```

Check that the sizes of the hyperparameters supplied in hyp match the sizes expected. The three parts `hyp.mean`, `hyp.cov` and `hyp.lik` are checked separately, and define empty entries if they don't exist.

6a ⟨*check hyperparameters* 6a⟩≡ (5b)

```
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
```

Inference is performed by calling the desired inference method `inf`. In training mode, we accept a failure of the inference method (and issue a warning), since during hyperparameter learning, hyperparameters causing a numerical failure may be attempted, but the `minimize` function may gracefully recover from this. During prediction, failure of the inference method is an error.

6b ⟨*inference* 6b⟩≡ (4a)

```
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
```

We copy the already computed negative log marginal likelihood to the first output argument, and if desired report its partial derivatives w.r.t. the hyperparameters if running in inference mode.

Predictions are computed in a loop over small batches to avoid memory problems for very large test sets.

7a ⟨*compute test predictions* 7a⟩≡ (4a)

```
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
  ⟨make predictions 7b⟩
  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
```

In every iteration of the above loop, we compute the predictions for all test points of the batch.

7b ⟨*make predictions* 7b⟩≡ (7a)

```
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
```

3 Inference Methods

Inference methods are responsible for computing the (approximate) posterior `post`, the (approximate) negative log marginal likelihood `nlZ` and its partial derivatives `dnlZ` w.r.t. the hyperparameters `hyp`. The arguments to the function are hyperparameters `hyp`, mean function `mean`, covariance function `cov`, likelihood function `lik` and training data `x` and `y`. Several inference methods are implemented and described this section.

8 ⟨*infMethods.m* 8⟩≡

```
% Inference methods: Compute the (approximate) posterior for a Gaussian process.
% Methods currently implemented include:
%
%   infExact         Exact inference (only possible with Gaussian likelihood)
%   infLaplace       Laplace's Approximation
%   infEP            Expectation Propagation
%   infVB            Variational Bayes Approximation
%
%   infFITC          Large scale regression with approximate covariance matrix
%   infFITC_Laplace  Large scale inference  with approximate covariance matrix
%   infFITC_EP       Large scale inference  with approximate covariance matrix
%
%   infMCMC     Markov Chain Monte Carlo and Annealed Importance Sampling
%               We offer two samplers.
%                 - hmc: Hybrid Monte Carlo
%                 - ess: Elliptical Slice Sampling
%               No derivatives w.r.t. to hyperparameters are provided.
%
%   infLOO      Leave-One-Out predictive probability and Least-Squares Approxim.
%
% The interface to the approximation methods is the following:
%
%   function [post nlZ dnlZ] = inf..(hyp, cov, lik, x, y)
%
% where:
%
%   hyp      is a struct of hyperparameters
%   cov      is the name of the covariance function (see covFunctions.m)
%   lik      is the name of the likelihood function (see likFunctions.m)
%   x        is a n by D matrix of training inputs
%   y        is a (column) vector (of size n) of targets
%
%   nlZ      is the returned value of the negative log marginal likelihood
%   dnlZ     is a (column) vector of partial derivatives of the negative
%               log marginal likelihood w.r.t. each hyperparameter
%   post     struct representation of the (approximate) posterior containing
%     alpha  is a (sparse or full column vector) containing inv(K)*m, where K
%               is the prior covariance matrix and m the approx posterior mean
%     sW     is a (sparse or full column) vector containing diagonal of sqrt(W)
%               the approximate posterior covariance matrix is inv(inv(K)+W)
%     L      is a (sparse or full) matrix, L = chol(sW*K*sW+eye(n))
%
% Usually, the approximate posterior to be returned admits the form
% N(m=K*alpha, V=inv(inv(K)+W)), where alpha is a vector and W is diagonal;
% if not, then L contains instead -inv(K+inv(W)), and sW is unused.
%
% For more information on the individual approximation methods and their
% implementations, see the separate inf??.m files. See also gp.m
```

```
%
⟨gpml copyright 5a⟩
```

Not all inference methods are compatible with all likelihood functions, e.g.. exact inference is only possible with Gaussian likelihood. In order to perform inference, each method needs various properties of the likelihood functions, section 4.

3.1 Exact Inference

For Gaussian likelihoods, GP inference reduces to computing mean and covariance of a multivariate Gaussian which can be done exactly by simple matrix algebra. The program `inf/infExact.m` does exactly this. If it is called with a likelihood function other than the Gaussian, it issues an error. The Gaussian posterior $q(\mathbf{f}|\mathcal{D}) = \mathcal{N}(\mathbf{f}|\mu, V)$ is exact.

9 ⟨*inf/infExact.m* 9⟩≡

```
function [post nlZ dnlZ] = infExact(hyp, mean, cov, lik, x, y)

% Exact inference for a GP with Gaussian likelihood. Compute a parametrization
% of the posterior, the negative log marginal likelihood and its derivatives
% w.r.t. the hyperparameters. See also "help infMethods".
%
⟨gpml copyright 5a⟩
%
% See also INFMETHODS.M.

if iscell(lik), likstr = lik{1}; else likstr = lik; end
if ~ischar(likstr), likstr = func2str(likstr); end
if ~strcmp(likstr,'likGauss')               % NOTE: no explicit call to likGauss
  error('Exact inference only possible with Gaussian likelihood');
end

[n, D] = size(x);
K = feval(cov{:}, hyp.cov, x);                      % evaluate covariance matrix
m = feval(mean{:}, hyp.mean, x);                          % evaluate mean vector

sn2 = exp(2*hyp.lik);                               % noise variance of likGauss
L = chol(K/sn2+eye(n));               % Cholesky factor of covariance with noise
alpha = solve_chol(L,y-m)/sn2;

post.alpha = alpha;                              % return the posterior parameters
post.sW = ones(n,1)/sqrt(sn2);                  % sqrt of noise precision vector
post.L  = L;                                        % L = chol(eye(n)+sW*sW'.*K)

if nargout>1                               % do we want the marginal likelihood?
  nlZ = (y-m)'*alpha/2 + sum(log(diag(L))) + n*log(2*pi*sn2)/2;  % -log marg lik
  if nargout>2                                        % do we want derivatives?
    dnlZ = hyp;                                 % allocate space for derivatives
    Q = solve_chol(L,eye(n))/sn2 - alpha*alpha';     % precompute for convenience
    for i = 1:numel(hyp.cov)
      dnlZ.cov(i) = sum(sum(Q.*feval(cov{:}, hyp.cov, x, [], i)))/2;
    end
    dnlZ.lik = sn2*trace(Q);
    for i = 1:numel(hyp.mean),
      dnlZ.mean(i) = -feval(mean{:}, hyp.mean, x, i)'*alpha;
    end
  end
end
```

3.2 Laplace's Approximation

For differentiable likelihoods, Laplace's approximation, approximates the posterior by a Gaussian centered at its mode and matching its curvature `infLaplace.m`.

More concretely, the mean of the posterior $q(\mathbf{f}|\mathcal{D}) = \mathcal{N}\big(f|\mu, V\big)$ is given by

$$\mu = \arg\min_{f} \phi(f), \text{ where } \phi(f) = \frac{1}{2}(f-m)^\top K^{-1}(f-m) - \sum_{i=1}^{n} \ln p(y_i|f_i) \stackrel{c}{=} -\ln[p(f)p(y|f)]. \quad (2)$$

The curvature $\frac{\partial^2\phi}{\partial f f^\top} = K^{-1} + W$ with $W_{ii} = -\frac{\partial^2}{\partial f_i^2} \ln p(y_i|f_i)$ serves as precision for the Gaussian posterior approximation $V = (K^{-1}+W)^{-1}$ and the marginal likelihood $Z = \int p(f)p(y|f)df$ is approximated by $Z \approx Z_{LA} = \int \tilde{\phi}(f)df$ where we use the 2nd order Taylor expansion at the mode μ given by $\tilde{\phi}(f) = \phi(\mu) + \frac{1}{2}(f-\mu)^\top V^{-1}(f-\mu) \approx \phi(f)$.

Laplace's approximation needs derivatives up to third order for the mode fitting procedure (Newton method)

$$d_k \;=\; \frac{\partial^k}{\partial f^k} \log p(y|f), \quad k = 0,1,2,3$$

and

$$d_k \;=\; \frac{\partial}{\partial \rho_i}\frac{\partial^k}{\partial f^k} \log p(y|f), \quad k = 0,1,2$$

evaluated at the latent location f and observed value y. The likelihood calls (see section 4)

- `[d0, d1, d2, d3] = lik(hyp, y, f, [], 'infLaplace')`

and

- `[d0, d1, d2] = lik(hyp, y, f, [], 'infLaplace', i)`

return exactly these values.

3.3 Expectation Propagation

The basic idea of Expectation Propagation (EP) is to replace the non-Gaussian likelihood terms $p(y_i|f_i)$ by Gaussian functions $t(f_i;\nu_i,\tau_i) = \exp(\nu_i f_i - \frac{1}{2}\tau_i f_i^2)$ and to adjust the parameters ν_i, τ_i such that the following identity holds:

$$\frac{1}{Z_{t,i}}\int f^k q_{-i}(f)\cdot t(f_i;\nu_i,\tau_i)df = \frac{1}{Z_{p,i}}\int f^k q_{-i}(f)\cdot p(y_i|f_i)df, \quad k = 1,2$$

with the so-called cavity distributions $q_{-i}(f) = \mathcal{N}(f|m,K)\prod_{j\neq i} t(f_j;\nu_j,\tau_j) \propto \mathcal{N}(f|\mu,V)/t(f_i;\nu_i,\tau_i)$ equal to the posterior divided by the ith Gaussian approximation function and the two normalisers $Z_{t,i} = \int q_{-i}(f)\cdot t(f_i;\nu_i,\tau_i)df$ and $Z_{p,i} = \int q_{-i}(f)\cdot p(y_i|f_i)df$.

In order to apply the moment matching steps in a numerically safe way, EP requires the expectations

$$d_k \;=\; \frac{\partial^k}{\partial \mu_i^k} \log \int p(y|f)\mathcal{N}(f|\mu,\sigma^2)df, \quad k = 0,1,2$$

and

$$d \;=\; \frac{\partial}{\partial \rho_i} \log \int p(y|f)\mathcal{N}(f|\mu,\sigma^2)df$$

which can be obtained by the likelihood calls (see section 4)

- `[d0, d1, d2] = lik(hyp, y, mu, s2, 'infEP')`

and

- `d = lik(hyp, y, mu, s2, 'infEP', i).`

3.4 Variational Bayes

Based on individual lower bounds to every likelihood

$$p(y|f) \geqslant t(f;\gamma) = \exp\left(\beta(\gamma)f - \frac{1}{2}s^2\gamma - \frac{1}{2}h(\gamma)\right) \propto \mathcal{N}(f|\beta\gamma,\gamma)$$

of scaled Gaussian form, one can construct a joint lower bound on the marginal likelihood

$$Z = \int \mathcal{N}(f|m,V)p(y|f)df \geqslant Z_{VB} = \int \mathcal{N}(f|m,V)t(f;\gamma)df$$

that can be maximised w.r.t. to the variational parameters γ. Whenever, the likelihood is log-concave, the maximisation problem in γ is concave. Details about $h(\gamma)$ and $\beta(\gamma)$ can be found in papers by Palmer et al. *Variational EM Algorithms for Non-Gaussian Latent Variable Models* , NIPS, 2006 and Nickisch & Seeger *Convex Variational Bayesian Inference for Large Scale Generalized Linear Models* , ICML, 2009.

In practice, we use a Newton algorithm requiring

$$\mathtt{dh}_k \;=\; \frac{\partial^k}{\partial\gamma^k}h(\gamma),\;\mathtt{db}_k \;=\; \frac{\partial^k}{\partial\gamma^k}\beta(\gamma),\quad k=0,1,2$$

and

$$\mathtt{d} \;=\; \frac{\partial}{\partial\rho_i}h(\gamma)$$

which are delivered by the likelihood calls (see section 4)

- `[dh0, db0, dh1, db1, dh2, db2] = lik(hyp, y, [], ga, 'infVB')`

and

- `d = lik(hyp, y, [], ga, 'infVB', i).`

3.5 FITC Approximations

One of the main problems with GP models is the high computational load for inference computations. In a setting with n training points x, exact inference with Gaussian likelihood requires $O(n^3)$ effort; approximations like Laplace of EP consist of a sequence of $O(n^3)$ operations.

There is a line of research with the goal to alleviate this burden by using approximate covariance functions $\tilde{k}$ instead of k. A review is given by Candela and Rasmussen *A Unifying View of Sparse Approximate Gaussian Process Regression* , JMLR, 2005. One basic idea in those approximations is to work with a set of m inducing inputs u with a reduced computational load of $O(nm^2)$. In the following, we will provide a rough idea of the FITC approximation used in the toolbox. Let K denote the $n \times n$ covariance matrix between the training points x, K_u the $m \times n$ covariance matrix between the n training points and the m inducing points, and K_{uu} the $m \times m$ covariance matrix between the m inducing points. The FITC approximation to the covariance is given by

$$K \approx \tilde{K} = Q + G, \; G = \text{diag}(g), \; g = \text{diag}(K - Q), \; Q = K_u^\top Q_{uu}^{-1} K_u, \; Q_{uu} = K_{uu} + \sigma_{n_u}^2 I,$$

where σ_{n_u} is the noise from the inducing inputs. Note that $\tilde{K}$ and K have the same diagonal elements $\text{diag}(\tilde{K}) = \text{diag}(K)$; all off-diagonal elements are the same as for Q. The toolbox offers FITC versions for regression with Gaussian likelihood `infFITC`, as well as for Laplace's approximation `infFITC_Laplace` and expectation propagation `infFITC_EP`.

4 Likelihood Functions

A likelihood function $\mathbb{P}_\rho(y|f)$ (with hyperparameters ρ) is a conditional density $\int \mathbb{P}_\rho(y|f)dy = 1$ defined for scalar latent function values f and outputs y. In the GPML toolbox, we use iid. likelihoods $\mathbb{P}_\rho(\mathbf{y}|\mathbf{f}) = \prod_{i=1}^n \mathbb{P}_\rho(y_i|f_i)$. The approximate inference engine does not explicitly distinguish between classification and regression likelihoods: it is fully generic in the likelihood allowing to use a single code in the inference step.

Likelihood functionality is needed both during inference and while predicting.

4.1 Prediction

A prediction at x_* conditioned on the data $\mathcal{D} = (X, y)$ (as implemented in `gp.m`) consists of the predictive mean μ_{y_*} and variance $\sigma^2_{y_*}$ which are computed from the Gaussian marginal approximation $\mathcal{N}(f_*|\mu_{f_*}, \sigma^2_{f_*})$ via

$$\begin{aligned} p(y_*|\mathcal{D}, x_*) &= \int p(y_*|f_*)p(f_*|\mathcal{D}, x_*)df_*. &(3)\\ &\approx \int p(y_*|f_*)\mathcal{N}(f_*|\mu_{f_*}, \sigma^2_{f_*})df_*. &(4) \end{aligned}$$

The moments are obtained by $\mu_{y_*} = \int y_* p(y_*|\mathcal{D}, x_*)dy_*$ and $\sigma^2_{y_*} = \int (y_* - \mu_{y_*})^2 p(y_*|\mathcal{D}, x_*)dy_*$. The likelihood call

- `[lp,ymu,ys2] = lik(hyp, [], fmu, fs2)`

does exactly this. Evaluation of the log of $p_{y_*} = p(y_*|\mathcal{D}, x_*)$ for values y_* can be done via

- `[lp,ymu,ys2] = lik(hyp, y, fmu, fs2)`

where `lp` contains the number $\ln p_{y_*}$.

The binary case is simple since $y_* \in \{-1, +1\}$ and $1 = p_{y_*} + p_{-y_*}$. Using $\pi_* = p_1$, we find

$$\begin{aligned} p_{y_*} &= \begin{cases} \pi_* & y_* = +1 \\ 1 - \pi_* & y_* = -1 \end{cases} \\ \mu_{y_*} &= \sum_{y_*=\pm1} y_* p(y_*|\mathcal{D}, x_*) = 2 \cdot \pi_* - 1 \in [-1, 1], \quad \text{and} \\ \sigma^2_{y_*} &= \sum_{y_*=\pm1} (y_* - \mu_{y_*})^2 p(y_*|\mathcal{D}, x_*) = 4 \cdot \pi_*(1 - \pi_*) \in [0, 1]. \end{aligned}$$

The continuous case for likelihoods depending on $r_* = |f_* - y_*|$ only is also simple. By noting that $p(y_*|f_*) = p(y_* + \rho|f_* + \rho)$, we can swap the order of integration and use the Gaussian marginal approximation $\mathcal{N}(f_*|\mu_{f_*}, \sigma^2_{f_*})$ to find

$$\begin{aligned} \mu_{y_*} &\approx \mu_{f_*}, \text{ and} \\ \sigma^2_{y_*} &\approx \sigma^2_{f_*} + \int y_*^2 p(y_*|0)dy_*. \end{aligned}$$

In the following, we will detail how and which likelihood functions are implemented in the GPML toolbox. Further, we will mention dependencies between likelihoods and inference methods and provide some analytical expressions in addition to some likelihood implementations.

4.2 Interface

The likelihoods are in fact the most challenging object in our implementation. Different inference algorithms require different aspects of the likelihood to be computed, therefore the interface is rather involved as detailed below.

14 ⟨*likFunctions.m* 14⟩≡

```
% likelihood functions are provided to be used by the gp.m function:
%
%   likErf          (Error function, classification, probit regression)
%   likLogistic     (Logistic,       classification, logit  regression)
%   likUni          (Uniform likelihood, classification)
%
%   likGauss        (Gaussian, regression)
%   likLaplace      (Laplacian or double exponential, regression)
%   likSech2        (Sech-square, regression)
%   likT            (Student's t, regression)
%
%   likPoisson      (Poisson regression, count data)
%
%   likMix          (Mixture of individual covariance functions)
%
% The likelihood functions have three possible modes, the mode being selected
% as follows (where "lik" stands for any likelihood function in "lik/lik*.m".):
%
% 1) With one or no input arguments:          [REPORT NUMBER OF HYPERPARAMETERS]
%
%    s = lik OR s = lik(hyp)
%
% The likelihood function returns a string telling how many hyperparameters it
% expects, using the convention that "D" is the dimension of the input space.
% For example, calling "likLogistic" returns the string '0'.
%
%
% 2) With three or four input arguments:                       [PREDICTION MODE]
%
%    lp = lik(hyp, y, mu) OR [lp, ymu, ys2] = lik(hyp, y, mu, s2)
%
% This allows to evaluate the predictive distribution. Let p(y_*|f_*) be the
% likelihood of a test point and N(f_*|mu,s2) an approximation to the posterior
% marginal p(f_*|x_*,x,y) as returned by an inference method. The predictive
% distribution p(y_*|x_*,x,y) is approximated by.
%   q(y_*) = \int N(f_*|mu,s2) p(y_*|f_*) df_*
%
%   lp = log( q(y) ) for a particular value of y, if s2 is [] or 0, this
%                     corresponds to log( p(y|mu) )
%   ymu and ys2       the mean and variance of the predictive marginal q(y)
%                     note that these two numbers do not depend on a particular
%                     value of y
%  All vectors have the same size.
%
%
% 3) With five or six input arguments, the fifth being a string [INFERENCE MODE]
%
% [varargout] = lik(hyp, y, mu, s2, inf) OR
% [varargout] = lik(hyp, y, mu, s2, inf, i)
%
% There are three cases for inf, namely a) infLaplace, b) infEP and c) infVB.
```

```
% The last input i, refers to derivatives w.r.t. the ith hyperparameter.
%
% a1) [lp,dlp,d2lp,d3lp] = lik(hyp, y, f, [], 'infLaplace')
% lp, dlp, d2lp and d3lp correspond to derivatives of the log likelihood
% log(p(y|f)) w.r.t. to the latent location f.
%   lp = log( p(y|f) )
%  dlp = d   log( p(y|f) ) / df
% d2lp = d^2 log( p(y|f) ) / df^2
% d3lp = d^3 log( p(y|f) ) / df^3
%
% a2) [lp_dhyp,dlp_dhyp,d2lp_dhyp] = lik(hyp, y, f, [], 'infLaplace', i)
% returns derivatives w.r.t. to the ith hyperparameter
%   lp_dhyp = d   log( p(y|f) ) / (     dhyp_i)
%  dlp_dhyp = d^2 log( p(y|f) ) / (df   dhyp_i)
% d2lp_dhyp = d^3 log( p(y|f) ) / (df^2 dhyp_i)
%
%
% b1) [lZ,dlZ,d2lZ] = lik(hyp, y, mu, s2, 'infEP')
% let Z = \int p(y|f) N(f|mu,s2) df then
%   lZ =     log(Z)
%  dlZ = d   log(Z) / dmu
% d2lZ = d^2 log(Z) / dmu^2
%
% b2) [dlZhyp] = lik(hyp, y, mu, s2, 'infEP', i)
% returns derivatives w.r.t. to the ith hyperparameter
% dlZhyp = d log(Z) / dhyp_i
%
%
% c1) [h,b,dh,db,d2h,d2b] = lik(hyp, y, [], ga, 'infVB')
% ga is the variance of a Gaussian lower bound to the likelihood p(y|f).
%   p(y|f) \ge exp( b*f - f.^2/(2*ga) - h(ga)/2 ) \propto N(f|b*ga,ga)
% The function returns the linear part b and the "scaling function" h(ga) and
% derivatives dh = d h/dga, db = d b/dga, d2h = d^2 h/dga and d2b = d^2 b/dga.
%
% c2) [dhhyp] = lik(hyp, y, [], ga, 'infVB', i)
% dhhyp = dh / dhyp_i, the derivative w.r.t. the ith hyperparameter
%
% Cumulative likelihoods are designed for binary classification. Therefore, they
% only look at the sign of the targets y; zero values are treated as +1.
%
% Some examples for valid likelihood functions:
%      lik = @likLogistic;
%      lik = {'likMix',{'likUni',@likErf}}
%      lik = {@likPoisson,'logistic'};
%
% See the help for the individual likelihood for the computations specific to
% each likelihood function.
%
⟨gpml copyright 5a⟩
```

4.3 Implemented Likelihood Functions

The following table enumerates all (currently) implemented likelihood functions that can be found at `lik/lik<NAME>.m` and their respective set of hyperparameters $\boldsymbol{\rho}$.

lik<NAME>	regression $y_i \in \mathbb{R}$	$\mathbb{P}_\rho(y_i\|f_i) =$	$\rho =$
Gauss	Gaussian	$\mathcal{N}(y_i\|f_i,\sigma^2) = \frac{1}{\sqrt{2\pi}\sigma}\exp\left(-\frac{(y_i-f_i)^2}{2\sigma^2}\right)$	$\{\ln\sigma\}$
Sech2	Sech-squared	$\frac{\tau}{2\cosh^2(\tau(y_i-f_i))}$, $\tau = \frac{\pi}{2\sigma\sqrt{3}}$	$\{\ln\sigma\}$
Laplace	Laplacian	$\frac{1}{2b}\exp\left(-\frac{\|y_i-f_i\|}{b}\right)$, $b = \frac{\sigma}{\sqrt{2}}$	$\{\ln\sigma\}$
T	Student's t	$\frac{\Gamma(\frac{\nu+1}{2})}{\Gamma(\frac{\nu}{2})}\frac{1}{\sqrt{\nu\pi}\sigma}\left(1+\frac{(y_i-f_i)^2}{\nu\sigma^2}\right)^{-\frac{\nu+1}{2}}$	$\{\ln(\nu-1),\ln\sigma\}$
lik<NAME>	classification $y_i \in \{\pm 1\}$	$\mathbb{P}_\rho(y_i\|f_i) =$	$\rho =$
Erf	Error function	$\int_{-\infty}^{y_i f_i}\mathcal{N}(t)dt$	$\emptyset$
Logistic	Logistic function	$\frac{1}{1+\exp(-y_i f_i)}$	$\emptyset$
Uni	Label noise	$\frac{1}{2}$	$\emptyset$
lik<NAME>	count data $y_i \in \mathbb{N}$	$\mathbb{P}_\rho(y_i\|f_i) =$	$\rho =$
Poisson	Poisson	$\mu^y \cdot \frac{e^{-\mu}}{y!}$, $\mu = e^f$ or $\mu = \log(1+e^f)$	$\emptyset$
Composite likelihood functions $[p_1(y_i\|f_i), p_1(y_i\|f_i), ..] \mapsto \mathbb{P}_\rho(y_i\|f_i)$			
Mix	Mixture	$\sum_j \alpha_j p_j(y_i\|f_i)$	$\{\ln\alpha_1, \ln\alpha_2, ..\}$

4.4 Usage of Implemented Likelihood Functions

Some code examples taken from `doc/usageLik.m` illustrate how to use simple and composite likelihood functions to specify a GP model.

Syntactically, a likelihood function `lf` is defined by

```
lk := 'func' | @func // simple

lf := {lk} | {param, lk} | {lk, {lk, .., lk}} // composite
```

i.e., it is either a string containing the name of a likelihood function, a pointer to a likelihood function or one of the former in combination with a cell array of likelihood functions and an additional list of parameters.

16 ⟨*doc/usageLik.m* 16⟩≡

```
% demonstrate usage of likelihood functions
%
% See also likFunctions.m.
%
⟨gpml copyright 5a⟩
clear all, close all
n = 5; f = randn(n,1);        % create random latent function values

% set up simple classification likelihood functions
yc = sign(f);
lc0 = {'likErf'};      hypc0 = [];    % no hyperparameters are needed
lc1 = {@likLogistic}; hypc1 = [];     % also function handles are OK
lc2 = {'likUni'};      hypc2 = [];
lc3 = {'likMix',{'likUni',@likErf}}; hypc3 = log([1,2]); %mixture

% set up simple regression likelihood functions
yr = f + randn(n,1)/20;
sn = 0.1;                                   % noise standard deviation
lr0 = {'likGauss'};   hypr0 = log(sn);
lr1 = {'likLaplace'}; hypr1 = log(sn);
lr2 = {'likSech2'};   hypr2 = log(sn);
nu = 4;                                 % number of degrees of freedom
lr3 = {'likT'};       hypr3 = [log(nu-1); log(sn)];
```

```
lr4 = {'likMix',{lr0,lr1}}; hypr4 = [log([1,2]),hypr0,hypr1];

% set up Poisson regression
yp = fix(abs(f)) + 1;
lp0 = {@likPoisson,'logistic'}; hypp0 = [];
lp1 = {@likPoisson,'exp'};      hypp1 = [];

% 0) specify the likelihood function
lik = lc0; hyp = hypc0; y = yc;
% lik = lr4; hyp = hypr4; y = yr;
% lik = lp1; hyp = hypp1; y = yp;

% 1) query the number of parameters
feval(lik{:})

% 2) evaluate the likelihood function on f
exp(feval(lik{:},hyp,y,f))

% 3a) evaluate derivatives of the likelihood
[lp,dlp,d2lp,d3lp] = feval(lik{:}, hyp, y, f, [], 'infLaplace');

% 3b) compute Gaussian integrals w.r.t. likelihood
mu = f; s2 = rand(n,1);
[lZ,dlZ,d2lZ] = feval(lik{:}, hyp, y, mu, s2, 'infEP');

% 3c) obtain lower bound on likelihood
ga = rand(n,1);
[h,b,dh,db,d2h,d2b] = feval(lik{:}, hyp, y, [], ga, 'infVB');
```

4.5 Compatibility Between Likelihoods and Inference Methods

The following table lists all possible combinations of likelihood function and inference methods.

Likelihood \ Inference	Exact FITC	EP FITC-EP	Laplace FITC-Laplace	variational Bayes	leave one out	MCMC	usage
Gaussian	✓	✓	✓	✓	✓	✓	regression
Sech-squared		✓	✓	✓	✓	✓	regression
Laplacian		✓		✓	✓	✓	regression
Student's t			✓	✓	✓	✓	regression
Error function		✓	✓	✓	✓	✓	probit regression
Logistic function		✓	✓	✓	✓	✓	logit regression
Uniform		✓	✓	✓	✓	✓	label noise
Poisson		✓	✓		✓	✓	Poisson regression
Mixture		✓	✓		✓	✓	general mixture

Exact inference is only tractable for Gaussian likelihoods. Expectation propagation together with Student's t likelihood is inherently unstable due to non-log-concavity. Laplace's approximation for Laplace likelihoods is not sensible because at the mode the curvature and the gradient can be undefined due to the non-differentiable peak of the Laplace distribution. Special care has been taken for the non-convex optimisation problem imposed by the combination Student's t likelihood and Laplace's approximation.

4.6 Gaussian Likelihood

The Gaussian likelihood is the simplest likelihood because the posterior distribution is not only Gaussian but can be computed analytically. In principle, the Gaussian likelihood would only be operated in conjunction with the exact inference method but we chose to provide compatibility with all other inference algorithms as well because it enables code testing and allows to switch between different regression likelihoods very easily.

18 ⟨*lik/likGauss.m* 18⟩≡

```
function [varargout] = likGauss(hyp, y, mu, s2, inf, i)

% likGauss - Gaussian likelihood function for regression. The expression for the
% likelihood is
%   likGauss(t) = exp(-(t-y)^2/2*sn^2) / sqrt(2*pi*sn^2),
% where y is the mean and sn is the standard deviation.
%
% The hyperparameters are:
%
% hyp = [  log(sn)  ]
%
% Several modes are provided, for computing likelihoods, derivatives and moments
% respectively, see likFunctions.m for the details. In general, care is taken
% to avoid numerical issues when the arguments are extreme.
%
⟨gpml copyright 5a⟩
%
% See also LIKFUNCTIONS.M.

if nargin<3, varargout = {'1'}; return; end   % report number of hyperparameters

sn2 = exp(2*hyp);

if nargin<5                                  % prediction mode if inf is not present
```

```
  ⟨Prediction with Gaussian likelihood 19a⟩
else
  switch inf
  case 'infLaplace'
    ⟨Laplace's method with Gaussian likelihood 19b⟩
  case 'infEP'
    ⟨EP inference with Gaussian likelihood 20a⟩
  case 'infVB'
    ⟨Variational Bayes inference with Gaussian likelihood 20b⟩
  end
end
```

19a ⟨*Prediction with Gaussian likelihood* 19a⟩≡ (18)

```
if numel(y)==0,  y = zeros(size(mu)); end
s2zero = 1; if nargin>3, if norm(s2)>0, s2zero = 0; end, end         % s2==0 ?
if s2zero                                                     % log probability
  lp = -(y-mu).^2./sn2/2-log(2*pi*sn2)/2; s2 = 0;
else
  lp = likGauss(hyp, y, mu, s2, 'infEP');                          % prediction
end
ymu = {}; ys2 = {};
if nargout>1
  ymu = mu;                                                    % first y moment
  if nargout>2
    ys2 = s2 + sn2;                                           % second y moment
  end
end
varargout = {lp,ymu,ys2};
```

The Gaussian likelihood function has a single hyperparameter ρ, the log of the noise standard deviation σ_n.

4.6.1 Exact Inference

Exact inference doesn't require any specific likelihood related code; all computations are done directly by the inference method, section 3.1.

4.6.2 Laplace's Approximation

19b ⟨*Laplace's method with Gaussian likelihood* 19b⟩≡ (18)

```
if nargin<6                                              % no derivative mode
  if numel(y)==0, y=0; end
  ymmu = y-mu; dlp = {}; d2lp = {}; d3lp = {};
  lp = -ymmu.^2/(2*sn2) - log(2*pi*sn2)/2;
  if nargout>1
    dlp = ymmu/sn2;                       % dlp, derivative of log likelihood
    if nargout>2                     % d2lp, 2nd derivative of log likelihood
      d2lp = -ones(size(ymmu))/sn2;
      if nargout>3                   % d3lp, 3rd derivative of log likelihood
        d3lp = zeros(size(ymmu));
      end
    end
  end
  varargout = {lp,dlp,d2lp,d3lp};
else                                                        % derivative mode
  lp_dhyp = (y-mu).^2/sn2 - 1;  % derivative of log likelihood w.r.t. hypers
```

```
  dlp_dhyp = 2*(mu-y)/sn2;                                  % first derivative,
  d2lp_dhyp = 2*ones(size(mu))/sn2;   % and also of the second mu derivative
  varargout = {lp_dhyp,dlp_dhyp,d2lp_dhyp};
end
```

4.6.3 Expectation Propagation

20a ⟨*EP inference with Gaussian likelihood* 20a⟩≡ (18)

```
if nargin<6                                                  % no derivative mode
  lZ = -(y-mu).^2./(sn2+s2)/2 - log(2*pi*(sn2+s2))/2;     % log part function
  dlZ = {}; d2lZ = {};
  if nargout>1
    dlZ  = (y-mu)./(sn2+s2);                       % 1st derivative w.r.t. mean
    if nargout>2
      d2lZ = -1./(sn2+s2);                         % 2nd derivative w.r.t. mean
    end
  end
  varargout = {lZ,dlZ,d2lZ};
else                                                            % derivative mode
  dlZhyp = ((y-mu).^2./(sn2+s2)-1) ./ (1+s2./sn2);   % deriv. w.r.t. hyp.lik
  varargout = {dlZhyp};
end
```

4.6.4 Variational Bayes

20b ⟨*Variational Bayes inference with Gaussian likelihood* 20b⟩≡ (18)

```
if nargin<6
  % variational lower site bound
  % t(s) = exp(-(y-s)^2/2sn2)/sqrt(2*pi*sn2)
  % the bound has the form: b*s - s.^2/(2*ga) - h(ga)/2 with b=y/ga
  ga = s2; n = numel(ga); b = y./ga; y = y.*ones(n,1);
  db = -y./ga.^2; d2b = 2*y./ga.^3;
  h = zeros(n,1); dh = h; d2h = h;           % allocate memory for return args
  id = ga(:)<=sn2+1e-8;                               % OK below noise variance
  h(id) = y(id).^2./ga(id) + log(2*pi*sn2); h(~id) = Inf;
  dh(id) = -y(id).^2./ga(id).^2;
  d2h(id) = 2*y(id).^2./ga(id).^3;
  id = ga<0; h(id) = Inf; dh(id) = 0; d2h(id) = 0;      % neg. var. treatment
  varargout = {h,b,dh,db,d2h,d2b};
else
  ga = s2; n = numel(ga);
  dhhyp = zeros(n,1); dhhyp(ga(:)<=sn2) = 2;
  dhhyp(ga<0) = 0;                % negative variances get a special treatment
  varargout = {dhhyp};                                   % deriv. w.r.t. hyp.lik
end
```

4.7 Laplace Likelihood

4.7.1 Laplace's Approximation

The following derivatives are needed:

$$
\begin{aligned}
\ln p(y|f) &= -\ln(2b) - \frac{|f-y|}{b} \\
\frac{\partial \ln p}{\partial f} &= \frac{\operatorname{sign}(f-y)}{b} \\
\frac{\partial^2 \ln p}{(\partial f)^2} &= \frac{\partial^3 \ln p}{(\partial f)^3} = \frac{\partial^3 \ln p}{(\partial \ln \sigma_n)(\partial f)^2} = 0 \\
\frac{\partial \ln p}{\partial \ln \sigma_n} &= \frac{|f-y|}{b} - 1
\end{aligned}
$$

4.7.2 Expectation Propagation

Expectation propagation requires integration against a Gaussian measure for moment matching. We need to evaluate $\ln Z = \ln \int \mathcal{L}(y|f, \sigma_n^2)\mathcal{N}(f|\mu, \sigma^2)df$ as well as the derivatives $\frac{\partial \ln Z}{\partial \mu}$ and $\frac{\partial^2 \ln Z}{\partial \mu^2}$ where $\mathcal{N}(f|\mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(f-\mu)^2}{2\sigma^2}\right)$, $\mathcal{L}(y|f, \sigma_n^2) = \frac{1}{2b}\exp\left(-\frac{|y-f|}{b}\right)$, and $b = \frac{\sigma_n}{\sqrt{2}}$. As a first step, we reduce the number of parameters by means of the substitution $\tilde{f} = \frac{f-y}{\sigma_n}$ yielding

$$
\begin{aligned}
Z &= \int \mathcal{L}(y|f, \sigma_n^2)\mathcal{N}(f|\mu, \sigma^2)df \\
&= \frac{1}{\sqrt{2\pi}\sigma}\frac{\sqrt{2}}{2\sigma_n}\int \exp\left(-\frac{(f-\mu)^2}{2\sigma^2}\right)\exp\left(-\sqrt{2}\frac{|f-y|}{\sigma_n}\right)df \\
&= \frac{\sqrt{2}}{2\sigma\sqrt{2\pi}}\int \exp\left(-\frac{(\sigma_n\tilde{f} + y - \mu)^2}{2\sigma^2}\right)\exp\left(-\sqrt{2}|\tilde{f}|\right)d\tilde{f} \\
&= \frac{\sigma_n}{\sigma\sigma_n\sqrt{2\pi}}\int \exp\left(-\frac{\sigma_n^2\left(\tilde{f} - \frac{\mu-y}{\sigma_n}\right)^2}{2\sigma^2}\right)\mathcal{L}(\tilde{f}|0,1)d\tilde{f} \\
&= \frac{1}{\sigma_n}\int \mathcal{L}(f|0,1)\mathcal{N}(f|\tilde{\mu}, \tilde{\sigma}^2)df \\
\ln Z &= \ln\tilde{Z} - \ln\sigma_n = \ln\int \mathcal{L}(f|0,1)\mathcal{N}(f|\tilde{\mu}, \tilde{\sigma}^2)df - \ln\sigma_n
\end{aligned}
$$

with $\tilde{\mu} = \frac{\mu - y}{\sigma_n}$ and $\tilde{\sigma} = \frac{\sigma}{\sigma_n}$. Thus, we concentrate on the simpler quantity $\ln \tilde{Z}$.

$$
\begin{aligned}
\ln Z &= \ln \int \exp\left(-\frac{(f-\tilde{\mu})^2}{2\tilde{\sigma}^2} - \sqrt{2}|f|\right) df \overbrace{- \ln \tilde{\sigma}\sqrt{2\pi} - \ln \sqrt{2}\sigma_n}^{C} \\
&= \ln \left[\int_{-\infty}^{0} \exp\left(-\frac{(f-\tilde{\mu})^2}{2\tilde{\sigma}^2} + \sqrt{2}f\right) df + \int_{0}^{\infty} \exp\left(-\frac{(f-\tilde{\mu})^2}{2\tilde{\sigma}^2} - \sqrt{2}f\right) df \right] + C \\
&= \ln \left[\int_{-\infty}^{0} \exp\left(-\frac{f^2 - 2(\overbrace{\tilde{\mu} + \tilde{\sigma}^2\sqrt{2}}^{m_-})f + \tilde{\mu}^2}{2\tilde{\sigma}^2}\right) df + \int_{0}^{\infty} \exp\left(-\frac{f^2 - 2(\overbrace{\tilde{\mu} - \tilde{\sigma}^2\sqrt{2}}^{m_+})f + \tilde{\mu}^2}{2\tilde{\sigma}^2}\right) df \right] + C \\
&= \ln \left[\exp\left(\frac{m_-^2}{2\tilde{\sigma}^2}\right) \int_{-\infty}^{0} \exp\left(-\frac{(f-m_-)^2}{2\tilde{\sigma}^2}\right) df + \exp\left(\frac{m_+^2}{2\tilde{\sigma}^2}\right) \int_{0}^{\infty} \exp\left(-\frac{(f-m_+)^2}{2\tilde{\sigma}^2}\right) df \right] - \frac{\tilde{\mu}^2}{2\tilde{\sigma}^2} + C \\
&= \ln \left[\exp\left(\frac{m_-^2}{2\tilde{\sigma}^2}\right) \int_{-\infty}^{0} \mathcal{N}(f|m_-, \tilde{\sigma}^2) df + \exp\left(\frac{m_+^2}{2\tilde{\sigma}^2}\right) \left(1 - \int_{-\infty}^{0} \mathcal{N}(f|m_+, \tilde{\sigma}^2) df\right) \right] - \frac{\tilde{\mu}^2}{2\tilde{\sigma}^2} - \ln \sqrt{2}\sigma_n \\
&= \ln \left[\exp\left(\frac{m_-^2}{2\tilde{\sigma}^2}\right) \Phi\left(\frac{m_-}{\tilde{\sigma}}\right) - \exp\left(\frac{m_+^2}{2\tilde{\sigma}^2}\right) \Phi\left(\frac{m_+}{\tilde{\sigma}}\right) + \exp\left(\frac{m_+^2}{2\tilde{\sigma}^2}\right) \right] - \frac{\tilde{\mu}^2}{2\tilde{\sigma}^2} - \ln \sqrt{2}\sigma_n
\end{aligned}
$$

Here, $\Phi(z) = \int_{-\infty}^{z} \mathcal{N}(f|0,1) df$ denotes the cumulative Gaussian distribution. Finally, we have

$$
\begin{aligned}
\ln Z &= \ln \left[\exp\left(-\sqrt{2}\tilde{\mu}\right) \Phi\left(\frac{m_-}{\tilde{\sigma}}\right) + \exp\left(\sqrt{2}\tilde{\mu}\right) \Phi\left(-\frac{m_+}{\tilde{\sigma}}\right) \right] + \tilde{\sigma}^2 - \ln \sqrt{2}\sigma_n \\
&= \ln \left[\exp\left(\underbrace{\ln \Phi(-z_+) + \sqrt{2}\tilde{\mu}}_{a_+}\right) + \exp\left(\underbrace{\ln \Phi(z_-) - \sqrt{2}\tilde{\mu}}_{a_-}\right) \right] + \tilde{\sigma}^2 - \ln \sqrt{2}\sigma_n \\
&= \ln(e^{a_+} + e^{a_-}) + \tilde{\sigma}^2 - \ln \sqrt{2}\sigma_n
\end{aligned}
$$

where $z_+ = \frac{\tilde{\mu}}{\tilde{\sigma}} + \tilde{\sigma}\sqrt{2} = \frac{\mu - y}{\sigma} + \frac{\sigma}{\sigma_n}\sqrt{2}$, $z_- = \frac{\tilde{\mu}}{\tilde{\sigma}} - \tilde{\sigma}\sqrt{2} = \frac{\mu - y}{\sigma} - \frac{\sigma}{\sigma_n}\sqrt{2}$ and $\tilde{\mu} = \frac{\mu - y}{\sigma_n}$, $\tilde{\sigma} = \frac{\sigma}{\sigma_n}$.

Now, using $\frac{d}{d\theta} \ln \Phi(z) = \frac{1}{\Phi(z)} \frac{d}{d\theta} \Phi(z) = \frac{\mathcal{N}(z)}{\Phi(z)} \frac{dz}{d\theta}$ we tackle first derivative

$$
\begin{aligned}
\frac{\partial \ln Z}{\partial \mu} &= \frac{e^{a_+} \frac{\partial a_+}{\partial \mu} + e^{a_-} \frac{\partial a_-}{\partial \mu}}{e^{a_+} + e^{a_-}} \\
\frac{\partial a_+}{\partial \mu} &= \frac{\partial}{\partial \mu} \ln \Phi(-z_+) + \frac{\sqrt{2}}{\sigma_n} \\
&= -\frac{\mathcal{N}(-z_+)}{\sigma \Phi(-z_+)} + \frac{\sqrt{2}}{\sigma_n} = -\frac{q_+}{\sigma} + \frac{\sqrt{2}}{\sigma_n} \\
\frac{\partial a_-}{\partial \mu} &= \frac{\partial}{\partial \mu} \ln \Phi(z_-) - \frac{\sqrt{2}}{\sigma_n} \\
&= \frac{\mathcal{N}(z_-)}{\sigma \Phi(z_-)} - \frac{\sqrt{2}}{\sigma_n} = \frac{q_-}{\sigma} - \frac{\sqrt{2}}{\sigma_n} \\
\frac{\partial a_\pm}{\partial \mu} &= \mp \frac{q_\pm}{\sigma} \pm \frac{\sqrt{2}}{\sigma_n}.
\end{aligned}
$$

as well as the second derivative

$$
\begin{aligned}
\frac{\partial^2 \ln Z}{\partial \mu^2} &= \frac{\frac{\partial}{\partial \mu}\left(e^{a_+}\frac{\partial a_+}{\partial \mu}\right) + \frac{\partial}{\partial \mu}\left(e^{a_-}\frac{\partial a_-}{\partial \mu}\right)}{e^{a_+} + e^{a_-}} - \left(\frac{\partial \ln Z}{\partial \mu}\right)^2 \\
\frac{\partial}{\partial \mu}\left(e^{a_\pm}\frac{\partial a_\pm}{\partial \mu}\right) &= e^{a_\pm}\left[\left(\frac{\partial a_\pm}{\partial \mu}\right)^2 + \frac{\partial^2 a_\pm}{\partial \mu^2}\right] \\
\frac{\partial^2 a_+}{\partial \mu^2} &= -\frac{1}{\sigma}\frac{\frac{\partial}{\partial \mu}\mathcal{N}(-z_+)\Phi(-z_+) - \frac{\partial}{\partial \mu}\Phi(-z_+)\mathcal{N}(-z_+)}{\Phi^2(-z_+)} \\
&= -\frac{1}{\sigma}\frac{\mathcal{N}(-z_+)\Phi(-z_+)\frac{\partial - z_+^2/2}{\partial \mu} - \mathcal{N}^2(-z_+)\frac{\partial - z_+}{\partial \mu}}{\Phi^2(-z_+)} \\
&= \frac{\mathcal{N}(-z_+)}{\sigma^2}\cdot\frac{\Phi(-z_+)z_+ - \mathcal{N}(-z_+)}{\Phi^2(-z_+)} = -\frac{q_+^2 - q_+ z_+}{\sigma^2} \\
\frac{\partial^2 a_-}{\partial \mu^2} &= \frac{1}{\sigma}\frac{\frac{\partial}{\partial \mu}\mathcal{N}(z_-)\Phi(z_-) - \frac{\partial}{\partial \mu}\Phi(z_-)\mathcal{N}(z_-)}{\Phi^2(z_-)} \\
&= \frac{1}{\sigma}\frac{\mathcal{N}(z_-)\Phi(z_-)\frac{\partial - z_-^2/2}{\partial \mu} - \mathcal{N}^2(z_-)\frac{\partial z_-}{\partial \mu}}{\Phi^2(z_-)} \\
&= \frac{\mathcal{N}(z_-)}{\sigma^2}\cdot\frac{-\Phi(z_-)z_- - \mathcal{N}(z_-)}{\Phi^2(z_-)} = -\frac{q_-^2 + q_- z_-}{\sigma^2} \\
\frac{\partial^2 a_\pm}{\partial \mu^2} &= -\frac{q_\pm^2 \mp q_\pm z_\pm}{\sigma^2}
\end{aligned}
$$

which can be simplified to

$$
\frac{\partial^2 \ln Z}{\partial \mu^2} = \frac{e^{a_+}b_+ + e^{a_-}b_-}{e^{a_+} + e^{a_-}} - \left(\frac{\partial \ln Z}{\partial \mu}\right)^2
$$

using

$$
\begin{aligned}
b_\pm = \left(\frac{\partial a_\pm}{\partial \mu}\right)^2 + \frac{\partial^2 a_\pm}{\partial \mu^2} &= \left(\mp\frac{q_\pm}{\sigma} \pm \frac{\sqrt{2}}{\sigma_n}\right)^2 - \frac{q_\pm^2 \mp q_\pm z_\pm}{\sigma^2} \\
&= \left(\frac{q_\pm}{\sigma} - \frac{\sqrt{2}}{\sigma_n}\right)^2 - \frac{q_\pm^2}{\sigma^2} \pm \frac{q_\pm z_\pm}{\sigma^2} \\
&= \frac{2}{\sigma_n^2} - \left(\frac{\sqrt{8}}{\sigma\sigma_n} \mp \frac{z_\pm}{\sigma^2}\right) q_\pm.
\end{aligned}
$$

We also need

$$
\frac{\partial \ln Z}{\partial \ln \sigma_n} = \frac{e^{a_+}\frac{\partial a_+}{\partial \ln \sigma_n} + e^{a_-}\frac{\partial a_-}{\partial \ln \sigma_n}}{e^{a_+} + e^{a_-}} - \frac{2\sigma^2}{\sigma_n^2} - 1.
$$

4.7.3 Variational Bayes

We need $h(\gamma)$ and its derivatives as well as $\beta(\gamma)$:

$$
\begin{aligned}
h(\gamma) &= \frac{2}{\sigma_n^2}\gamma + \ln(2\sigma_n^2) + y^2\gamma^{-1} \\
h'(\gamma) &= \frac{2}{\sigma_n^2} - y^2\gamma^{-2} \\
h''(\gamma) &= 2y^2\gamma^{-3} \\
\beta(\gamma) &= y\gamma^{-1}
\end{aligned}
$$

4.8 Student's t Likelihood

The likelihood has two hyperparameters (both represented in the log domain to ensure positivity): the degrees of freedom ν and the scale σ_n with mean y (for $\nu > 1$) and variance $\frac{\nu}{\nu-2}{\sigma_n}^2$ (for $\nu > 2$).

$$
p(y|f) = Z \cdot \left(1 + \frac{(f-y)^2}{\nu\sigma_n^2}\right)^{-\frac{\nu+1}{2}}, \quad Z = \frac{\Gamma\left(\frac{\nu+1}{2}\right)}{\Gamma\left(\frac{\nu}{2}\right)\sqrt{\nu\pi\sigma_n^2}}
$$

4.8.1 Laplace's Approximation

For the mode fitting procedure, we need derivatives up to third order; the hyperparameter derivatives at the mode require some mixed derivatives. All in all, using $r = y - f$, we have

$$
\begin{aligned}
\ln p(y|f) &= \ln\Gamma\left(\frac{\nu+1}{2}\right) - \ln\Gamma\left(\frac{\nu}{2}\right) - \frac{1}{2}\ln\nu\pi\sigma_n^2 - \frac{\nu+1}{2}\ln\left(1 + \frac{r^2}{\nu\sigma_n^2}\right) \\
\frac{\partial \ln p}{\partial f} &= (\nu+1)\frac{r}{r^2 + \nu\sigma_n^2} \\
\frac{\partial^2 \ln p}{(\partial f)^2} &= (\nu+1)\frac{r^2 - \nu\sigma_n^2}{(r^2 + \nu\sigma_n^2)^2} \\
\frac{\partial^3 \ln p}{(\partial f)^3} &= 2(\nu+1)\frac{r^3 - 3r\nu\sigma_n^2}{(r^2 + \nu\sigma_n^2)^3} \\
\frac{\partial \ln p}{\partial \ln \nu} &= \frac{\partial Z}{\partial \ln \nu} - \frac{\nu}{2}\ln\left(1 + \frac{r^2}{\nu\sigma_n^2}\right) + \frac{\nu+1}{2}\cdot\frac{r^2}{r^2 + \nu\sigma_n^2} \\
\frac{\partial Z}{\partial \ln \nu} &= \frac{\nu}{2}\frac{d\ln\Gamma\left(\frac{\nu+1}{2}\right)}{d\ln\nu} - \frac{\nu}{2}\frac{d\ln\Gamma\left(\frac{\nu}{2}\right)}{d\ln\nu} - \frac{1}{2} \\
\frac{\partial^3 \ln p}{(\partial \ln \nu)(\partial f)^2} &= \nu\frac{r^2(r^2 - 3(\nu+1){\sigma_n}^2) + \nu\sigma_n^2}{(r^2 + \nu\sigma_n^2)^3} \\
\frac{\partial \ln p}{\partial \ln \sigma_n} &= (\nu+1)\frac{r^2}{r^2 + \nu\sigma_n^2} - 1 \\
\frac{\partial^3 \ln p}{(\partial \ln \sigma_n)(\partial f)^2} &= 2\nu\sigma_n^2(\nu+1)\frac{\nu\sigma_n^2 - 3r^2}{(r^2 + \nu\sigma_n^2)^3}
\end{aligned}
$$

4.9 Cumulative Logistic Likelihood

The likelihood has one hyperparameter (represented in the log domain), namely the standard deviation σ_n

$$
p(y|f) = Z \cdot \cosh^{-2}\left(\tau(f-y)\right), \ \tau = \frac{\pi}{2\sigma_n\sqrt{3}}, \ Z = \frac{\pi}{4\sigma_n\sqrt{3}}
$$

4.9.1 Laplace's Approximation

The following derivatives are needed where $\phi(x) \equiv \ln(\cosh(x))$

$$
\begin{aligned}
\ln p(y|f) &= \ln(\pi) - \ln(4\sigma_n\sqrt{3}) - 2\phi\left(\tau(f-y)\right) \\
\frac{\partial \ln p}{\partial f} &= 2\tau\phi'\left(\tau(f-y)\right) \\
\frac{\partial^2 \ln p}{(\partial f)^2} &= -2\tau^2\phi''\left(\tau(f-y)\right) \\
\frac{\partial^3 \ln p}{(\partial f)^3} &= 2\tau^3\phi'''\left(\tau(f-y)\right) \\
\frac{\partial^3 \ln p}{(\partial \ln \sigma_n)(\partial f)^2} &= 2\tau^2\left(2\phi''\left(\tau(f-y)\right) + \tau(f-y)\phi'''\left(\tau(f-y)\right)\right) \\
\frac{\partial \ln p}{\partial \ln \sigma_n} &= 2\tau(f-y)\phi'\left(\tau(f-y)\right) - 1
\end{aligned}
$$

5 Mean Functions

A mean function $m_\phi : \mathcal{X} \to \mathbb{R}$ (with hyperparameters ϕ) of a GP f is a scalar function defined over the whole domain $\mathcal{X}$ that computes the expected value $m(\mathbf{x}) = \mathbb{E}[f(\mathbf{x})]$ of f for the input $\mathbf{x}$.

5.1 Interface

In the `GPML` toolbox, a mean function $m : \mathcal{X} \to \mathbb{R}$ needs to implement evaluation $\mathbf{m} = m_\phi(\mathbf{X})$ and first derivatives $\mathbf{m}_i = \frac{\partial}{\partial \phi_i}\mathbf{m}$ with respect to the components i of the parameter $\phi \in \Phi$ as detailed below.

26 ⟨*meanFunctions.m* 26⟩≡

```
% mean functions to be use by Gaussian process functions. There are two
% different kinds of mean functions: simple and composite:
%
% simple mean functions:
%
%   meanZero      - zero mean function
%   meanOne       - one mean function
%   meanConst     - constant mean function
%   meanLinear    - linear mean function
%
% composite covariance functions (see explanation at the bottom):
%
%   meanScale     - scaled version of a mean function
%   meanPow       - power of a mean function
%   meanProd      - products of mean functions
%   meanSum       - sums of mean functions
%   meanMask      - mask some dimensions of the data
%
% Naming convention: all mean functions are named "mean/mean*.m".
%
%
% 1) With no or only a single input argument:
%
%    s = meanNAME  or  s = meanNAME(hyp)
%
% The mean function returns a string s telling how many hyperparameters hyp it
% expects, using the convention that "D" is the dimension of the input space.
% For example, calling "meanLinear" returns the string ’D’.
%
% 2) With two input arguments:
%
%    m = meanNAME(hyp, x)
%
% The function computes and returns the mean vector where hyp are the
% hyperparameters and x is an n by D matrix of cases, where D is the dimension
% of the input space. The returned mean vector is of size n by 1.
%
% 3) With three input arguments:
%
%    dm = meanNAME(hyp, x, i)
%
% The function computes and returns the n by 1 vector of partial derivatives
% of the mean vector w.r.t. hyp(i) i.e. hyperparameter number i.
%
```

```
% See also doc/usageMean.m.
%
⟨gpml copyright 5a⟩
```

5.2 Implemented Mean Functions

We offer simple and composite mean functions producing new mean functions $m(\mathbf{x})$ from existing mean functions $\mu_j(\mathbf{x})$. All code files are named according to the pattern mean/mean<NAME>.m for simple identification. This modular specification allows to define affine mean functions $m(\mathbf{x}) = c + \mathbf{a}^\top \mathbf{x}$ or polynomial mean functions $m(\mathbf{x}) = (c + \mathbf{a}^\top \mathbf{x})^2$. All currently available mean functions are summarised in the following table.

Simple mean functions $m(\mathbf{x})$			
<NAME>	Meaning	$m(\mathbf{x}) =$	Φ
Zero	mean vanishes always	0	$\emptyset$
One	mean equals 1	1	$\emptyset$
Const	mean equals a constant	c	$c \in \mathbb{R}$
Linear	mean linearly depends on $\mathbf{x} \in \mathcal{X} \subseteq \mathbb{R}^D$	$\mathbf{a}^\top \mathbf{x}$	$\mathbf{a} \in \mathbb{R}^D$

Composite mean functions $[\mu_1(\mathbf{x}), \mu_2(\mathbf{x}), ..] \mapsto m(\mathbf{x})$			
<NAME>	Meaning	$m(\mathbf{x}) =$	Φ
Scale	scale a mean	$\alpha\mu(\mathbf{x})$	$\alpha \in \mathbb{R}$
Sum	add up mean functions	$\sum_j \mu_j(\mathbf{x})$	$\emptyset$
Prod	multiply mean functions	$\prod_j \mu_j(\mathbf{x})$	$\emptyset$
Pow	raise a mean to a power	$\mu(\mathbf{x})^d$	$\emptyset$
Mask	act on components $I \subseteq [1, 2, .., D]$ of $\mathbf{x} \in \mathcal{X} \subseteq \mathbb{R}^D$ only	$\mu(\mathbf{x}_I)$	$\emptyset$

5.3 Usage of Implemented Mean Functions

Some code examples taken from doc/usageMean.m illustrate how to use simple and composite mean functions to specify a GP model.

Syntactically, a mean function mf is defined by

```
mn := 'func' | @func // simple

mf := {mn} | {mn, {param, mf}} | {mn, {mf, .., mf}} // composite
```

i.e., it is either a string containing the name of a mean function, a pointer to a mean function or one of the former in combination with a cell array of mean functions and an additional list of parameters.

27 ⟨*doc/usageMean.m* 27⟩≡

```
% demonstrate usage of mean functions
%
% See also meanFunctions.m.
%
⟨gpml copyright 5a⟩
clear all, close all
n = 5; D = 2; x = randn(n,D);            % create a random data set

% set up simple mean functions
m0 = {'meanZero'};  hyp0 = [];       % no hyperparameters are needed
m1 = {'meanOne'};   hyp1 = [];       % no hyperparameters are needed
mc = {@meanConst};  hypc = 2;  % also function handles are possible
ml = {@meanLinear}; hypl = [2;3];              % m(x) = 2*x1 + 3*x2

```

```
% set up composite mean functions
msc = {'meanScale',{m1}};      hypsc = [3; hyp1];      % scale by 3
msu = {'meanSum',{m0,mc,ml}};  hypsu = [hyp0; hypc; hypl];    % sum
mpr = {@meanProd,{mc,ml}};     hyppr = [hypc; hypl];      % product
mpo = {'meanPow',{3,msu}};     hyppo = hypsu;          % third power
mask = [0,1,0]; %   binary mask excluding all but the 2nd component
mma = {'meanMask',{mask,mpo{:}}}; hypma = hyppo;

% 0) specify mean function
% mean = m0;  hyp = hyp0;
% mean = msu; hyp = hypsu;
% mean = mpr; hyp = hyppr;
mean = mpo; hyp = hyppo;

% 1) query the number of parameters
feval(mean{:})

% 2) evaluate the function on x
feval(mean{:},hyp,x)

% 3) compute the derivatives w.r.t. to hyperparameter i
i = 2; feval(mean{:},hyp,x,i)
```

6 Covariance Functions

A covariance function $k_\psi : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ (with hyperparameters ψ) of a GP f is a scalar function defined over the whole domain $\mathcal{X}^2$ that computes the covariance $k(\mathbf{x}, \mathbf{x}') = \mathbb{V}[f(\mathbf{x}), f(\mathbf{x}')] = \mathbb{E}[(f(\mathbf{x}) - m(\mathbf{x}))(f(\mathbf{x}') - m(\mathbf{x}'))]$ of f between the inputs $\mathbf{x}$ and $\mathbf{x}'$.

6.1 Interface

Again, the interface is simple since only evaluation of the full covariance matrix $\mathbf{K} = k_\psi(\mathbf{X})$ and its derivatives $\mathbf{K}_i = \frac{\partial}{\partial \psi_i}\mathbf{K}$ as well as cross terms $\mathbf{k}_* = k_\psi(\mathbf{X}, \mathbf{x}_*)$ and $k_{**} = k_\psi(\mathbf{x}_*, \mathbf{x}_*)$ for prediction are required.

29 ⟨*covFunctions.m* 29⟩≡

```
% covariance functions to be use by Gaussian process functions. There are two
% different kinds of covariance functions: simple and composite:
%
% simple covariance functions:
%   covConst      - covariance for constant functions
%   covLIN        - linear covariance function
%   covLINard     - linear covariance function with ARD
%   covLINone     - linear covariance function with bias
%   covMaterniso  - Matern covariance function with nu=1/2, 3/2 or 5/2
%   covNNone      - neural network covariance function
%   covNoise      - independent covariance function (i.e. white noise)
%   covPeriodic   - smooth periodic covariance function (1d) with unit period
%   covPoly       - polynomial covariance function
%   covPPiso      - piecewise polynomial covariance function (compact support)
%   covRQard      - rational quadratic covariance function with ARD
%   covRQiso      - isotropic rational quadratic covariance function
%   covSEard      - squared exponential covariance function with ARD
%   covSEiso      - isotropic squared exponential covariance function
%   covSEisoU     - as above but without latent scale
%
% composite (meta) covariance functions (see explanation at the bottom):
%   covScale      - scaled version of a covariance function
%   covProd       - products of covariance functions
%   covSum        - sums of covariance functions
%   covADD        - additive covariance function
%   covMask       - mask some dimensions of the data
%
% special purpose (wrapper) covariance functions
%   covFITC       - to be used in conjunction with infFITC for large scale
%                   regression problems; any covariance can be wrapped by
%                   covFITC such that the FITC approximation is applicable
%
% Naming convention: all covariance functions are named "cov/cov*.m". A trailing
% "iso" means isotropic, "ard" means Automatic Relevance Determination, and
% "one" means that the distance measure is parameterized by a single parameter.
%
% The covariance functions are written according to a special convention where
% the exact behaviour depends on the number of input and output arguments
% passed to the function. If you want to add new covariance functions, you
% should follow this convention if you want them to work with the function gp.
% There are four different ways of calling the covariance functions:
%
% 1) With no (or one) input argument(s):
```

```
%
%   s = cov
%
% The covariance function returns a string s telling how many hyperparameters it
% expects, using the convention that "D" is the dimension of the input space.
% For example, calling "covRQard" returns the string '(D+2)'.
%
% 2) With two input arguments:
%
%   K = cov(hyp, x) equivalent to K = cov(hyp, x, [])
%
% The function computes and returns the covariance matrix where hyp are
% the hyperparameters and x is an n by D matrix of cases, where
% D is the dimension of the input space. The returned covariance matrix is of
% size n by n.
%
% 3) With three input arguments:
%
%   Ks  = cov(hyp, x, xs)
%   kss = cov(hyp, xs, 'diag')
%
% The function computes test set covariances; kss is a vector of self covariances
% for the test cases in xs (of length ns) and Ks is an (n by ns) matrix of cross
% covariances between training cases x and test cases xs.
%
% 4) With four input arguments:
%
%    dKi   = cov(hyp, x, [], i)
%    dKsi  = cov(hyp, x, xs, i)
%    dkssi = cov(hyp, xs, 'diag', i)
%
% The function computes and returns the partial derivatives of the
% covariance matrices with respect to hyp(i), i.e. with
% respect to the hyperparameter number i.
%
% Covariance functions can be specified in two ways: either as a string
% containing the name of the covariance function or using a cell array. For
% example:
%
%  cov = 'covRQard';
%  cov = {'covRQard'};
%  cov = {@covRQard};
%
% are supported. Only the second and third form using the cell array can be used
% for specifying composite covariance functions, made up of several
% contributions. For example:
%
%       cov = {'covScale', {'covRQiso'}};
%       cov = {'covSum', {'covRQiso','covSEard','covNoise'}};
%       cov = {'covProd',{'covRQiso','covSEard','covNoise'}};
%       cov = {'covMask',{mask,'covSEiso'}}
%  q=1; cov = {'covPPiso',q};
%  d=3; cov = {'covPoly',d};
%       cov = {'covADD',{[1,2],'covSEiso'}};
%       cov = {@covFITC, {@covSEiso}, u};  where u are the inducing inputs
%
% specifies a covariance function which is the sum of three contributions. To
% find out how many hyperparameters this covariance function requires, we do:
```

```
%
%   feval(cov{:})
%
% which returns the string '3+(D+1)+1' (i.e. the 'covRQiso' contribution uses
% 3 parameters, the 'covSEard' uses D+1 and 'covNoise' a single parameter).
%
% See also doc/usageCov.m.
%
⟨gpml copyright 5a⟩
```

6.2 Implemented Covariance Functions

Similarly to the mean functions, we provide a whole algebra of covariance functions $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ with the same generic name pattern `cov/cov<NAME>.m` as before.

Besides a long list of simple covariance functions, we also offer a variety of composite covariance functions as shown in the following table.

Simple covariance functions $k(\mathbf{x}, \mathbf{x}')$			
`<NAME>`	Meaning	$k(\mathbf{x}, \mathbf{x}') =$	$\boldsymbol{\psi}$
`Zero`	mean vanishes always	0	$\emptyset$
`Noise`	additive measurement noise	$\sigma_f{}^2\delta(\mathbf{x} - \mathbf{x}')$	$\ln \sigma_f$
`Const`	covariance equals a constant	$\sigma_f{}^2$	$\ln \sigma_f$
`LIN`	linear, $\mathcal{X} \subseteq \mathbb{R}^D$	$\mathbf{x}^\top \mathbf{x}'$	$\emptyset$
`LINard`	linear with diagonal weighting, $\mathcal{X} \subseteq \mathbb{R}^D$	$\mathbf{x}^\top \boldsymbol{\Lambda}^{-2} \mathbf{x}'$	$\{\ln \lambda_1, .., \ln \lambda_D\}$
`LINone`	linear with bias, $\mathcal{X} \subseteq \mathbb{R}^D$	$(\mathbf{x}^\top \mathbf{x}' + 1)/\ell^2$	$\ln \ell$
`Poly`	polynomial covariance, $\mathcal{X} \subseteq \mathbb{R}^D$	$\sigma_f^2 (\mathbf{x}^\top \mathbf{x}' + c)^d$	$\{\ln c, \ln \sigma_f\}$
`SEard`	full squared exponential, $\mathcal{X} \subseteq \mathbb{R}^D$	$\sigma_f^2 \exp\left(-\frac{1}{2}(\mathbf{x} - \mathbf{x}')^\top \boldsymbol{\Lambda}^{-2} (\mathbf{x} - \mathbf{x}')\right)$	$\{\ln \lambda_1, .., \ln \lambda_D, \ln \sigma_f\}$
`SEiso`	diagonal squared exponential, $\mathcal{X} \subseteq \mathbb{R}^D$	$\sigma_f^2 \exp\left(-\frac{1}{2\ell^2}(\mathbf{x} - \mathbf{x}')^\top (\mathbf{x} - \mathbf{x}')\right)$	$\{\ln \ell, \ln \sigma_f\}$
`SEisoU`	squared exponential, $\mathcal{X} \subseteq \mathbb{R}^D$	$\exp(-\frac{1}{2\ell^2} \mathbf{x}^\top \mathbf{x}')$	$\ln \ell$
`RQard`	rational quadratic, $\mathcal{X} \subseteq \mathbb{R}^D$	$\sigma_f^2 \left(1 + \frac{1}{2\alpha}(\mathbf{x} - \mathbf{x}')^\top \boldsymbol{\Lambda}^{-2} (\mathbf{x} - \mathbf{x}')\right)^{-\alpha}$	$\{\ln \lambda_1, .., \ln \lambda_D, \ln \sigma_f, \ln \alpha\}$
`RQiso`	rational quadratic, $\mathcal{X} \subseteq \mathbb{R}^D$	$\sigma_f^2 \left(1 + \frac{1}{2\alpha\ell^2}(\mathbf{x} - \mathbf{x}')^\top (\mathbf{x} - \mathbf{x}')\right)^{-\alpha}$	$\{\ln \ell, \ln \sigma_f, \ln \alpha\}$
`Materniso`	Matérn, $\mathcal{X} \subseteq \mathbb{R}^D$, $f_1(t) = 1$, $f_3(t) = 1 + t$, $f_5(t) = f_3(t) + \frac{t^2}{3}$	$\sigma_f^2 f_d(r_d) \exp(-r_d)$, $r_d = \sqrt{\frac{d}{\ell^2}(\mathbf{x} - \mathbf{x}')^\top (\mathbf{x} - \mathbf{x}')}$	$\{\ln \ell, \ln \sigma_f\}$
`NNone`	neural net, $\mathcal{X} \subseteq \mathbb{R}^D$, $f(\mathbf{x}) = 1 + \mathbf{x}^\top \boldsymbol{\Lambda}^{-2} \mathbf{x}$	$\sigma_f^2 \sin^{-1}\left(\frac{\mathbf{x}^\top \boldsymbol{\Lambda}^{-2} \mathbf{x}'}{\sqrt{f(\mathbf{x}) f(\mathbf{x}')}}\right)$	$\{\ln \ell, \ln \sigma_f\}$
`Periodic`	periodic, $\mathcal{X} \subseteq \mathbb{R}$	$\sigma_f^2 \exp\left(-\frac{2}{\ell^2} \sin^2[\frac{\omega}{2\pi}(\mathbf{x} - \mathbf{x}')]\right)$	$\{\ln \ell, \ln \omega, \ln \sigma_f\}$
`PPiso`	compact support, piecewise polynomial $f_v(r)$, $\mathcal{X} \subseteq \mathbb{R}$,	$\sigma_f^2 \max(0, 1 - r) \cdot f_v(r)$, $r = \frac{\lVert \mathbf{x} - \mathbf{x}' \rVert}{\ell}$, $j = \lfloor \frac{D}{2} \rfloor + v + 1$	$\{\ln \ell, \ln \sigma_f\}$
Composite covariance functions $[\kappa_1(\mathbf{x}, \mathbf{x}'), \kappa_2(\mathbf{x}, \mathbf{x}'), ..] \mapsto k(\mathbf{x}, \mathbf{x}')$			
`<NAME>`	Meaning	$k(\mathbf{x}, \mathbf{x}') =$	Φ
`Scale`	scale a covariance	$\alpha\kappa(\mathbf{x}, \mathbf{x}')$	$\alpha \in \mathbb{R}$
`Sum`	add up covariance functions	$\sum_j \kappa_j(\mathbf{x}, \mathbf{x}')$	$\emptyset$
`Prod`	multiply covariance functions	$\prod_j \kappa_j(\mathbf{x}, \mathbf{x}')$	$\emptyset$
`Mask`	act on components $I \subseteq [1, 2, .., D]$ of $\mathbf{x} \in \mathcal{X} \subseteq \mathbb{R}^D$ only	$\kappa(\mathbf{x}_I, \mathbf{x}'_I)$	$\emptyset$
`ADD`	additive, $\mathcal{X} \subseteq \mathbb{R}^D$, index degree set $\mathcal{D} = \{1, .., D\}$	$\sum_{d \in \mathcal{D}} \sigma_{f_d}^2 \sum_{\lvert I \rvert = d} \prod_{i \in I} \kappa(x_i, x'_i; \psi_i)$	$\{\psi_1, .., \psi_D, \ln \sigma_{f_1}, .., \ln \sigma_{f_{\lvert \mathcal{D} \rvert}}\}$

The additive covariance function $k(\mathbf{x}, \mathbf{x}')$ starts from a one-dimensional covariance function $\kappa(x_i, x'_i, \psi_i)$ acting on a single component $i \in [1, .., D]$ of $\mathbf{x}$. From that, we define covariance functions $\kappa_I(\mathbf{x}_I, \mathbf{x}_I) = \prod_{i \in I} \kappa(x_i, x'_i, \psi_i)$ acting on vector-valued inputs $\mathbf{x}_I$. The sums of exponential size can efficiently be computed using the Newton-Girard formulae. Samples functions drawn from a GP with additive covariance are additive functions. The number of interacting variables $\lvert I \rvert$ is a measure of how complex the additive functions are.

6.3 Usage of Implemented Covariance Functions

Some code examples taken from `doc/usageCov.m` illustrate how to use simple and composite covariance functions to specify a GP model.

Syntactically, a covariance function `cf` is defined by

```
cv := 'func' | @func // simple
```

```
cf := {cv} | {cv, {param, cf}} | {cv, {cf, .., cf}} // composite
```

i.e., it is either a string containing the name of a covariance function, a pointer to a covariance function or one of the former in combination with a cell array of covariance functions and an additional list of parameters.

32 ⟨*doc/usageCov.m* 32⟩≡

```
% demonstrate usage of covariance functions
%
% See also covFunctions.m.
%
⟨gpml copyright 5a⟩
clear all, close all
n = 5; D = 3; x = randn(n,D); xs = randn(3,D);  % create a data set

% set up simple covariance functions
cn  = {'covNoise'}; sn = .1;  hypn = log(sn);  % one hyperparameter
cc  = {@covConst};   sf = 2;  hypc = log(sf); % function handles OK
cl  = {@covLIN};              hypl = []; % linear is parameter-free
cla = {'covLINard'}; L = rand(D,1); hypla = log(L);  % linear (ARD)
clo = {@covLINone}; ell = .9; hyplo = log(ell);  % linear with bias
cp  = {@covPoly,3}; c = 2; hypp = log([c;sf]);   % third order poly
cga = {@covSEard};   hypga = log([L;sf]);       % Gaussian with ARD
cgi = {'covSEiso'};  hypgi = log([ell;sf]);    % isotropic Gaussian
cgu = {'covSEisoU'}; hypgu = log(ell);   % isotropic Gauss no scale
cra = {'covRQard'}; al = 2; hypra = log([L;sf;al]); % ration. quad.
cri = {@covRQiso};          hypri = log([ell;sf;al]);   % isotropic
cm  = {'covMaterniso',3}; hypm = log([ell;sf]);  % Matern class q=3
cnn = {'covNNone'}; hypnn = log([L;sf]);           % neural network
cpe = {'covPeriodic'}; om = 2; hyppe = log([ell;om;sf]); % periodic
ccc = {'covPPiso',2}; hypcc = hypm; % compact support poly degree 2

% set up composite covariance functions
csc = {'covScale',{cgu}};    hypsc = [log(3); hypgu];  % scale by 9
csu = {'covSum',{cn,cc,cl}}; hypsu = [hypn; hypc; hypl];      % sum
cpr = {@covProd,{cc,ccc}};   hyppr = [hypc; hypcc];       % product
mask = [0,1,0]; %   binary mask excluding all but the 2nd component
cma = {'covMask',{mask,cgi{:}}}; hypma = hypgi;
% additive based on SEiso using unary and pairwise interactions
cad = {'covADD',{[1,2],'covSEiso'}};

% 0) specify covariance function
cov = cma; hyp = hypma;

% 1) query the number of parameters
feval(cov{:})

% 2) evaluate the function on x
feval(cov{:},hyp,x)

% 3) evaluate the function on x and xs to get cross-terms
kss = feval(cov{:},hyp,xs,'diag')
Ks  = feval(cov{:},hyp,x,xs)

% 4) compute the derivatives w.r.t. to hyperparameter i
i = 1; feval(cov{:},hyp,x,[],i)
```