The GPML Toolbox version 4.2

Carl Edward Rasmussen & Hannes Nickisch

April 4, 2022

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.
6.3 Usage of Implemented Covariance Functions ........................................ 46

7 Hyperpriors ................................................................. 48
   7.1 Interface ......................................................................... 48
   7.2 Implemented Hyperpriors ...................................................... 50
   7.3 Usage of Implemented Hyperpriors .............................................. 50
1 Gaussian Process Training and Prediction

The \texttt{gpml} toolbox contains a single user function \texttt{gp} described in section\cite{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\cite{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 \texttt{mean}, \texttt{cov} and \texttt{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 is either empty or non-existent it represents zero hyperparameters. When working with FITC approximate inference, the inducing inputs \(x_u\) can also be treated as hyperparameters for some common stationary covariances.

**Hyperparameter Prior Distributions:** When optimising the marginal likelihood w.r.t. hyperparameters, it is sometimes useful to softly constrain the hyperparameters by means of prior knowledge. A prior is a probability distribution over individual or a group of hyperparameters, section\cite{7}.

**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\cite{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\cite{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\cite{6}.

Inference methods, see section\cite{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 \texttt{post} with the fields \texttt{post.alpha}, \texttt{post.sW} and \texttt{post.L}. Often, starting from the Gaussian prior \(p(f) = \mathcal{N}(f|m, K)\) the approximate posterior admits the form

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

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

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

```matlab
function [varargout] = gp(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 struct of column vectors of mean/cov/lik hyperparameters
% inf function specifying the inference method
% mean prior mean function
% cov prior covariance 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 struct of column vectors of partial derivatives of the negative
% log marginal likelihood w.r.t. mean/cov/lik hyperparameters
% 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
```

It offers facilities for training the hyperparameters of a GP model as well as predictions at unseen inputs as detailed in the following help.
Depending on the number of input parameters, \texttt{gp} knows whether it is operated in training or in prediction mode. The high level 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.

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

If there is more than a single output dimension in \texttt{y}, we call multiple instances of \texttt{gp} with shared hyperparameters and settings each computing the corresponding result with scalar output.
Next, we set some useful default values for empty arguments, and convert \texttt{inf} and \texttt{lik} to function handles and \texttt{mean} and \texttt{cov} to cell arrays if necessary. Initialize variables.

\begin{verbatim}
6a⟩⟨process input arguments⟩≡
1 if isempty(mean), mean = {@meanZero}; end % set default mean
2 if ischar(mean) || isa(mean, 'function_handle'), mean = {mean}; end % make cell
3 if isempty(cov), error('Covariance function cannot be empty'); end % no default
4 if ischar(cov) || isa(cov,'function_handle'), cov = {cov}; end % make cell
5 cstr = cov{1}; if isa(cstr,'function_handle'), cstr = func2str(cstr); end
6 if (strcmp(cstr,'covFITC') || strcmp(cstr,'apxSparse')) && isfield(hyp,'xu')
7 cov{3} = hyp.xu; %use hyp.xu
8 end
9 if isempty(inf), inf = {@infGaussLik}; end % set default inference method
10 if ischar(inf), inf = str2func(inf); end % convert into function handle
11 if ischar(inf) || isa(inf,'function_handle'), inf = {inf}; end % make cell
12 istr = inf{1}; if isa(istr,'function_handle'), istr = func2str(istr); end
13 if strcmp(istr,'infPrior')
14 istr = inf{2}; if isa(istr,'function_handle'), istr = func2str(istr); end
15 end
16 if isempty(lik), lik = {@likGauss}; end % set default lik
17 if ischar(lik) || isa(lik,'function_handle'), lik = {lik}; end % make cell
18 lstr = lik{1}; if isa(lstr,'function_handle'), lstr = func2str(lstr); end
19
20 D = size(x,2);
21 if strcmp(lstr,'likErf') || strcmp(lstr,'likLogistic') % only some inf* possible
22 D = 0; xg = cov{3}; p = numel(xg); for i=1:p, D = D+size(xg{i},2); end % dims
23 end
\end{verbatim}

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

\begin{verbatim}
6b⟩⟨check hyperparameters⟩≡
1 if ~isfield(hyp,'mean'), hyp.mean = []; end % check the hyp specification
2 if eval(feval(mean{:})) ~= numel(hyp.mean)
3 error('Number of mean function hyperparameters disagree with mean function')
4 end
5 if ~isfield(hyp,'cov'), hyp.cov = []; end
6 if eval(feval(cov{:})) ~= numel(hyp.cov)
7 error('Number of cov function hyperparameters disagree with cov function')
8 end
9 if ~isfield(hyp,'lik'), hyp.lik = []; end
10 if eval(feval(lik{:})) ~= numel(hyp.lik)
11 error('Number of lik function hyperparameters disagree with lik function')
12 end
\end{verbatim}

Inference is performed by calling the desired inference method \texttt{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.

\begin{verbatim}
6c⟩⟨inference⟩≡
1 try % call the inference method
2 % issue a warning if a classification likelihood is used in conjunction with
3 % labels different from +1 and -1
4 if strcmp(lstr,'likErf') || strcmp(lstr,'likLogistic')
5 if ~isstruct(y)
6 uy = unique(y);
7 if any( uy~=+1 & uy~=-1 )
8 warning('You try classification with labels different from \{+1,-1\}')
\end{verbatim}
9          end
10       end
11       end
12 if nargin>7 % compute marginal likelihood and its derivatives only if needed
13     if isstruct(y)
14       post = y; % reuse a previously computed posterior approximation
15     else
16       post = feval(inf{:}, hyp, mean, cov, lik, x, y);
17     end
18 end
19 else
20     if nargout<=1
21       [post nlZ] = feval(inf{:}, hyp, mean, cov, lik, x, y); dnlZ = {};
22     else
23       [post nlZ dnlZ] = feval(inf{:}, hyp, mean, cov, lik, x, y);
24     end
25 end
26 catch
27   msgstr = lasterr;
28   if nargin>7, error('Inference method failed [%s]', msgstr); else
29     warning('Inference method failed [%s] .. attempting to continue',msgstr)
30   varargout = {NaN, vec2any(hyp,zeros(numel(any2vec(hyp)),1))}; return % go on
31 end
32
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.

⟨compute test predictions⟩≡
1 alpha = post.alpha; L = post.L; sW = post.sW;
2 if issparse(alpha) % handle things for sparse representations
3   nz = alpha ~= 0; % determine nonzero indices
4 if issparse(L), L = full(L(nz,nz)); end % convert L and sW if necessary
5 if issparse(sW), sW = full(sW(nz)); end
6 else nz = true(size(alpha,1),1); end % non-sparse representation
7 if isempty(L) % in case L is not provided, we compute it
8   K = feval(cov{:}, hyp.cov, x(nz,:));
9   L = chol(eye(sum(nz))+sW*sW'.*K);
10 end
11 %verify whether L contains valid Cholesky decomposition or something different
12 Lchol = isnumeric(L) && all(all(tril(L,-1)==0)&diag(L)'>0&isreal(diag(L))');
13 ns = size(xs,1); % number of data points
14 if strncmp(cstr,'apxGrid',7), xs = apxGrid('idx2dat',cov{3},xs); end % expand
15 nperbatch = 1000; % number of data points per mini batch
16 nact = 0; % number of already processed test data points
17 ymu = zeros(ns,1); ys2 = ymu; fmu = ymu; fs2 = ymu; lp = ymu; % allocate mem
18 while nact<ns % process minibatches of test cases to save memory
19     id = (nact+1):min(nact+nperbatch,ns); % data points to process
20     ⟨make predictions⟩
21     nact = id(end); % set counter to index of last processed data point
22 end
23 if nargin<9
24   varargout = {ymu, ys2, fmu, fs2, [], post}; % assign output arguments
25 else
26   varargout = {ymu, ys2, fmu, fs2, lp, post};
27 end
In every iteration of the above loop, we compute the predictions for all test points of the batch.

1. \( k_{ss} = \text{feval}(\text{cov}(:,), \text{hyp.cov}, \text{xs(id,:)}, \text{'diag'}) \); % self-variance
2. \( \text{if strcmp(cstr,'covFITC') || strcmp(cstr,'apxSparse')} \) % cross-covariances
3. \( K_{s} = \text{feval}(\text{cov}(:,), \text{hyp.cov}, \text{x}, \text{xs(id,:)}) \); \( K_{s} = K_{s(nz,:)} \); % res indep. of x
4. else
5. \( K_{s} = \text{feval}(\text{cov}(:,), \text{hyp.cov}, \text{x(nz,:)}, \text{xs(id,:)}) \); % avoid computation
6. end
7. \( m_{s} = \text{feval}(\text{mean}(:,), \text{hyp.mean}, \text{xs(id,:)}) \);
8. \( N = \text{size}(\text{alpha},2) \); % number of alphas (usually 1; more in case of sampling)
9. \( F_{mu} = \text{repmat}(m_{s},1,N) + K_{s} \ast \text{full}(\text{alpha(nz,:)}) \); % conditional mean \( f_{s} | f \)
10. \( f_{mu(id)} = \text{sum}(F_{mu},2)/N \); % predictive means
11. \( k_{ss} = \text{feval}(\text{cov}(:,), \text{hyp.cov}, \text{xs(id,:)}), \text{xs(id,:)}) \);
12. \( V = L'\text{(repmat}(sW,1,\text{length(id)})*K_{s}) \);
13. \( f_{s2}(id) = k_{ss} - \text{sum}(V*V,1)' \); % predictive variances
14. else % L is not triangular => use alternative parametrisation
15. \( f_{s2}(id) = k_{ss} + \text{sum}(K_{s} \ast \text{L}_{K_{s}},1)' \); % predictive variances
16. end
17. \( f_{s2}(id) = \text{max}(f_{s2}(id),0) \); % remove numerical noise i.e. negative variances
18. \( F_{s2} = \text{repmat}(f_{s2}(id),1,N) \); % we have multiple values in case of sampling
19. \( f_{s2}(id) = \text{max}(f_{s2}(id),0) \); % remove numerical noise i.e. negative variances
20. \( f_{mu(id)} = \text{sum}(\text{reshape}(\text{Lp},[],N),2)/N \); % log probability; sample averaging
21. \( y_{mu(id)} = \text{sum}(\text{reshape}(Y_{mu},[],N),2)/N \); % predictive mean \( y_{s} | y \) and ..
22. \( y_{s2(id)} = \text{sum}(\text{reshape}(Y_{s2},[],N),2)/N \); % .. variance
3 Inference Methods

Inference methods are responsible for computing the (approximate) posterior $p$, the (approximate) negative log marginal likelihood $\ln Z$ and its partial derivatives $\frac{\partial}{\partial \text{hyperparameters}} \ln Z$ w.r.t. the hyperparameters $\text{hyp}$. The arguments to the function are hyperparameters $\text{hyp}$, mean function $\text{mean}$, covariance function $\text{cov}$, likelihood function $\text{lik}$ and training data $\text{x}$ and $\text{y}$. Several inference methods are implemented and described this section.

% Inference methods: Compute the (approximate) posterior for a Gaussian process.
% Methods currently implemented include:
% infGaussLik Exact inference (only possible with Gaussian likelihood)
% infLaplace Laplace’s Approximation
% infEP Expectation Propagation
% infVB Variational Bayes Approximation
% infKL Kullback-Leibler optimal Approximation
% 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.
% infPrior Perform inference with hyperparameter prior.
% The interface to the approximation methods is the following:
% function [post n1Z d1Z] = inf...(hyp, mean, cov, lik, x, y)
% where:
% hyp is a struct of hyperparameters
% mean is the name of the mean function (see meanFunctions.m)
% 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
% n1Z is the returned value of the negative log marginal likelihood
% d1Z 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 $\text{inv}(\text{K})*(\text{mu-m})$, where K is the prior covariance matrix, m the prior mean, and mu the approx posterior mean
% sW is a (sparse or full column) vector containing diagonal of $\text{sqrt}(\text{W})$ the approximate posterior covariance matrix is $\text{inv}(\text{inv}(\text{K})+\text{W})$
% L is a (sparse or full) triangular matrix, $\text{L} = \text{chol}(sW*K*sW+\text{eye}(n))$, or a full matrix, $\text{L} = -\text{inv}(\text{K}+\text{inv}(\text{W}))$
% or a function $\text{L}(A)$ of a matrix A such that $-(\text{K}+\text{inv}(\text{W}))*\text{L}(A) = \text{A}$
% Usually, the approximate posterior to be returned admits the form $\text{N}((\text{mu}+\text{K})*\text{alpha}, \text{W}=\text{inv}(\text{inv}(\text{K})+\text{W}))$, where alpha is a vector and W is diagonal.
% For more information on the individual approximation methods and their
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 with Gaussian likelihood

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 \( q(f|D) = N(f|\mu, V) \) is exact.

```
function [post n1Z dn1L] = infGaussLik(hyp, mean, cov, lik, x, y, opt)
% 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. The function takes a specified covariance function (see covFunctions.m) and likelihood function (see likFunctions.m), and is designed to be used with gp.m.

if nargin<7, opt = []; end % make sure parameter exists
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);
[m, dm] = feval(mean{:}, hyp.mean, x); % evaluate mean vector and deriv
sn2 = exp(2*hyp.lik); W = ones(n,1)/sn2; % noise variance of likGauss
K = apx(hyp,cov,x,opt); % set up covariance approximation
[ldB2, solveKiW,dW,dhyp,post.L] = K.fun(W); % obtain functionality depending on W
alpha = solveKiW(y-m);
post.alpha = K.P(alpha); % return the posterior parameters
post.sW = sqrt(W); % sqrt of noise precision vector
if nargout>1 % do we want the marginal likelihood?
    n1Z = (y-m)'*alpha/2 + ldB2 + n*log(2*pi*sn2)/2; % -log marginal likelihood
end
if nargout>2 % do we want derivatives?
    dn1Z = dhypr(alpha); dn1L.mean = -dm(alpha);
    dn1L.lik = -sn2*(alpha'*alpha) - 2*sum(dW)/sn2 + n;
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 \( \inf/\inf \text{Laplace.m} \).

More concretely, the mean of the posterior \( q(f|\mathcal{D}) = \mathcal{N}(f|\mu, V) \) is – defining \( \ell_i(f_i) = \ln p(y_i|f_i) \) and \( \ell(f) = \sum_{i=1}^{n} \ell_i(f_i) \) – given by

\[
\mu = \arg \min_f \phi(f), \quad \text{where} \quad \phi(f) = \frac{1}{2} (f - m)^\top K^{-1} (f - m) - \ell(f) = -\ln[p(f)p(y|f)],
\]

which we abbreviate by \( \mu \leftarrow \mathcal{L}(\ell) \). The curvature \( \frac{\partial^2 \phi}{\partial f^2} = K^{-1} + W \) with \( W_{i,i} = -\frac{\partial^2}{\partial \tau_i^2} \ln p(y_i|f_i) \) 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 \( Z \approx Z_{\text{LA}} = \int \hat{\phi}(f)df \) where we use the 2nd order Taylor expansion at the mode \( \mu \) given by \( \hat{\phi}(f) = \phi(\mu) + \frac{1}{2}(f - \mu)^\top V^{-1}(f - \mu) \approx \phi(f) \).

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

\[
d_k = \frac{\partial}{\partial \nu_i} \frac{\partial^k}{\partial f_k} \log p(y|f), \quad k = 0, 1, 2
\]

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

- \([d0, d1, d2, d3] = \text{lik(hyp, y, f, [], ’infLaplace’)}\)

and

- \([d0, d1, d2] = \text{lik(hyp, y, f, [], ’infLaplace’, i)}\)

return exactly these values.

3.3 Expectation Propagation

The basic idea of Expectation Propagation (EP) as implemented in \( \text{inf/infEP.m} \) 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 natural parameters \( \nu_i, \tau_i \) such that the following identity holds:

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

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

\[
\nu_i, \tau_i = \arg \min_{\nu, \tau} \text{KL}[q_{-i}(f)p(y_i|f_i)/Z_{p,i}||q_{-i}(f)t(f_i; \nu, \tau)/Z_{t,i}].
\]

In order to apply the moment matching steps in a numerically safe way, EP requires the derivatives of the expectations w.r.t. the Gaussian mean parameter \( \mu \)

\[
d_k = \frac{\partial^k}{\partial \mu^k} \log \int p(y|f)\mathcal{N}(f|\mu, \sigma^2) df, \quad k = 0, 1, 2
\]
and the \( i \)th likelihood hyperparameter \( \rho_i \)

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

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

- \([d0, d1, d2] = \text{lik}(\text{hyp}, y, \mu, s^2, '\text{infEP}')\)

and

- \(d = \text{lik}(\text{hyp}, y, \mu, s^2, '\text{infEP}', i)\).

### 3.4 Kullback Leibler Divergence Minimisation

Another well known approach to approximate inference implemented \text{inf/infKL.m} in attempts to directly find the closest Gaussian \( q(f|\mathcal{D}) = N(f|\mu, V) \) to the exact posterior \( p(f|\mathcal{D}) \) w.r.t. to some proximity measure or equivalently to maximise a lower bound \( Z(\mu, V) \) to the marginal likelihood \( Z \) as described in Nickisch & Rasmussen \textit{Approximations for Binary Gaussian Process Classification}, JMLR, 2008. In particular, one minimises \( KL(N(f|\mu, V)||p(f|\mathcal{D})) \) which amounts to minimising \(-\log Z(\mu, V)\) as defined by:

\[
-\log Z = -\log \int p(f)p(y|f) df = -\log \int q(f|\mathcal{D}) \frac{p(f)}{q(f|\mathcal{D})} p(y|f) df \\
\]

\[
\leq \int q(f|\mathcal{D}) \ln \frac{p(f|\mathcal{D})}{q(f|\mathcal{D})} df - \int q(f|\mathcal{D}) \ln p(y|f) df =: -\log Z(\mu, V)
\]

\[
= KL(N(f|\mu, V)||N(f|m, K)) - \sum_{i=1}^{n} \int N(f_i|\mu_i, v_{ii}) \ln p(y_i|f_i) df_i, \ v_{ii} = |V|_{ii}
\]

\[
= \frac{1}{2} \left( \text{tr}(VK^{-1} - I) - \ln |VK^{-1}| \right) + \frac{1}{2} (\mu - m)^\top K^{-1} (\mu - m) - \sum_{i=1}^{n} \ell_{KL}(\mu_i, v_{ii})
\]

where \( \ell_{KL}(\mu_i) = \int N(f_i|\mu_i, v_{ii}) \ell_i(f_i) df_i \) is the convolution of the log likelihood \( \ell_i \) with the Gaussian \( N \) and \( v = \text{dg}(V) \). Equivalently, one can view \( \ell_{KL} \) as a smoothed version of \( \ell \) with univariate smoothing kernel \( N \).

From Challis & Barber \textit{Concave Gaussian Variational Approximations for Inference in Large Scale Bayesian Linear Models}, AISTATS, 2011 we know that the mapping \( (\mu, L) \mapsto -\log Z(\mu, L^\top L) \) is jointly convex whenever the likelihoods \( f_i \mapsto \mathbb{P}(y_i|f_i) \) are log concave. In particular, this implies that every \( (\mu_i, s_i) \mapsto -\ell_{KL}(\mu_i, s_i^2) \) is jointly convex.

We use an optimisation algorithm similar to EP (section 3.3) where we minimise the local KL-divergence the other way round \( \mu_i, s_i = \arg\min_{\mu_i, s_i} KL[N(f|\mu, s^2)||\ell_i(f)] \). This view was brought forward by Tom Minka \textit{Convex Divergence measures and message passing}, MSR-TR, 2005. The KL minimisation constitutes a jointly convex 2d optimisation problem solved by \text{klmin} using a scaled Newton approach which is included as a sub function in \text{inf/infKL.m}. The smoothed likelihood \( \ell_{KL}(\mu_i, v_{ii}) \) is implemented as a meta likelihood in \text{likKL}; it uses Gaussian-Hermite quadrature to compute the required integrals. Note that – as opposed to EP – Gaussian-Hermite quadrature is appropriate since we integrate against the \( \mathbb{P}(y_i|f_i) \) (which can be well approximated by a polynomial) instead of \( \mathbb{P}(y_i|f_i) \) itself. The algorithm is – again unlike EP – provably convergent for log-concave likelihoods (e.g. \text{likGauss}, \text{likLaplace}, \text{likSech2}, \text{likLogistic}, \text{likPoisson}) since it can be regarded as coordinate descent with guaranteed decrease in the objective in every step. Due to the complex update computations, \text{infKL} can be quite slow although it has the same \( O(n^3) \) asymptotic complexity as EP and Laplace.
3.5 Variational Bayes

One can drive the bounding even further by means of local quadratic lower bounds to the log likelihood $\ell(f) = \ln p(y|f)$. Suppose that we use a super-Gaussian likelihood $p(y|f)$ i.e. likelihoods that can be lower bounded by Gaussians of any width $w$ (e.g. likLaplace, likT, likLogistic, likSech2). Formally, that means there are $b, z \in \mathbb{R}$ such that

$$\rho(f) = \ln p(y|f - z) - bf$$

is symmetric and $\sqrt{t} \mapsto \rho(t)$ is a convex function for all $f \geq 0$. As a result, we obtain the following exact representation of the likelihood

$$\ell(f) = \ln p(y|f) = \max_{w > 0} \left( (b + wz)f - \frac{wf^2}{2} - \frac{1}{2} h(\gamma) \right),$$

which can be derived by convex duality and assuming the likelihoods to be super-Gaussian. Details 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].

The bottom line is that we can treat the variational bounding as a sequence of Laplace approximations with the “variational Bayes” log likelihood

$$\ell_{\text{VB}}(f_i) = \ell(g_i) + b_i(f_i - g_i), \quad g = \text{sgn}(f - z) \odot \sqrt{(f - z)^2 + v + z}$$

instead of the usual likelihood $\ell(f_i) = \ln p(y|f_i)$ i.e. we solve $\mu \leftarrow \mathcal{L}(\ell_{\text{VB}})$ instead of $\mu \leftarrow \mathcal{L}(\ell)$. See section 3.2. In the code of inf/infVB.m, the likelihood is implemented in the function likVB.

At the end, the optimal value of $W$ can be obtained analytically via $w_i = |b_i - \ell'(g_i)|/|g_i - z_i|$.

For the minimisation in inf/infVB.m, we use a provably convergent double loop algorithm, where in the inner loop a nonlinear least squares problem (convex for log-concave likelihoods) is solved using inf/infLaplace.m such that $\mu \leftarrow \mathcal{L}(\ell_{\text{VB}})$ and in the outer loop, we compute $v \leftarrow dg((K^{-1} + W)^{-1})$. The only requirement to the likelihood function is that it returns the values $z$ and $b$ required by the bound which are delivered by the call (see section 4)

- $[b, z] = \text{lik}(\text{hyp, y, [ ], ga, \text{'infVB'}})$

The negative marginal likelihood upper bound $-\ln Z_{\text{VB}}$ is obtained by integrating the prior times the exact representation of the likelihood

$$p(y|f) = \max_{\gamma > 0} q(y|f, \gamma), \quad q(y|f, \gamma) = N(f|v, \gamma) \exp \left( -\frac{h(\gamma)}{2} - \frac{v^2}{2\gamma} \right) \sqrt{2\pi\gamma}, \quad \gamma = \frac{1}{w}, \quad v = b\gamma + z$$

w.r.t. the latent variables $f$ yielding

$$-\ln Z_{\text{VB}} = -\ln \int N(f|m, K) \prod_{i=1}^{n} q_i(y_i|f_i, \gamma_i) df$$

$$= -\ln N(m|v, K + \Gamma) + \frac{1}{2} \left( h(\gamma) - w^T v^2 - 1^T \ln 2\pi\gamma \right).$$

3.6 Compatibility Between Inference Methods and Covariance Approximations

Another kind of approximation is needed to render an inference method scalable. We have two approximation schemes which in fact approximate the covariance to make it amenable to large number of training data points. The following table shows the compatibility between some inference methods and two major groups of covariance approximations we will discuss in the next two sections.
### 3.7 Sparse Covariance 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 or 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[1]. 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

\[
K \approx \tilde{K} = Q + G, \ G = \text{diag}(g), \ g = \text{diag}(K - Q), \ Q = K_u^T Q_{uu}^{-1} K_u, \ Q_{uu} = K_{uu} + \sigma^2_{nu} I,
\]

where \( \sigma_{nu} \) 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 \( Q \). Internally, the necessary covariance evaluations are performed by a meta covariance function \( \text{cov/apxSparse.m} \). The toolbox offers FITC versions for regression with Gaussian likelihood \( \text{inf/infGaussLik.m} \), as well as for Laplace’s approximation \( \text{inf/infLaplace.m} \).

The user can decide whether to treat the inducing inputs \( u \) as fixed or as hyperparameters. The latter allows to adjust the inducing inputs \( u \) w.r.t. the marginal likelihood. As detailed in the documentation of \( \text{inf/apx.m} \), \( u \) is treated as fixed if it is passed as the 2nd parameter of \( \text{apxSparse(co, xu, ..)} \). If the hyperparameter structure \( \text{hyp} \) contains a field \( \text{hyp.xu} \) in inference method calls such as \( \text{infGaussLik(hyp, ..)} \) or inference/prediction calls like \( \text{gp(hyp, @infGaussLik, ..)} \) the inducing inputs \( u \) are treated as hyperparameters and can be optimised. See \( \text{doc/demoSparse.m} \) for an illustration.

### 3.8 Grid-Based Covariance Approximations

Another way to bring down computational costs is to take advantage of grid structure \( x \). For example, in geostatistics or image processing, the training data \( x \in \mathbb{R}^{n \times D} \) could be a complete 2d lattice of size \( n_1 \times n_2 \) as given by the axes \( g_1 \in \mathbb{R}^{n_1} \), \( g_2 \in \mathbb{R}^{n_2} \) so that \( n = N = n_1 \cdot n_2 \), \( D = 2 \) and \( x = [\text{vec}(g_1^T), \text{vec}(1g_2^T)] \). In general, a \( p \)-dimensional grid \( U \in \mathbb{R}^{N \times D} \) is specified by a set of axis matrices \( \{g_i \in \mathbb{R}^{n_i \times D_1}\}_{i=1..p} \) so that \( N = \prod_{i=1}^p n_i \) and \( D = \sum_{i=1}^p D_1 \) where the axes do not need to be 1d nor do their components need to be sorted. As a consequence, \( U \) represents a Cartesian product of its axes \( U = g_1 \times g_2 \times \ldots \times g_p \). The \( \text{cov/apxGrid.m} \) covariance function represents a Kronecker product covariance matrix

\[
K_{U,U} = K_p \otimes \ldots \otimes K_2 \otimes K_1
\]

whose factorisation structure is given by the grid \( x_U \). The gain in computational efficiency is due to the fact that matrix-vector product, determinant, inverse and eigenvalue computations decompose so that many operations with an overall cost of \( O(N^3) \) now only cost \( O(\sum_{i=1}^p n_i^3) \).

---

For off-grid data points, we can still take advantage of the computational properties of a grid-based covariance matrix $K_{U,U}$ via the structured kernel interpolation (SKI) framework aka KISS-GP by Wilson and Nickisch\textsuperscript{3} with extensions\textsuperscript{4}. Here, the $n \times n$ covariance $K$ is obtained from the $N \times N$ grid covariance $K_{U,U}$ by interpolation $K \approx W_X K_{U,U} W_X^T$, where $K_{U,U}$ is a covariance matrix formed by evaluating the user-specified kernel over a set of latent inducing inputs $U$, with locations that have been chosen to create algebraic structure in $K_{U,U}$ that we can exploit for efficiency. Here, the interpolation matrix $W_X \in \mathbb{R}^{n \times N}$ is extremely sparse; i.e., for local cubic interpolation $W_X$ contains only $4^D$ nonzeros per row, where $D$ is the data dimension. In addition $W_X$ is row-normalised $1_n = W_X 1_N$. The structure in $K_{U,U}$ alongside the sparsity of $W_X$, allows for very fast MVMs with the SKI approximate covariance matrix $K$ over the inputs $x$ enabling fast inference and prediction.

Internally, we use a meta covariance function $cov/apxGrid.m$ to represent the Kronecker covariance matrix and a Gaussian regression inference method $inf/infGaussLik.m$. We also support incomplete grids where $n < N$. A good starting point is Yunus Saatçi’s PhD thesis\textsuperscript{5}. For incomplete grids, we use the interpolation-based extensions by Wilson et al.\textsuperscript{6} where conjugate gradients and a determinant approximations are used. See doc/demoGrid1d.m and doc/demoGrid2d.m for an illustration. We also offer non-Gaussian likelihoods as described by Seth Flaxman\textsuperscript{6} so that $inf/infLaplace.m$ can be used.

### 3.9 State Space Representation of GPs

GP models with covariance functions with a Markovian structure can be transformed into equivalent discrete state space models where inference can be done in linear time $O(n)$. Exact models can be derived for sum, product, linear, noise, constant, Matérn (half-integer), Ornstein–Uhlenbeck, and Wiener covariance functions. Other common covariance functions can be approximated by their Markovian counterparts, including squared exponential, rational quadratic, and periodic covariance functions.

A state model describes the evolution of a dynamical system at different time instances $t_i$, $i = 1, 2, \ldots$ by

$$ f_i \sim P(f_i|f_{i-1}), \quad y_i \sim P(y_i|f_i), $$

where $f_i := f(t_i) \in \mathbb{R}^d$ and $f_0 \sim P(f_0)$ with $f_i$ being the latent (hidden/unobserved) variable and $y_i$ being the observed variable. In continuous time, a simple dynamical system able to represent many covariance functions is given by the following linear time-invariant stochastic differential equation:

$$ \dot{f}(t) = F f(t) + L w(t), \quad y_i = H f(t_i) + \epsilon_i, $$

where $w(t)$ is an $s$-dimensional white noise process, the measurement noise $\epsilon_i \sim N(0, \sigma_i^2)$ is Gaussian, and $F \in \mathbb{R}^{d \times d}$, $L \in \mathbb{R}^{d \times s}$, $H \in \mathbb{R}^{1 \times d}$ are the feedback, noise effect, and measurement matrices, respectively. The initial state is distributed according to $f_0 \sim N(0, P_0)$.

The latent GP is recovered by $f(t) = H f(t)$ and $w(t) \in \mathbb{R}^s$ is a multivariate white noise process with spectral density matrix $Q_c \in \mathbb{R}^{s \times s}$. For discrete values, this translates into

$$ f_i \sim N(A_{i-1} f_{i-1}, Q_{i-1}), \quad y_i \sim P(y_i|HLf_i), $$

\textsuperscript{3}Kernel Interpolation for Scalable Structured Gaussian Processes, ICML, 2015
\textsuperscript{4}Thoughts on Massively Scalable Gaussian Processes, TR, 2015.
\textsuperscript{5}Scalable Inference for Structured Gaussian Process Models, University of Cambridge, 2011
\textsuperscript{6}Fast Kernel Learning for Multidimensional Pattern Extrapolation, NIPS, 2014
\textsuperscript{6}Fast Kronecker inference in Gaussian processes with non-Gaussian likelihoods, ICML, 2015
Since the matrix exponential \( \psi \) variance matrices \( Q \)
are internally, we use a meta covariance function
and Kalman smoothing as detailed in Algorithms 1 and 2.
From the discrete set of matrices, all the necessary computations can be done using Kalman filtering
by convolution interpolation.

In practice, the evaluation of the \( n \Delta t \)
with \( f_0 \sim N(0, P_0) \). The discrete-time matrices are
\[
    A_i = A_i|\Delta t_i| = e^{\Delta t_iF},
    Q_i = \int_0^{\Delta t_i} e^{(\Delta t_k - \tau)F} L Q_c L^T e^{(\Delta t_k - \tau)F^T} d\tau,
\]
where \( \Delta t_i = t_{i+1} - t_i \geq 0 \).

For stationary covariances \( k(t, t') = k(t - t') \), the stationary state is distributed by \( f_\infty \sim N(0, P_\infty) \)
and the stationary covariance can be found by solving the Lyapunov equation
\[
    \dot{P}_\infty = FP_\infty + P_\infty F^T + L Q_c L^T = 0,
\]
which leads to the identity \( Q_i = P_\infty - A_i P_\infty A_i^T \).

In practice, the evaluation of the \( n \) discrete-time transition matrices \( A_i = e^{\Delta t_i F} \) and the noise covariance matrices \( Q_i \) (in the stationary case) for different values of \( \Delta t_i \) is a computational challenge. Since the matrix exponential \( \psi : s \mapsto e^{sX} \) is smooth, its evaluation can be accurately approximated
by convolution interpolation.

From the discrete set of matrices, all the necessary computations can be done using Kalman filtering
and Kalman smoothing as detailed in Algorithms 1 and 2.

Internally, we use a meta covariance function \texttt{cov/apxState.m} to represent the state space represen-
A good starting point is Arno Solin’s PhD thesis\cite{Solin2016}. See doc/demoState.m for an illustration. We also offer non-Gaussian likelihoods\cite{Solin2018} so that inf/infLaplace.m and inf/infVB.m can be used. EP is not fully functional; we offer single-sweep EP aka assumed density filtering (ADF).

\begin{itemize}
\item \cite{Solin2016} Stochastic Differential Equation Methods for Spatio-Temporal Gaussian Process Regression, Aalto University, 2016
\item \cite{Solin2018} State Space Gaussian Processes with Non-Gaussian Likelihood, ICML, 2018
\end{itemize}
4 Likelihood Functions

A likelihood function \( p_{\rho}(y|f) \) (with hyperparameters \( \rho \)) is a conditional density \( \int p_{\rho}(y|f)dy = 1 \) defined for scalar latent function values \( f \) and outputs \( y \). In the GPML toolbox, we use iid. likelihoods \( p_{\rho}(y|f) = \prod_{i=1}^{n} p_{\rho}(y_{i}|f_{i}) \). 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 \( x_{s} \) conditioned on the data \( D = (X,y) \) (as implemented in \texttt{gp.m}) consists of the predictive mean \( \mu_{y_{*}} \) and variance \( \sigma_{y_{*}}^{2} \), which are computed from the the latent marginal moments \( \mu_{f_{*}}, \sigma_{f_{*}}^{2} \), i.e. the Gaussian marginal approximation \( \mathcal{N}(f_{*}|\mu_{f_{*}},\sigma_{f_{*}}^{2}) \) via

\[
p(y_{*}|D,x_{s}) = \int p(y_{*}|f_{*})p(f_{*}|D,x_{s})df_{*} \approx \int p(y_{*}|f_{*})\mathcal{N}(f_{*}|\mu_{f_{*}},\sigma_{f_{*}}^{2})df_{*}.
\]

(3)

The moments are given by \( \mu_{y_{*}} = \int y_{*}p(y_{*}|D,x_{s})dy_{*} \) and \( \sigma_{y_{*}}^{2} = \int (y_{*} - \mu_{y_{*}})^{2}p(y_{*}|D,x_{s})dy_{*} \). The likelihood call

- \([lp,ymu,ys2] = \text{lik}(\text{hyp}, [], \text{fmu}, \text{fs2})\)

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

- \([lp,ymu,ys2] = \text{lik}(\text{hyp}, y, \text{fmu}, \text{fs2})\)

where \( lp \) contains the number \( \ln p_{y_{*}} \).

Using the moments of the likelihood \( \mu(f_{*}) = \int y_{*}p(y_{*}|f_{*})dy_{*} \) and \( \sigma^{2}(f_{*}) = \int (y_{*} - \mu(f_{*}))^{2}p(y_{*}|f_{*})dy_{*} \), we obtain for the predictive moments the following (exact) expressions

\[
\mu_{y_{*}} = \int \mu(f_{*})p(f_{*}|D,x_{s})df_{*}, \text{ and }
\]

\[
\sigma_{y_{*}}^{2} = \int \left[ \sigma^{2}(f_{*}) + (\mu(f_{*}) - \mu_{y_{*}})^{2} \right] p(f_{*}|D,x_{s})df_{*}.
\]

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

\[
p_{y_{*}} \begin{cases} 
\pi_{y_{*}} & y_{*} = +1 \\
1 - \pi_{y_{*}} & y_{*} = -1 
\end{cases}
\]

\[
\mu_{y_{*}} = \sum_{y_{*} = \pm 1} y_{*}p(y_{*}|D,x_{s}) = 2 \cdot \pi_{y_{*}} - 1 \in [-1,1], \text{ and }
\]

\[
\sigma_{y_{*}}^{2} = \sum_{y_{*} = \pm 1} (y_{*} - \mu_{y_{*}})^{2}p(y_{*}|D,x_{s}) = 4 \cdot \pi_{y_{*}}(1 - \pi_{y_{*}}) \in [0,1].
\]

2. The continuous case for homoscedastic likelihoods depending on \( r_{*} = y_{*} - f_{*} \) only and having noise variance \( \sigma^{2}(f_{*}) = \sigma_{n}^{2} \) is also simple since the identity \( p(y_{*}|f_{*}) = p(y_{*} - f_{*}|0) \) allows to substitute \( y_{*} \leftarrow y_{*} + f_{*} \) yielding \( \mu(f_{*}) = f_{*} + \int y_{*}p(y_{*}|0)dy_{*} \) and assuming \( \int y_{*}p(y_{*}|0)dy_{*} = 0 \) we arrive at

\[
\mu_{y_{*}} = \mu_{f_{*}}, \text{ and }
\]

\[
\sigma_{y_{*}}^{2} = \sigma_{f_{*}}^{2} + \sigma_{n}^{2}.
\]
3. The generalised linear model (GLM) case is also feasible. Evaluation of the predictive distribution is done by quadrature

\[ p(y_* = \int p(y_*|f_*)p(f_*|D, x_*)df_* \approx \int p(y_*|f_*)N(f_*|\mu_{f_*}, \sigma^2_{f_*})df_*. \]

For GLMs the mean is given by \( \mu(f_*) = g(f_*) \) and the variance is usually given by a simple function of the mean \( \sigma^2(f_*) = v(g(f_*)) \), hence we use Gaussian-Hermite quadrature with \( N(f_*|\mu_{f_*}, \sigma^2_{f_*}) \approx p(f_*|D, x_*) \) to compute

\[ \mu_{y_*} = \int g(f_*)p(f_*|D, x_*)df_*, \text{ and} \]

\[ \sigma^2_{y_*} = \int \left[ v(g(f_*)) + (g(f_*) - \mu_{y_*})^2 \right] p(f_*|D, x_*)df_* \neq v(\mu_{y_*}). \]

4. Finally the warped Gaussian likelihood predictive distribution with strictly monotonically increasing warping function \( g \) is given by the expression

\[ p(y_*|D, x_*) = g'(y_*)N(g(y_*)|\mu_{f_*}, \sigma^2_n + \sigma^2_{f_*}) \]

so that the predictive moments can be computed by Gaussian-Hermite quadrature.

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.

\[
\langle \text{likFunctions.m} \rangle \equiv
\]

1 \% likelihood functions are provided to be used by the gp.m function:
2 \%
3 \% likErf (Error function, classification, probit regression)
4 \% likLogistic (Logistic, classification, logit regression)
5 \% likUni (Uniform likelihood, classification)
6 \%
7 \% likGauss (Gaussian, regression)
8 \% likGaussWarp (Warped Gaussian, regression)
9 \% likGumbel (Gumbel likelihood for extremal values)
10 \% likLaplace (Laplacian or double exponential, regression)
11 \% likSech2 (Sech-square, regression)
12 \% likT (Student’s t, regression)
13 \%
14 \% likPoisson (Poisson regression, count data)
15 \% likNegBinom (Negativ binomial regression, count data)
16 \% likGamma (Nonnegative regression, positive data)
17 \% likExp (Nonnegative regression, positive data)
18 \% likInvGauss (Nonnegative regression, positive data)
19 \% likLogGauss (Nonnegative regression, positive data)
20 \% likBeta (Beta regression, interval data)
21 \%
22 \% likMix (Mixture of individual likelihood 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]

   \[ s = \text{lik} \text{ OR } s = \text{lik}(\text{hyp}) \]

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]

   \[ \text{lp} = \text{lik}(\text{hyp}, \text{y}, \text{mu}) \text{ OR } \text{lp, ymu, ys2} = \text{lik}(\text{hyp}, \text{y}, \text{mu}, \text{s2}) \]

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 \( p(f_*|x_*, y) \) as returned by an inference method. The predictive distribution \( p(y_*|x_*, y) \) is approximated by.

   \[ q(y_*) = \int N(f_*|\mu, s2) p(y_*|f_*) \, df_* \]

   \[ \text{lp} = \log( q(y) ) \text{ for a particular value of } y, \text{ if } s2 \text{ is } [] \text{ or } 0, \text{ this corresponds to } \log( p(y|\mu) ) \]

   \[ \text{ymu} \text{ and } \text{ys2} \text{ the mean and variance of the predictive marginal } q(y) \]

   \[ \text{note that these two numbers do not depend on a particular value of } y \]

   \[ \text{All vectors have the same size.} \]

3) With five or six input arguments, the fifth being a string [INFERENCE MODE]

   \[ [\text{varargout}] = \text{lik}(\text{hyp}, \text{y}, \text{mu}, \text{s2}, \text{inf}) \text{ OR } [\text{varargout}] = \text{lik}(\text{hyp}, \text{y}, \text{mu}, \text{s2}, \text{inf}, \text{i}) \]

There are three cases for \text{inf}, namely a) \text{infLaplace}, b) \text{infEP} and c) \text{infVB}. The last input \text{i}, refers to derivatives w.r.t. the \text{i}th hyperparameter.

a1) \[ [\text{lp}, \text{dlp}, \text{d2lp}, \text{d3lp}] = \text{lik}(\text{hyp}, \text{y}, \text{f}, [], \text{',infLaplace'}) \]

   \[ \text{lp, dlp, d2lp and d3lp correspond to derivatives of the log likelihood} \]

   \[ \text{lp} = \log( p(y|f) ) \text{ w.r.t. to the latent location } f. \]

   \[ \text{dlp} = \frac{\text{d} \log( p(y|f) )}{\text{df}} \]

   \[ \text{d2lp} = \frac{\text{d}^2 \log( p(y|f) )}{\text{df}^2} \]

   \[ \text{d3lp} = \frac{\text{d}^3 \log( p(y|f) )}{\text{df}^3} \]

a2) \[ [\text{lp}, \text{dlp}, \text{d2lp}, \text{d3lp}] = \text{lik}(\text{hyp}, \text{y}, \text{f}, [], \text{',infLaplace'}, \text{i}) \]

   \[ \text{return derivatives w.r.t. to the } \text{i}th \text{ hyperparameter} \]

b1) \[ [\text{lZ}, \text{dlZ}, \text{d2lZ}] = \text{lik}(\text{hyp}, \text{y}, \text{mu}, \text{s2}, \text{',infEP'}) \]

   \[ \text{let } Z = \int p(y|f) N(f|\mu, s2) \, df \text{ then} \]

   \[ \text{lZ} = \log(Z) \]

   \[ \text{dlZ} = \frac{\text{d} \log(Z)}{\text{dmu}} \]
% d2Z = \frac{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 = \frac{d \log(Z)}{d hyp_i}

% c1) [b,z] = lik(hyp, y, [], ga, 'infVB')
% ga is the variance of a Gaussian lower bound to the likelihood p(y|f).
% p(y|f) \geq \exp( b*(f+z) - (f+z)^2/(2*ga) - h(ga)/2 ) \propto N(f|b*ga-z,ga)
% The function returns the linear part b and z.

% 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.
%
% See also USAGELIK

help likFunctions
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 ρ.

| lik<NAME>    | regression y_i ∈ R | p_ρ(y_i|f_i) = | ρ =               |
|-----------------|---------------------|-----------------|------------------|
| Gauss           | Gaussian            | N(y_i|f_i, σ^2) = 1/(2πσ^2) exp (−(y_i−f_i)^2/(2σ^2)) | [ln σ]           |
| GaussWarp       | Warped Gaussian     | N(g_0(y_i|f_i, σ^2)g_0(y_i) = [θ_1,...,θ_n, ln σ]            |
| Gumbel          | Gumbel              | N(2/σ^6 exp(−z_i − e^{-z_i}), z_i = γ + s^2(y_i−f_i)/σ^6, | [ln σ]           |
| Sech2           | Sech-squared        | 2 cosh^2(τ(y_i−f_i)), τ = 2/σ√3 | [ln σ]           |
| Laplace         | Laplacian           | 1/(2b)^{1/2} exp (−|y_i−f_i|^1/2), b = σ/√2 | [ln σ]           |
| T               | Student’s t         | (y_i−f_i) 1/(1+y_i−f_i)^2 | [ln(ν−1), ln σ] |

| lik<NAME>    | classification y_i ∈ {±1} | p_ρ(y_i|f_i) = | ρ =               |
|-----------------|-----------------------------|-----------------|------------------|
| Erf             | Error function              | erf(y_i|f_i) | ∅               |
| Logistic        | Logistic function           | 1−exp(−y_i|f_i) | ∅               |
| Uni             | Label noise                 | 1/2            | ∅               |

| lik<NAME>    | count data y_i ∈ N | p_ρ(y_i|f_i) = | ρ =               |
|-----------------|---------------------|-----------------|------------------|
| Poisson         | Poisson            | μe^{−μ}/μ! | ∅               |
| NegBinom        | Negative Binomial   | μ^{r+1}/r!(r+μ) | [ln r]           |

| lik<NAME>    | nonnegative data y_i ∈ R \{0\} | p_ρ(y_i|f_i) = | ρ =               |
|-----------------|---------------------------------|-----------------|------------------|
| Weibull         | Weibull, y_i = Γ(1+1/κ) | κ/μy_i|y_i/μ|κ−1 | [ln κ]           |
| Gamma           | Gamma                          | μ^{−α} exp (−y_i|μ) | [ln α]           |
| Exp             | Exponential                    | μ^{−1} exp (−y_i|μ) | ∅               |
| InvGauss        | Inverse Gaussian              | √2πy_i | exp (−y_i^2/(2πy_i)) | [ln λ]           |

| lik<NAME>    | interval data y_i ∈ [0,1] | p_ρ(y_i|f_i) = | ρ =               |
|-----------------|----------------------------|-----------------|------------------|
| Beta            | Beta                          | Γ(Φ(1−y_i|1−μ))μ^{1−μ}(1−y_i)(1−μ)^{Φ−1} | [ln Φ]           |

| Composite       | likelihood functions | p_ρ(y_i|f_i) = | [ln α_1, ln α_2, ..]          |
|-----------------|---------------------|-----------------|------------------|
| Mixture         | Mixture             | Σ_1 α_i p_i(y_i|f_i) |               |

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

```matlab
lf := 'func' | @func // simple
lf := {lf} | {param, lf} | {lf, {lf, .., lf}} // 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.

```matlab
doc/usageLik.m
1 % demonstrate usage of likelihood functions
2 %
3 % See also likFunctions.m.
4 %
5 <gpm copyright> 5a
6 clear all, close all
7 n = 5; f = randn(n,1); % create random latent function values
8```
% 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);
u = 4; % number of degrees of freedom
lr3 = {'likT'}; hypr3 = [log(u-1); log(sn)];
lr4 = {'likMix',{lr0,lr1}}; hypr4 = [log([1,2]);hypr0;hypr1];

a = 1; % set up warped Gaussian with g(y) = y + a*sign(y).*y.^2
lr5 = {'likGaussWarp',[poly2]}; hypr5 = log([a;sn]);
lr6 = {'likGumbel','+'}; hypr6 = log(sn);

% set up Poisson/negative binomial regression
yp = fix(abs(f)) + 1;
lp0 = {likPoisson,'logistic'}; hypp0 = [];
lp1 = {likPoisson,'logistic2',0.1}; hypp1 = [];
lp2 = {likPoisson,'exp'}; hypp2 = [];
lp3 = {likNegBinom,'logistic2'}; hypp3 = [];

% set up other GLM likelihoods for positive or interval regression
lg1 = {likGamma,'logistic'}; al = 2; hyp.lik = log(al);
lg2 = {likInvGauss,'exp'}; lam = 1.1; hyp.lik = log(lam);
lg3 = {likBeta,'expexp'}; phi = 2.1; hyp.lik = log(phi);
lg4 = {likBeta,'logit'}; phi = 4.7; hyp.lik = log(phi);
lg5 = {likWeibull,{'logistic2',0.01}}; ka = 0.5; hyp.lik = log(ka);

% 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,[],'infLaplace');

% 3b) compute Gaussian integrals w.r.t. likelihood
mu = f; s2 = rand(n,1);
[lZ,d1Z,d21Z] = feval(lik{:},hyp,y,mu,s2,'infEP');

% 3c) obtain lower bound on likelihood
ga = rand(n,1);
[b,z] = 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.

<table>
<thead>
<tr>
<th>Likelihood \ Inference</th>
<th>Gaussian Likelihood</th>
<th>Laplace</th>
<th>VB</th>
<th>EP</th>
<th>KL</th>
<th>MCMC</th>
<th>LOO</th>
<th>Type, Output Domain</th>
<th>Alternative Names</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gaussian</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>regression, R</td>
<td></td>
</tr>
<tr>
<td>Warped Gaussian</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>regression, R</td>
<td></td>
</tr>
<tr>
<td>Gumbel</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>regression, R</td>
<td></td>
</tr>
<tr>
<td>Sch-squared</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>regression, R</td>
<td></td>
</tr>
<tr>
<td>Laplacian</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>regression, R</td>
<td>double exponential</td>
</tr>
<tr>
<td>Student's t</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>regression, R</td>
<td></td>
</tr>
<tr>
<td>Mixture</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>mixing meta likelihood</td>
<td></td>
</tr>
<tr>
<td>Error function</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>classification, [±1]</td>
<td>probit regression</td>
</tr>
<tr>
<td>Logistic function</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>classification, [±1]</td>
<td>logit regression</td>
</tr>
<tr>
<td>Uniform</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>classification, [±1]</td>
<td>label noise</td>
</tr>
<tr>
<td>Weibull</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>positive data, R_+{0}</td>
<td>nonnegative regression</td>
</tr>
<tr>
<td>Gamma</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>positive data, R_+{0}</td>
<td>nonnegative regression</td>
</tr>
<tr>
<td>Exp</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>positive data, R_+{0}</td>
<td>nonnegative regression</td>
</tr>
<tr>
<td>Inverse Gaussian</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>positive data, R_+{0}</td>
<td>nonnegative regression</td>
</tr>
<tr>
<td>Poisson</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>count data, N</td>
<td>Poisson regression</td>
</tr>
<tr>
<td>Negative Binomial</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>count data, N</td>
<td>negative binomial regression</td>
</tr>
<tr>
<td>Beta</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>interval data, [0,1]</td>
<td>beta regression</td>
</tr>
</tbody>
</table>

(✓)* EP might not converge in some cases since quadrature is used.

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 is 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.

```matlab
lik/likGauss.m
24
1 function [varargout] = likGauss(hyp, y, mu, s2, inf, i)
2
3 % likGauss - Gaussian likelihood function for regression. The expression for the
4 % likelihood is
5 % likGauss(t) = exp(-(t-y)^2/2*sn^2) / sqrt(2*pi*sn^2),
6 % where y is the mean and sn is the standard deviation.
7 %
8 % The hyperparameters are:
9 %
10 % hyp = [ log(sn) ]
11 %
12 % Several modes are provided, for computing likelihoods, derivatives and moments
13 % respectively, see likFunctions.m for the details. In general, care is taken
14 % to avoid numerical issues when the arguments are extreme.
15 %
16 \langle gpml copyright \rangle
17 %
```
% 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
else
  switch inf
    case 'infLaplace'
      Laplace's method with Gaussian likelihood
    case 'infEP'
      EP inference with Gaussian likelihood
    case 'infVB'
      Variational Bayes inference with Gaussian likelihood
  end
end

if isempty(y), y = zeros(size(mu)); end
s2zero = 1; if nargin>3&&numel(s2)>0&&norm(s2)>eps, s2zero = 0; 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 \( \rho \), the log of the noise standard deviation \( \sigma_n \).

### 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

\( \text{Laplace's method with Gaussian likelihood} \)
Starting from the likelihood \( p(y|f) \) we are sometimes facing the situation where the data \( y \in Y \subseteq \mathbb{R} \) is not distributed according to \( p(y|f) \) but some nonlinear transformation of the data \( g(y) = z \) so that \( z \sim p(z|f) \). Here, the warping function \( g : Y \rightarrow \mathbb{R} \) needs to be strictly monotonically increasing i.e. \( g'(y) > 0 \). Formally, we start from the fact that \( p(z|f) \) integrates to one and use the derivative \( dz = g'(y)dy \) to substitute

\[
\int p(z|f)dz = 1 = \int p_g(y|f)dy, \, p_g(y|f) = p(g(y)|f)g'(y)
\]

where we have defined the log warped likelihood \( \ln p_g(y|f) = \ln p(g(y)|f) + \ln g'(y) \). The interesting bit is that approximate inference methods such as infExact, infLaplace, infEP, infVB, infKL remain fully feasible; only prediction and derivatives become more involved. The usual GP inference is recovered by using the identity warping function \( g : y \mapsto y \). The construction works in principle for any likelihood but our implementation in likGaussWarp is limited to the Gaussian likelihood.
Hyperparameter derivatives

Hyperparameter derivatives for infLaplace are obtained as follows

\[
\frac{\partial}{\partial \theta} \ln \frac{\partial^k}{\partial f^k} p_g(y|f) = \frac{\partial}{\partial \theta} \ln \frac{\partial^k}{\partial f^k} p(g(y)|f) + \frac{\partial}{\partial \theta} \frac{\partial^k}{\partial f^k} \ln g'(y), \quad k = 0, 1, 2
\]

\[
= -\frac{\partial^{k+1}}{\partial f^{k+1}} \ln p(g(y)|f) \frac{\partial}{\partial \theta} g(y) + \frac{\partial}{\partial \theta} \frac{\partial^k}{\partial f^k} \ln g'(y).
\]

Similarly for infEP the derivatives are given by

\[
\frac{\partial}{\partial \theta} \ln \int p_g(y|f) N(f|\mu, \sigma^2) df = \frac{\partial}{\partial \theta} \ln \int p(g(y)|f) N(f|\mu, \sigma^2) df + \frac{\partial}{\partial \theta} \ln g'(y)
\]

\[
= -\frac{\partial}{\partial \mu} \ln \int p(g(y)|f) N(f|\mu, \sigma^2) df \frac{\partial}{\partial \theta} g(y) + \frac{\partial}{\partial \theta} \ln g'(y).
\]

This trick above works for any homoscedastic likelihood where \( p(y|f) = p(y + \beta|f + \beta) \) such as likGauss, likLaplace, likSech2 and likT.

Predictive moments

As detailed in [4] the predictive distribution is – for Gaussian likelihood – given by

\[
p(z_*|D, x_*) = \int p(z_*|f_*) p(f_*|D, x_*) df_* = \int \mathcal{N}(z_*|f_*, \sigma_n^2) \mathcal{N}(f_*|\mu_f, \sigma_f^2) df_*
\]

\[
= \mathcal{N}(z_*|\mu_{f_*}, \sigma_n^2 + \sigma_f^2), \quad \text{where } z_* = g(y_*)
\]

\[
p(y_*|D, x_*) = g'(y_*) \mathcal{N}(g(y_*)|\mu_{f_*}, \sigma_n^2 + \sigma_f^2).
\]

Hence, the predictive moments are obtained by the 1d integrals

\[
\mu_{y_*} = \int y_* g'(y_*) \mathcal{N}(g(y_*)|\mu_{f_*}, \sigma_n^2 + \sigma_f^2) dy_*
\]

\[
= \int g^{-1}(z_*) \mathcal{N}(z_*|\mu_{f_*}, \sigma_n^2 + \sigma_f^2) dz_*, \quad \text{and}
\]

\[
\sigma_{y_*}^2 = \int (y_* - \mu_{y_*})^2 g'(y_*) \mathcal{N}(g(y_*)|\mu_{f_*}, \sigma_n^2 + \sigma_f^2) dy_*
\]

\[
= \int (g^{-1}(z_*) - \mu_{y_*})^2 \mathcal{N}(z_*|\mu_{f_*}, \sigma_n^2 + \sigma_f^2) dz_*.
\]

4.8 Gumbel Likelihood

Distributions of extrema are well captured by the Gumbel distribution

\[
p(y) = \frac{1}{\beta} \exp \left( -z - e^{-z} \right), \quad z = s \frac{y - \eta}{\beta}, \quad s \in \{\pm 1\}
\]

with mean \( \mu = \eta + \beta \gamma \) and variance \( \sigma^2 = \pi^2 \beta^2 / 6 \) where \( \gamma = 0.57721566490153 \) denotes Euler–Mascheroni’s constant. Skewness is approximately given by 1.1395s where \( s \) is a sign switching between left and right skewness and kurtosis is 12/5. The final expression for the Gumbel likelihood is

\[
p(y|f) = \frac{\pi}{\sigma \sqrt{6}} \exp \left( -z - e^{-z} \right), \quad z = \gamma + s \frac{\pi}{\sigma \sqrt{6}} (y - f), \quad s \in \{\pm 1\}.
\]
4.9 Laplace Likelihood

Laplace’s Approximation

The following derivatives are needed:

\[
\begin{align*}
\ln p(y|f) &= -\ln(2b) - \frac{|f - y|}{b} \\
\frac{\partial \ln p}{\partial f} &= \text{sign}(f - y) \\
\frac{\partial^2 \ln p}{(\partial f)^2} &= \frac{\partial^3 \ln p}{(\partial f)^3} = \frac{\partial^3 \ln p}{(\partial \ln \sigma_n)(\partial f)^2} = 0 \\
\frac{\partial \ln p}{\partial \ln \sigma_n} &= \frac{|f - y|}{b} - 1
\end{align*}
\]

Expectation Propagation

Expectation propagation requires integration against a Gaussian measure for moment matching. We need to evaluate \( \ln Z = \ln \int \mathcal{L}(y|f, \sigma^2_n)N(f|\mu, \sigma^2)df \) as well as the derivatives \( \frac{\partial \ln Z}{\partial \mu} \) and \( \frac{\partial^2 \ln Z}{\partial \mu^2} \) where \( N(f|\mu, \sigma^2) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(f-\mu)^2}{2\sigma^2}\right) \), \( \mathcal{L}(y|f, \sigma^2_n) = \frac{1}{2b} \exp\left(-\frac{|y-f|}{b}\right) \), 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

\[
\begin{align*}
Z &= \int \mathcal{L}(y|f, \sigma^2_n)N(f|\mu, \sigma^2)df \\
&= \frac{1}{\sqrt{2\pi}\sigma} \frac{\sqrt{2}}{2\sigma\sqrt{2\pi}} \int \exp\left(-\frac{(f-\mu)^2}{2\sigma^2}\right) \exp\left(-\frac{|f - y|}{\sigma_n}\right) df \\
&= \frac{\sqrt{2}}{2\sigma\sqrt{2\pi}} \int \exp\left(-\frac{(\sigma_n \tilde{f} + y - \mu)^2}{2\sigma^2}\right) \exp\left(-\sqrt{2}|\tilde{f}|\right) d\tilde{f} \\
&= \frac{\sigma_n}{\sigma\sigma_n\sqrt{2\pi}} \int \exp\left(-\frac{\sigma_n^2 \left(\tilde{f} - \frac{y - \mu}{\sigma_n}\right)^2}{2\sigma^2}\right) \mathcal{L}(\tilde{f}|0, 1) d\tilde{f} \\
&= \frac{1}{\sigma_n} \int \mathcal{L}(f|0, 1)N(f|\tilde{\mu}, \tilde{\sigma}^2)df \\
\ln Z &= \ln Z - \ln \sigma_n = \ln \int \mathcal{L}(f|0, 1)N(f|\tilde{\mu}, \tilde{\sigma}^2)df - \ln \sigma_n
\end{align*}
\]
with \( \tilde{\mu} = \frac{\mu - y}{\sigma_n} \) and \( \tilde{\sigma} = \frac{\sigma}{\sigma_n} \). Thus, we concentrate on the simpler quantity \( \ln \tilde{Z} \).

\[
\ln Z = \ln \left[ \exp \left( -\frac{(f - \tilde{\mu})^2}{2\tilde{\sigma}^2} - \sqrt{2f} \right) \right] - \ln \sqrt{2\pi} - \ln \sqrt{2\sigma_n}
\]

\[
\ln Z = \ln \left[ \exp \left( -\frac{(f - \tilde{\mu})^2}{2\tilde{\sigma}^2} + \sqrt{2f} \right) \right] + C
\]

\[
\ln Z = \ln \left[ \exp \left( -\frac{(f - \tilde{\mu} + \tilde{\sigma}^2 \sqrt{2})^2}{2\tilde{\sigma}^2} + \tilde{\mu}^2 \right) \right] + C
\]

\[
\ln Z = \ln \left[ \exp \left( \frac{m_+}{2\tilde{\sigma}^2} \right) \right] - \ln \sqrt{2\sigma_n}
\]

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

\[
\ln Z = \ln \left[ \exp \left( -\frac{(f - \tilde{\mu})^2}{\tilde{\sigma}^2} + \sqrt{2f} \right) \right] + \tilde{\sigma}^2 - \ln \sqrt{2\sigma_n}
\]

\[
\ln Z = \ln \left[ \exp \left( \sqrt{2\tilde{\mu}} + \sqrt{2\tilde{\mu}} \right) \right] + \tilde{\sigma}^2 - \ln \sqrt{2\sigma_n}
\]

\[
\ln Z = \ln \left[ \exp \left( \frac{\sqrt{2\tilde{\mu}}}{\sigma} \right) \right] + \tilde{\sigma}^2 - \ln \sqrt{2\sigma_n}
\]

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

Now, using \( \frac{d}{d\tilde{\sigma}} \ln \Phi(z) = \frac{1}{\Phi(z)} \frac{d}{d\tilde{\sigma}} \Phi(z) = \frac{N(z)}{\Phi(z)} \) we tackle first derivative

\[
\frac{\partial \ln Z}{\partial \tilde{\mu}} = e^{a} \frac{\partial a_+}{\partial \tilde{\mu}} + e^{a_-} \frac{\partial a_-}{\partial \tilde{\mu}}
\]

\[
\frac{\partial a_+}{\partial \tilde{\mu}} = \frac{\partial}{\partial \tilde{\mu}} \ln \Phi(-z_+) + \sqrt{2} \frac{N(-z_+)}{\sigma_n} = -\frac{q_+}{\sigma} + \frac{\sqrt{2}}{\sigma_n}
\]

\[
\frac{\partial a_-}{\partial \tilde{\mu}} = \frac{\partial}{\partial \tilde{\mu}} \ln \Phi(z_-) - \sqrt{2} \frac{N(z_-)}{\sigma_n} = -\frac{q_-}{\sigma} - \frac{\sqrt{2}}{\sigma_n}
\]

\[
\frac{\partial a_+}{\partial \tilde{\mu}} = \pm \frac{q_+}{\sigma} \pm \frac{\sqrt{2}}{\sigma_n}.
\]
as well as the second derivative

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

\[
\frac{\partial}{\partial \mu} \left( e^{a_+ \frac{\partial a_+}{\partial \mu}} + e^{a_- \frac{\partial a_-}{\partial \mu}} \right) = e^{a_+} \left[ \frac{\partial a_+}{\partial \mu} \right]^2 + e^{a_-} \left[ \frac{\partial a_-}{\partial \mu} \right]^2
\]

\[
\frac{\partial^2 a_+}{\partial \mu^2} = -\frac{1}{\sigma} \frac{\partial^2}{\partial \mu^2} N(-z_+) \Phi(-z_+) \Phi(-z_+ - \frac{z_+}{\sigma})
\]

\[
= -\frac{1}{\sigma} \frac{\partial^2}{\partial \mu^2} N(-z_+) \Phi(-z_+) \Phi(-z_+ - \frac{z_+}{\sigma}) \frac{\partial -z_+}{\partial \mu}
\]

\[
= \frac{N(-z_+)}{\sigma^2} \Phi(-z_+) \Phi(-z_+ - \frac{z_+}{\sigma}) = -q_+^2 - q_+ z_+
\]

\[
\frac{\partial^2 a_-}{\partial \mu^2} = \frac{1}{\sigma} \frac{\partial^2}{\partial \mu^2} N(z_-) \Phi(z_-) \Phi(z_-) \Phi(z_-)
\]

\[
= \frac{N(z_-)}{\sigma^2} \Phi(z_-) \Phi(z_-) \Phi(z_-) = -q_-^2 + q_- z_-
\]

\[
\frac{\partial^2 a_\pm}{\partial \mu^2} = -q_\pm^2 \mp q_\pm z_\pm
\]

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

\[
b_\pm = \left( \frac{\partial a_\pm}{\partial \mu} \right)^2 + \frac{\partial^2 a_\pm}{\partial \mu^2} = \left( \pm \frac{q_\pm}{\sigma} \pm \frac{\sqrt{2}}{\sigma_\gamma} \right)^2 - \frac{q_\pm^2 \mp q_\pm z_\pm}{\sigma^2}
\]

\[
= \left( \frac{q_\pm}{\sigma} - \frac{\sqrt{2}}{\sigma_\gamma} \right)^2 - \frac{q_\pm^2 \mp q_\pm z_\pm}{\sigma^2}
\]

\[
= \frac{2}{\sigma_\gamma^2} - \left( \frac{\sqrt{8}}{\sigma_\gamma^2} \mp \frac{z_\pm}{\sigma^2} \right) q_\pm.
\]

We also need

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

Variational Bayes

We need \( h(\gamma) \) and its derivatives as well as \( \beta(\gamma) \):
\[ h(\gamma) = \frac{2}{\sigma_n^2} \gamma + \ln(2\sigma_n^2) + y^2\gamma^{-1} \]
\[ h'(\gamma) = \frac{2}{\sigma_n^2} - y^2\gamma^{-2} \]
\[ h''(\gamma) = 2y^2\gamma^{-3} \]
\[ \beta(\gamma) = y\gamma^{-1} \]

4.10 Student’s t Likelihood

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

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

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

\[ \frac{\partial \ln p}{\partial f} = \frac{\nu}{\nu \sigma_n^2} \frac{r}{r^2 + \nu \sigma_n^2} \]
\[ \frac{\partial^2 \ln p}{(\partial f)^2} = \frac{\nu}{\nu \sigma_n^2} \frac{r^2 - \nu \sigma_n^2}{(r^2 + \nu \sigma_n^2)^2} \]
\[ \frac{\partial^3 \ln p}{(\partial f)^3} = 2 \frac{\nu}{\nu \sigma_n^2} \frac{3r \sigma_n^2}{(r^2 + \nu \sigma_n^2)^3} \]

\[ \frac{\partial \ln p}{\partial \ln \nu} = \frac{\partial Z}{\partial \ln \nu} = \frac{\nu}{2} \left(1 + \frac{r^2}{\nu \sigma_n^2}\right) + \frac{\nu + 1}{2} \cdot \frac{r^2}{r^2 + \nu \sigma_n^2} \]
\[ \frac{\partial Z}{\partial \ln \nu} = \frac{\nu}{2} \frac{\partial \ln \Gamma \left(\frac{\nu + 1}{2}\right)}{\partial \ln \nu} - \frac{\nu}{2} \frac{\partial \ln \Gamma \left(\frac{\nu}{2}\right)}{\partial \ln \nu} - \frac{1}{2} \]
\[ \frac{\partial^3 \ln p}{(\partial \ln \nu)(\partial f)^2} = \frac{\nu}{2} \frac{\partial^2 \ln \Gamma \left(\frac{\nu + 1}{2}\right)}{\partial \ln \nu^2} + \frac{\nu}{2} \frac{\partial \ln \Gamma \left(\frac{\nu}{2}\right)}{\partial \ln \nu} - \frac{1}{2} \]
\[ \frac{\partial \ln p}{\partial \ln \sigma_n} = \frac{\nu}{\nu \sigma_n^2} \frac{r^2}{r^2 + \nu \sigma_n^2} - 1 \]
\[ \frac{\partial^3 \ln p}{(\partial \ln \sigma_n)(\partial f)^2} = 2 \nu \sigma_n^2 \left(\nu + 1\right) \frac{\nu \sigma_n^2 - 3r^2}{(r^2 + \nu \sigma_n^2)^3} \]

4.11 Cumulative Logistic Likelihood

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

\[ p(y|f) = Z \cdot \cosh^{-2} (\tau(f - y)), \quad \tau = \frac{\pi}{2\sigma_n \sqrt{3}}, \quad Z = \frac{\pi}{4\sigma_n \sqrt{3}} \]
Laplace’s Approximation

The following derivatives are needed where \( \phi(x) \equiv \ln(\cosh(x)) \)
\[
\ln p(y|f) = \ln(\pi) - \ln(4\sigma_n \sqrt{3}) - 2\phi(\tau(f - y))
\]
\[
\frac{\partial \ln p}{\partial f} = 2\tau \phi'(\tau(f - y))
\]
\[
\frac{\partial^2 \ln p}{(\partial f)^2} = -2\tau^2 \phi''(\tau(f - y))
\]
\[
\frac{\partial^3 \ln p}{(\partial f)^3} = 2\tau^3 \phi'''(\tau(f - y))
\]
\[
\frac{\partial^3 \ln p}{(\partial \ln \sigma_n)^3} = 2\tau^2 (2\phi''(\tau(f - y)) + \tau(f - y)\phi'''(\tau(f - y)))
\]
\[
\frac{\partial \ln p}{\partial \ln \sigma_n} = 2\tau(\tau(f - y)\phi'(\tau(f - y)) - 1)
\]

4.12 GLM Likelihoods: Poisson, Negative Binomial, Weibull, Gamma, Exponential, Inverse Gaussian and Beta

Data \( y \) from a space other than \( \mathbb{R} \) e.g. \( \mathbb{N}, \mathbb{R}_+ \) or \( [0, 1] \) can be modeled using generalised linear model likelihoods \( p(y|f) \) where the expected value \( E[y] = \mu \) is related to the underlying Gaussian process \( f \) by means of an inverse link function \( \mu = g(f) \). Typically, the likelihoods are from an exponential family, hence the variance \( \mathbb{V}[y] = v(\mu) \), is a simple function of the mean \( \mu \) as well as higher order moments such as skewness \( S[y] = s(\mu) \) and kurtosis \( K[y] = k(\mu) \).

Here, we directly specify the inverse link function \( \mu = g(f) \) defining the mapping from the GP \( f \) to the mean intensity \( \mu \). For numerical reasons, we work with the log of the inverse link function \( \ln g(f) = \ln \mu \) and use its derivatives \( h', h'' \) and \( h''' \) for subsequent computations. In the table below, we have summarised the GLM likelihood expressions, the moments, the range of their variables and the applicable inverse link functions.

| Likelihood       | \( \phi(\mu) = \) | \( \psi(\mu) = \) | \( \kappa(\mu) = \) | \( p(\mu|f) = \) | \( \mu \in \) | \( \mu \in \) | Inverse Links |
|------------------|------------------|------------------|------------------|------------------|-------------|-------------|---------------|
| Poisson          | \( f \)          | \( \mu \)        | \( s(\mu) \)     | \( \ln \exp(\mu/f)\) | \( R \) \( \in \) | \( \exp, \logistic* \) |
| Neg. Binomial    | \( \ln \frac{\mu}{\mu + 1} \) | \( \sqrt{\frac{\mu}{\mu + 1}} \) | \( \frac{\mu}{\mu + 1} \) | \( \phi(\mu)=\frac{\mu}{\mu + 1} \) | \( R \) \( \in \) | \( \exp, \logistic* \) |
| Weibull          | \( \ln \frac{\mu}{\mu + 1} \) | \( \sqrt{\frac{\mu}{\mu + 1}} \) | \( \frac{\mu}{\mu + 1} \) | \( \phi(\mu)=\frac{\mu}{\mu + 1} \) | \( R \) \( \in \) | \( \exp, \logistic* \) |
| Gamma            | \( \frac{\mu}{\mu} \) | \( \frac{\mu}{\mu} \) | \( \frac{\mu}{\mu} \) | \( \phi(\mu)=\frac{\mu}{\mu} \) | \( R \) \( \in \) | \( \exp, \logistic* \) |
| Exponential      | \( \frac{\mu}{\mu} \) | \( \frac{\mu}{\mu} \) | \( \frac{\mu}{\mu} \) | \( \phi(\mu)=\frac{\mu}{\mu} \) | \( R \) \( \in \) | \( \exp, \logistic* \) |
| Inv. Gauss       | \( \ln \frac{\mu}{\mu} \) | \( \frac{\mu}{\mu} \) | \( \frac{\mu}{\mu} \) | \( \phi(\mu)=\frac{\mu}{\mu} \) | \( R \) \( \in \) | \( \exp, \logistic* \) |
| Log Gauss        | \( \ln \frac{\mu}{\mu} \) | \( \frac{\mu}{\mu} \) | \( \frac{\mu}{\mu} \) | \( \phi(\mu)=\frac{\mu}{\mu} \) | \( R \) \( \in \) | \( \exp, \logistic* \) |
| Beta             | \( \ln \frac{\mu}{\mu} \) | \( \frac{\mu}{\mu} \) | \( \frac{\mu}{\mu} \) | \( \phi(\mu)=\frac{\mu}{\mu} \) | \( R \) \( \in \) | \( \exp, \logistic* \) |

4.12.1 Inverse Link Functions

Possible inverse link functions and their properties (\( \cup \) convex, \( \cap \) concave, \( \uparrow \) monotone) are summarised below:

<table>
<thead>
<tr>
<th>util/glm_invlink_*</th>
<th>( g(f) = \mu = )</th>
<th>( g : \mathbb{R} \rightarrow )</th>
<th>( g ) is</th>
<th>( h(f) = \ln \mu = )</th>
<th>( h ) is</th>
</tr>
</thead>
<tbody>
<tr>
<td>exp</td>
<td>( e^f )</td>
<td>( \mathbb{R}_+ )</td>
<td>( \cup, \uparrow )</td>
<td>( f )</td>
<td>( \cup, \cap, \uparrow )</td>
</tr>
<tr>
<td>logistic</td>
<td>( \ell(f) = \ln(1 + e^f) )</td>
<td>( \mathbb{R}_+ )</td>
<td>( \cup, \uparrow )</td>
<td>( \ln(\ln(1 + e^f)) )</td>
<td>( \cap, \uparrow )</td>
</tr>
<tr>
<td>logistic2</td>
<td>( \ell(f + \alpha(f)) )</td>
<td>( \mathbb{R}_+ )</td>
<td>( \uparrow )</td>
<td>( \ln(\ell(f + \alpha(f))) )</td>
<td>( \cap, \uparrow )</td>
</tr>
<tr>
<td>expexp</td>
<td>( \exp(-e^{-f}) )</td>
<td>( [0, 1] )</td>
<td>( \uparrow )</td>
<td>( -e^{-f} )</td>
<td>( \cup, \uparrow )</td>
</tr>
<tr>
<td>logit</td>
<td>( 1/(1 + e^{-f}) )</td>
<td>( [0, 1] )</td>
<td>( \uparrow )</td>
<td>( -\ln(1 + e^{-f}) )</td>
<td>( \cup, \uparrow )</td>
</tr>
</tbody>
</table>
Please see doc/usageLik.m for how to specify the pair likelihood function and link function in GPML.

**Exponential inverse link: exp**

For \( g(f) = e^f \) things are simple since \( h(f) = f, \ h'(f) = 1 \) and \( h''(f) = h'''(f) = 0. \)

**Logistic inverse link: logistic**

For \( g(f) = \ln(1 + e^f) \) the derivatives of \( h(f) \) are given by

\[
\begin{align*}
  h(f) &= \ln(\ln(1 + e^f)) \\
  h'(f) &= \frac{1}{\ln(1 + e^f)} s(-f), \ s(f) = \frac{1}{1 + e^f}, \ s'(f) = \frac{-e^f}{(1 + e^f)^2} = -s(-f)s(f) \\
  h''(f) &= \frac{1}{\ln(1 + e^f)} \frac{e^{-f}}{(1 + e^f)^2} - \frac{1}{\ln^2(1 + e^f)} \frac{e^f}{1 + e^f} \frac{1}{1 + e^{-f}} \\
  &= h'(f) [s(f) - h'(f)] \\
  h'''(f) &= h''(f) [s(f) - h'(f)] + h'(f) \left[ \frac{-e^f}{(1 + e^f)^2} - h''(f) \right] \\
  &= h''(f) [s(f) - 2h'(f)] - h'(f)s(f)s(-f).
\end{align*}
\]

Note that \( g(f) = e^{h(f)} = \ln(1 + e^f) \) is convex and \( h(f) = \ln(\ln(1 + e^f)) \) with

\[
  h''(f) = \frac{1}{\ln(1 + e^f)} \left( 1 - \frac{e^f}{\ln(1 + e^f)} \right) \frac{1}{1 + e^f} \frac{1}{1 + e^{-f}} \leq 0
\]

is concave since \( e^f \geq \ln(1 + e^f) \) for all \( f \in \mathbb{R} \).

**Twice logistic inverse link: logistic2**

Note that \( h(f) = \ln(\ell(f + af\ell(f))) \) is – according to Seeger et al.\(^9\) – concave.

**Double negative exponential inverse link: expexp**

For \( g(f) = \exp(-e^{-f}) \) the derivatives of \( h(f) \) are given by

\[
\begin{align*}
  h(f) &= -e^{-f} \\
  h'(f) &= -h(f) \\
  h''(f) &= h(f) \\
  h'''(f) &= -h(f)
\end{align*}
\]

\(^9\)Bayesian Intermittent Demand Forecasting, NIPS, 2016
Logit regression inverse link: \texttt{logit}

For } g(f) = 1/(1 + e^{-f}) \text{ the derivatives of } h(f) \text{ can be computed using the logistic inverse link function } h_\ell(f) \text{ since } h(f) = f - \exp(h_\ell(f))

\begin{align*}
h(f) &= f - e^{h_\ell(f)} \\
h'(f) &= 1 - e^{h_\ell(f)} h_\ell'(f) \\
h''(f) &= -e^{h_\ell(f)}[h_\ell'(f)^2 + h_\ell''(f)] = e^{h_\ell(f)} s_\ell(-f) s_\ell^2(f) \\
h'''(f) &= -e^{h_\ell(f)}[3 h_\ell'(f) h_\ell''(f) + h_\ell'''(f)]
\end{align*}

4.12.2 Poisson Likelihood

Count data } y \in \mathbb{N}^n \text{ can be modeled in the GP framework using the Poisson distribution } p(y) = \mu^y e^{-\mu}/y! \text{ with mean/variance } \mathbb{E}[y] = \mathbb{V}[y] = \mu, \text{ skewness } S[y] = 1/\sqrt{\mu} \text{ and kurtosis } K[y] = 1/\mu \text{ leading to the likelihood}

\begin{align*}
p(y|f) &= \mu^y \exp(-\mu)/y!, \quad \mu = g(f) \\
⇔ \ln p(y|f) &= y \cdot \ln g(f) - g(f) - \ln \Gamma(y + 1).
\end{align*}

For Laplace’s method to work, we need the first three derivatives of the log likelihood } \ln p(y|f) \text{, where } h(f) = \ln g(f)

\begin{align*}
\ln p(y|f) &= y \cdot h(f) - \exp(h(f)) - \ln \Gamma(y + 1) \\
\frac{\partial}{\partial f} \ln p(y|f) &= h'(f) [y - \exp(h(f))] \\
\frac{\partial^2}{\partial f^2} \ln p(y|f) &= h''(f) [y - \exp(h(f))] - [h'(f)]^2 \exp(h(f)) \\
\frac{\partial^3}{\partial f^3} \ln p(y|f) &= h'''(f) [y - \exp(h(f))] - 3h'(f) \cdot h''(f) \exp(h(f)) - [h'(f)]^3 \exp(h(f)) \\
&- h''(f) [y - \exp(h(f))] - h'(f) [h'(f)^2 + 3h''(f)] \exp(h(f)).
\end{align*}

Note that if } \ln \mu = h(f) \text{ is concave and } \mu = g(f) \text{ is convex then the Poisson likelihood } p(y|f) \text{ is log-concave in } f \text{ which is the case for both } \exp \text{ and } \texttt{logistic}.

4.12.3 Weibull Likelihood

Nonnegative data } y \in \mathbb{R}_+ \text{ such as time-to-failure can be modeled in the GP framework using the Weibull distribution } p(y) = \kappa/\lambda (y/\lambda)^{\kappa-1} e^{-(y/\lambda)\kappa} \text{ with shape parameter } \kappa > 0, \text{ scale parameter } \lambda > 0, \text{ mean } \mathbb{E}[y] = \lambda \gamma_1 = \mu \text{ where } \gamma_j = \Gamma(1 + j/\kappa), \text{ variance } \mathbb{V}[y] = \lambda^2 \gamma_2 - \mu^2 = \mu^2 (\gamma_2/\gamma_1^2 - 1), \text{ skewness } S[y] = (\gamma_3 - 3\gamma_1\gamma_2 + 2\gamma_1^3)/(\gamma_2 - \gamma_1^2)^{3/2} \text{ and kurtosis } K[y] = (\gamma_4 - 4\gamma_1\gamma_3 + 12\gamma_1^2 \gamma_2 - 3\gamma_2^2 - 6\gamma_1^3)/(\gamma_2 - \gamma_1^2)^2. \text{ Using the substitution } \mu = \lambda \gamma_1 \Leftrightarrow 1/\lambda = \gamma_1/\mu, \text{ we obtain}

\begin{align*}
p(y|f) &= \gamma_1 \frac{\kappa}{\mu} \left(\gamma_1 \frac{y}{\mu}\right)^{\kappa-1} \exp \left(-\left(\gamma_1 \frac{y}{\mu}\right)^\kappa\right), \quad \mu = g(f) > 0 \\
⇔ \ln p(y|f) &= \ln \left(\gamma_1 \frac{\kappa}{\mu}\right) + (\kappa - 1) \ln \left(\gamma_1 \frac{y}{\mu}\right) - \left(\gamma_1 \frac{y}{\mu}\right)^\kappa.
\end{align*}

Note that the Weibull likelihood } p(y|f) \text{ is log-concave in } f \text{ neither for the } \exp \text{ nor for the } \texttt{logistic} inverse link.
4.12.4 Gamma Likelihood

Nonnegative data $y \in \mathbb{R}_+$ can be modeled in the GP framework using the Gamma distribution $p(y) = 0^{-\alpha}/\Gamma(\alpha) y^{\alpha-1} e^{-y/\theta}$ with shape parameter $\alpha > 0$, scale parameter $\theta > 0$, mean $\mathbb{E}[y] = \alpha \theta = \mu$, variance $\mathbb{V}[y] = \alpha \theta^2 = \mu^2/\alpha$, skewness $\mathbb{S}[y] = 2/\sqrt{\alpha}$ and kurtosis $\mathbb{K}[y] = 6/\alpha$. Using the substitution $\mu = \alpha \theta \iff \alpha/\mu = 1/\theta$, we obtain

$$p(y|f) = \frac{\alpha^\alpha y^{\alpha-1}}{\Gamma(\alpha)} \mu^{-\alpha} \exp\left(-\frac{y \alpha}{\mu}\right), \mu = g(f) > 0$$

$$\iff \ln p(y|f) = -\alpha \left(\ln \mu + \frac{y}{\mu}\right) - \ln Z_\alpha(y), \ln Z_\alpha(y) = \ln \Gamma(\alpha) - \alpha \ln \alpha + (1 - \alpha) \ln y.$$

Note that if $\ln \mu = h(f)$ was convex and $\mu = g(f)$ was concave then the Gamma likelihood $p(y|f)$ would be log-concave in $f$ which is not the case for both $\exp$ and $\logistic$.

4.12.5 Exponential Likelihood

Nonnegative data $y \in \mathbb{R}_+$ can be modeled in the GP framework using the Exponential distribution $p(y) = 0^{-1} e^{-y/\theta}$ with scale parameter $\theta > 0$, mean $\mathbb{E}[y] = \theta = \mu$, variance $\mathbb{V}[y] = \mu^2$, skewness $\mathbb{S}[y] = 2$ and kurtosis $\mathbb{K}[y] = 6$. We obtain

$$p(y|f) = \mu^{-1} \exp\left(-\frac{y}{\mu}\right), \mu = g(f) > 0$$

$$\iff \ln p(y|f) = -\ln \mu - \frac{y}{\mu}.$$

Note that for $\exp$ (but not for $\logistic$) the likelihood is log-concave. The exponential distribution corresponds to the Gamma distribution with $\alpha = 1$ and the Weibull distribution with $\kappa = 1$.

4.12.6 Inverse Gaussian Likelihood

Nonnegative data $y \in \mathbb{R}_+^n$ can be modeled in the GP framework using the Inverse Gaussian distribution $p(y) = \sqrt{\lambda/(2\pi y^2)} \exp(-\lambda(y - \mu)^2/(2\mu^2y))$ with shape parameter $\lambda > 0$, mean parameter $\mu > 0$, mean $\mathbb{E}[y] = \mu$, variance $\mathbb{V}[y] = \mu^3/\lambda$, skewness $\mathbb{S}[y] = 3\sqrt{\mu/\lambda}$ and kurtosis $\mathbb{K}[y] = 15\mu/\lambda$. We obtain

$$p(y|f) = \sqrt{\lambda/2\pi y^3} \exp\left(-\frac{\lambda(y - \mu)^2}{2\mu^2 y}\right), \mu = g(f) > 0$$

$$\iff \ln p(y|f) = -\frac{\lambda(y - \mu)^2}{2\mu^2 y} - \ln Z_\alpha(y), \ln Z_\alpha(y) = -\frac{1}{2}(\ln \lambda - \ln 2\pi y^3).$$

The inverse Gaussian likelihood is in general not log-concave in $f$ for both $\exp$ and $\logistic$.

4.12.7 Log Gaussian Likelihood

Nonnegative data $y \in \mathbb{R}_+^n$ can be modeled in the GP framework using the Log-normal distribution $p(y) = \exp\left(-\log \mu - \sigma_n^2/2 - \log y^2/(2\sigma_n^2)\right)/(y \sigma_n \sqrt{2\pi})$ with scale parameter $\sigma_n > 0$, and mean
parameter $\mu > 0$, mean $E[y] = \mu$, variance $V[y] = (e^{\sigma^2_n} - 1)\mu^2$, skewness $S[y] = (e^{\sigma^2_n} + 2)e^{\sigma^2_n} - 6$. We obtain

$$p(y|f) = \frac{1}{y\sigma_n \sqrt{2\pi}} \exp\left(-\frac{(\log \mu - \sigma_n^2/2 - \log y)^2}{2\sigma_n^2}\right), \mu = g(f) > 0$$

$$\Leftrightarrow \ln p(y|f) = -\frac{(\log \mu - \sigma_n^2/2 - \log y)^2}{2\sigma_n^2} - \ln Z(y), \ln Z(y) = y\sigma_n \sqrt{2\pi}.$$ 

The log Gaussian likelihood is log-concave in $f$ for the exp link function.

### 4.12.8 Beta Likelihood

Interval data $y \in [0, 1]^n$ can be modeled in the GP framework using the Beta distribution $p(y) = y^{\alpha-1}(1-y)^{\beta-1}/B(\alpha, \beta)$ with shape parameters $\alpha, \beta > 0$, mean $E[y] = \alpha/(\alpha + \beta)$ and variance $V[y] = \alpha\beta/[(\alpha + \beta)^2(\alpha + \beta + 1)]$ and $1/B(\alpha, \beta) = \Gamma(\alpha + \beta)/[\Gamma(\alpha)\Gamma(\beta)]$. Reparametrising using the mean parameter $\mu = E[y] = \alpha/(\alpha + \beta)$, the shape parameter $\phi = \alpha + \beta$, the variance $V[y] = \mu(1-\mu)/(1+\phi)$ and hence

$$p(y|f) = \frac{\Gamma(\phi)}{\Gamma(\mu\phi)\Gamma((1-\mu)\phi)}y^{\mu\phi-1}(1-y)^{(1-\mu)\phi-1}, \mu = g(f) > 0$$

$$\Leftrightarrow \ln p(y|f) = \ln \Gamma(\phi) - \ln \Gamma(\mu\phi) - \ln \Gamma((1-\mu)\phi) + (\mu\phi - 1) \ln y + ((1-\mu)\phi - 1) \ln(1-y).$$ 

The Beta likelihood is in general not log-concave in $f$ for both exp and logistic.
5 Mean Functions

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

5.1 Interface

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

```matlab
% 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
meanPoly - polynomial mean function
meanDiscrete - precomputed mean for discrete data
meanGP - predictive mean of another GP
meanGPexact - predictive mean of a regression GP
meanNN - nearest neighbor mean function
meanWSPC - weighted sum of projected cosines

% Composite mean functions (see explanation at the bottom):
meanScale - scaled version of a mean function
meanSum - sum of mean functions
meanProd - product of mean functions
meanPow - power of a mean function
meanMask - mask some dimensions of the data
meanPref - difference mean for preference learning
meanWarp - warped mean function

% Naming convention: all mean functions are named "mean/mean*.m".

1) With no or only a single input argument:
\[ s = \text{meanNAME} \] or \[ s = \text{meanNAME}(\text{hyp}) \]

2) With two input arguments and one output argument:
\[ m = \text{meanNAME}(\text{hyp}, \text{x}) \]
```

The function computes and returns the mean vector \( m \) with components \( m(i) = m(x(i,:)) \) where \( \text{hyp} \) are the hyperparameters and \( x \) is an \( n \) by \( D \) matrix of data, where \( D \) is the dimension of the input space. The returned mean.
vector m is of size n by 1.

3) With two input arguments and two output arguments:

[m, dm] = meanNAME(hyp, x)

The function computes and returns the mean vector m as in 2) above. In addition to that, the (linear) directional derivative function dm is returned. The call dhyp = dm(q) for a direction vector q of size n by 1 returns a vector of directional derivatives dhyp = d (q'*m(x)) / d hyp of the same size as the hyperparameter vector hyp. The components of dhyp are defined as follows: dhyp(i) = q'*( d m(x) / d hyp(i) ).

See also USAGEMEAN

help meanFunctions

5.2 Implemented Mean Functions

We offer simple and composite mean functions producing new mean functions \( m(x) \) from existing mean functions \( \mu_j(x) \). 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 \( m(x) = c + a'x \) or polynomial mean functions \( m(x) = (c + a'x)^2 \). All currently available mean functions are summarised in the following table.

| Simple mean functions \( m(x) \) | \( m(x) \) = | \( \phi \) |
|---------------------------------|-----------------|
| 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 | \( a'x \) | \( a \in \mathbb{R}^D \) |
| Poly  | mean polynomially depends on | \( \sum_d a_dx^d \) | \( a \in \mathbb{R}^{D \times a} \) |
| Discrete | precomputed mean for discrete data | \( \mu_x \) | \( \mu \in \mathbb{R}^s \) |
| GP | predictive mean of another GP | \( \int y \cdot p(y|D,x)dy \) | \( \emptyset \) |
| GPexact | predictive mean of a regression GP | \( \int y \cdot p(y|D,x)dy \) | \( \rho, \psi, \sigma_m \) |
| NN | nearest neighbor for a set \( (z_i, m_i) \in \mathcal{X} \times \mathbb{R} \) | \( m_i, i = \arg\min_d d(x, z_i) \) | \( \emptyset \) |
| WSPC | weighted sum of d projected cosines | \( \cos(Wx + b) \) | \( W \in \mathbb{R}^{d \times D}, a, b \in \mathbb{R}^d \) |

| Composite mean functions \( [\mu_1(x), \mu_2(x), ..] \mapsto m(x) \) | \( m(x) \) = | \( \phi \) |
|-----------------|-----------------|
| Scale | scale a mean | \( \alpha \mu(x) \) | \( \alpha \in \mathbb{R} \) |
| Sum | add up mean functions | \( \sum_i \mu_i(x) \) | \( \emptyset \) |
| Prod | multiply mean functions | \( \prod_i \mu_i(x) \) | \( \emptyset \) |
| Pow  | raise a mean to a power | \( \mu(x)^d \) | \( \emptyset \) |
| Mask | act on components \( I \subseteq [1, 2, .., D] \) of \( x \in \mathcal{X} \subseteq \mathbb{R}^D \) only | \( \mu(x_I) \) | \( \emptyset \) |
| Pref | preference learning mean \( x = [x_1; x_2], x_i \subseteq \mathbb{R}^{D\times2} \) | \( \mu(x_1) - \mu(x_2) \) | \( \emptyset \) |
| Warp | warped mean | \( g[\mu(x)] \) | \( \emptyset \) |

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
\[
\text{mn} := '\text{func}' | \ast\text{func} \ // \text{simple}
\]

\[
\text{mf} := \{\text{mn}\} | \{\text{mn}, \{\text{param}, \text{mf}\}\} | \{\text{mn}, \{\text{mf}, \ldots, \text{mf}\}\} \ // \text{composite}
\]
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.

```matlab
% demonstrate usage of mean functions
% See also meanFunctions.m.

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
mp = {@meanPoly,2}; hypp = [1;1;2;3]; % m(x) = x1+x2+2*x1^2+3*x2^2
mn = {@meanNN,[1,0; 0,1],[0.9,0.5]}; hypn = []; % nearest neighbor
s = 12; hypd = randn(s,1); % discrete mean with 12 hypers
md = {'meanDiscrete',s};

temp = randn(2*n,D); yt = sign(temp(:,1)-temp(:,2)); % training data
mg = {@meanGP,hyp,@infEP,@meanZero,@covSEiso,@likErf,temp,yt};

temp = [0;0; log(0.1)]; % regression GP predictive mean
xt = randn(2*n,D); yt = xt(:,1).*xt(:,2); % training data

temp = {@meanGPexact,@meanZero,@covSEiso,temp,yt};

% 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 = [false,true]; % mask excluding all but the 2nd component
mma = {'meanMask',mask,ml}; hypma = hypl(mask);
mpf = {@meanPref,ml}; hyppf = 2; % linear pref with slope
mpw = {@meanWarp,ml,@sin,@cos};hypwp = 2; % sin of linear

% 0) specify mean function
mean = md; hyp = hypd; x = randn([1,s],n,1);
mean = mn; hyp = hypn;
mean = mg; hyp = hypg;
mean = me; hyp = hype;
mean = m0; hyp = hyp0;
mean = msu; hyp = hypsu;
mean = mpr; hyp = hyppr;
mean = mpo; hyp = hyppo;
mean = mpf; hyp = hyppf;

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

% 2) evaluate the function on x
feval(mean{:},hyp,x)
```
52  % 3) evaluate and compute the derivatives
53  mean = m1; hyp = hyp1;
54  [m, dm] = feval(mean{:}, hyp, x)
6 Covariance Functions

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

6.1 Interface

Again, the interface is simple since only evaluation of the full covariance matrix \( K = k_\Psi(\mathcal{X}) \) and its derivatives \( K_i = \frac{\partial}{\partial \Psi_i} K \) as well as cross terms \( k_* = k_\Psi(\mathcal{X}, x_*) \) and \( k_{**} = k_\Psi(x_*, x_*) \) for prediction are required.

```
% Covariance functions to be use by Gaussian process functions. There are two
% different kinds of covariance functions: simple and composite:
1) Elementary and standalone covariance functions:
% covZero - zero covariance function
% covEye - unit covariance function
% covOne - unit constant covariance function
% covDiscrete - precomputed covariance for discrete data

2) Composite covariance functions:
% covScale - scaled version of a covariance function
% covSum - sums of covariance functions
% covProd - products of covariance functions
% covMask - mask some dimensions of the data
% covPref - difference covariance for preference learning
% covPER - make stationary covariance periodic
% covADD - additive covariance function
% covWarp - warp input to a covariance function

3) Mahalanobis distance based covariances and their modes
% covMaha - generic "mother" covariance
% * eye - unit length scale
% * iso - isotropic length scale
% * ard - automatic relevance determination
% * prot - (low-rank) projection in input space
% * fact - factor analysis covariance
% * vlen - spatially varying length scale
% covGE - Gamma exponential covariance
% covMatern - Matern covariance function with \( \nu = 1/2, 3/2 \) or 5/2
% covPP - piecewise polynomial covariance function (compact support)
% covRQ - rational quadratic covariance function
% covSE - squared exponential covariance function

4) Dot product based covariances and their modes
% covDot - generic "mother" covariance
% * eye - unit length scale
% * iso - isotropic length scale
% * ard - automatic relevance determination
% * pro - (low-rank) projection in input space
% * fac - factor analysis covariance
% covLIN - linear covariance function
% covPoly - polynomial covariance function
```
5) Time series covariance functions on the positive real line

- covFBM - fractional Brownian motion covariance
- covULL - underdamped linear Langevin process covariance
- covW - i-times integrated Wiener process covariance
- covOU - i-times integrated Ornstein-Uhlenbeck process covariance

6) Standalone covariances

- covNNone - neural network covariance function
- covLINone - linear covariance function with bias
- covPeriodic - smooth periodic covariance function (1d)
- covPeriodicNoDC - as above but with zero DC component and properly scaled
- covCos - sine periodic covariance function (1d) with unit period
- covGabor - Gabor covariance function

7) Shortcut covariances assembled from library

- covConst - covariance for constant functions
- covNoise - independent covariance function (i.e. white noise)
- covPERiso - make isotropic stationary covariance periodic
- covPERard - make ARD stationary covariance periodic
- covMaterniso - Matern covariance function with nu=1/2, 3/2 or 5/2
- covMaternard - Matern covariance function with nu=1/2, 3/2 or 5/2 with ARD
- covPPiso - piecewise polynomial covariance function (compact support)
- covPPard - piecewise polynomial covariance function (compact support)
- covRQiso - isotropic rational quadratic covariance function
- covRQard - rational quadratic covariance function with ARD
- covSEiso - isotropic squared exponential covariance function
- covSEisoU - same as above but without latent scale
- covSEard - squared exponential covariance function with ARD
- covSEvlen - spatially varying lengthscale squared exponential
- covSEproj - projection squared exponential covariance function
- covLINiso - linear covariance function
- covLINard - linear covariance function with ARD
- covGaborsio - isotropic Gabor covariance function
- covSM - spectral mixture covariance function

8) Special purpose (approximation) covariance functions

- apxSparse - sparse approximation: to be used for large scale inference problems with inducing points aka FITC
- apxGrid - grid interpolation: to be used for large scale inference problems with Kronecker/Toeplitz/BTTB covariance matrix

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

2) With two input arguments and one output argument:

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)'.

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 and one output argument:

Kz = cov(hyp, x, z)
kx = cov(hyp, x, 'diag')

The function computes test set covariances; kx is a vector of self covariances
for the test cases in x (of length n) and Kz is an (n by nz) matrix of cross
covariances between training cases x and test cases z.

4) With two output arguments:

[K,dK] = cov(hyp, x) equivalent to [K,dK] = cov(hyp, x, [])

[K,dK] = cov(hyp, x, z)

[K,dK] = cov(hyp, x, 'diag')

The function computes and returns the covariances K as in 3) above.
In addition to that, the (linear) directional derivative **function** dK is
returned. The two possible calls dhyp = dK(Q) and [dhyp,dx] = dK(Q) for a
direction Q of the same size as K are possible. The return arguments dhyp
and dx are the directional derivatives dhyp = d trace(Q*K) / d hyp and
dx = d trace(Q*K) / d x are of the same size as the hyperparameter
vector hyp and the input data x, respectively. The components of dhyp and
dx are defined as follows: dhyp(i) = trace(Q'*( d K / d hyp(i) ))
and dx(i,j) = trace(Q'*( d K / d x(i,j) )).

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 = {@apxSparse, @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 USAGECOV

help covFunction
6.2 Implemented Covariance Functions

Similarly to the mean functions, we provide a whole algebra of covariance functions \( k : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R} \) with the same generic name pattern \texttt{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.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Parameters</th>
<th>Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{RBF} )</td>
<td>Fractional Brownian motion covariance with Hurst index ( H )</td>
<td>( \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R} )</td>
<td>( \text{cov} )</td>
</tr>
<tr>
<td>( \text{ULL} )</td>
<td>Underdamped linear Langevin process covariance</td>
<td>( \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R} )</td>
<td>( \text{cov} )</td>
</tr>
<tr>
<td>( \text{Spline} )</td>
<td>Neural net, ( \mathcal{X} \subseteq \mathbb{R}^D )</td>
<td>( \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R} )</td>
<td>( \text{cov} )</td>
</tr>
<tr>
<td>( \text{Linear} )</td>
<td>Linear with bias, ( \mathcal{X} \subseteq \mathbb{R}^D )</td>
<td>( \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R} )</td>
<td>( \text{cov} )</td>
</tr>
<tr>
<td>( \text{Periodic} )</td>
<td>Periodic, ( \mathcal{X} \subseteq \mathbb{R}^D )</td>
<td>( \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R} )</td>
<td>( \text{cov} )</td>
</tr>
<tr>
<td>( \text{Gabor} )</td>
<td>Gabor function, ( \mathcal{X} \subseteq \mathbb{R}^D, \mathbf{A}, \mathbf{p} \in \mathbb{R}^D )</td>
<td>( \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R} )</td>
<td>( \text{cov} )</td>
</tr>
</tbody>
</table>
The spectral mixture covariance \( \text{covSM} \) was introduced by Wilson & Adams [Gaussian Process Kernels for Pattern Discovery and Extrapolation], ICML, 2013.

The periodic covariance function \( \text{covPER} \) starts from a stationary covariance function that depends on the data only through a distance \( r^2 = (x - x^*)^\top A^{-2}(x - x^*) \) such as \( \text{covMatern} \), \( \text{covPP} \), \( \text{covRQ} \), \( \text{covSE} \) and turns them into a periodic covariance function by embedding the data \( x \in \mathbb{R}^D \) into a periodic high-dimensional space \( x_p = u(x) \in \mathbb{R}^{2D} \) by a function \( u(x) = 2\pi \text{diag}(p^{-1})x \).

The additive covariance function \( \text{covADD} \) starts from a one-dimensional covariance function \( \kappa(x_1, x_1', \psi_i) \) acting on a single component \( i \in [1, \ldots, D] \) of \( x \). From that, we define covariance functions \( \kappa_1(x_1, x_1') = \prod_{i=1}^{D} \kappa(x_1, x_1', \psi_i) \) acting on vector-valued inputs \( x_1 \). 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 \( |I| \) 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 \( \text{cf} \) is defined by

\[
\text{cv} := \text{'func'} \mid @\text{func} \mid \text{simple} \mid \text{composite} \mid \text{i.e., it is either a string containing the name of a covariance function, a pointer to a covariance function or one of the former in combination with a cell array of covariance functions and an additional list of parameters.}
\]

46 doc/usageCov.m

1 \% demonstrate usage of covariance functions
2 \%
3 \% See also covFunctions.m.
4 \%
5 \%
6 clear all, close all
7 n = 5; D = 3; x = randn(n,D); xs = randn(3,D); \% create a data set
8
% set up simple covariance functions
9 cn = {'covNoise'}; sn = .1; hypn = log(sn); % one hyperparameter
10 cc = @covConst; sf = 2; hypc = log(sf); % function handles OK
11 ce = @covEye; hype = [] ; % identity
12 cl = @covLIN; hypl = [] ; % linear is parameter-free
13 cla = {'covLINard'}; L = rand(D,1); hypla = log(L); % linear (ARD)
14 cli = {'covLINiso'}; l = rand(1); hypli = log(l); % linear iso
15 cp = @covPoly,3; c = 2; hypp = log([c;sf]); % third order poly
16 cga = @covSEard; hypga = log([L;sf]); % Gaussian with ARD
17 cvl = {'covSEvlen', {@meanLinear}}; hypvl = [1;2;1; 0]; % var lenscale
18 cd = @covDiscrete,s; % discrete covariance function
19 \% 0) specify a covariance function
20 \% cov = cma; hyp = hypma;
21 \% cov = cci; hyp = hypcc;
22 \% cov = csm; hyp = hypsm;
23 \% cov = cds; hyp = hypdo;
24 % 1) query the number of parameters
25 feval(cov{:})
7 Hyperpriors

A hyperprior \( p(\theta) \) with \( \theta = [\rho, \phi, \psi] \) is a joint probability distribution over the likelihood hyperparameters \( \rho \), the mean hyperparameters \( \phi \) and the covariance hyperparameters \( \psi \). We concentrate on factorial priors \( p(\theta) = \prod_j p_j(\theta_j) \). Hyperpriors can be used to regularise the optimisation of the hyperparameters via the marginal likelihood \( Z(\theta) \) so that \( p(\theta)Z(\theta) \) is maximised instead. As we wish to perform unconstrained optimisation, we require (mainly) smooth hyperpriors with infinite support.

7.1 Interface

In the GPML toolbox, a prior distribution \( p(\theta) \) needs to implement the evaluation of the log density \( \ln p(\theta) \) and its first derivative \( \frac{\partial}{\partial \theta} \ln p(\theta) \). In addition, we require sampling capabilities i.e. the generation of \( \theta \sim p(\theta) \).

48

\begin{verbatim}
1 % prior distributions to be used for hyperparameters of Gaussian processes
2 % using infPrior.
3 % There are two different kinds of prior distributions: simple and composite:
4 %
5 % simple prior distributions:
6 %
7 % priorGauss - univariate Gaussian
8 % priorLaplace - univariate Laplace
9 % priorT - univariate Student’s t
10 %
11 % priorSmoothBox1 - univariate interval (linear decay in log domain)
12 % priorSmoothBox2 - univariate interval (quadr. decay in log domain)
13 %
14 % priorGamma - univariate Gamma, IR+
15 % priorWeibull - univariate Weibull, IR+
16 % priorInvGauss - univariate Inverse Gaussian, IR+
17 % priorLogNormal - univariate Log-normal, IR+
18 %
19 % priorClamped or - fix hyperparameter to its current value by setting
20 % priorDelta derivatives to zero, no effect on marginal likelihood
21 %
22 % priorGaussMulti - multivariate Gauss
23 % priorLaplaceMulti - multivariate Laplace
24 % priorTMulti - multivariate Student’s t
25 %
26 % priorClampedMulti or - fix hyperparameter to its current value by setting
27 % priorDeltaMulti derivatives to zero, no effect on marginal likelihood
28 %
29 % priorEqualMulti or - make several hyperparameters have the same value by
30 % priorSameMulti same derivative, no effect on marginal likelihood
\end{verbatim}
% composite prior distributions (see explanation at the bottom):
% priorMix - nonnegative mixture of priors
% priorTransform - prior on g(t) rather than t

% Naming convention: all prior distributions are named "prior/prior*.m".

% 1) With only a fixed input arguments:
% r = priorNAME(par1,par2,parN)
% The function returns a random sample from the distribution for e.g.
% random restarts, simulations or optimisation initialisation.

% 2) With one additional input arguments:
% [lp,dlp] = priorNAME(par1,par2,parN, t)
% The function returns the log density at location t along with its first
% derivative.

% See also USAGEPRIOR, INFPRIOR

help priorDistributions
### 7.2 Implemented Hyperpriors

All code files are named according to the pattern `prior/prior<NAME>.m` for simple identification. All currently available hyperpriors are summarised in the following table.

<table>
<thead>
<tr>
<th>Simple hyperpriors p(\theta)</th>
<th>Univariate hyperpriors defined over the whole reals with mean ( \mu ) and variance ( \sigma^2 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>(&lt;NAME&gt;)</td>
<td>Meaning</td>
</tr>
<tr>
<td>Gauss</td>
<td>normally distributed hyperparameter ( \theta \in \mathbb{R} )</td>
</tr>
<tr>
<td>Laplace</td>
<td>double exponentially distributed hyperparameter ( \theta \in \mathbb{R} )</td>
</tr>
<tr>
<td>T</td>
<td>Student's t distributed hyperparameter ( \theta \in \mathbb{R} )</td>
</tr>
</tbody>
</table>

| Univariate hyperpriors with effective bounded support but defined over the whole real line |
|-----------------------------|--------------------------------------------------|
| SmoothBox1 | interval hyperparameter \( \theta \in \mathbb{R} \) i.e. \( \theta \in [a, b] \) | \( \frac{1}{\sqrt{2\pi\sigma^2}} \exp \left( -\frac{\theta^2}{2\sigma^2} \right) \) | \( \mu = \frac{a+b}{2}, \sigma^2 = \frac{a-b}{2} \) |
| SmoothBox2 | localised hyperparameter \( \theta \in \mathbb{R} \) i.e. \( \theta \in [a, b] \) | \( \frac{1}{\sqrt{2\pi\sigma^2}} \exp \left( -\frac{\theta^2}{2\sigma^2} \right) \) | \( \mu = \frac{a+b}{2}, \sigma^2 = \frac{a-b}{2} \) |

| Univariate hyperpriors supported only over the positive reals |
|-----------------------------|--------------------------------------------------|
| Gamma | Gamma hyperparameter \( \theta \in \mathbb{R}_+ \) | \( \frac{1}{\Gamma(\lambda)} \theta^{\lambda-1} \exp(-\theta) \) | \( k \in \mathbb{R}_+, t \in \mathbb{R}_+ \) |
| Weibull | Weibull hyperparameter \( \theta \in \mathbb{R}_+ \) | \( \frac{1}{\Gamma(\lambda)} \theta^{\lambda-1} \exp(-\lambda\theta) \) | \( k \in \mathbb{R}_+, \lambda \in \mathbb{R}_+ \) |
| InverseGauss | inverse Gaussian hyperparameter \( \theta \in \mathbb{R}_+ \) | \( \frac{1}{\sqrt{2\pi\sigma^2}} \exp \left( -\frac{\theta^2}{2\sigma^2} \right) \) | \( k \in \mathbb{R}_+, \lambda \in \mathbb{R}_+ \) |
| LogNormal | log-normal hyperparameter \( \theta \in \mathbb{R}_+ \) | \( \frac{1}{\sqrt{2\pi\sigma^2}} \exp \left( -\frac{\theta^2}{2\sigma^2} \right) \) | \( k \in \mathbb{R}_+, \sigma^2 \in \mathbb{R}_+ \) |

| Multivariate hyperpriors supported all over \( \mathbb{R}^D \) with mean \( \mu \) and covariance \( \Sigma \) |
|-----------------------------|--------------------------------------------------|
| GaussMulti | multivariate normal distribution \( \theta \in \mathbb{R}^D \) | \( \frac{1}{(2\pi)^{D/2}|\Sigma|^{1/2}} \exp \left( -\frac{1}{2} (\theta - \mu)^\top \Sigma^{-1} (\theta - \mu) \right) \) | \( \mu \in \mathbb{R}^D, \Sigma \in \mathbb{R}^{D \times D} \) |
| LaplaceMulti | multivariate Laplace distribution \( \theta \in \mathbb{R}^D \) | \( \frac{1}{\sqrt{2|\Sigma|}} \exp \left(-\frac{1}{2} \|L^{1/2}(\theta - \mu)\|_2 \right) \) | \( \mu \in \mathbb{R}^D, \Sigma \in \mathbb{R}^{D \times D} \) |
| TMulti | multivariate Student's t distribution \( \theta \in \mathbb{R}^D \) | \( \left( |v-2|\pi|\Sigma| \right)^{-1/2} \left( 1 + \frac{|\theta - \mu|^2}{\nu|\Sigma|^{-1}(\theta - \mu)} \right)^{-\frac{v+D}{2}} \) | \( \mu \in \mathbb{R}^D, \Sigma \in \mathbb{R}^{D \times D}, v \in \mathbb{R}_+ \) |

| Improper hyperpriors used to fix the value of a particular hyperparameter |
|-----------------------------|--------------------------------------------------|
| Delta | clamped hyperparameter \( \theta = \theta_0 \in \mathbb{R} \) | \( \delta(\theta - \theta_0) \) | \( \emptyset \) |
| DeltaMulti | clamped hyperparameter \( \theta = \theta_0 \in \mathbb{R}^D \) | \( \delta(\theta - \theta_0) \) | \( \emptyset \) |
| ClampedMulti | same hyperparameter \( \theta = \theta_0 \in \mathbb{R}^D \) | \( \delta(\theta - \theta_0) \) | \( \emptyset \) |
| SameMulti | same hyperparameter \( \theta = \theta_0 \in \mathbb{R}^D \) | \( \delta(\theta - \theta_0) \) | \( \emptyset \) |

| Composite hyperpriors \( [\eta_1(\theta), \eta_2(\theta), \ldots] \to p(\theta) \) |
|-----------------------------|--------------------------------------------------|
| Transform | prior distribution on \( g(\theta) \) instead of \( \theta \) | \( \eta(g(\theta)) \) | \( \emptyset \) |
| Mix | mixture distribution | \( \sum_i w_i \eta_i(\theta) \) | \( \emptyset \) |

The `priorSmoothBox2` is a Gauss-uniform sandwich obtained by complementing a uniform distribution on \([a, b]\) with two Gaussian halves at each side. The parameter \( \eta \) balances the probability mass between the constituents so that \( \eta/(\eta + 1) \) is used for the box and \( 1/(\eta + 1) \) for the Gaussian sides. Its brother `priorSmoothBox1` is the product of two sigmoidal functions.

The `priorDelta` or equivalently `priorClamped` can be used to exclude some hyperparameters from the optimisation. Their values are clamped to \( \theta_0 \) and the derivative vanishes. There are also multivariate counterparts `priorDeltaMulti` and `priorClampedMulti`.

### 7.3 Usage of Implemented Hyperpriors

Some code examples taken from `doc/usagePrior.m` illustrate how to use univariate, multivariate and composite priors on hyperparameters. Syntactically, a hyperprior \( \text{hp} \) is defined by

```matlab
func := Dist       // prior distributions in prior/
| Clamped | Delta // predefined for fixing the hyperparameter
pr := 'func'       | @func                              // univariate hyperprior
```

The hyperprior \( \text{hp} \) is defined by the function \( \text{func} \), which can be any of the implemented hyperpriors. The hyperparameter \( \theta \) is then sampled from the prior distribution \( p(\theta) \) defined by \( \text{func} \).
i.e., it is either a string containing the name of a hyperprior function, a pointer to a hyperprior function or one of the former in combination with a cell array of hyperprior functions and an additional list of parameters. Furthermore, we have multivariate hyperprior variants and 2 (equivalent) predefined hyperpriors allowing to exclude variables from optimisation.
par = {mean, cov, lik, x, y}; mfun = @minimize; % input for GP function

% a) plain marginal likelihood optimisation (maximum likelihood)
im = @infExact; % inference method
hyp_plain = feval(mfun, hyp, @gp, -10, im, par{:}); % optimise

% b) regularised optimisation (maximum a posteriori) with 1d priors
prior.mean = {pg;pc}; % Gaussian prior for first, clamp second par
prior.cov = {p1;[]}; % box prior for first, nothing for second par
im = {@infPrior,@infExact,prior}; % inference method
hyp_p1 = feval(mfun, hyp, @gp, -10, im, par{:}); % optimise

% c) regularised optimisation (maximum a posteriori) with Nd priors
prior = [];
% clear the structure
prior.multi{1} = {@priorTMulti,[mu;mu],diag([s2,s2]),nu,...
  struct('mean',[1,2])}; % use hyper struct
% Equivalent shortcut (same mu and s2 for all dimensions)
prior.multi{1} = {@priorTMulti,mu,s2,nu,struct('mean',[1,2])};
% multivariate Gaussian prior jointly on 1st and 3rd hyper
prior.multi{2} = {@priorGaussMulti,[mu;mu],diag([s2,s2]),...
  [1,3]}; % use unwrapped hyper vector
% Equivalent shortcut (same mu and s2 for all dimensions)
prior.multi{2} = {@priorGaussMulti,mu,s2,[1,3]};
im = {@infPrior,@infExact,prior}; % inference method
hyp_pN = feval(mfun, hyp, @gp, -10, im, par{:}); % optimise

[any2vec(hyp), any2vec(hyp_plain), any2vec(hyp_p1), any2vec(hyp_pN)]