The GPML Toolbox version 3.1

Carl Edward Rasmussen & Hannes Nickisch

February 18, 2011

Abstract

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

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

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

Contents

1 Gaussian Process Training and Prediction

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

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

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

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

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

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

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

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

In such cases, the entire posterior can be computed from the two vectors `post.alpha` and `post.s`; the inference method may optionally also return $L = \text{chol}(\text{diag}(\mathbf{s})K \, \text{diag}(\mathbf{s}) + I)$.

If on the other hand the posterior doesn't admit the above form, then `post.L` returns the matrix $L = -(K + W^{-1})^{-1}$ (and `post.s` is unused). In addition, a sparse representation of the posterior may be used, in which case the non-zero elements of the `post.alpha` vector indicate the active entries.

2 The gp Function

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

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

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

4b ⟨*gp function help* 4b⟩≡   (4a)
```
 1 % Gaussian Process inference and prediction. The gp function provides a
 2 % flexible framework for Bayesian inference and prediction with Gaussian
 3 % processes for scalar targets, i.e. both regression and binary
 4 % classification. The prior is Gaussian process, defined through specification
 5 % of its mean and covariance function. The likelihood function is also
 6 % specified. Both the prior and the likelihood may have hyperparameters
 7 % associated with them.
 8 %
 9 % Two modes are possible: training or prediction: if no test cases are
10 % supplied, then the negative log marginal likelihood and its partial
11 % derivatives w.r.t. the hyperparameters is computed; this mode is used to fit
12 % the hyperparameters. If test cases are given, then the test set predictive
13 % probabilities are returned. Usage:
14 %
15 %   training: [nlZ dnlZ          ] = gp(hyp, inf, mean, cov, lik, x, y);
16 % prediction: [ymu ys2 fmu fs2   ] = gp(hyp, inf, mean, cov, lik, x, y, xs);
17 %         or: [ymu ys2 fmu fs2 lp] = gp(hyp, inf, mean, cov, lik, x, y, xs, ys);
18 %
19 % where:
20 %
21 %   hyp      column vector of hyperparameters
22 %   inf      function specifying the inference method
23 %   cov      prior covariance function (see below)
24 %   mean     prior mean function
25 %   lik      likelihood function
26 %   x        n by D matrix of training inputs
27 %   y        column vector of length n of training targets
28 %   xs       ns by D matrix of test inputs
29 %   ys       column vector of length nn of test targets
30 %
31 %   nlZ      returned value of the negative log marginal likelihood
32 %   dnlZ     column vector of partial derivatives of the negative
33 %                log marginal likelihood w.r.t. each hyperparameter
34 %   ymu      column vector (of length ns) of predictive output means
35 %   ys2      column vector (of length ns) of predictive output variances
36 %   fmu      column vector (of length ns) of predictive latent means
37 %   fs2      column vector (of length ns) of predictive latent variances
38 %   lp       column vector (of length ns) of log predictive probabilities
39 %
```

4

```
40 %   post     struct representation of the (approximate) posterior
41 %              3rd output in training mode and 6th output in prediction mode
42 %
43 % See also covFunctions.m, infMethods.m, likFunctions.m, meanFunctions.m.
44 %
45 ⟨gpml copyright 5a⟩
```

5a   *⟨gpml copyright 5a⟩≡*   (4b 8 9 13 15 23 24 26 29)

```
1 % Copyright (c) by Carl Edward Rasmussen and Hannes Nickisch, 2011-02-18
```

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

5b   *⟨initializations 5b⟩≡*   (4a)

```
1 ⟨minimalist usage 5c⟩
2 ⟨process input arguments 5d⟩
3 ⟨check hyperparameters 5e⟩
```

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

5c   *⟨minimalist usage 5c⟩≡*   (5b)

```
1 if nargin<7 || nargin>9
2   disp('Usage: [nlZ dnlZ          ] = gp(hyp, inf, mean, cov, lik, x, y);')
3   disp('   or: [ymu ys2 fmu fs2   ] = gp(hyp, inf, mean, cov, lik, x, y, xs);')
4   disp('   or: [ymu ys2 fmu fs2 lp] = gp(hyp, inf, mean, cov, lik, x, y, xs, ys);')
5   return
6 end
```

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

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

```
1 if isempty(inf),  inf = @infExact; else                       % set default inf
2   if iscell(inf), inf = inf{1}; end                    % cell input is allowed
3   if ischar(inf), inf = str2func(inf); end       % convert into function handle
4 end
5 if isempty(mean), mean = {@meanZero}; end                    % set default mean
6 if ischar(mean) || isa(mean, 'function_handle'), mean = {mean}; end  % make cell
7 if isempty(cov), error('Covariance function cannot be empty'); end  % no default
8 if ischar(cov)  || isa(cov,  'function_handle'), cov  = {cov};  end  % make cell
9 cov1 = cov{1}; if isa(cov1, 'function_handle'), cov1 = func2str(cov1); end
10 if strcmp(cov1,'covFITC'); inf = @infFITC; end      % only one possible inf alg
11 if isempty(lik),  lik = @likGauss; else                       % set default lik
12   if iscell(lik), lik = lik{1}; end                    % cell input is allowed
13   if ischar(lik), lik = str2func(lik); end       % convert into function handle
14 end
15 D = size(x,2);
```

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

5e   *⟨check hyperparameters 5e⟩≡*   (5b)

```
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
```

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

6a   ⟨*inference* 6a⟩≡   (4a)
```
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(func2str(lik),'likErf') || strcmp(func2str(lik),'likLogistic')
5      uy = unique(y);
6      if any( uy~=+1 & uy~=-1 )
7        warning('You attempt classification using labels different from {+1,-1}\n')
8      end
9    end
10   if nargin>7   % compute marginal likelihood and its derivatives only if needed
11     post = inf(hyp, mean, cov, lik, x, y);
12   else
13     if nargout==1
14       [post nlZ] = inf(hyp, mean, cov, lik, x, y); dnlZ = {};
15     else
16       [post nlZ dnlZ] = inf(hyp, mean, cov, lik, x, y);
17     end
18   end
19 catch
20   msgstr = lasterr;
21   if nargin > 7, error('Inference method failed [%s]', msgstr); else
22     warning('Inference method failed [%s] .. attempting to continue',msgstr)
23     dnlZ = struct('cov',0*hyp.cov, 'mean',0*hyp.mean, 'lik',0*hyp.lik);
24     varargout = {NaN, dnlZ}; return                    % continue with a warning
25   end
26 end
```

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

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

6b   ⟨*compute test predictions* 6b⟩≡   (4a)
```
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)); end                     % non-sparse representation
7  if numel(L)==0                        % 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 Ltril = all(all(tril(L,-1)==0));              % is L an upper triangular matrix?
12 ns = size(xs,1);                                      % number of data points
```

6

```
13 nperbatch = 1000;                            % number of data points per mini batch
14 nact = 0;                        % number of already processed test data points
15 ymu = zeros(ns,1); ys2 = ymu; fmu = ymu; fs2 = ymu; lp = ymu;    % allocate mem
16 while nact<ns                % process minibatches of test cases to save memory
17   id = (nact+1):min(nact+nperbatch,ns);            % data points to process
18   ⟨make predictions 7⟩
19   nact = id(end);           % set counter to index of last processed data point
20 end
21 if nargin<9
22   varargout = {ymu, ys2, fmu, fs2, [], post};        % assign output arguments
23 else
24   varargout = {ymu, ys2, fmu, fs2, lp, post};
25 end
```

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

7   ⟨make predictions 7⟩≡   (6b)
```
1 kss = feval(cov{:}, hyp.cov, xs(id,:), 'diag');              % self-variance
2 Ks  = feval(cov{:}, hyp.cov, x(nz,:), xs(id,:));        % cross-covariances
3 ms = feval(mean{:}, hyp.mean, xs(id,:));
4 fmu(id) = ms + Ks'*full(alpha(nz));                       % predictive means
5 if Ltril           % L is triangular => use Cholesky parameters (alpha,sW,L)
6   V  = L'\(repmat(sW,1,length(id)).*Ks);
7   fs2(id) = kss - sum(V.*V,1)';                          % predictive variances
8 else                 % L is not triangular => use alternative parametrisation
9   fs2(id) = kss + sum(Ks.*(L*Ks),1)';                    % predictive variances
10 end
11 fs2(id) = max(fs2(id),0);   % remove numerical noise i.e. negative variances
12 if nargin<9
13   [lp(id) ymu(id) ys2(id)] = lik(hyp.lik, [], fmu(id), fs2(id));
14 else
15   [lp(id) ymu(id) ys2(id)] = lik(hyp.lik, ys(id), fmu(id), fs2(id));
16 end
```

3 Inference Methods

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

8   ⟨*infMethods.m* 8⟩≡

```
1  % Inference methods: Compute the (approximate) posterior for a Gaussian process.
2  % Methods currently implemented include:
3  %
4  %   infExact     Exact inference (only possible with Gaussian likelihood)
5  %   infFITC      Large scale regression with approximate covariance matrix
6  %   infLaplace   Laplace's Approximation
7  %   infEP        Expectation Propagation
8  %   infVB        Variational Bayes
9  %
10 %   infLOO       Leave-One-Out predictive probability and Least-Squares Approxim.
11 %
12 % The interface to the approximation methods is the following:
13 %
14 %   function [post nlZ dnlZ] = inf..(hyp, cov, lik, x, y)
15 %
16 % where:
17 %
18 %   hyp      is a struct of hyperparameters
19 %   cov      is the name of the covariance function (see covFunctions.m)
20 %   lik      is the name of the likelihood function (see likFunctions.m)
21 %   x        is a n by D matrix of training inputs
22 %   y        is a (column) vector (of size n) of targets
23 %
24 %   nlZ      is the returned value of the negative log marginal likelihood
25 %   dnlZ     is a (column) vector of partial derivatives of the negative
26 %               log marginal likelihood w.r.t. each hyperparameter
27 %   post     struct representation of the (approximate) posterior containing
28 %     alpha  is a (sparse or full column vector) containing inv(K)*m, where K
29 %               is the prior covariance matrix and m the approx posterior mean
30 %     sW     is a (sparse or full column) vector containing diagonal of sqrt(W)
31 %               the approximate posterior covariance matrix is inv(inv(K)+W)
32 %     L      is a (sparse or full) matrix, L = chol(sW*K*sW+eye(n))
33 %
34 % Usually, the approximate posterior to be returned admits the form
35 % N(m=K*alpha, V=inv(inv(K)+W)), where alpha is a vector and W is diagonal;
36 % if not, then L contains instead -inv(K+inv(W)), and sW is unused.
37 %
38 % For more information on the individual approximation methods and their
39 % implementations, see the separate inf??.m files. See also gp.m
40 %
41  ⟨gpml copyright 5a⟩
```

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

3.1 Exact Inference

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

9 ⟨*inf/infExact.m* 9⟩≡

```matlab
1  function [post nlZ dnlZ] = infExact(hyp, mean, cov, lik, x, y)
2
3  % Exact inference for a GP with Gaussian likelihood. Compute a parametrization
4  % of the posterior, the negative log marginal likelihood and its derivatives
5  % w.r.t. the hyperparameters. See also "help infMethods".
6  %
7  ⟨gpml copyright 5a⟩
8  %
9  % See also INFMETHODS.M.
10
11 likstr = lik; if ~ischar(lik), likstr = func2str(lik); end
12 if ~strcmp(likstr,'likGauss')              % NOTE: no explicit call to likGauss
13   error('Exact inference only possible with Gaussian likelihood');
14 end
15
16 [n, D] = size(x);
17 K = feval(cov{:}, hyp.cov, x);                     % evaluate covariance matrix
18 m = feval(mean{:}, hyp.mean, x);                       % evaluate mean vector
19
20 sn2 = exp(2*hyp.lik);                              % noise variance of likGauss
21 L = chol(K/sn2+eye(n));              % Cholesky factor of covariance with noise
22 alpha = solve_chol(L,y-m)/sn2;
23
24 post.alpha = alpha;                        % return the posterior parameters
25 post.sW = ones(n,1)/sqrt(sn2);              % sqrt of noise precision vector
26 post.L  = L;                                  % L = chol(eye(n)+sW*sW'.*K)
27
28 if nargout>1                             % do we want the marginal likelihood?
29   nlZ = (y-m)'*alpha/2 + sum(log(diag(L))) + n*log(2*pi*sn2)/2;  % -log marg lik
30   if nargout>2                                       % do we want derivatives?
31     dnlZ = hyp;                              % allocate space for derivatives
32     Q = solve_chol(L,eye(n))/sn2 - alpha*alpha';    % precompute for convenience
33     for i = 1:numel(hyp.cov)
34       dnlZ.cov(i) = sum(sum(Q.*feval(cov{:}, hyp.cov, x, [], i)))/2;
35     end
36     dnlZ.lik = sn2*trace(Q);
37     for i = 1:numel(hyp.mean),
38       dnlZ.mean(i) = -feval(mean{:}, hyp.mean, x, i)'*alpha;
39     end
40   end
41 end
```

3.2 Laplace's Approximation

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

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

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

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

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

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

and

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

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

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

and

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

return exactly these values.

3.3 Expectation Propagation

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

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

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

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

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

and

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

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

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

and

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

3.4 Variational Bayes

Based on individual lower bounds to every likelihood

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

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

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

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

In practice, we use a Newton algorithm requiring

$$dh_k = \frac{\partial^k}{\partial\gamma^k}h(\gamma), \ db_k = \frac{\partial^k}{\partial\gamma^k}\beta(\gamma), \quad k = 0,1,2$$

and

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

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

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

and

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

4 Likelihood Functions

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

Likelihood functionality is needed both during inference and while predicting.

4.1 Prediction

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

$$p(y_*|\mathcal{D}, x_*) \quad = \quad \int p(y_*|f_*)p(f_*|\mathcal{D}, x_*)df_*. \tag{3}$$

$$\approx \quad \int p(y_*|f_*)\mathcal{N}(f_*|\mu_{f_*}, \sigma^2_{f_*})df_*. \tag{4}$$

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

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

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

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

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

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

$$p_{y_*} \quad = \quad \begin{cases} \pi_* & y_* = +1 \\ 1 - \pi_* & y_* = -1 \end{cases}$$

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

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

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

$$\mu_{y_*} \quad \approx \quad \mu_{f_*}, \text{ and}$$

$$\sigma^2_{y_*} \quad \approx \quad \sigma^2_{f_*} + \int y_*^2 p(y_*|0)dy_*.$$

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.

13 ⟨*likFunctions.m* 13⟩≡

```
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 %
6 %    likGauss        (Gaussian, regression)
7 %    likLaplace      (Laplacian or double exponential, regression)
8 %    likSech2        (Sech-square, regression)
9 %    likT            (Student's t, regression)
10 %
11 % The likelihood functions have three possible modes, the mode being selected
12 % as follows (where "lik" stands for any likelihood function in "lik/lik*.m".):
13 %
14 % 1) With one or no input arguments:          [REPORT NUMBER OF HYPERPARAMETERS]
15 %
16 %    s = lik OR s = lik(hyp)
17 %
18 % The likelihood function returns a string telling how many hyperparameters it
19 % expects, using the convention that "D" is the dimension of the input space.
20 % For example, calling "likLogistic" returns the string '0'.
21 %
22 %
23 % 2) With three or four input arguments:                    [PREDICTION MODE]
24 %
25 %    lp = lik(hyp, y, mu) OR [lp, ymu, ys2] = lik(hyp, y, mu, s2)
26 %
27 % This allows to evaluate the predictive distribution. Let p(y_*|f_*) be the
28 % likelihood of a test point and N(f_*|mu,s2) an approximation to the posterior
29 % marginal p(f_*|x_*,x,y) as returned by an inference method. The predictive
30 % distribution p(y_*|x_*,x,y) is approximated by.
31 %   q(y_*) = \int N(f_*|mu,s2) p(y_*|f_*) df_*
32 %
33 %    lp = log( q(y) ) for a particular value of y, if s2 is [] or 0, this
34 %                     corresponds to log( p(y|mu) )
35 %   ymu and ys2       the mean and variance of the predictive marginal q(y)
36 %                     note that these two numbers do not depend on a particular
37 %                     value of y
38 %  All vectors have the same size.
39 %
40 %
41 % 3) With five or six input arguments, the fifth being a string [INFERENCE MODE]
42 %
43 % [varargout] = lik(hyp, y, mu, s2, inf) OR
44 % [varargout] = lik(hyp, y, mu, s2, inf, i)
45 %
46 % There are three cases for inf, namely a) infLaplace, b) infEP and c) infVB.
47 % The last input i, refers to derivatives w.r.t. the ith hyperparameter.
48 %
49 % a1) [sum(lp),dlp,d2lp,d3lp] = lik(hyp, y, f, [], 'infLaplace')
50 % lp, dlp, d2lp and d3lp correspond to derivatives of the log likelihood
51 % log(p(y|f)) w.r.t. to the latent location f.
```

```
52 %   lp = log( p(y|f) )
53 %  dlp = d   log( p(y|f) ) / df
54 % d2lp = d^2 log( p(y|f) ) / df^2
55 % d3lp = d^3 log( p(y|f) ) / df^3
56 %
57 % a2) [lp_dhyp,d2lp_dhyp] = lik(hyp, y, f, [], 'infLaplace', i)
58 % returns derivatives w.r.t. to the ith hyperparameter
59 %   lp_dhyp = d   log( p(y|f) ) / (df   dhyp_i)
60 % d2lp_dhyp = d^3 log( p(y|f) ) / (df^2 dhyp_i)
61 %
62 %
63 % b1) [lZ,dlZ,d2lZ] = lik(hyp, y, mu, s2, 'infEP')
64 % let Z = \int p(y|f) N(f|mu,s2) df then
65 %   lZ =     log(Z)
66 %  dlZ = d   log(Z) / dmu
67 % d2lZ = d^2 log(Z) / dmu^2
68 %
69 % b2) [dlZhyp] = lik(hyp, y, mu, s2, 'infEP', i)
70 % returns derivatives w.r.t. to the ith hyperparameter
71 % dlZhyp = d log(Z) / dhyp_i
72 %
73 %
74 % c1) [h,b,dh,db,d2h,d2b] = lik(hyp, y, [], ga, 'infVB')
75 % ga is the variance of a Gaussian lower bound to the likelihood p(y|f).
76 %   p(y|f) \ge exp( b*f - f.^2/(2*ga) - h(ga)/2 ) \propto N(f|b*ga,ga)
77 % The function returns the linear part b and the "scaling function" h(ga) and
78 % derivatives dh = d h/dga, db = d b/dga, d2h = d^2 h/dga and d2b = d^2 b/dga.
79 %
80 % c2) [dhhyp] = lik(hyp, y, [], ga, 'infVB', i)
81 % dhhyp = dh / dhyp_i, the derivative w.r.t. the ith hyperparameter
82 %
83 % Cumulative likelihoods are designed for binary classification. Therefore, they
84 % only look at the sign of the targets y; zero values are treated as +1.
85 %
86 % See the help for the individual likelihood for the computations specific to
87 % each likelihood function.
88 %
89 ⟨gpml copyright 5a⟩
```

4.3 Implemented Likelihood Functions

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

<NAME>	regression $y_i \in \mathbb{R}$	$\mathbb{P}_\rho(y_i\|f_i) =$	$\rho =$
Gauss	Gaussian	$\mathcal{N}(y_i\|f_i, \sigma^2) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(y_i - f_i)^2}{2\sigma^2}\right)$	$\{\ln \sigma\}$
Sech2	Sech-squared	$\frac{\tau}{2\cosh^2(\tau(y_i - f_i))}, \; \tau = \frac{\pi}{2\sigma\sqrt{3}}$	$\{\ln \sigma\}$
Laplace	Laplacian	$\frac{1}{2b} \exp\left(-\frac{\|y_i - f_i\|}{b}\right), \; b = \frac{\sigma}{\sqrt{2}}$	$\{\ln \sigma\}$
T	Student's t	$\frac{\Gamma(\frac{\nu+1}{2})}{\Gamma(\frac{\nu}{2})} \frac{1}{\sqrt{\nu\pi}\sigma} \left(1 + \frac{(y_i - f_i)^2}{\nu\sigma^2}\right)^{-\frac{\nu+1}{2}}$	$\{\ln(\nu - 1), \ln \sigma\}$
<NAME>	classification $y_i \in \{\pm 1\}$	$\mathbb{P}_\rho(y_i\|f_i) =$	$\rho =$
Erf	Error function	$\int_{-\infty}^{y_i f_i} \mathcal{N}(t)dt$	$\emptyset$
Logistic	Logistic function	$\frac{1}{1+\exp(-y_i f_i)}$	$\emptyset$

14

4.4 Compatibility Between Likelihoods and Inference Methods

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

Likelihood \ Inference	Exact	EP	Laplace	Variational Bayes	regression
Gaussian	✓	✓	✓	✓	regression
Sech-squared		✓	✓	✓	regression
Laplacian		✓		✓	regression
Student's t			✓	✓	regression
Error function		✓	✓	✓	probit regression
Logistic function		✓	✓	✓	logit regression

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

4.5 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.

15 ⟨*lik/likGauss.m* 15⟩≡

```
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 likelihoods.m for the details. In general, care is taken
14 % to avoid numerical issues when the arguments are extreme.
15 %
16 % See also likFunctions.m.
17 %
18 ⟨gpml copyright 5a⟩
19
20 if nargin<2, varargout = {'1'}; return; end   % report number of hyperparameters
21
22 sn2 = exp(2*hyp);
23
24 if nargin<5                              % prediction mode if inf is not present
25   ⟨Prediction with Gaussian likelihood 16a⟩
26 else
27   switch inf
```

```
28    case 'infLaplace'
29        ⟨Laplace's method with Gaussian likelihood 16b⟩
30    case 'infEP'
31        ⟨EP inference with Gaussian likelihood 17a⟩
32    case 'infVB'
33        ⟨Variational Bayes inference with Gaussian likelihood 17b⟩
34    end
35 end
```

16a   ⟨*Prediction with Gaussian likelihood* 16a⟩≡   (15)
```
1 if numel(y)==0,  y = zeros(size(mu)); end
2 s2zero = 1; if nargin>3, if norm(s2)>0, s2zero = 0; end, end        % s2==0 ?
3 if s2zero                                                          % log probability
4   lp = -(y-mu).^2./sn2/2-log(2*pi*sn2)/2; s2 = 0;
5 else
6   lp = likGauss(hyp, y, mu, s2, 'infEP');                         % prediction
7 end
8 ymu = {}; ys2 = {};
9 if nargout>1
10   ymu = mu;                                                       % first y moment
11   if nargout>2
12     ys2 = s2 + sn2;                                               % second y moment
13   end
14 end
15 varargout = {lp,ymu,ys2};
```

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

4.5.1 Exact Inference

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

4.5.2 Laplace's Approximation

16b   ⟨*Laplace's method with Gaussian likelihood* 16b⟩≡   (15)
```
1 if nargin<6                                                        % no derivative mode
2   if numel(y)==0, y=0; end
3   ymmu = y-mu; dlp = {}; d2lp = {}; d3lp = {};
4   lp = -ymmu.^2/(2*sn2) - log(2*pi*sn2)/2;
5   if nargout>1
6     dlp = ymmu/sn2;                           % dlp, derivative of log likelihood
7     if nargout>2                      % d2lp, 2nd derivative of log likelihood
8       d2lp = -ones(size(ymmu))/sn2;
9       if nargout>3                    % d3lp, 3rd derivative of log likelihood
10        d3lp = zeros(size(ymmu));
11      end
12    end
13  end
14  varargout = {sum(lp),dlp,d2lp,d3lp};
15 else                                                              % derivative mode
16  lp_dhyp = (y-mu).^2/sn2 - 1;  % derivative of log likelihood w.r.t. hypers
17  d2lp_dhyp = 2*ones(size(mu))/sn2;   % and also of the second mu derivative
18  varargout = {lp_dhyp,d2lp_dhyp};
19 end
```

4.5.3 Expectation Propagation

17a   *⟨EP inference with Gaussian likelihood* 17a⟩≡   (15)

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

4.5.4 Variational Bayes

17b   *⟨Variational Bayes inference with Gaussian likelihood* 17b⟩≡   (15)

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

4.6 Laplace Likelihood

4.6.1 Laplace's Approximation

The following derivatives are needed:

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

4.6.2 Expectation Propagation

Expectation propagation requires integration against a Gaussian measure for moment matching.

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

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

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

$$
\begin{aligned}
\ln Z &= \ln \int \exp\left(-\frac{(f - \tilde{\mu})^2}{2\tilde{\sigma}^2} - \sqrt{2}|f|\right) df \overbrace{- \ln \tilde{\sigma}\sqrt{2\pi} - \ln \sqrt{2}\sigma_n}^{C} \\
&= \ln\left[\int_{-\infty}^0 \exp\left(-\frac{(f - \tilde{\mu})^2}{2\tilde{\sigma}^2} + \sqrt{2}f\right) df + \int_0^\infty \exp\left(-\frac{(f - \tilde{\mu})^2}{2\tilde{\sigma}^2} - \sqrt{2}f\right) df\right] + C \\
&= \ln\left[\int_{-\infty}^0 \exp\left(-\frac{f^2 - 2(\overbrace{\tilde{\mu} + \tilde{\sigma}^2\sqrt{2}}^{m_-})f + \tilde{\mu}^2}{2\tilde{\sigma}^2}\right) df + \int_0^\infty \exp\left(-\frac{f^2 - 2(\overbrace{\tilde{\mu} - \tilde{\sigma}^2\sqrt{2}}^{m_+})f + \tilde{\mu}^2}{2\tilde{\sigma}^2}\right) df\right] + C \\
&= \ln\left[\exp\left(\frac{m_-^2}{2\tilde{\sigma}^2}\right)\int_{-\infty}^0 \exp\left(-\frac{(f - m_-)^2}{2\tilde{\sigma}^2}\right) df + \exp\left(\frac{m_+^2}{2\tilde{\sigma}^2}\right)\int_0^\infty \exp\left(-\frac{(f - m_+)^2}{2\tilde{\sigma}^2}\right) df\right] - \frac{\tilde{\mu}^2}{2\tilde{\sigma}^2} + C \\
&= \ln\left[\exp\left(\frac{m_-^2}{2\tilde{\sigma}^2}\right)\int_{-\infty}^0 \mathcal{N}(f|m_-, \tilde{\sigma}^2) df + \exp\left(\frac{m_+^2}{2\tilde{\sigma}^2}\right)\left(1 - \int_{-\infty}^0 \mathcal{N}(f|m_+, \tilde{\sigma}^2) df\right)\right] - \frac{\tilde{\mu}^2}{2\tilde{\sigma}^2} - \ln \sqrt{2}\sigma_n \\
&= \ln\left[\exp\left(\frac{m_-^2}{2\tilde{\sigma}^2}\right)\Phi\left(\frac{m_-}{\tilde{\sigma}}\right) - \exp\left(\frac{m_+^2}{2\tilde{\sigma}^2}\right)\Phi\left(\frac{m_+}{\tilde{\sigma}}\right) + \exp\left(\frac{m_+^2}{2\tilde{\sigma}^2}\right)\right] - \frac{\tilde{\mu}^2}{2\tilde{\sigma}^2} - \ln \sqrt{2}\sigma_n
\end{aligned}
$$

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

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

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

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

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

as well as the second derivative

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

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

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

$$= -\frac{1}{\sigma}\frac{\mathcal{N}(-z_+)\Phi(-z_+)\frac{\partial -z_+^2/2}{\partial \mu} - \mathcal{N}^2(-z_+)\frac{\partial -z_+}{\partial \mu}}{\Phi^2(-z_+)}$$

$$= \frac{\mathcal{N}(-z_+)}{\sigma^2}\cdot\frac{\Phi(-z_+)z_+ - \mathcal{N}(-z_+)}{\Phi^2(-z_+)} = -\frac{q_+^2 - q_+z_+}{\sigma^2}$$

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

$$= \frac{1}{\sigma}\frac{\mathcal{N}(z_-)\Phi(z_-)\frac{\partial -z_-^2/2}{\partial \mu} - \mathcal{N}^2(z_-)\frac{\partial z_-}{\partial \mu}}{\Phi^2(z_-)}$$

$$= \frac{\mathcal{N}(z_-)}{\sigma^2}\cdot\frac{-\Phi(z_-)z_- - \mathcal{N}(z_-)}{\Phi^2(z_-)} = -\frac{q_-^2 + q_-z_-}{\sigma^2}$$

$$\frac{\partial^2 a_\pm}{\partial \mu^2} = -\frac{q_\pm^2 \mp q_\pm z_\pm}{\sigma^2}$$

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

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

$$= \left(\frac{q_\pm}{\sigma} - \frac{\sqrt{2}}{\sigma_n}\right)^2 - \frac{q_\pm^2}{\sigma^2} \pm \frac{q_\pm z_\pm}{\sigma^2}$$

$$= \frac{2}{\sigma_n^2} - \left(\frac{\sqrt{8}}{\sigma\sigma_n} \mp \frac{z_\pm}{\sigma^2}\right)q_\pm.$$

We also need

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

4.6.3 Variational Bayes

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

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

4.7 Student's t Likelihood

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

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

4.7.1 Laplace's Approximation

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

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

4.8 Cumulative Logistic Likelihood

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

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

4.8.1 Laplace's Approximation

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

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

5 Mean Functions

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

5.1 Interface

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

23   ⟨*meanFunctions.m* 23⟩≡
```
1 % mean functions to be use by Gaussian process functions. There are two
2 % different kinds of mean functions: simple and composite:
3 %
4 % simple mean functions:
5 %
6 %   meanZero      - zero mean function
7 %   meanOne       - one mean function
8 %   meanConst     - constant mean function
9 %   meanLinear    - linear mean function
10 %
11 % composite covariance functions (see explanation at the bottom):
12 %
13 %   meanScale     - scaled version of a mean function
14 %   meanPow       - power of a mean function
15 %   meanProd      - products of mean functions
16 %   meanSum       - sums of mean functions
17 %   meanMask      - mask some dimensions of the data
18 %
19 % Naming convention: all mean functions are named "mean/mean*.m".
20 %
21 %
22 % 1) With no or only a single input argument:
23 %
24 %    s = meanNAME   or   s = meanNAME(hyp)
25 %
26 % The mean function returns a string s telling how many hyperparameters hyp it
27 % expects, using the convention that "D" is the dimension of the input space.
28 % For example, calling "meanLinear" returns the string 'D'.
29 %
30 % 2) With two input arguments:
31 %
32 %    m = meanNAME(hyp, x)
33 %
34 % The function computes and returns the mean vector where hyp are the
35 % hyperparameters and x is an n by D matrix of cases, where D is the dimension
36 % of the input space. The returned mean vector is of size n by 1.
37 %
38 % 3) With three input arguments:
39 %
40 %    dm = meanNAME(hyp, x, i)
41 %
42 % The function computes and returns the n by 1 vector of partial derivatives
43 % of the mean vector w.r.t. hyp(i) i.e. hyperparameter number i.
44 %
```

```
45 % See also doc/usageMean.m.
46 %
47 ⟨gpml copyright 5a⟩
```

5.2 Implemented Mean Functions

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

Simple mean functions $m(\mathbf{x})$			
`<NAME>`	Meaning	$m(\mathbf{x}) =$	Φ
`Zero`	mean vanishes always	0	$\emptyset$
`One`	mean equals 1	1	$\emptyset$
`Const`	mean equals a constant	c	$c \in \mathbb{R}$
`Linear`	mean linearly depends on $\mathbf{x} \in \mathcal{X} \subseteq \mathbb{R}^D$	$\mathbf{a}^\top \mathbf{x}$	$\mathbf{a} \in \mathbb{R}^D$
Composite mean functions $[\mu_1(\mathbf{x}), \mu_2(\mathbf{x}), ..] \mapsto m(\mathbf{x})$			
`<NAME>`	Meaning	$m(\mathbf{x}) =$	Φ
`Scale`	scale a mean	$\alpha\mu(\mathbf{x})$	$\alpha \in \mathbb{R}$
`Sum`	add up mean functions	$\sum_j \mu_j(\mathbf{x})$	$\emptyset$
`Prod`	multiply mean functions	$\prod_j \mu_j(\mathbf{x})$	$\emptyset$
`Pow`	raise a mean to a power	$\mu(\mathbf{x})^d$	$\emptyset$
`Mask`	act on components $I \subseteq [1, 2, .., D]$ of $\mathbf{x} \in \mathcal{X} \subseteq \mathbb{R}^D$ only	$\mu(\mathbf{x}_I)$	$\emptyset$

5.3 Usage of Implemented Mean Functions

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

Syntactically, a mean function `mf` is defined by

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

```
mf := {mn} | {mn, {param, mf}} | {mn, {mf, .., mf}} // composite
```
i.e., it is either a string containing the name of a mean function, a pointer to a mean function or one of the former in combination with a cell array of mean functions and an additional list of parameters.

24   ⟨doc/usageMean.m 24⟩≡
```
1 % demonstrate usage of mean functions
2 %
3 % See also meanFunctions.m.
4 %
5 ⟨gpml copyright 5a⟩
6 clear all, close all
7 n = 5; D = 2; x = randn(n,D);            % create a random data set
8
9 % set up simple mean functions
10 m0 = {'meanZero'};   hyp0 = [];       % no hyperparameters are needed
11 m1 = {'meanOne'};    hyp1 = [];       % no hyperparameters are needed
12 mc = {@meanConst};   hypc = 2;  % also function handles are possible
13 ml = {@meanLinear};  hypl = [2;3];             % m(x) = 2*x1 + 3*x2
14
```

```matlab
15 % set up composite mean functions
16 msc = {'meanScale',{m1}};         hypsc = [3; hyp1];        % scale by 3
17 msu = {'meanSum',{m0,mc,ml}};  hypsu = [hyp0; hypc; hypl];    % sum
18 mpr = {@meanProd,{mc,ml}};       hyppr = [hypc; hypl];      % product
19 mpo = {'meanPow',{3,msu}};       hyppo = hypsu;           % third power
20 mask = [0,1,0]; %   binary mask excluding all but the 2nd component
21 mma = {'meanMask',{mask,mpo{:}}}; hypma = hyppo;
22
23 % 0) specify mean function
24 % mean = m0;  hyp = hyp0;
25 % mean = msu; hyp = hypsu;
26 % mean = mpr; hyp = hyppr;
27 mean = mpo; hyp = hyppo;
28
29 % 1) query the number of parameters
30 feval(mean{:})
31
32 % 2) evaluate the function on x
33 feval(mean{:},hyp,x)
34
35 % 3) compute the derivatives w.r.t. to hyperparameter i
36 i = 2; feval(mean{:},hyp,x,i)
```

6 Covariance Functions

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

6.1 Interface

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

26   ⟨*covFunctions.m* 26⟩≡

```
1  % covariance functions to be use by Gaussian process functions. There are two
2  % different kinds of covariance functions: simple and composite:
3  %
4  % simple covariance functions:
5  %   covConst       - covariance for constant functions
6  %   covLIN         - linear covariance function
7  %   covLINard      - linear covariance function with ARD
8  %   covLINone      - linear covariance function with bias
9  %   covMaterniso   - Matern covariance function with nu=1/2, 3/2 or 5/2
10 %   covNNone       - neural network covariance function
11 %   covNoise       - independent covariance function (i.e. white noise)
12 %   covPeriodic    - smooth periodic covariance function (1d) with unit period
13 %   covPoly        - polynomial covariance function
14 %   covPPiso       - piecewise polynomial covariance function (compact support)
15 %   covRQard       - rational quadratic covariance function with ARD
16 %   covRQiso       - isotropic rational quadratic covariance function
17 %   covSEard       - squared exponential covariance function with ARD
18 %   covSEiso       - isotropic squared exponential covariance function
19 %   covSEisoU      - as above but without latent scale
20 %
21 % composite (meta) covariance functions (see explanation at the bottom):
22 %   covScale       - scaled version of a covariance function
23 %   covProd        - products of covariance functions
24 %   covSum         - sums of covariance functions
25 %   covADD         - additive covariance function
26 %   covMask        - mask some dimensions of the data
27 %
28 % special purpose (wrapper) covariance functions
29 %   covFITC        - to be used in conjunction with infFITC for large scale
30 %                      regression problems; any covariance can be wrapped by
31 %                      covFITC such that the FITC approximation is applicable
32 %
33 % Naming convention: all covariance functions are named "cov/cov*.m". A trailing
34 % "iso" means isotropic, "ard" means Automatic Relevance Determination, and
35 % "one" means that the distance measure is parameterized by a single parameter.
36 %
37 % The covariance functions are written according to a special convention where
38 % the exact behaviour depends on the number of input and output arguments
39 % passed to the function. If you want to add new covariance functions, you
40 % should follow this convention if you want them to work with the function gp.
41 % There are four different ways of calling the covariance functions:
42 %
43 % 1) With no (or one) input argument(s):
```

```
44 %
45 %    s = covNAME
46 %
47 % The covariance function returns a string s telling how many hyperparameters it
48 % expects, using the convention that "D" is the dimension of the input space.
49 % For example, calling "covRQard" returns the string '(D+2)'.
50 %
51 % 2) With two input arguments:
52 %
53 %    K = covNAME(hyp, x) equivalent to K = covNAME(hyp, x, [])
54 %
55 % The function computes and returns the covariance matrix where hyp are
56 % the hyperparameters and x is an n by D matrix of cases, where
57 % D is the dimension of the input space. The returned covariance matrix is of
58 % size n by n.
59 %
60 % 3) With three input arguments:
61 %
62 %    Ks  = covNAME(hyp, x, xs)
63 %    kss = covNAME(hyp, xs, 'diag')
64 %
65 % The function computes test set covariances; kss is a vector of self covariances
66 % for the test cases in xs (of length ns) and Ks is an (n by ns) matrix of cross
67 % covariances between training cases x and test cases xs.
68 %
69 % 4) With four input arguments:
70 %
71 %     dKi   = covNAME(hyp, x, [], i)
72 %     dKsi  = covNAME(hyp, x, xs, i)
73 %     dkssi = covNAME(hyp, xs, 'diag', i)
74 %
75 % The function computes and returns the partial derivatives of the
76 % covariance matrices with respect to hyp(i), i.e. with
77 % respect to the hyperparameter number i.
78 %
79 % Covariance functions can be specified in two ways: either as a string
80 % containing the name of the covariance function or using a cell array. For
81 % example:
82 %
83 %   cov = 'covRQard';
84 %   cov = {'covRQard'};
85 %   cov = {@covRQard};
86 %
87 % are supported. Only the second and third form using the cell array can be used
88 % for specifying composite covariance functions, made up of several
89 % contributions. For example:
90 %
91 %        cov = {'covScale', {'covRQiso'}};
92 %        cov = {'covSum', {'covRQiso','covSEard','covNoise'}};
93 %        cov = {'covProd',{'covRQiso','covSEard','covNoise'}};
94 %        cov = {'covMask',{mask,'covSEiso'}}
95 %   q=1; cov = {'covPPiso',q};
96 %   d=3; cov = {'covPoly',d};
97 %        cov = {'covADD',{[1,2],'covSEiso'}};
98 %        cov = {@covFITC, {@covSEiso}, u};  where u are the inducing inputs
99 %
100 % specifies a covariance function which is the sum of three contributions. To
101 % find out how many hyperparameters this covariance function requires, we do:
```

```
102 %
103 %    feval(cov{:})
104 %
105 % which returns the string '3+(D+1)+1' (i.e. the 'covRQiso' contribution uses
106 % 3 parameters, the 'covSEard' uses D+1 and 'covNoise' a single parameter).
107 %
108 % See also doc/usageCov.m.
109 %
110 ⟨gpml copyright 5a⟩
```

6.2 Implemented Covariance Functions

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

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

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

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

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

```
    cf := {cv} | {cv, {param, cf}} | {cv, {cf, .., cf}} // composite
```
i.e., it is either a string containing the name of a covariance function, a pointer to a covariance function or one of the former in combination with a cell array of covariance functions and an additional list of parameters.

29   ⟨*doc/usageCov.m* 29⟩≡

```matlab
1  % demonstrate usage of covariance functions
2  %
3  % See also covFunctions.m.
4  %
5  ⟨gpml copyright 5a⟩
6  clear all, close all
7  n = 5; D = 3; x = randn(n,D); xs = randn(3,D);  % create a data set
8
9  % set up simple covariance functions
10 cn  = {'covNoise'}; sn = .1;  hypn = log(sn);  % one hyperparameter
11 cc  = {@covConst};   sf = 2;  hypc = log(sf); % function handles OK
12 cl  = {@covLIN};              hypl = []; % linear is parameter-free
13 cla = {'covLINard'}; L = rand(D,1); hypla = log(L);  % linear (ARD)
14 clo = {@covLINone}; ell = .9; hyplo = log(ell);  % linear with bias
15 cp  = {@covPoly,3}; c = 2; hypp = log([c;sf]);   % third order poly
16 cga = {@covSEard};   hypga = log([L;sf]);       % Gaussian with ARD
17 cgi = {'covSEiso'};  hypgi = log([ell;sf]);     % isotropic Gaussian
18 cgu = {'covSEisoU'}; hypgu = log(ell);   % isotropic Gauss no scale
19 cra = {'covRQard'}; al = 2; hypra = log([L;sf;al]); % ration. quad.
20 cri = {@covRQiso};            hypri = log([ell;sf;al]);   % isotropic
21 cm  = {'covMaterniso',3}; hypm = log([ell;sf]);  % Matern class q=3
22 cnn = {'covNNone'}; hypnn = log([L;sf]);             % neural network
23 cpe = {'covPeriodic'}; om = 2; hyppe = log([ell;om;sf]); % periodic
24 ccc = {'covPPiso',2}; hypcc = hypm; % compact support poly degree 2
25
26 % set up composite covariance functions
27 csc = {'covScale',{cgu}};     hypsc = [log(3); hypgu];  % scale by 9
28 csu = {'covSum',{cn,cc,cl}}; hypsu = [hypn; hypc; hypl];      % sum
29 cpr = {@covProd,{cc,ccc}};   hyppr = [hypc; hypcc];       % product
30 mask = [0,1,0]; %   binary mask excluding all but the 2nd component
31 cma = {'covMask',{mask,cgi{:}}}; hypma = hypgi;
32 % additive based on SEiso using unary and pairwise interactions
33 cad = {'covADD',{[1,2],'covSEiso'}};
34
35 % 0) specify covariance function
36 cov = cma; hyp = hypma;
37
38 % 1) query the number of parameters
39 feval(cov{:})
40
41 % 2) evaluate the function on x
42 feval(cov{:},hyp,x)
43
44 % 3) evaluate the function on x and xs to get cross-terms
45 kss = feval(cov{:},hyp,xs,'diag')
46 Ks  = feval(cov{:},hyp,x,xs)
47
48 % 4) compute the derivatives w.r.t. to hyperparameter i
49 i = 1; feval(cov{:},hyp,x,[],i)
```