```
function [max_lik, theta_opt, m_star, K_line] = ...
    fitGPR(X, y, K_type, theta0, bound_theta, X_star)
% input:
% X: Training input X, where each row x corresponds to one input case.
% y: Training output vector corresponding to each row of the input
 matrix X.
% K_type:  One of the three kernels defined in part (a)
% theta0: initial parameters of the kernel
% bound_theta = bounds on the parameters of the kernel
% x_star: test set

% output:
% max_lik: maxima of marginal likelihood
% theta_opt: optimal hyperparameters
% m_star: posterior mean
% K_post: posterior covariance

N = size(X,1); % number of rows of X

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% 1 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%

% sigma^2:

sigma2_0 = 2;
sigma2_bounds = [0, 10];

% add sigma^2 initial value to initial values
theta0(end+1) = sigma2_0;
% add sigma^2 bound to theta bounds
bound_theta(:,end+1) = sigma2_bounds;

% define inputs for fmicon function:
A = [];
b = [];
Aeq = [];
beq = [];
lb = bound_theta(1,:);
ub = bound_theta(2,:);
nonlcon = [];
options =
 optimoptions('fmincon','Display','iter','SpecifyObjectiveGradient',true);

fun = @(theta) to_minimize(theta, X, y, K_type); % fun contains f and
 g

% perform minimization
theta_opt = fmincon(fun,theta0,A,b,Aeq,beq,lb,ub,nonlcon,options);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% 2 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%
theta = theta_opt;
```

```
sigma2 = theta(end);

% find K with optimal parametrs
if strcmp(K_type,'squaredexponential')
    K = sqrdexp(X, X, theta(1), theta(2), 1);
elseif strcmp(K_type,'linearkernel')
    K = linearkernel(X, X, theta(1), theta(2), theta(3), 1);
elseif strcmp(K_type,'periodickernel')
    K = periodickernel(X, X, theta(1), theta(2), theta(3), 1);
end

% maxima marginal likelihood
max_lik = marginal_likelihood(K, y, sigma2, N);

% find K* and K** with optimal parametrs
if strcmp(K_type,'squaredexponential')
    K_star = sqrdexp(X_star, X, theta(1), theta(2), 1);
    K_star_star = sqrdexp(X_star, X_star, theta(1), theta(2), 1);
elseif strcmp(K_type,'linearkernel')
    K_star = linearkernel(X_star, X, theta(1), theta(2), theta(3), 1);
    K_star_star = linearkernel(X_star, X_star, theta(1), theta(2),
 theta(3), 1);
elseif strcmp(K_type,'periodickernel')
    K_star = periodickernel(X_star, X, theta(1), theta(2), theta(3),
 1);
    K_star_star = periodickernel(X_star, X_star, theta(1), theta(2),
 theta(3), 1);
end

Ky = K + sigma2 * ones(size(K));
L = chol(Ky); % cholesky factorization

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% 3 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%
alpha = L'\(L\y);
v = L\K_star';

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% 4 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%
m_star = K_star * alpha;
K_line = K_star_star * (v' * v);

end


function [f,g] = to_minimize(theta_f, X, y, K_type)

% sigma^2 is the last parameter
sigma2 = theta_f(end);

% define f and g, the inputs to use fmicon
len_X = size(X,1); % number of rows of X

if strcmp(K_type,'squaredexponential')
```

```
    K = sqrdexp(X, X, theta_f(1), theta_f(2), 1);
    dK = sqrdexp(X, X, theta_f(1), theta_f(2), 0);
elseif strcmp(K_type,'linearkernel')
    K = linearkernel(X, X, theta_f(1), theta_f(2), theta_f(3), 1);
    dK = linearkernel(X, X, theta_f(1), theta_f(2), theta_f(3), 0);
elseif strcmp(K_type,'periodickernel')
    K = periodickernel(X, X, theta_f(1), theta_f(2), theta_f(3), 1);
    dK = periodickernel(X, X, theta_f(1), theta_f(2), theta_f(3), 0);
end

len_k = length(K);
log_p = marginal_likelihood(K, y, sigma2, len_X);

f = -log_p; % maximize log_p == minimize f

K_sigma = K + sigma2 * ones(size(K));

%gradiet:
if strcmp(K_type,'squaredexponential')
    grad_sigma0 = derivative_f_theta(dK(1:len_k,:), y, K_sigma);
    grad_l = derivative_f_theta(dK(len_k+1:end,:), y, K_sigma);
    g_K =[grad_sigma0, grad_l];

elseif strcmp(K_type,'linearkernel')
    grad_sigma0 = derivative_f_theta(dK(1:len_k,:), y, K_sigma);
    grad_sigma1 = derivative_f_theta(dK(len_k+1:2*len_k,:), y,
 K_sigma);
    grad_p = derivative_f_theta(dK(2*len_k+1:end,:), y, K_sigma);
    g_K =[grad_sigma0, grad_sigma1, grad_p];

elseif strcmp(K_type,'periodickernel')
    grad_sigma0 = derivative_f_theta(dK(1:len_k,:), y, K_sigma);
    grad_l = derivative_f_theta(dK(len_k+1:2*len_k,:), y, K_sigma);
    grad_p = derivative_f_theta(dK(2*len_k+1:end,:), y, K_sigma);
    g_K =[grad_sigma0, grad_l, grad_p];
end

grad_sigma2 = derivative_f_sigma2 (X, y, K, sigma2);
g = [g_K, grad_sigma2];

end

function [result] = derivative_f_sigma2 (X, y, K, sigma0)
% sigma2 = sigma^2. this function find the d f / d sigma^2, where f =
 -log p
N = size(X,1); % number of rows of X

% ATTENTION: this is the first try using diff. This is too long for
 big matrices, so we calulate
% the derivative by hand (see notes)
%
% % def syms to use the function diff
% syms sigma2_
%
```

```
% f = 0.5 * (y'* inv(K + sigma2_.*ones(size(K)))) * y ...
%     + 0.5 * log(det(K + sigma2_*ones(size(K)))) + 0.5 * N *
 log(2*pi);
% df_dsigma2 = diff(f,sigma2_);
%
% sigma2_= sigma0;
% % in this way df_dsigma2 is calcluated in sigma0
% result = double(subs(df_dsigma2));

% derivative by hand (see notes)
K_inv = inv(K + sigma0.*ones(size(K)));
der_inv_k = - K_inv * ones(size(K)) * K_inv;
result = 0.5 * (y' * der_inv_k * y  + trace(inv(K +
 sigma0.*ones(size(K)))));

end

function[df_dtheta] =  derivative_f_theta(dK, y, K_sigma)
% gradient function

df_dtheta = + 0.5 * y' * inv(K_sigma) * dK * (K_sigma\y)...
      + 0.5 * trace(K_sigma\dK);
% note that we use + and not - becuase - would be log p and f = - log
 p
end

function [log_p] = marginal_likelihood(K, y, sigma2, len_X)

K_sigma = K + sigma2 * ones(size(K));
if det(K_sigma) == 0 %otherwise is log(0) is -Inf
    log_p = -0.5 * y' * (K_sigma\y) - 0.5 * len_X * log(2*pi);
else
    log_p = -0.5 * y' * (K_sigma\y) - 0.5 * log(det(K_sigma)) - 0.5 *
 len_X * log(2*pi);
end

end
```

```
Not enough input arguments.

Error in fitGPR (line 17)
N = size(X,1); % number of rows of X
```

Published with MATLAB® R2020a