```
% example file to illustrate the use of multi-task Gaussian Process models
% with a convoluted squared exponential covariance function
%
%
% by Robert Duerichen
% 18/11/2013
clear

path_gpml = 'E:\OneDrive - hnu.edu.cn\tools\matlabcourse\GPML_matlab\gpml-matlab-v4.
2-2018-06-11';                    % please insert here path of GPML Toolbox

% add folders of MTGP and GPML Toolbox
if ~isunix  % windows system
    addpath(genpath('..\..\'));
    addpath(genpath(path_gpml));
else        % linux system
    addpath(genpath('../../'));
    addpath(genpath(path_gpml));
end

% please uncomment one of the following cases to demonstrate the use of
% MTGPs with convoluted kernels
MTGP_case = 1;
% case1: 3 signals assuming that they are uncorrelated but with further optimization
%           and SE cov. func with isotropic length-scale hyperparameter for
%           all tasks
% case2: 3 signals assuming that they are uncorrelated but with further optimization
%           and SE cov. func with non-isotropic length-scale hyperparameter
switch MTGP_case
    case 1
        opt.cov_func = 1;               % select cov. function
        opt.se_hyp = 1;                 % initial value for lenght scale
hyperparameter
    case 2
        opt.cov_func = 2;               % select cov. function
        opt.se_hyp = [1 0.5 0.05];      % initial value for lenght scale
hyperparameter
                                        % 3 squared exponential hyperpara. for 3
tasks
%         opt.se_hyp = [1 1 1];       % initial value for lenght scale hyperparameter
```

```
    otherwise
        error('Unknown example case. MTGP_case has to be between 1 and 2.');
end

% generate example data
t = (0:0.01:2)'; % 201×1
y1 =  sin(2*pi*0.1*t)+1/100*randn([1,length(t)])';
y2 =  sin(2*pi*0.1*t+0.2*pi)-0.10*sin(2*pi*1*t+0.1*pi)+1/50*randn([1,length(t)])';
y3 =  sin(2*pi*0.1*t+0.2*pi)-0.10*sin(2*pi*3.1*t+0.1*pi)+1/20*randn([1,length(t)])';

y = [y1 y2 y3]; % 201×3

%% initial parameter
opt.training_data{1} = 1:20:200;        % training data - task 1  1×10
opt.training_data{2} = 1:10:200;        % training data - task 2  1×20
opt.training_data{3} = 1:8:200;         % training data - task 3  1×25
opt.init_num_opt = 200;                 % number of optimization steps
opt.show = 1;                           % show result plot
opt.start = 1;                          % start index for prediction
opt.end = 200;                          % end index for prediction
opt.random = 0;                         % if 1 - hyp for correlation and SE will set
randomly
opt.num_rep = 1;                        % number of trails for selecting hyp

% init values for hyperparameter (if opt.random ~= 1)
opt.cc_hyp = [1/0 1/0 0 1];             % assumes that all signals are independent
opt.noise_lik = 0.1;                    % inital noise hyperparameter


num_dim = size(opt.training_data,2); 3
num_cc_hyp = sum(1:size(opt.training_data,2)); sum(1:3) = 6
num_se_hyp = length(opt.se_hyp); % case1: num_se_hyp = 1; case2: num_se_hyp = 3

%% define range for hyperparameters if random = 1 is selected
random_bounds.cc = [-5,5];
random_bounds.SE = [0,2];

%% generate training and test data with labels
% also substract mean
10×2 x_train = [t(opt.training_data{1},1), ...
                % 1×10
```

```
    ones(length(opt.training_data{1}),1)];

y_train_mean(1) = mean(y(opt.training_data{1},1));
y_train = y(opt.training_data{1},1)-y_train_mean(1); % y: 201×3, y_train: 10×1

               t(1:200)
x_test = [t(opt.start:opt.end) ones(opt.end-opt.start+1,1)];
10×1 y_test = y(opt.start:opt.end,1)-y_train_mean(1);

                   3
for cnt_dim = 2:num_dim  num_dim = size(opt.training_data, 2); % the number of the training datasets
        x_train = [x_train(:,1) x_train(:,2);...
             t(opt.training_data{cnt_dim},1) ...
             ones(length(opt.training_data{cnt_dim}),1)*cnt_dim];

                     2                                  2        2
        y_train_mean(cnt_dim) = mean(y(opt.training_data{cnt_dim},cnt_dim));
        y_train = [y_train; y(opt.training_data{cnt_dim},cnt_dim)-...
            y_train_mean(cnt_dim)];
                                                                          x_train: 55×3
                                                                          y_train: 55×1
        x_test = [x_test(:,1) x_test(:,2);...                             x_test: 600×3
            t(opt.start:opt.end) ones(opt.end-opt.start+1,1)*cnt_dim];    y_test: 600×1
        y_test = [y_test; y(opt.start:opt.end,cnt_dim)-y_train_mean(cnt_dim)];
end

%% init covariance functions and hyperparameters
switch opt.cov_func
        case 1      % SE with one length scale parameter for all tasks
            disp('Covariance Function: K = CC(l) x (SE_U(t))');
            covfunc = {'MTGP_covProd',{'MTGP_covCC_chol_nD','MTGP_covSEisoU'}};
            hyp.cov(1:num_cc_hyp) = opt.cc_hyp(1:num_cc_hyp);
            hyp.cov(num_cc_hyp+1) = log(opt.se_hyp);

        case 2      % SE with different length scale parameter for all tasks
            disp('Covariance Function: K = CC(l) x (SE_conU(t))');
            covfunc = {'MTGP_covProd',{'MTGP_covCC_chol_nD','MTGP_covSEconU'}};
            hyp.cov(1:num_cc_hyp) = opt.cc_hyp;
            hyp.cov(num_cc_hyp+1:num_cc_hyp+num_se_hyp) = log(opt.se_hyp);
end

% likelihood function
likfunc = @likGauss;
hyp.lik = log(opt.noise_lik);
```

```
% optimize hyperparameters
for cnt_rep = 1:opt.num_rep
    disp(['Number of rep: ',num2str(cnt_rep)]);
    % if opt. random == 1 - hyper will be choosen randomly
    if opt.random
        % random hyp for first correlation term
        hyp.cov(1:6)  = rand(6,1)*(random_bounds.cc(2)-random_bounds.cc(1))+...
            random_bounds.cc(1);
        hyp.cov(numel(opt.cc_hyp)+1:numel(opt.cc_hyp)+numel(opt.se_hyp))  = log(rand(numel(opt.se_hyp),1)*(random_bounds.SE(2)-random_bounds.SE(1))+...
            random_bounds.SE(1));
    end

    % optimize hyperparameter
    [results.hyp{cnt_rep}] = minimize(hyp, @MTGP, -opt.init_num_opt, @MTGP_infExact, [], covfunc, likfunc, x_train,y_train);

    % training
    results.nlml(cnt_rep) = MTGP(results.hyp{cnt_rep}, @MTGP_infExact, [], covfunc, likfunc, x_train, y_train);
end
% find best  nlml
[results.nlml, best_hyp] =min(results.nlml);
results.hyp = results.hyp{best_hyp};

%% perform prediction
600×1 [results.m, results.s2, fmu, fs2, results.p] = MTGP(results.hyp, @MTGP_infExact, [], covfunc, likfunc, x_train, y_train, x_test, y_test);

% reshape of results
300×2 results.m = reshape(results.m,[opt.end-opt.start+1 num_dim]);
results.s2 = reshape(results.s2,[opt.end-opt.start+1 num_dim]);
results.p = exp(reshape(results.p,[opt.end-opt.start+1 num_dim]));

%% compute RMSE for training and test data for each dimension
for cnt_dim = 1:num_dim 3
    results.m(:,cnt_dim) = results.m(:,cnt_dim) + y_train_mean(cnt_dim);

    index_test_data = [opt.start:opt.end];
```

```
    index_test_data(ismember(index_test_data,opt.training_data{cnt_dim})) = [];

    results.rmse_test(cnt_dim) = rms(results.m(index_test_data-opt.start+1,cnt_dim)-...
        y(index_test_data,cnt_dim));

    results.rmse_train(cnt_dim) = rms(results.m(opt.training_data{cnt_dim}-opt.start+1,cnt_dim)-...
        y(opt.training_data{cnt_dim},cnt_dim));
        y(opt.training_data{cnt_dim}+opt.start-1, cnt_dim)

end

% compute resulting K_f matrix
vec_dim = 1:num_dim;
L = zeros(num_dim,num_dim);
for cnt_dim = 1:num_dim
    L(cnt_dim,1:vec_dim(cnt_dim)) = [results.hyp.cov(sum(vec_dim(1:cnt_dim-1))+1:sum(vec_dim(1:cnt_dim-1))+vec_dim(cnt_dim))];
end

results.K_f =  L*L';

% print results on console:
disp(['Estimated cross correlation covariance K_f:']);
results.K_f

if opt.cov_func == 3
    for cnt_dim = 1:num_dim
        disp(['Noise level for signal ',num2str(cnt_dim),': ',num2str(exp(2*results.hyp.cov(end-num_dim+cnt_dim)))]);
    end
end

%% plot basic results
if opt.show == 1
    h=figure('Position',[1 1 1400 800]);
    for cnt_dim = 1:num_dim
        % plot prediction results
        subplot(2,num_dim,cnt_dim);
```

```
        min_val = min(results.m(:,cnt_dim))-abs(min(results.s2(:,cnt_dim)));
        max_val = max(results.m(:,cnt_dim))+abs(max(results.s2(:,cnt_dim)));

        hTlines = plot([t(opt.training_data{cnt_dim}) t(opt.training_data↙
{cnt_dim})]',...
            [ min_val max_val]','Color',[0.85 0.85 0.5]);
        hTGroup = hggroup;
        set(hTlines,'Parent',hTGroup);
        set(get(get(hTGroup,'Annotation'),'LegendInformation'),...
            'IconDisplayStyle','on');
        hold on;
        f = [results.m(:,cnt_dim)+2*sqrt(results.s2(:,cnt_dim)); flip(results.m(:,↙
cnt_dim)-2*sqrt(results.s2(:,cnt_dim)),1)];
        fill([t(opt.start:opt.end); flip(t(opt.start:opt.end),1)], f, [0.7 0.7↙
0.7],'EdgeColor','none')

        % plot mean org signal
        plot(t(opt.start:opt.end),y(opt.start:opt.end,cnt_dim),'b');

        % plot mean predicted signal
        plot(t(opt.start:opt.end),results.m(1:opt.end-opt.start+1,cnt_dim),'r');

        if cnt_dim == 1
            legend('training data','95% conf. int.','org. values','pred.↙
values','Orientation','horizontal','Location',[0.45 0.49 0.15 0.04]);
        end
        axis tight

        title(['S',num2str(cnt_dim),': RMSE_{train}: ',num2str(results.rmse_train↙
(cnt_dim)),...
            ' - RMSE_{test}: ',num2str(results.rmse_test(cnt_dim))]);
        xlabel('time [s]');
        ylabel('amplitude y [mm]');


        subplot(2,num_dim,num_dim+cnt_dim);
        min_val = min(results.p(:,cnt_dim));
        max_val = max(results.p(:,cnt_dim));
        plot([t(opt.training_data{cnt_dim}) t(opt.training_data{cnt_dim})]',...
```

```
            [ min_val max_val]','Color',[0.85 0.85 0.5]);
        hold on
        plot(t(opt.start:opt.end),results.p(1:opt.end-opt.start+1,cnt_dim),'b');

        title('probability p');
        axis tight
        xlabel('time [s]');
        ylabel('p');
    end
end
```