```
% example file to illustrate the use of multi-task Gaussian Process models
%
% by Robert Duerichen
% 18/11/2013

~~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~~

clear
close all; clc

% load data
load('test_data.mat');
```

t : 200 × 1
y: 200 × 4

uncorrelated: opt.cc_hyp = [1| 0 1| 0 0 1];

```
% please select one of the following cases to demonstrate the use of MTGPs
MTGP_case = 1;
% case1: 2 signals assuming that they are uncorrelated, no optimization is
%            done and k_t = SE cov. func
% case2: 2 signals assuming that they are uncorrelated but with further optimization
%            and k_t = SE cov. func
% case3: 2 signals assuming that they are uncorrelated but with further optimization
%            and k_t = quasi-periodic cov. func
% case4: 3 signals, assuming that all are uncorrelated, with optimization and
%            with different sample frequencies - k_t = quasi-periodic cov. func
% case5: 3 signals, assuming that all are uncorrelated, with optimization and
%            with different sample frequencies - noise is added two signal 2
%            - quasi-periodic cov. func with individual noise term for each
%            signal

switch MTGP_case
    case 1
```

```
t(1) = 6.0815
t(30) = 17.2353 opt.init_num_opt = 0;            % number of optimization steps
t(60) = 28.7738 opt.training_data{1} = 1:60;  % index of know training points of signal 1 1×60
t(90) = 40.3122 opt.training_data{2} = 30:90; % index of know training points of signal 2 1×61
                opt.cov_func = 1;                % select cov. function
    case 2
        opt.init_num_opt = 200;          % number of optimization steps
        opt.training_data{1} = [1:60];  % index of know training points of signal 1
        opt.training_data{2} = [30:90]; % index of know training points of signal 2
        opt.cov_func = 1;                % select cov. function
    case 3
        opt.init_num_opt = 200;          % number of optimization steps
        opt.training_data{1} = [1:60];  % index of know training points of signal 1
        opt.training_data{2} = [30:90]; % index of know training points of signal 2
        opt.cov_func = 2;                % select cov. function
    case 4
        opt.init_num_opt = 200;          % number of optimization steps
        opt.training_data{1} = [1:60];  % index of know training points of signal 1
        opt.training_data{2} = [30:90];  % index of know training points of signal 2
        opt.training_data{3} = [60:5:150];% index of know training points of signal↙
3
        opt.cov_func = 2;                % select cov. function
    case 5
        seed=0; rng(seed); %rng('shuffle')
        opt.init_num_opt = 200;          % number of optimization steps
        opt.training_data{1} = [1:60];  % index of know training points of signal 1
        opt.training_data{2} = [30:90];  % index of know training points of signal 2
        opt.training_data{3} = [60:5:150];% index of know training points of signal↙
3
        opt.cov_func = 3;                % select cov. function
        y(:,2) = y(:,2)+rand(length(y),1); % add noise for illustration purpose on↙
signal 2
    otherwise
        error('Unknown example case. MTGP_case has to be between 1 and 5.');
end

%% initial parameter
opt.show = 1;                  % show result plot
opt.start = 1; 3 t(1) = 6.0815   % start index for prediction
opt.end = 150;  t(150) = 63.3892 % 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.per_hyp = 6;
opt.se_hyp = 1;
opt.cc_hyp = [1 0 1 0 0 1];     % assumes that all signals are independent
opt.noise_lik = 0.1;
opt.noise_hyp = 0.001;
```

MTGP_covCC_chol_nD

1

0 1

0 0 1

opt.training_data 1×2 cell

opt.training_data{1} = 1:60

opt.training_data{2} = 30:90

```
num_dim = size(opt.training_data,2);
num_cc_hyp = sum(1:size(opt.training_data,2));
random_bounds.cc = [-5,5];
random_bounds.SE = [0,10];
```

2

sum(1:2)=3

% the relationship between y_test(450×1) and y(200×4)

% case4

>>sum(y_test(1:150)+y_train_mean(1)-y(1:150,1))

ans =

-3.2752e-15

>>sum(y_test(151:300)+y_train_mean(2)-y(1:150,2))

ans =

5.3776e-17

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

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

1:60

60

1:60

1:60

1:150 3:150 #

1:150 3:150

第1组训练样本

串联所有组的训练样本

```
for cnt_dim = 2:num_dim
    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];
    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_test = [x_test(:,1) x_test(:,2);...
        t(opt.start:opt.end) ones(opt.end-opt.start+1,1)*cnt_dim];
    y_test = [y_test; y(opt.start:opt.end,cnt_dim)-y_train_mean(cnt_dim)];
end
```

2

% 串联来自不同组的训练集的x, x_train - 121×2

30:90 2

61

y_train_mean(2)

30:90 2

y: 200×4

% 串联来自不同组的训练集的y, y_train - 121×1

% 串联不同组的测试集的x, x_test - 300×2

t(1:150) t(3:150)

% 串联不同组的测试集的y, y_test - 300×1

```
%% init covariance functions and hyperparameters
switch opt.cov_func      feval(covfunc{:})
        case 1
            disp('Covariance Function: K = CC(l) x (SE_U(t))');
            covfunc = {'MTGP_covProd',{'MTGP_covCC_chol_nD','MTGP_covSEisoU'}};
                                  3                             3
            hyp.cov(1:num_cc_hyp) = opt.cc_hyp(1:num_cc_hyp);
            hyp.cov(num_cc_hyp+1) = log(sqrt(opt.se_hyp));

        case 2
            disp('Covariance Function: K = CC(l) x (Per_UU(t)*SE_U(t))');
            covfunc = {'MTGP_covProd',↙
{'MTGP_covCC_chol_nD','MTGP_covPeriodicisoUU','MTGP_covSEisoU'}};
            hyp.cov(1:num_cc_hyp) = opt.cc_hyp(1:num_cc_hyp);
            hyp.cov(num_cc_hyp+1) = log(opt.per_hyp);
            hyp.cov(num_cc_hyp+2) = log(opt.se_hyp);
        case 3
            disp('Covariance Function: K = CC(l) x (Per_UU(t)*SE_U(t)) + Noise(t)');
            covfunc_per = {'MTGP_covProd',↙
{'MTGP_covCC_chol_nD','MTGP_covPeriodicisoUU','MTGP_covSEisoU'}};
            covfunc = {'MTGP_covSum',{covfunc_per,'MTGP_covNoise'}};
            hyp.cov(1:num_cc_hyp) = opt.cc_hyp(1:num_cc_hyp);
            hyp.cov(num_cc_hyp+1) = log(opt.per_hyp);
            hyp.cov(num_cc_hyp+2) = log(opt.se_hyp);
            hyp.cov(num_cc_hyp+3:num_cc_hyp+2+num_dim) = log(sqrt(opt.noise_hyp));
end

% likelihood function
likfunc = @likGauss;
hyp.lik = log(opt.noise_lik); log(0.1)

% 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:num_cc_hyp) = rand(num_cc_hyp,1)*(random_bounds.cc(2)-↙
random_bounds.cc(1))+...
```

```
            random_bounds.cc(1);
        if opt.cov_func == 1
            hyp.cov(num_cc_hyp+1) = rand(1)*(random_bounds.SE(2)-random_bounds.SE↙
(1))+...
                random_bounds.SE(1);
        else
            hyp.cov(num_cc_hyp+2) = rand(1)*(random_bounds.SE(2)-random_bounds.SE↙
(1))+...
                random_bounds.SE(1);
        end
    end

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

results.hyp: 1×cnt_rep cell
results.nlml: 1×cnt_rep vector

```
    % 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
```

300×1
296×1

```
[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
```

In general, shape: (opt.end-opt.start+1) × num_dim

opt.start=1, opt.end=150

```
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]));
```

150×2 (results.m)
150×2 (results.s2)
150×2
148×2

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

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

num_dim=size(opt.training_data,2) % the number of the training datasets

2

1:150 3:150

3:150

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

opt.training_data{1} = 1:60 index_test_data = 61:150

opt.training_data{2} = 30:90 index_test_data = [1:29 91:150] [3:29 91:150]

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

$rms(x) = \sqrt{(1/N*\Sigma|x_i|^2)}$, i=1,2,...,N 3

```
    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));
```

rms(results.m(opt.training_data{cnt_dim}, cnt_dim),...

y(opt.training_data{cnt_dim}+opt.start-1,cnt_dim));

opt.training_data{1} = 1:60;

opt.training_data{2} = 30:90;

```
end
```

rms: 1:148

y: 3:150

```
% compute resulting K_f matrix
vec_dim = [1:num_dim];
```

2

num_dim = size(opt.training_data, 2) % the number of the training datasets

```
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
```

2 1 2 1 2 1

```
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
```

2

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

num_dim = size(opt.training_data, 2) % the number of the training datasets

```
        % 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)); flipdim(results.m↙
(:,cnt_dim)-2*sqrt(results.s2(:,cnt_dim)),1)]; % 95% confidence interval
        fill([t(opt.start:opt.end); flipdim(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));

        % 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));

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

    saveas(h,['demoMTGP_case',num2str(MTGP_case),'.fig']);


end
```