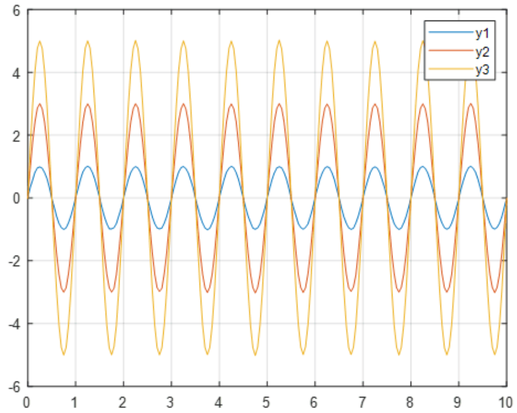
```
% example file to illustrate the use of multi-task Gaussian Process models
% to analyse the correlation between three signals
%
% within this example three sinusodial tasks are given, there:
%   task1:  has no phase shift - is fixed
%   task2:  is phase shifted by 1/4*phase_shift
%   task3:  is phase shifted by 1*phase_shift
%
%
% by Robert Duerichen
% 31/01/2014
```

1/2*phase_shift
2*phase_shift



```
close all; 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

phase_shift = 0:0.05:1;         % define phase shift vector % 1×21

scale1 = 1;                     % y scaline of signal 1
scale2 = 3;                     % y scaline of signal 2
scale3 = 5;                     % y scaline of signal 3
t = (0:0.05:10)';               % define time vector % 201×1

%% options for MTGP
opt.init_num_opt = 500;         % number of optimization steps
opt.training_data{1} = 1:50;    % index of know training points of signal 1
opt.training_data{2} = 1:50;    % index of know training points of signal 2 % 1×50
opt.training_data{3} = 1:50;    % index of know training points of signal 2
opt.show = ~~0~~;1              % show result plot
opt.start = 1;                  % start index for prediction
```

>>t

0
0.05
0.1
0.15
0.2
0.25
0.3
0.35
0.4
0.45
0.5
0.55
0.6
0.65
0.7
0.75
0.8
0.85
0.9
0.95
1
1.05
…

MTGP_covCC_chol_nD

```
1
0 1
0 0 1
```

```
opt.end = 50;                   % end index for prediction
opt.random = 1;                 % if 1 - hyp for correlation and SE will set↙
randomly
opt.num_rep = 30;               % number of trails for selecting hyp

% init values for hyperparameter, only relevant if opt.random ~= 1
opt.se_hyp = 0.1;               % init hyp for SE
opt.cc_hyp = [1/0 1/0 0 1];     % init hyp for correlation
opt.noise_lik = 0.01;           % init hyp for lik   assumes that all signals are independent

                                                3
num_cc_hyp = sum(1:size(opt.training_data,2));        % define number of correlation↙
hyperparameters  sum(1:3) = 6
random_bounds.cc = [-1,1];                             % define bounds for random↙
estimation of correlation hyp
random_bounds.SE = [0,0.1];                            % define bounds for random↙
estimation of SE hyp
random_bounds.noise = [0,0.01];                        % define bounds for random↙
estimation of lik hyp

seed=0; rng(seed); %rng('shuffle')
%% loop over data
for count = 1:length(phase_shift) 21
    shift  = phase_shift(count);
    % generate phase shifted data
    y1 = scale1*sin(2*pi*t)+randn([length(t),1])/(100);
           1
    y2 = scale2*sin(2*pi*t+0.5*shift*2*pi)+randn([length(t),1])/(100);
           3
    y3 = scale3*sin(2*pi*t+2*shift*2*pi)+randn([length(t),1])/(100);
           5

    y = [y1,y2,y3];
    num_dim = size(y,2); 3      % define number of tasks

    %% generate training and test data with labels
    % also substract mean
                         1:50
    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)];
```

```
>>phase_shift'

     0
0.0500
0.1000
0.1500
0.2000
0.2500
0.3000
0.3500
0.4000
0.4500
0.5000
0.5500
0.6000
0.6500
0.7000
0.7500
0.8000
0.8500
0.9000
0.9500
1.0000
```

```
    y_test = y(opt.start:opt.end,1)-y_train_mean(1);

    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

    %% init covariance functions and hyperparameters
    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);

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

    % optimize hyperparameters
    for cnt_rep = 1:opt.num_rep 30
        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))+...                      6×1
                random_bounds.cc(1);

            hyp.cov(num_cc_hyp+1) = rand(1)*(random_bounds.SE(2)-random_bounds.SE↙
(1))+...
                random_bounds.SE(1);
```

```
%              hyp.lik(1) = rand(1)*(random_bounds.noise(2)-random_bounds.noise(1))↙
+...
%                  random_bounds.noise(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                                            @MTGP_infExact
150×1 [results.m, results.s2, fmu, fs2, results.p] = MTGP(results.hyp, ~~@infExact~~, [],↙
covfunc, likfunc, x_train, y_train, x_test, y_test);

    % reshape of results
50×3  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);
                    1
        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));

    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';

    % MTGP correlation coefficients - not normalized
    MTGP_cc(count,1)  = results.K_f(2,1);
    MTGP_cc(count,2)  = results.K_f(3,1);
    MTGP_cc(count,3)  = results.K_f(3,2);
    est_hyp(count,:)  = results.hyp.cov(1:6);

    % normalization of K_f matrix
    [a, Kc_n]= normalize_Kc(est_hyp(count,:),num_dim);
    % print results on console:
    disp('Estimated cross correlation covariance Kc_n:');
    Kc_n

    % MTGP correlation coefficients - normalized
    MTGP_cc_n(count,1)  = Kc_n(2,1);
    MTGP_cc_n(count,2)  = Kc_n(3,1);
    MTGP_cc_n(count,3)  = Kc_n(3,2);

    % Pearsons correlation coefficient of the output function
    a = corrcoef(results.m(:,1),results.m(:,2));
    Pear_cc_output(count,1)  = a(2);
    a = corrcoef(results.m(:,1),results.m(:,3));
    Pear_cc_output(count,2)  = a(2);
    a = corrcoef(results.m(:,2),results.m(:,3));
    Pear_cc_output(count,3)  = a(2);
```

```
        % Pearsons correlation coefficient of the training data
        a = corrcoef(y1,y2);
        Pear_cc_input(count,1) = a(2);
        a = corrcoef(y1,y3);
        Pear_cc_input(count,2) = a(2);
        a = corrcoef(y2,y3);
        Pear_cc_input(count,3) = a(2);

        MTGP_results{count} = results;

        clear results
    end

    %% plot results
if opt.show == 1
    figure
    str_title = {'correlation y1-y2','correlation y1-y3','correlation y2-y3'};
    for cnt= 1:3
        subplot(3,1,cnt);

        plot(2*phase_shift*360,Pear_cc_input(:,cnt));
        hold on
        plot(2*phase_shift*360,Pear_cc_output(:,cnt),'rd:');
        plot(2*phase_shift*360,MTGP_cc_n(:,cnt),'*g--');
        plot(2*phase_shift*360,MTGP_cc(:,cnt),'om--');

        ylabel('correlation');
        xlabel('phase shift');
        legend('Pearsons CC_{input}','Pearsons CC_{output}','MTGP_{normalized}','MTGP_↙
    {not-normalized}');
        title(str_title{cnt})
    end
end
```