# Exercise 3-2, a), b), c): Gaussian Process Regression

## Table of Contents


..... 1
2a) ..... 1
2b) ..... 4
2c) ..... 5

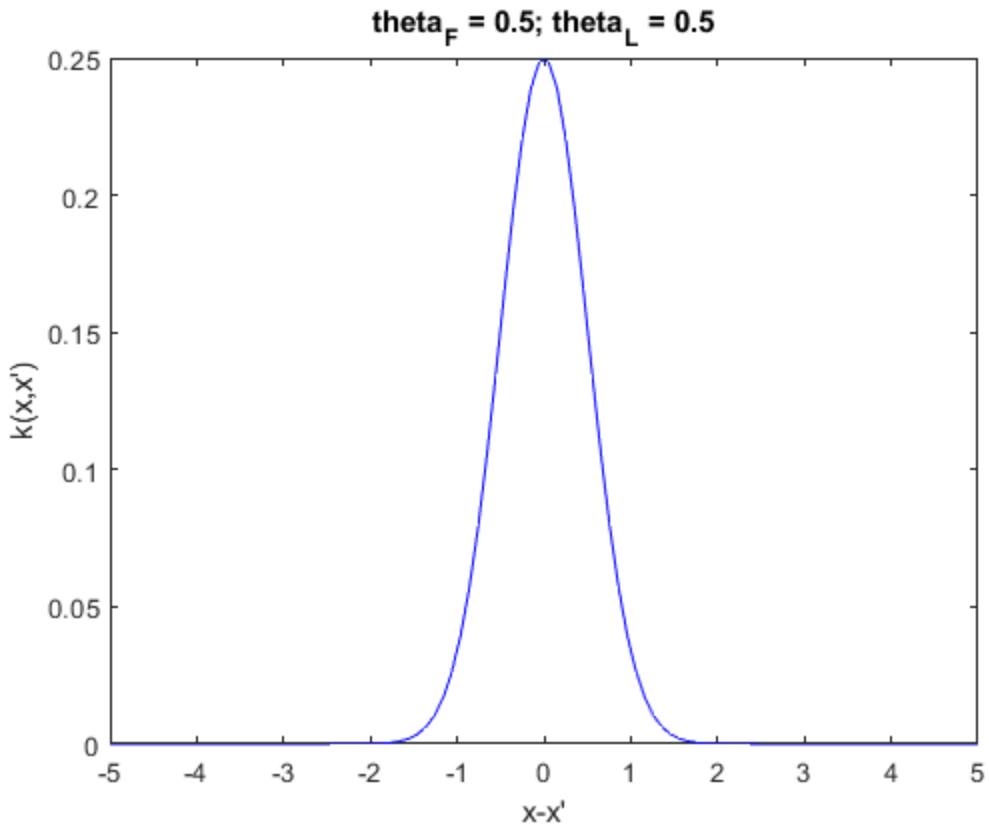


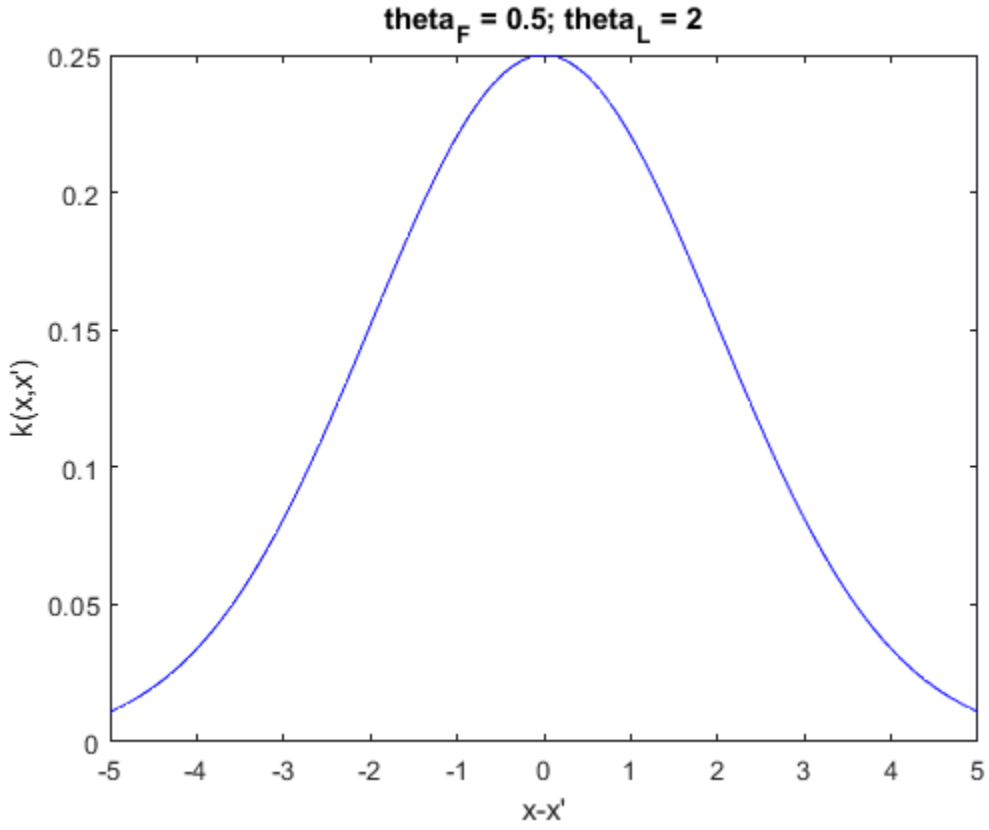
Felix Wittich, 28.06.2022

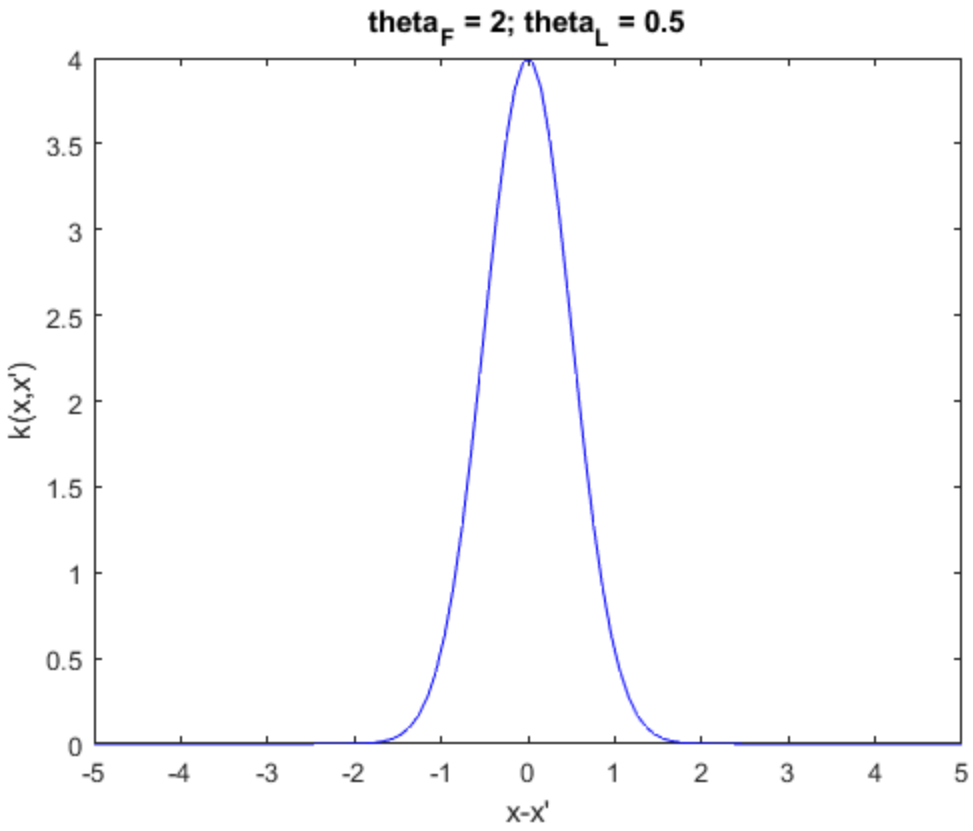

```
clear; close all; clc
```

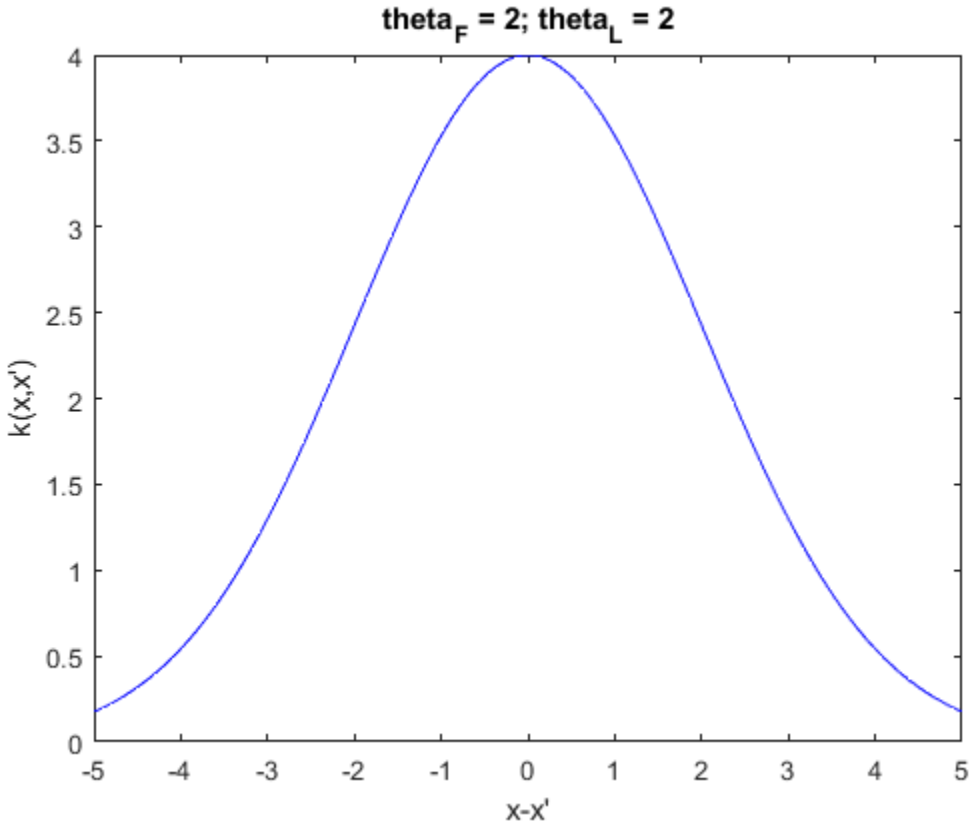
# 2a)

```
% Plot the Kernel Function for fixed x1 = 0 and vary x2 in {-5,5} for
% varying theta_f and theta_l value

x1 = 0;
x2 = linspace(-5,5,100)';
k = zeros(100,1);

% Use a (nested) for-loop to plot the kernel for different combinations of
% the kernel parameters
for j = [0.5  2]
    for m = [0.5  2]
        for i = 1:length(x2)
            k(i) = SqExpKernel(x1,x2(i),j,m);
        end
        figure
        plot(x2,k,'b')
        title("theta_F = " + j +"; theta_L = " + m)
        xlabel("x-x'")
        ylabel("k(x,x')")
    end
end
```


theta_F = 0.5; theta_L = 0.5
k(x,x')
x-x'
0.25
0.2
0.15
0.1
0.05
0
-5
-4
-3
-2
-1
0
1
2
3
4
5


theta_F = 0.5; theta_L = 2
k(x,x')
x-x'
0.25
0.2
0.15
0.1
0.05
0
-5
-4
-3
-2
-1
0
1
2
3
4
5

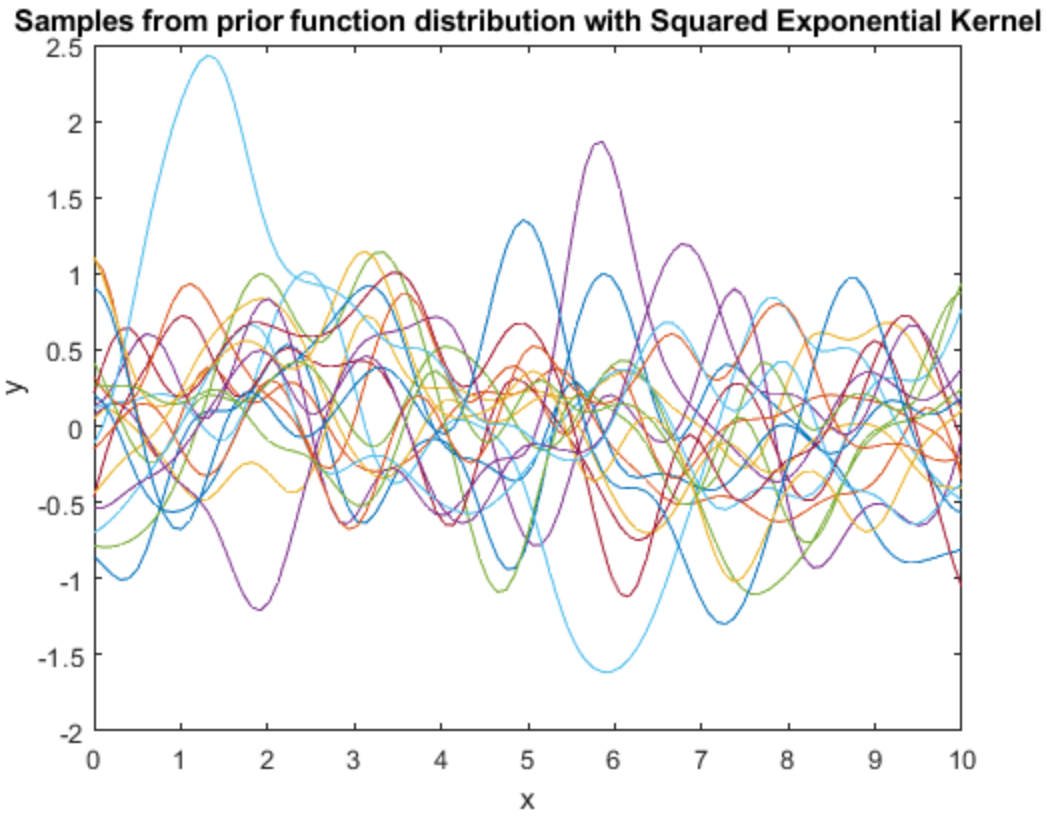

theta_F = 2; theta_L = 0.5
k(x,x')
x-x'
4
3.5
3
2.5
2
1.5
1
0.5
0
-5
-4
-3
-2
-1
0
1
2
3
4
5


theta_F = 2; theta_L = 2
k(x,x')
x-x'
4
3.5
3
2.5
2
1.5
1
0.5
0
-5
-4
-3
-2
-1
0
1
2
3
4
5

# 2b)

Now we want to sample from the prior distribution over functions of the Gaussian process: The multivariate Gaussian distribution is defined by its mean and covariance. The mean is assumed to be constant 0 and the covariance matrix is defined by the kernel function.

```
% Construct the covariance matrix with a nested for loop.
N = 100;
X = linspace(0,10,N);
samples = 20;
mu = zeros(N,1);
for i = 1:N
    for j = 1:N
        cov(i,j) = SqExpKernel(X(i),X(j),0.5,0.5);
    end
end

% Use the mvnrnd() function to sample from the multivariate Gaussian
% distribution
Z = mvnrnd(mu,cov,samples);

% Plot the sampled functions
figure
plot(X,Z)
title('Samples from prior function distribution with Squared Exponential
 Kernel')
xlabel('x')
ylabel('y')
```

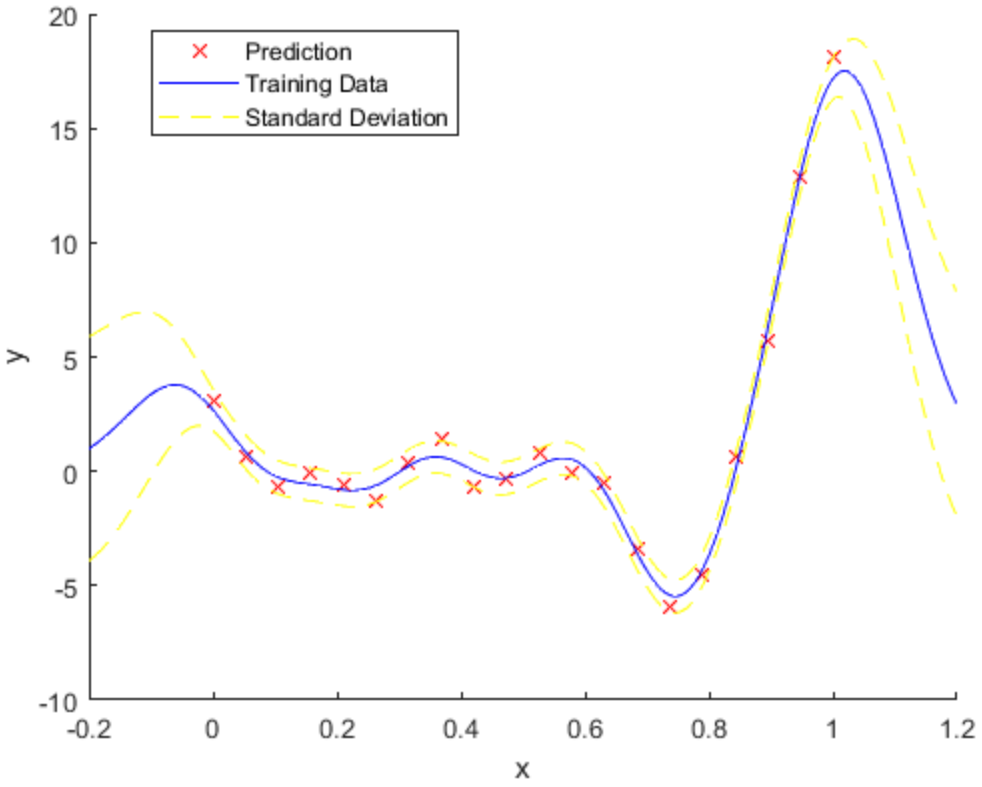



# 2c)

Conditional distribution: The joint distribution of our training data y and the output we want to predict f_* is, as we know, also multivariate Gaussian distributed with the covariance matrix C. From the joint distribution we can derive the conditional distribution to make new predictions f_* given query inputs x_*. For this exercise use the function 'GenerateNonlinData' to generate training data. We assume additive Gaussian noise for the output.

```
% Generate non linear data
[xTrain,yTrain] = GenerateNonlinData(20,1);
noiseP = 1;
thetaF = 5;
thetaL = 0.1;
N = 200;

xPred = linspace(-0.2,1.2,N);

% Make predictions using your implementation of the gprPred() function
for i = 1:N
    [yPred(i),varPred(i)] =
 gprPred(xTrain,yTrain,xPred(i),thetaF,thetaL,noiseP);
end

std = sqrt(varPred);

figure
hold on
```

```
plot(xTrain,yTrain,'rx')
plot(xPred,yPred,'b')
plot(xPred,yPred-std,'y--')
plot(xPred,yPred+std,'y--')
legend('Prediction','Training Data','Standard Deviation','location','best')
xlabel('x')
ylabel('y')

% Function to make predictions using GPR with the squared exponential kernel
function [mu_pred,sig_pred] = gprPred(x,y,xNew,thetaF,thetaL,noiseP)
    N = length(x);
    covn = zeros(N);

    % Construct a (nested) for-loop to construct the covariance matrix K_y
    for i = 1:N
        for j = 1:N
            covn(i,j) = SqExpKernel(x(i),x(j),thetaF,thetaL);
        end
    end

    % Add noise to the covariance matrix K_y
    covn = covn + noiseP*eye(N);

    % Create k_* with the new observation x_*
    kNew = zeros(N,1);
    for i = 1:N
        kNew(i) = SqExpKernel(xNew,x(i),thetaF,thetaL);
    end

    % Make a prediction for the mean and the variance
    mu_pred = kNew'*inv(covn)*y;
    sig_pred = SqExpKernel(xNew,xNew,thetaF,thetaL) - kNew'*inv(covn)*kNew;

end


% Generate nonlinear data
function [x,y] = GenerateNonlinData(N,noise)

% Nonlinear test function (http://www.sfu.ca/~ssurjano/forretal08.html)

x = linspace(0,1,N)';
y = ((6*x-2).^2).*sin(12*x-4) + noise*randn(N,1);

end

% Implementation of the squared exponential Kernel Function
function k = SqExpKernel(x1,x2,thetaF,thetaL)

    r = x1-x2;
    k = thetaF^2*exp((-1/2*r'*r)/thetaL^2);

end
```




*Published with MATLAB® R2021b*