Gaussian Process Regression

June 22, 2016

In [1]:
```
# Run this cell first (shit+enter) to import all packages

from numpy import array, matrix, meshgrid, invert, ones,\
hstack, vstack, dot, linspace, sin, diag, abs, arange,\
cos, exp, identity
import numpy as np
from numpy.linalg import inv
from numpy.ma import masked_where
from matplotlib.pyplot import plot, fill_between

%matplotlib inline
```

1 Gaussian Process Regression Demonstration

1.1 About

This notebook is supplementary material for the lecture "Maschinelles Lernen 2" at the Karlsruhe Insitute of Technology. It is intended to give the class' participants a feeling for the discussed algorithm and a brief introduction to numerical computation with python and jupyter. You are invited to modify the code but please do not share it outside the scope of the lecture. To install it on windows, it is recommended to use the anaconda distribution. Please inform us if you encounter problems with this notebook.

1.2 Kernel functions

The kernel functions in this notebook use a python decorator that transforms the *first two* arguments into matrices which yield all combinations of the input vectors when compared elementwise (cf. Cartesian product). Note that this works only for vectors and not for matrices (see this thread to handle higher dimensions).

Example:

$$k\left((x_1,x_2)^T,(y_1,y_2)^T\right) \rightarrow \begin{pmatrix} k(x_1,y_1) & k(x_1,y_2) \\ k(x_2,y_1) & k(x_2,y_2) \end{pmatrix}$$

In [2]:
```
# Press shift+enter to execute this cell
# and add the kernel functions
```

```python
def kernel_function(f):
    def decorator(x,y,*args, **kwargs):
        xx,yy = meshgrid(x,y)
        val = f(xx,yy, *args, **kwargs)
        return matrix(val)
    return decorator

class kernels(object):
    @kernel_function
    def periodic(x,y):
        return np.exp(np.cos((x-y)))

    @kernel_function
    def squared_exponential(x,y):
        return np.exp(-((x-y)**2)/2)

    @kernel_function
    def exponential(x,y):
        return np.exp(-((x-y))/0.5)

    @kernel_function
    def symmetric(x,y):
        return np.exp(-((abs(x)-abs(y))**2)/2)

    @kernel_function
    def squared_exponential_param(x,y,signal_variance=1.0,
                                  length_scale=0.1):
        return signal_variance * \
               np.exp(-((x-y)**2)/length_scale)
```

1.3 Regression Examples

1.3.1 Squared Exponential

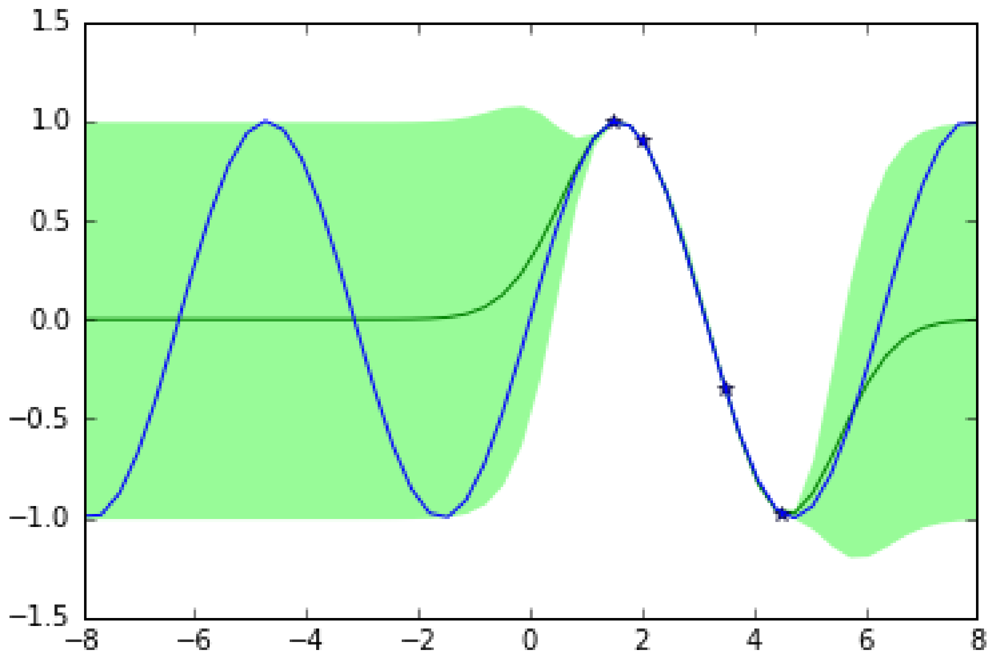
Kernel function **(stationary)**:

$$k(d) = \exp\left(-\frac{d^2}{2}\right), \quad \text{where } d = \|x - y\|$$

```python
In [3]: latent = lambda x: sin(x) # the latent function

        # input values of the training instances
        X = array([1.5,2,3.5,4.5])


        ## Uncomment the following two lines to see an example
        ## showing that it is often necessary to determine the
        ## hyperparameters of akernel:
```

```
# latent = lambda x: exp(-((x)**2)/0.25)
# X = array([-1,-0.25,0,0.25])


Y = latent(X)  # Compute the function values at

# for plotting, evaluate the GP at many x-values
x = linspace(-8,8,50)

kernel = kernels.squared_exponential

cov_X = kernel(X,X)
cov_Xx = kernel(x,X)
cov_x = kernel(x,x)

mean = cov_Xx.T * inv(cov_X) * Y.reshape(-1,1)
var = cov_x  - cov_Xx.T * inv(cov_X) * cov_Xx


upper = mean+var.diagonal().T
lower = mean-var.diagonal().T


plot(x,mean,'-g')
fill_between(x, upper.A.flatten(), lower.A.flatten(),
             lw=0, facecolor='palegreen', interpolate=True)
plot(X,Y,'*b')
plot(x,latent(x),'-b')
```

```
Out[3]: [<matplotlib.lines.Line2D at 0x112087fd0>]
```



1.3.2 A symmetric kernel

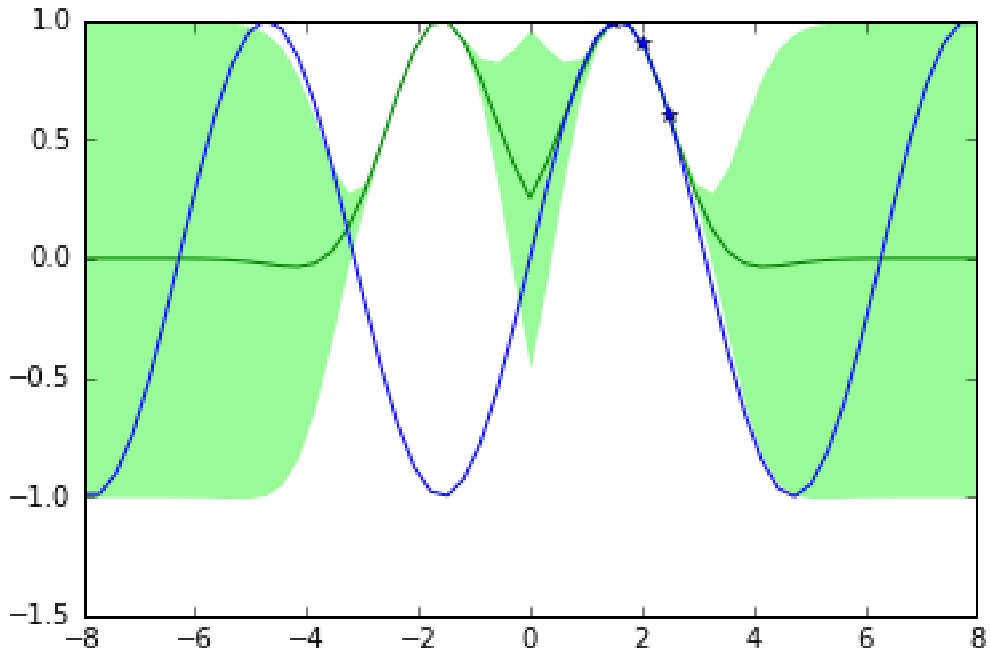
Kernel function **(non-stationary)**:

$$k(x,y) = \exp\left(-\frac{(|x|-|y|)^2}{2}\right)$$

```
In [4]: latent = lambda x: sin(x)

        X = array([1.5,2,2.5])
        Y = latent(X)
        x = linspace(-8,8,55)

        kernel = kernels.symmetric

        cov_X = kernel(X,X)
        cov_Xx = kernel(x,X)
        cov_x = kernel(x,x)

        mean = cov_Xx.T * inv(cov_X) * Y.reshape(-1,1)
        var = cov_x  - cov_Xx.T * inv(cov_X) * cov_Xx


        upper = mean+var.diagonal().T
        lower = mean-var.diagonal().T
```

```
plot(x,mean,'-g')
fill_between(x, upper.A.flatten(), lower.A.flatten(),
             lw=0, facecolor='palegreen', interpolate=True)
plot(X,Y,'*b')
plot(x,latent(x),'-b')
```

Out[4]: [<matplotlib.lines.Line2D at 0x1121dddd8>]



1.3.3 Periodic kernel

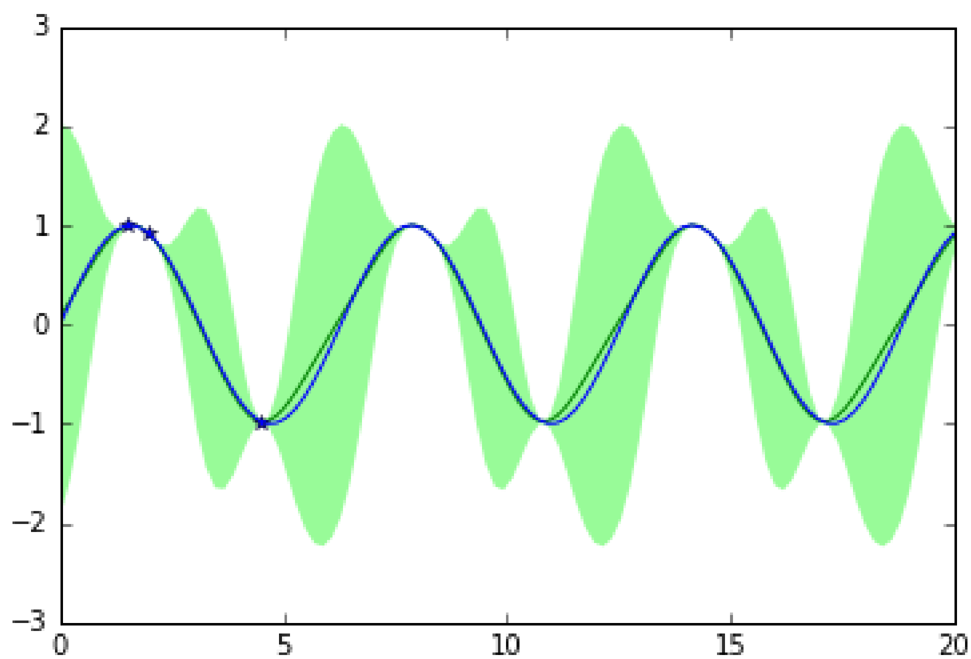
Kernel function **(stationary)**:

$$k(d) = \exp(\cos(d))$$

In [5]:
```
latent = lambda x: sin(x)

## Experiment with higher frequencies!
# latent = lambda x: sin(x*2)

X = array([1.5,2,4.5])
Y = latent(X)
x = linspace(0,20,100)

kernel = kernels.periodic
```

```
cov_X = kernel(X,X)
cov_Xx = kernel(x,X)
cov_x = kernel(x,x)

mean = cov_Xx.T * inv(cov_X) * Y.reshape(-1,1)
var = cov_x  - cov_Xx.T * inv(cov_X) * cov_Xx


upper = mean+var.diagonal().T
lower = mean-var.diagonal().T


plot(x,mean,'-g')
fill_between(x, upper.A.flatten(), lower.A.flatten(),
             lw=0, facecolor='palegreen', interpolate=True)
plot(X,Y,'*b')
plot(x,latent(x),'-b')
```

Out[5]: [<matplotlib.lines.Line2D at 0x11224d898>]



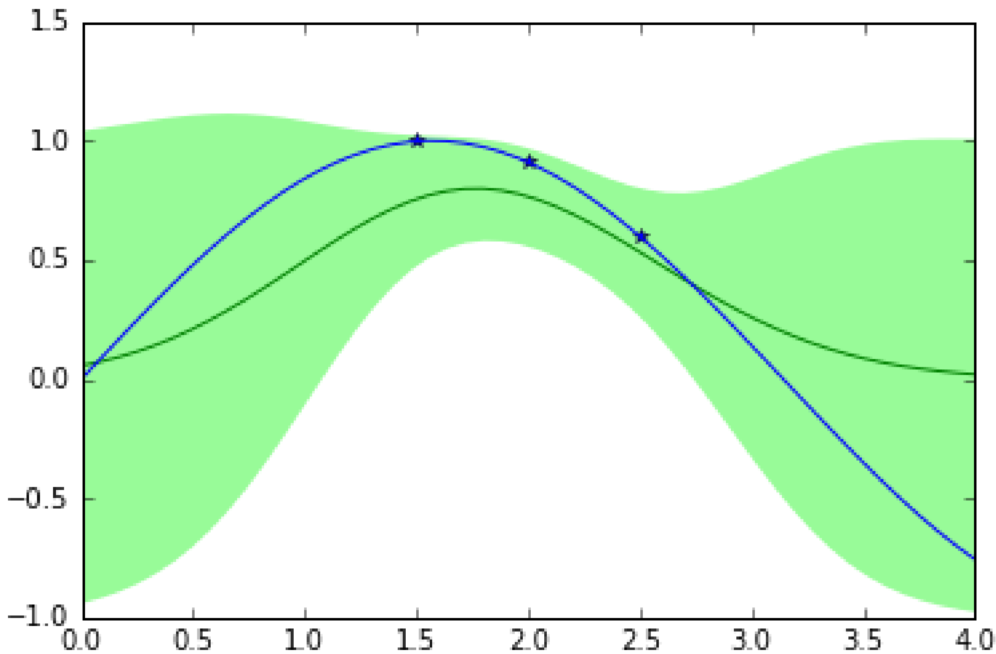
1.3.4 Paramterized Squared Exponential

Kernel **(stationary)**:

$$k(d) = \sigma_f^2 \cdot \exp(\frac{-d^2}{2 \cdot l^2}) + \sigma_n^2 \delta(d)$$

In [6]:
```
latent = lambda x: sin(x)

X = array([1.5,2,2.5])
Y = latent(X)
x = linspace(0,4,50)


# setup the hyperparameters

signal_variance=1
length_scale=1.0
noise_variance=.5


# Shortcut (no need to modify successing code)
kernel = lambda a,b: \
    kernels.squared_exponential_param(a,b, signal_variance,
                                      length_scale)


cov_X = kernel(X,X) + identity(X.size)*noise_variance
cov_Xx = kernel(x,X)
cov_x = kernel(x,x)

mean = cov_Xx.T * inv(cov_X) * Y.reshape(-1,1)
var = cov_x  - cov_Xx.T * inv(cov_X) * cov_Xx


upper = mean+var.diagonal().T
lower = mean-var.diagonal().T



plot(x,mean,'-g')
fill_between(x, upper.A.flatten(), lower.A.flatten(),
             lw=0, facecolor='palegreen', interpolate=True)
plot(X,Y,'*b')
plot(x,latent(x),'-b')
```

Out[6]:
```
[<matplotlib.lines.Line2D at 0x11234dba8>]
```



1.3.5 Polynomial Kernel

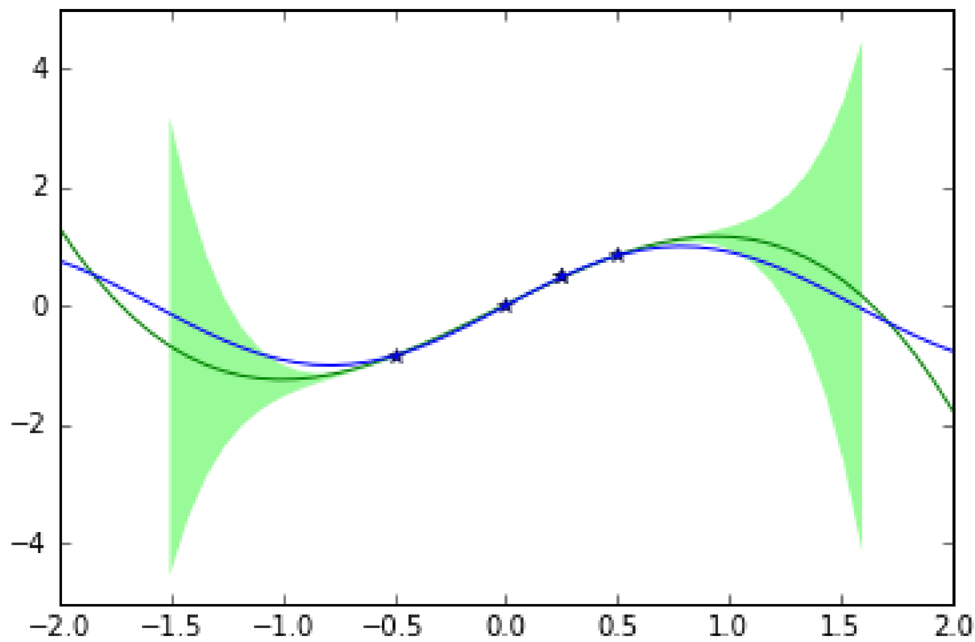
Kernel **(?)**:

$$k(x, y) = \phi(x)^t \cdot \phi(y) + \sigma_n^2 \delta(d)$$

```
In [7]: @kernel_function
        def polynomial(x,y,deg):
            """
            Polynomial kernel (inefficient as it uses for-loops)
            """
            result = ones(x.shape)
            for i in range(x.shape[0]):
                for j in range(x.shape[1]):
                    phi_x = x[i,j]**arange(deg+1)
                    phi_y = y[i,j]**arange(deg+1)
                    result[i,j] = dot(phi_x,phi_y)

            return result

        latent = lambda x:sin(x*2)

        X = array([-0.5, 0.25, 0., 0.25, 0.5])
        Y = latent(X)
        x = linspace(-2,2,50)
```

```
# Experiment with the noise value!
# See what happen for noise=0 (numerical instability).
noise=0.001

kernel = lambda a,b: polynomial(a,b, 3)


cov_X = kernel(X,X) + diag(ones((X.size)))*noise
cov_Xx = kernel(x,X)
cov_x = kernel(x,x)

mean = cov_Xx.T * inv(cov_X) * Y.reshape(-1,1)
var = cov_x  - cov_Xx.T * inv(cov_X) * cov_Xx

plot(x,mean,'-g')
plot(X,Y,'*b')
plot(x,latent(x),'-b')


# Note: Here we cut of values >/< +/- 5 as
# the variance increases quickly!

upper = (mean+var.diagonal().T).A.flatten()
lower = (mean-var.diagonal().T).A.flatten()

upper = masked_where(upper > 5.0, upper)
lower = masked_where(lower < -5.0, lower)

fill_between(x, upper, lower, lw=0,
             facecolor='palegreen', interpolate=True)
```

```
Out[7]: <matplotlib.collections.PolyCollection at 0x11245b390>
```




```
In [ ]:
```