# Lab session 1: Gaussian Process models with GPy

GP Summer School – Kampala, 6-9th of August 2013

The aim of this lab session is to illustrate the concepts seen during the lectures. We will focus on three aspects of GPs: the kernel, the random sample paths and the GP regression model.

Since the background of the attendees is very diverse, this session will attempt to cover a large spectrum from the very basic of GPs to more technical questions. This in mind, the difficulty of the questions has been ranked from $\star$ (easy) to $\star\star\star$ (difficult). Feel free to skip the parts that are either too easy or too technical.

## 1 Getting started: the covariance function

We first open *ipython* and import the libraries we will need:

```
import numpy as np
import pylab as pb
import GPy

pb.ion()                        # Open one thread per plot
```

The online documentation of GPy is available from the SheffieldML github page: `https://github.com/SheffieldML/GPy`.

Lets start with defining a squared-exponential kernel (ie rbf or Gaussian) in one dimension:

```
d = 1                           # input dimension
var = 1.                        # variance
theta = 0.2                     # lengthscale

k = GPy.kern.rbf(d,var,theta)
```

A summary of the kernel can be obtained using the command `print k`. It is also possible to plot the kernel as a function of one of its inputs (whilst fixing the other) with `k.plot()`.

The value of the kernel parameters can be accessed and modified using `k[".*var"]` where the string in square brackets is a regular expression matching the parameter name as it appears in `print k`. To get an insight on the effect of the parameters their shape, try:

```
k = GPy.kern.rbf(d)           # By default, the parameters are set to 1.

theta = np.asarray([0.2,0.5,1.,2.,4.])
for t in theta:
    k[".*lengthscale"]=t
    k.plot()

pb.legend(theta)
```

**Question 1**

- ⋆ What is the effect of the lengthscale parameter?
- ⋆ Similarly, change the previous bit of code to see the influence of the variance parameter.

Many kernels are already implemented in GPy. Instead of `rbf`, try plotting the following ones `exponential`, `Matern32`, `Matern52`, `Brownian`, `linear`, `bias`, `rbfcos`, `periodic_matern32`, etc... Some of these kernels, such as *rbfcos*, are not parametrized by a variance and a lengthscale. Furthermore, not all kernels are stationary (ie they can't all be written as $k(x,y) = f(x-y)$, see for example the *Brownian kernel*) so it may be interesting to change the value of the fixed input:

```
kb = GPy.kern.Brownian(input_dim=1)
kb.plot(x = 2.,plot_limits=[0,5])
kb.plot(x = 4.,plot_limits=[0,5],ls="--",color="r")
pb.ylim([-0.1,5.1])
```

Let $X$ be a $n \times d$ numpy array. Given a kernel $k$, the covariance matrix associated to $X$ is obtained with `C = k.K(X,X)`. The positive semi-definiteness of $k$ ensures that $C$ is a positive semi-definite (psd) matrix regardless of the initial points $X$. This can be checked numerically by looking at the eigenvalues:

```
k = GPy.kern.Matern52(input_dim=2)
X = np.random.rand(50,2)      # 50*2 matrix of iid uniform
C = k.K(X,X)
np.linalg.eigvals(C)          # Computes the eigenvalues of a matrix
```

**Question 2**

⋆⋆ Is the sum of two psd matrices also psd?

⋆⋆⋆ Show that the product of 2 kernels is also a valid covariance structure.

In GPy, the sum and the product of kernels can be achieved using the usual $+$ and $\times$ operators. For example, we can define two new kernels as `ksum = k + kb` and `kprod = k * kb`.

## 2 Sample paths from a GP

A psd-matrix can be seen as the covariance of a Gaussian vector. For example, we can simulate sample paths from a GP as follows:

```
k = GPy.kern.rbf(input_dim=1,lengthscale=0.2)

X = np.linspace(0.,1.,500)    # 500 points evenly spaced over [0,1]
X = X[:,None]                 # reshape X to make it n*D

mu = np.zeros((500))          # vector of the means
C = k.K(X,X)                  # covariance matrix

# Generate 20 sample path with mean mu and covariance C
Z = np.random.multivariate_normal(mu,C,20)

pb.figure()                   # open new plotting window
for i in range(20):
    pb.plot(X[:],Z[i,:])
```

**Question 3**

⋆ Investigate the influence of the choice of the kernel (and its parameters) on the sample paths.

⋆⋆ Can you tell the covariance structures that have been used for generating the sample paths shown in Figure 1?

## 3 GP regression model

We will now see how to create a GP regression model with GPy. We consider the toy function $f(x) = -\cos(\pi x) + \sin(4\pi x)$ over $[0, 1]$ and we assume we have the following observations (note that the observations $Y$ usually include some noise):

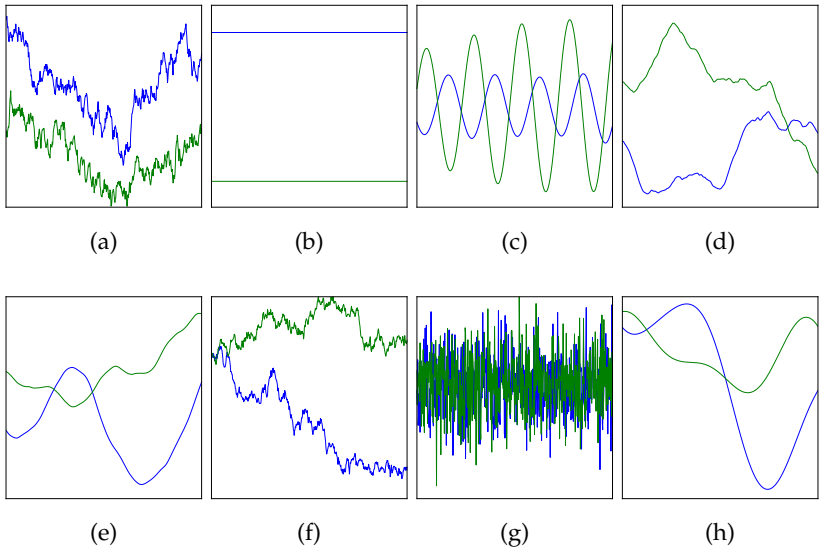
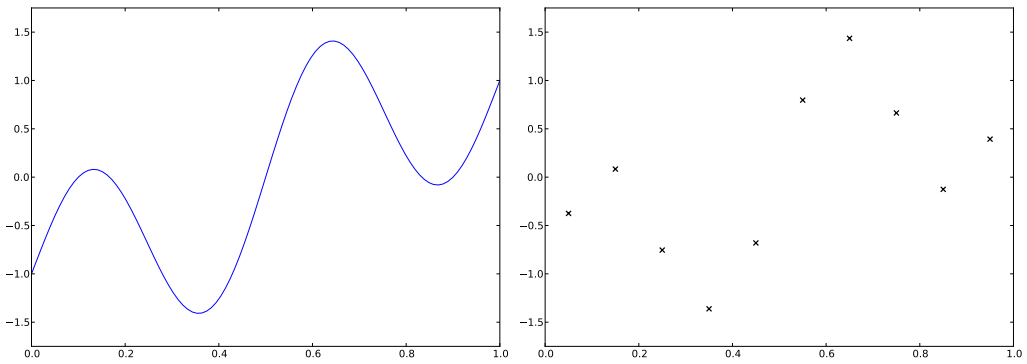



Figure 1: Examples of sample paths from centred GPs with various kernels.

```
X = np.linspace(0.05,0.95,10)[:,None]
Y = -np.cos(np.pi*X) +np.sin(4*np.pi*X) + np.random.randn(10,1)*0.2
pb.figure()
pb.plot(X,Y,"kx",mew=1.5)
```




(a) test function (b) Observations

Figure 2: test function and training set.

A GP regression model based on an squared-exponential kernel can be defined as follows:

```
k = GPy.kern.rbf(input_dim=1, variance=1., lengthscale=.2)
m = GPy.models.GPRegression(X,Y,k)
```

As previously, the commands `print m` and `m.plot()` are available to obtain a summary of the model. Note that by default the model includes some observation noise with variance 1. Furthermore, the predictions of the model for a new set of points $Xp$ (a $m \times D$ array) can be computed using `Yp, Vp, up95, lo95 = m.predict(Xp)`.

**Question 4**

- ⋆ What do you think about this first fit? Does the prior given by the GP seem to be adapted?
- ⋆ The parameters of the models can be modified using a regular expression matching the parameters names (for example `m["noise"] = 0.001`). Change the values of the parameters to obtain a better fit.
- ⋆ What difference does it make in the fit to have a noise equal to zero?.

As in Section 2, random sample paths from the conditional GP can be obtained using `np.random.multivariate_normal(mu[:,0],C)` where the mean vector and covariance matrix *mu*, *C* are given by `mu, C, up95, lo95 = m.predict(Xp,full_cov=True)`.

As we have seen during the lectures, the parameters values can be estimated by maximizing the likelihood of the observations. Since we don't want one of the variances to become negative during the optimization, we first need to constrain all parameters to be positive before running the optimisation. We will assume first a free noise model. We can define such a model by fixing the noise term to be zero:

```
m.constrain_positive("")      # "" is a regex matching all parameter names
m.constrain_fixed("noise",0)
```

Now we can optimize the model:

```
m.optimize()
m.plot()
```

To include noise in the model, we will constrain the noise as positive. Before optimizing again, we will randomize the initial parameter values. It might also be helpful to set a small initial value for the noise:

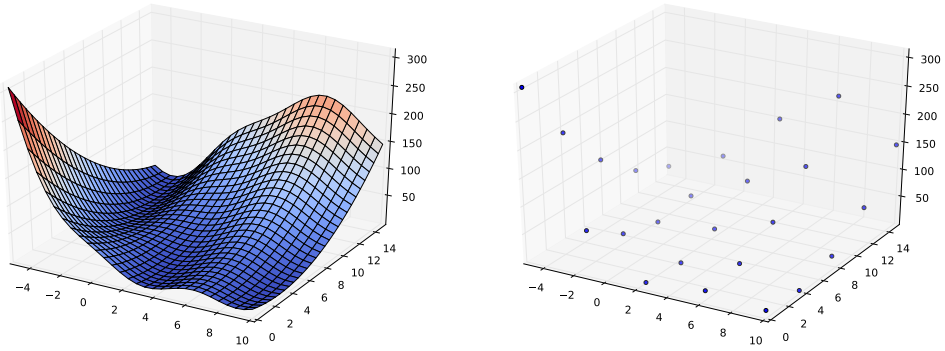
```
m.constrain_positive("noise")
m.randomize()
m["noise"] = .01
m.optimize()
m.plot()
```

The parameters obtained after optimisation can be compared with the values obtained by hand. As perviously, you can modify the kernel used for building the model to investigate its influence on the model.

# 4 Uncertainty propagation

Let $X$ be a random variable defined over $\mathbb{R}$ and $f$ be a function $\mathbb{R} \to \mathbb{R}$. Uncertainty propagation is the study of the distribution of the random variable $f(X)$.

We will see in this section the advantage of using a model when only a few observations of $f$ are available. We consider here the 2-dimensional Branin test function defined over $[-5, 10] \times [0, 15]$ and a set of 25 observations as seen in Figure 3.




(a) Branin function (b) Observations

Figure 3: Branin test function and training set.

```
# Definition of the Branin test function
def branin(X):
    y = (X[:,1]-5.1/(4*np.pi**2)*X[:,0]**2+5*X[:,0]/np.pi-6)**2
    y += 10*(1-1/(8*np.pi))*np.cos(X[:,0])+10
    return(y)

# Training set defined as a 5*5 grid:
xg1 = np.linspace(-5,10,5)
xg2 = np.linspace(0,15,5)

X = np.zeros((xg1.size * xg2.size,2))
for i,x1 in enumerate(xg1):
    for j,x2 in enumerate(xg2):
        X[i+xg1.size*j,:] = [x1,x2]

Y = branin(X)[:,None]
```

We assume here that we are interested in the distribution of $f(U)$ where $U$ is a random variable with uniform distribution over the input space of $f$. We will focus on the computation of two quantities: $\mathrm{E}[f(U)]$ and $\mathrm{P}(f(U) > 200)$.

## 4.1 Computation of $\mathrm{E}[f(U)]$

The expectation of $f(U)$ is given by $\int f(x)\mathrm{d}x$. A basic approach to approximate this integral is to compute the mean of the 25 observations: `np.mean(Y)` . Since the points are distributed on a grid, this can be seen as the approximation of the integral by a rough Riemann sum. The result can be compared with the actual mean of the Branin function which is 54.31.

Alternatively, we can fit a GP model and compute the integral of the best predictor by Monte Carlo sampling:

```
# Fit a GP
kg = GPy.kern.rbf(input_dim=2, ARD = True)
kb = GPy.kern.bias(input_dim=2)

k = kg + kb
k.plot()

m = GPy.models.GPRegression(X,Y,k,normalize_Y=True)
m.constrain_bounded("rbf_var",1e-3,1e5)
m.constrain_bounded("bias_var",1e-3,1e5)
m.constrain_bounded("rbf_len",.1,200.)
m.constrain_fixed("noise",1e-5)

m.randomize()
m.optimize()

m.plot()

# Compute the mean of model prediction on 1e5 Monte Carlo samples
Xp = np.random.uniform(size=(1e5,2))
Xp[:,0] = Xp[:,0]*15-5
Xp[:,1] = Xp[:,1]*15
Yp = m.predict(Xp)[0]
np.mean(Yp)
```

**Question 5**

- ⋆ Has the approximation of the mean been improved by using the GP model?

⋆⋆⋆ One particular feature of GPs we have not use for now is their prediction variance. Can you use it to define some confidence intervals around the previous result?

## 4.2 Computation of $\mathrm{P}(f(U) > 200)$

In various cases it is interesting to look at the probability that $f$ is greater than a given threshold. For example, assume that $f$ is the response of a physical model representing the maximum constraint in a structure depending on some parameters of the system such as Young's modulus of the material (say $Y$) and the force applied on the structure (say $F$). If the later are uncertain, the probability of failure of the structure is given by $\mathrm{P}(f(Y,F) > f_{max})$ where $f_{max}$ is the maximum acceptable constraint.

**Question 6**

- ⋆ As previously, use the 25 observations to compute a rough estimate of the probability that $f(U) > 200$.

- ⋆ Compute the probability that the best predictor is greater than the threshold.
- ⋆⋆ Compute some confidence intervals for the previous result

These two values can be compared with the actual value $\mathrm{P}(f(U) > 200) = 1.23\ 10^{-2}$.

We now assume that we have an extra budget of 10 evaluations of $f$ and we want to use these new evaluations to improve the accuracy of the previous result.

**Question 7**

- ⋆ Given the previous GP model, where is it interesting to add the new observations if we want to improve the accuracy of the estimator and reduce its variance?
- ⋆⋆⋆ Can you think about (and implement!) a procedure that updates sequentially the model with new points in order to improve the estimation of $\mathrm{P}(f(U) > 200)$?