# 00_gp_overview

January 15, 2025

## 1 Introduction to Gaussian Processes

### 1.1 Overview

Let $f : \mathbb{R} \to \mathbb{R}$ be a real valued function, such that $x \mapsto f(x)$. A Gaussian Process (GP) is a collection of random variables, such that any finite subset thereof follows a multivariate Gaussian distribution.

GPs generalise the concept of a Gaussian distribution to functions, i.e. instead of modeling discrete data points, it models an entire function. Therefore, assume a Gaussian Process (GP) is placed over f, in symbols:

$$f \sim \mathrm{GP}(m(x), k(x)).$$

where $m(x)$ is called *mean function*, and $k(x)$ is the so-called kernel (or covariance function)

In this short note, we shall see how a GP looks like.

### 1.2 Assumptions

Although there are a number of available kernels, we shall consider a *squared exponential* one given its versatility and widespread use:

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

where $C$ is the kernel variance, and $l$ is the length scale.

Whilst, we assume null mean function $m(x) = 0 \, \forall x$, as commonly done in the absence of prior knowledge.

To assess how these realisation behaves let's import a few Python packages and modules.

## 2 Hands-on Example

To assess how these GP realisations behave let's import a few Python packages and modules.

[1]:
```
import sys
sys.path.append("..")

from typing import Tuple, List

import torch
```

```
from torch import distributions as dist

import pyro
import pyro.contrib.gp as gp

from src.inference import kernel
from src.view import inspect_data, inspect_gp_samples

# reproducibility
pyro.set_rng_seed(0)
```

```
/home/ale/pythonEnv/tut/lib/python3.12/site-packages/tqdm/auto.py:21:
TqdmWarning: IProgress not found. Please update jupyter and ipywidgets. See
https://ipywidgets.readthedocs.io/en/stable/user_install.html
  from .autonotebook import tqdm as notebook_tqdm
```

In the following, we shall vary the parameters of the kernel ($C$, and $l$) to see how GP samples modifies.

### 2.1 Generation of Parameters

Let's code a function for this purpose, and draw some parameters.

[2]:
```
def param_gp() -> Tuple[torch.Tensor, torch.Tensor, List[float]]:
    """
    Returns the parameters for the Gaussian process.

    Returns
    -------
    tuple of (torch.Tensor, torch.Tensor, list of float)
        - A tensor of length scales.
        - A tensor of variances.
        - A list of noise variances.
    """
    return torch.linspace(3, 15, 5, dtype=torch.float64), \
        torch.linspace(0.5, 2.5, 5, dtype=torch.float64), \
            [0.4e-4]*5

var_list, len_list, noise_list = param_gp() # prepare parameters
```

### 2.2 Sampling a GP

Any sample of $f$ obeys a multivariate distribution such that:

$$f \sim \mathcal{N}(m(x), k(x, x')).$$

Normally, we cannot observe the values of $f$, but only noisy samples. Therefore, we add random noise to the kernel, and the covariance matrix becomes:

$$k(x, x') \mapsto k(x, x') + \sigma_n^2 \mathbf{1}$$

[3]:
```
def sample_gp(x: torch.Tensor, var: float, length: float, noise: float) ->␣
↪torch.Tensor:
    """
    Samples from a Gaussian Process with the specified parameters.

    Parameters
    ----------
    x : torch.Tensor
        The input locations for the Gaussian process.
    var : float
        The variance (amplitude) of the kernel.
    length : float
        The length scale of the kernel.
    noise : float
        The noise variance to be added to the kernel diagonal.

    Returns
    -------
    torch.Tensor
        Samples from the Gaussian process with the given parameters.
    """
    cov = kernel(X=x, Z=x, var=var, length=length, noise=noise, jitter=1.e-6)
    return dist.MultivariateNormal(loc=torch.zeros_like(x),␣
↪covariance_matrix=cov).sample()
```

### 2.3 Visual Inspection

To display the GPs, we shall sample over, e.g., $x \in [-3, 3]$.

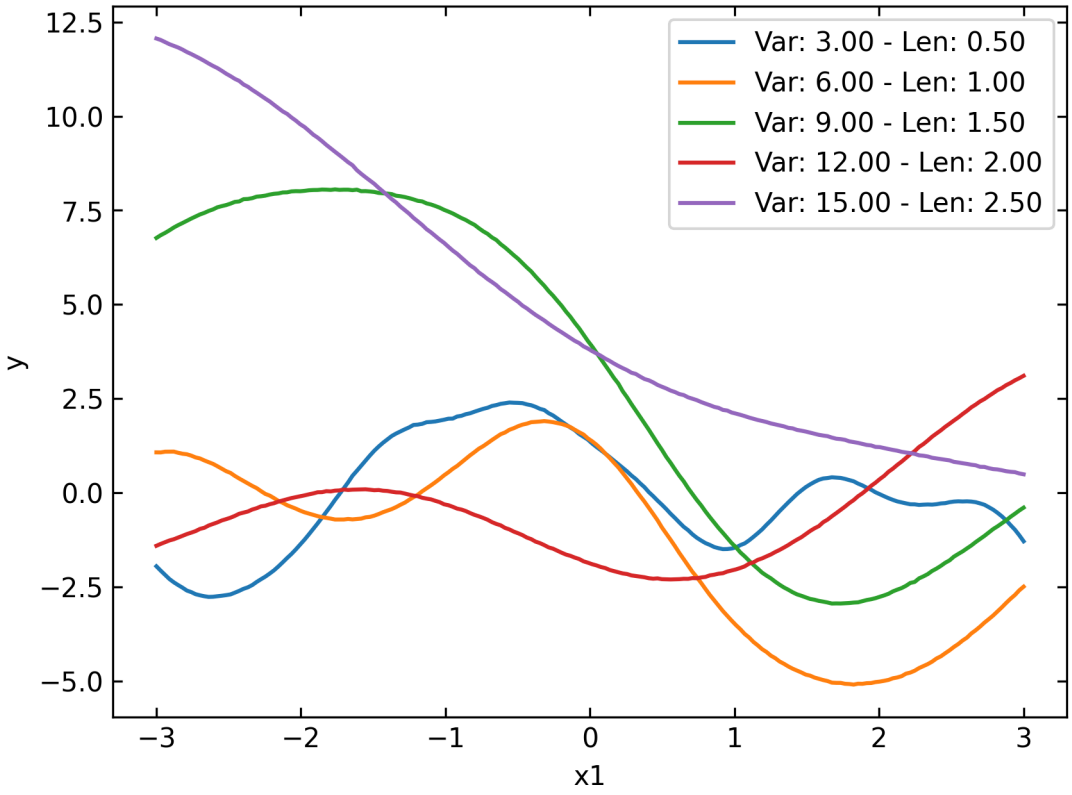
[4]:
```
x = torch.linspace(-3., 3., 200) # sample location
```

First, we sample the distribution for different parameters (at the same location, though) using list comprehension, and inspect:

[5]:
```
y_sample = [sample_gp(x, v, l, n) for v, l, n in zip(var_list, len_list,␣
↪noise_list)]
```

Then, we inspect the samples.

[6]:
```
inspect_gp_samples(x, y_sample, var_list, len_list)
```


Var: 3.00 - Len: 0.50
Var: 6.00 - Len: 1.00
Var: 9.00 - Len: 1.50
Var: 12.00 - Len: 2.00
Var: 15.00 - Len: 2.50
12.5
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
y
−3
−2
−1
0
1
2
3
x1