TUM-DI-LAB Documentation

K. Harsha, A. Grundner, K. Wang

Jun 24, 2018

CONTENTS

CHAPTER

ONE

INTRODUCTION TO GAUSSIAN PROCESS

Define Gaussian processes and properties

CHAPTER TWO

SIMPLE EXAMPLE OF A GAUSSIAN PROCESS

The following example illustrates how we move from process to distribution and also shows that the Gaussian process defines a distribution over functions.

$f \sim \mathcal{GP}(m, k)$

$m(x) = \frac{x^2}{4}$

$k(x, x') = exp(-\frac{1}{2}(x - x')^2)$

$y = f + \epsilon$

$\epsilon \sim \mathcal{N}(0, \sigma^2)$

```
In [1]: import numpy as np
        import scipy as sp
        import matplotlib.pyplot as plt
```

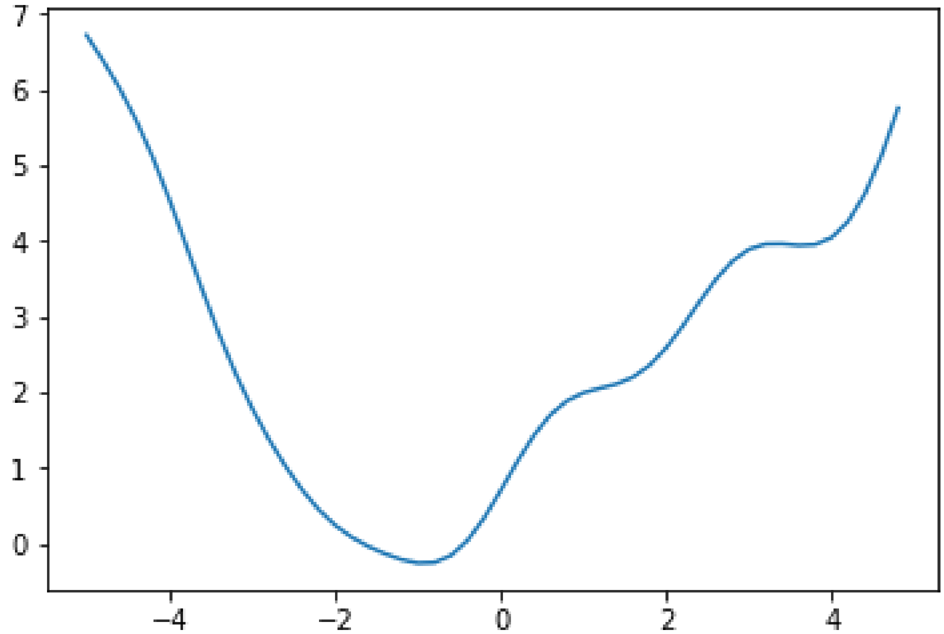
```
In [2]: x = np.arange(-5,5,0.2)
        n = x.size
        s = 1e-9
```

```
In [3]: m = np.square(x) * 0.25
```

```
In [4]: a = np.repeat(x, n).reshape(n, n)
        k = np.exp(-0.5*np.square(a - a.transpose())) + s*np.identity(n)
```

```
In [5]: r = np.random.multivariate_normal(m, k, 1)
        y = np.reshape(r, n)
```

```
In [6]: plt.plot(x,y)
        plt.show()
```




```
In [ ]:
```

CHAPTER

THREE

LINEAR OPERATORS ON GPS

Description of covariance transformations

CHAPTER

FOUR

PARAMETER ESTIMATION FOR A LINEAR OPERATOR USING GAUSSIAN PROCESSES

Assumptions about the linear operator:

$\mathcal{L}_x^\phi u(x) = f(x)$

$u(x) \sim \mathcal{GP}(0, k_{uu}(x, x', \theta))$

$f(x) \sim \mathcal{GP}(0, k_{ff}(x, x', \theta, \phi))$

$y_u = u(X_u) + \epsilon_u; \epsilon_u \sim \mathcal{N}(0, \sigma_u^2 I)$

$y_f = f(X_f) + \epsilon_f; \epsilon_f \sim \mathcal{N}(0, \sigma_f^2 I)$

Taking a simple operator as example:

$\mathcal{L}_x^\phi := \phi \cdot + \frac{d}{dx}\cdot$

$u(x) = sin(x)$

$f(x) = \phi sin(x) + cos(x)$

Problem at hand:

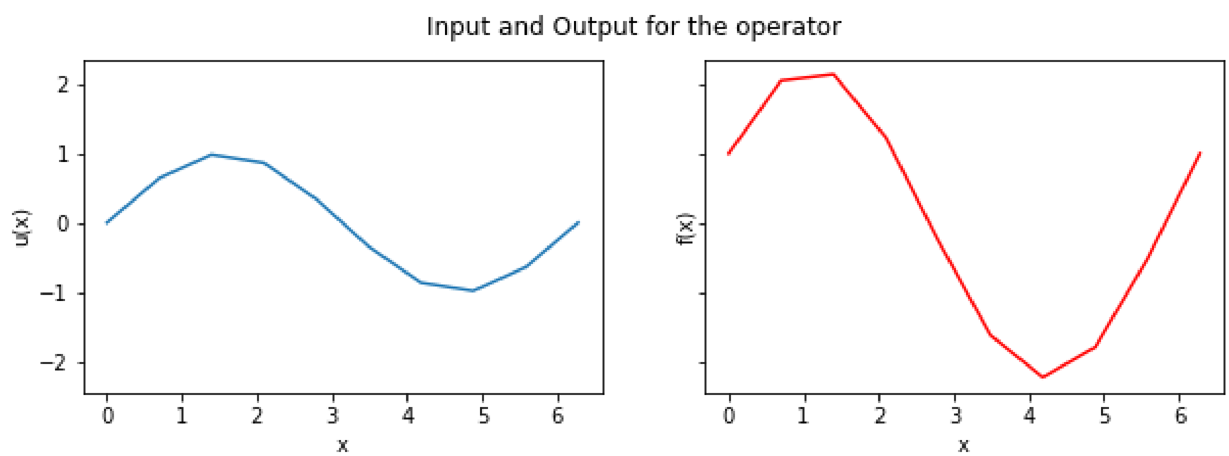
Given $\{X_u, y_u\}$ and $\{X_f, y_f\}$, estimate ϕ.

4.1 step 1: simulate data

Use $\phi = 2$

```
In [2]: x_u = np.linspace(0,2*np.pi,10)
        y_u = np.sin(x_u)
        x_f = np.linspace(0,2*np.pi, 10)
        y_f = 2.0*np.sin(x_f) + np.cos(x_f)
```

```
In [4]: plt.show()
```



4.2 step 2: create covariance matrix

This step uses information about $\mathcal{L}_x^\phi$ but not about $u(x)$ or $f(x)$.

$k_{uu}(x_i, x_j; \theta) = \theta exp(-\frac{1}{2}(x_i - x_j)^2)$

```
In [5]: x_i, x_j, theta, phi = sp.symbols('x_i x_j theta phi')
        kuu_sym = theta*sp.exp(-1/(2)*((x_i - x_j)**2))
        kuu_fn = sp.lambdify((x_i, x_j, theta), kuu_sym, "numpy")
        def kuu(x, theta):
            k = np.zeros((x.size, x.size))
            for i in range(x.size):
                for j in range(x.size):
                    k[i,j] = kuu_fn(x[i], x[j], theta)
            return k
```

$$
\begin{aligned}
&k_{ff}(x_i, x_j; \theta, \phi) \\
&= \mathcal{L}_{x_i}^\phi \mathcal{L}_{x_j}^\phi k_{uu}(x_i, x_j; \theta) \\
&= \mathcal{L}_{x_i}^\phi \left(\phi k_{uu} + \frac{\partial}{\partial x_j} k_{uu}\right) \\
&= \phi^2 k_{uu} + \phi \frac{\partial}{\partial x_j} k_{uu} + \phi \frac{\partial}{\partial x_i} k_{uu} + \frac{\partial}{\partial x_i}\frac{\partial}{\partial x_j} k_{uu}
\end{aligned}
$$

More explicit calculations follow:

$$
\begin{aligned}
&\mathcal{L}_{x_i}^\phi \mathcal{L}_{x_j}^\phi \left[\theta exp(-\frac{1}{2}(x_i - x_j)^2)\right] \\
&= \mathcal{L}_{x_i}^\phi \left[\theta exp(-\frac{1}{2}(x_i - x_j)^2) \left(\phi + (-\frac{1}{2})2(x_i - x_j)(-1)\right)\right] \\
&= \mathcal{L}_{x_i}^\phi \left[\theta exp(-\frac{1}{2}(x_i - x_j)^2)(\phi + x_i - x_j)\right] \\
&= \phi\theta exp(-\frac{1}{2}(x_i - x_j)^2)(\phi + x_i - x_j) + \theta exp(-\frac{1}{2}(x_i - x_j)^2)\left[-\frac{1}{2}2(x_i - x_j)(\phi + x_i - x_j) + 1\right] \\
&= \theta exp(-\frac{1}{2}(x_i - x_j)^2)\left[\phi^2 - (x_i - x_j)^2 + 1\right]
\end{aligned}
$$

```
In [6]: kff_sym = phi**2*kuu_sym + phi*sp.diff(kuu_sym, x_j) + phi*sp.diff(kuu_sym, x_i) + sp.diff(ku
        kff_fn = sp.lambdify((x_i, x_j, theta, phi), kff_sym, "numpy")
        def kff(x, theta, phi):
            k = np.zeros((x.size, x.size))
            for i in range(x.size):
                for j in range(x.size):
                    k[i,j] = kff_fn(x[i], x[j], theta, phi)
            return k
```

$$
\begin{aligned}
&k_{fu}(x_i, x_j; \theta, \phi) \\
&= \mathcal{L}_{x_i}^\phi k_{uu}(x_i, x_j; \theta)
\end{aligned}
$$

$= \phi k_{uu} + \frac{\partial}{\partial x_i} k_{uu}$
$= \mathcal{L}_{x_i}^{\phi} \left[\theta exp(-\frac{1}{2}(x_i - x_j)^2) \right]$
$= \theta exp(-\frac{1}{2}(x_i - x_j)^2) \left[(-\frac{1}{2}) 2(x_i - x_j) + \phi \right]$
$= \theta exp(-\frac{1}{2}(x_i - x_j)^2)(\phi - x_i + x_j)$

```
In [7]: kfu_sym = phi*kuu_sym + sp.diff(kuu_sym, x_i)
        kfu_fn = sp.lambdify((x_i, x_j, theta, phi), kfu_sym, "numpy")
        def kfu(x1, x2, theta, phi):
            k = np.zeros((x1.size, x2.size))
            for i in range(x1.size):
                for j in range(x2.size):
                    k[i,j] = kfu_fn(x1[i], x2[j], theta, phi)
            return k
```

$k_{uf}(x_i, x_j; \theta, \phi)$
$= \mathcal{L}_{x_j}^{\phi} k_{uu}(x_i, x_j; \theta)$
$= \mathcal{L}_{x_j}^{\phi} \left[\theta exp(-\frac{1}{2}(x_i - x_j)^2) \right]$
$= \theta exp(-\frac{1}{2}(x_i - x_j)^2) \left[(-\frac{1}{2}) 2(x_i - x_j)(-1) + \phi \right]$
$= \theta exp(-\frac{1}{2}(x_i - x_j)^2)(\phi + x_i - x_j)$

```
In [8]: def kuf(x1, x2, theta, phi):
            return kfu(x1,x2,theta,phi).T
```

4.3 step 3: define negative log marginal likelihood

$$K = \begin{bmatrix} k_{uu}(X_u, X_u; \theta) + \sigma_u^2 I & k_{uf}(X_u, X_f; \theta, \phi) \\ k_{fu}(X_f, X_u; \theta, \phi) & k_{ff}(X_f, X_f; \theta, \phi) + \sigma_f^2 I \end{bmatrix}$$

For simplicity, assume $\sigma_u = \sigma_f$.

$\mathcal{NLML} = \frac{1}{2} \left[log|K| + y^T K^{-1} y + N log(2\pi) \right]$

where $y = \begin{bmatrix} y_u \\ y_f \end{bmatrix}$

```
In [9]: def nlml(params, x1, x2, y1, y2, s):
            params = np.exp(params)
            K = np.block([
                [kuu(x1, params[0]) + s*np.identity(x1.size), kuf(x1, x2, params[0], params[1])],
                [kfu(x1, x2, params[0], params[1]), kff(x2, params[0], params[1]) + s*np.identity(x2
            ])
            y = np.concatenate((y1, y2))
            val = 0.5*(np.log(abs(np.linalg.det(K))) + np.mat(y) * np.linalg.inv(K) * np.mat(y).T)
            return val.item(0)

In [10]: nlml((1, 2), x_u, x_f, y_u, y_f, 1e-6)

Out[10]: 1121127.9918793645
```

4.4 step 4: optimise hyperparameters

```
In [11]: m = minimize(nlml, np.random.rand(2), args=(x_u, x_f, y_u, y_f, 1e-6), method="Nelder-Mead")

In [12]: m

Out[12]: final_simplex: (array([[-1.45260947,  0.69314849],
                [-1.45254371,  0.69314842],
```

```
                [-1.45254438,  0.69314847]]), array([-54.80042072, -54.80042072, -54.8004207 ]))
                     fun: -54.80042072210781
                 message: 'Optimization terminated successfully.'
                    nfev: 131
                     nit: 68
                  status: 0
                 success: True
                       x: array([-1.45260947,  0.69314849])
```

```
In [13]: np.exp(m.x)
```

```
Out[13]: array([0.23395898, 2.00000261])
```

Limitations of available tools

Linear PDEs

Non-linear PDEs

PDEs without discretization

Results and Analysis

Conclusion