

# Mathematical foundations of Machine Learning 2022 – lesson 2

Shai Dekel



TEL AVIV UNIVERSITY

# Gaussian Process for Noisy Time Series forecasting

# Multivariate Normal Distribution

Let  $y = (y_1, y_2, \dots, y_n) \in \mathbb{R}^n$  be a random vector such that

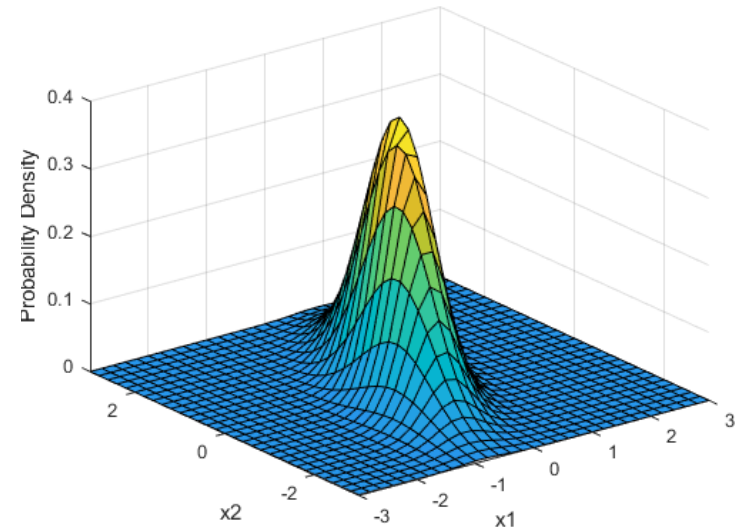
$$y \sim \mathcal{N}(\mu, \Sigma).$$

$$\mu = E(y) = (E(y_1), \dots, E(y_n)) = (\mu_1, \dots, \mu_n),$$

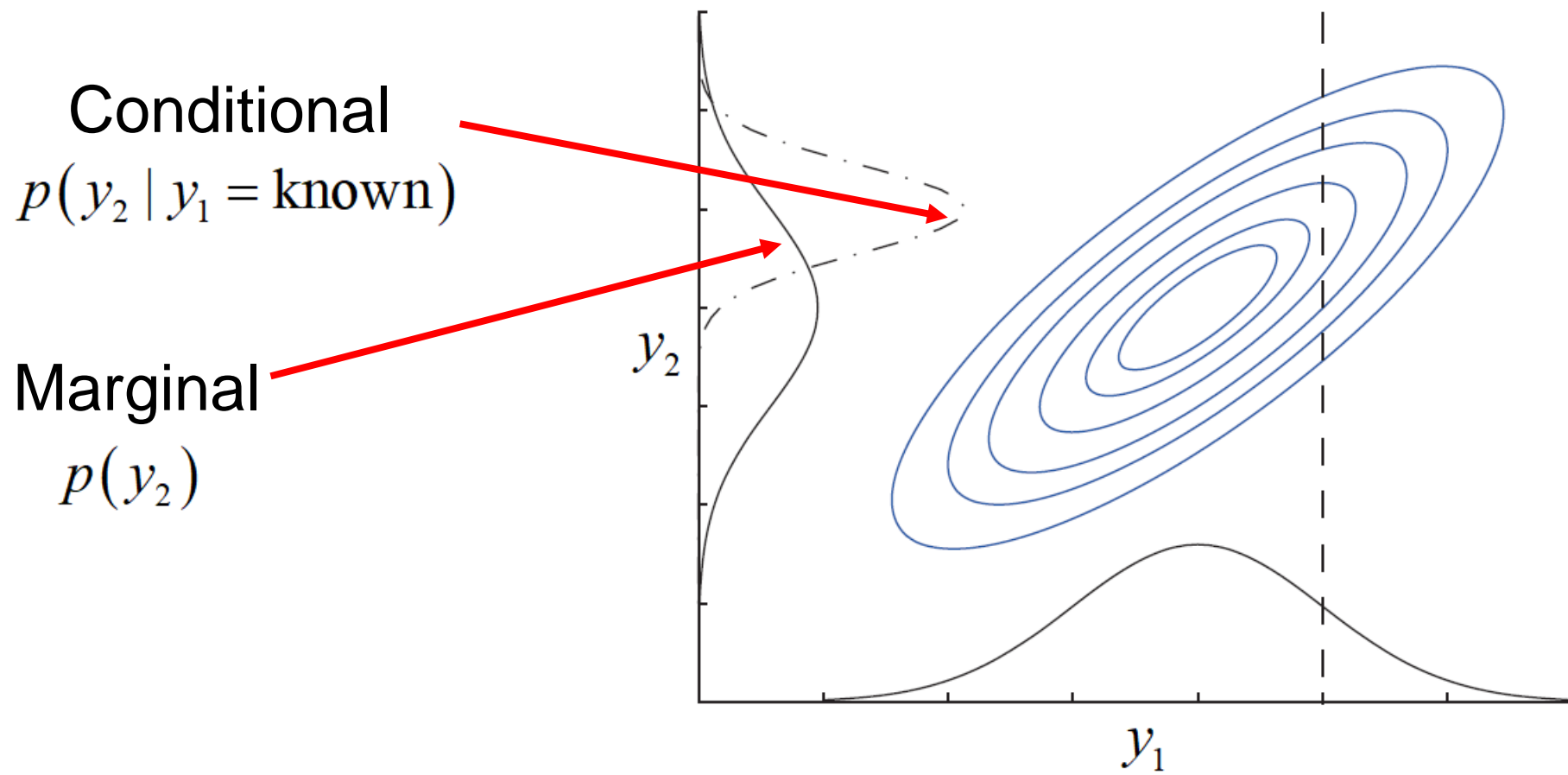
$$\Sigma_{i,j} = E((y_i - \mu_i)(y_j - \mu_j)) = \text{cov}(y_i, y_j).$$

Density function

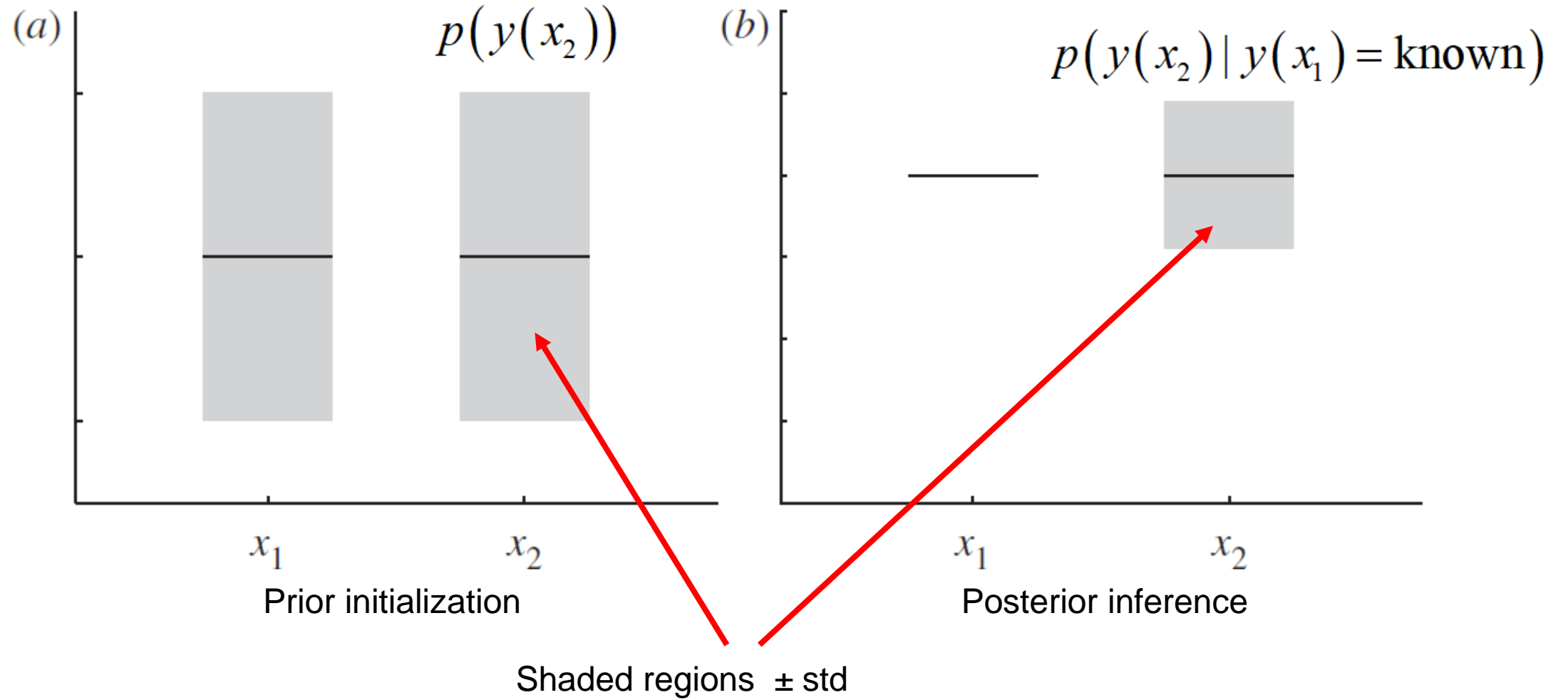
$$p(y) = \frac{1}{\sqrt{(2\pi)^n |\Sigma|}} e^{-\frac{1}{2}(y-\mu)^T \Sigma^{-1}(y-\mu)}$$



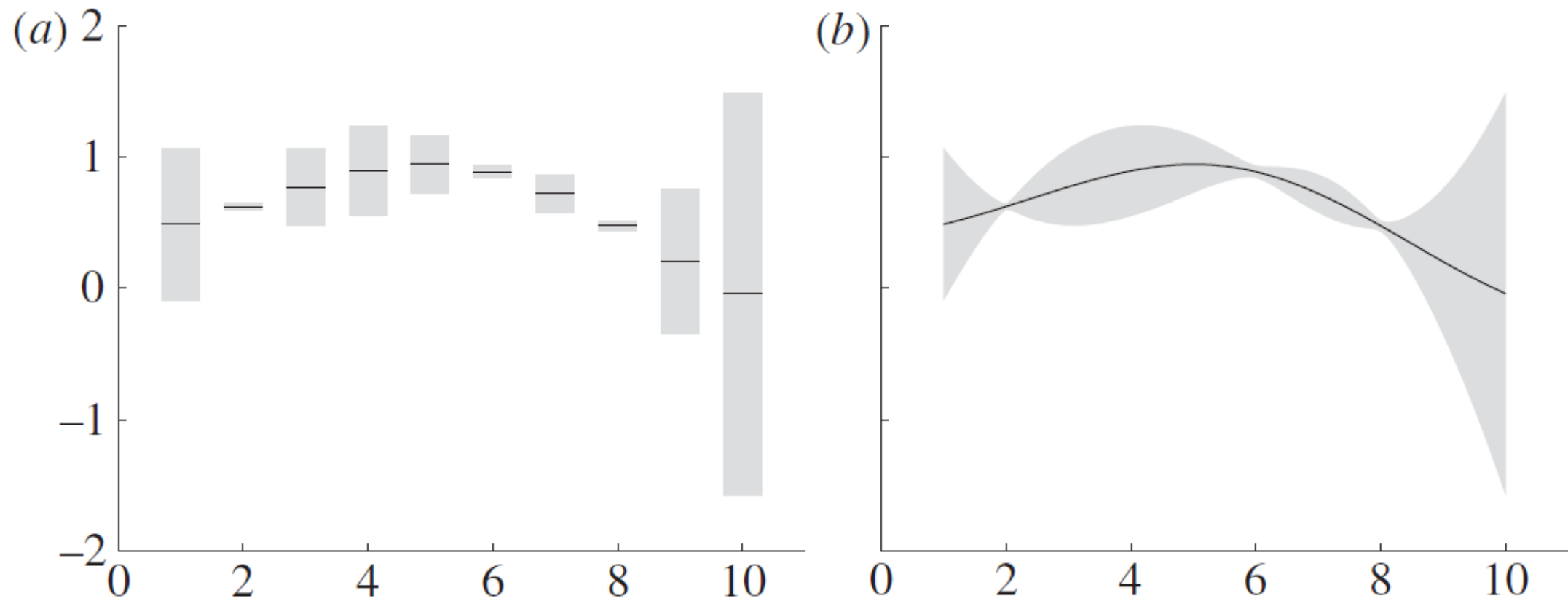
# Marginal/Conditional distributions



# GP – same example as a time-series



# Qualitative view with more time samples...



Posterior after observations at  $x=2,6,8$

# How does it work?

We have measurements at times  $x = (x_1, x_2, \dots, x_k)$ , with observations  $y$  and we want to forecast  $y_*$  at times  $x_* = (x_{k+1}, \dots, x_n)$ .

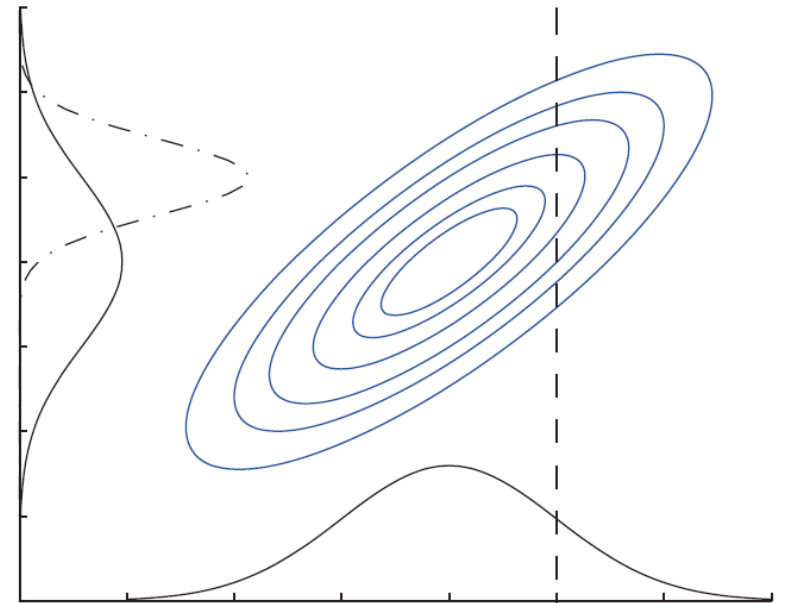
We assume the normal distribution

$$p\left(\begin{bmatrix} y \\ y_* \end{bmatrix}\right) = \mathcal{N}\left(\begin{bmatrix} \mu(x) \\ \mu(x_*) \end{bmatrix}, \begin{bmatrix} \Sigma(x, x) & \Sigma(x, x_*) \\ \Sigma(x_*, x) & \Sigma(x_*, x_*) \end{bmatrix}\right),$$

and then extract the conditional (posterior) distribution

$$m_* = \mu(x_*) + \Sigma(x_*, x)\Sigma(x, x)^{-1}(y - \mu(x))$$

$$\sigma_*^2 = \Sigma(x_*, x_*) - \Sigma(x_*, x)\Sigma(x, x)^{-1}\Sigma(x, x_*)$$



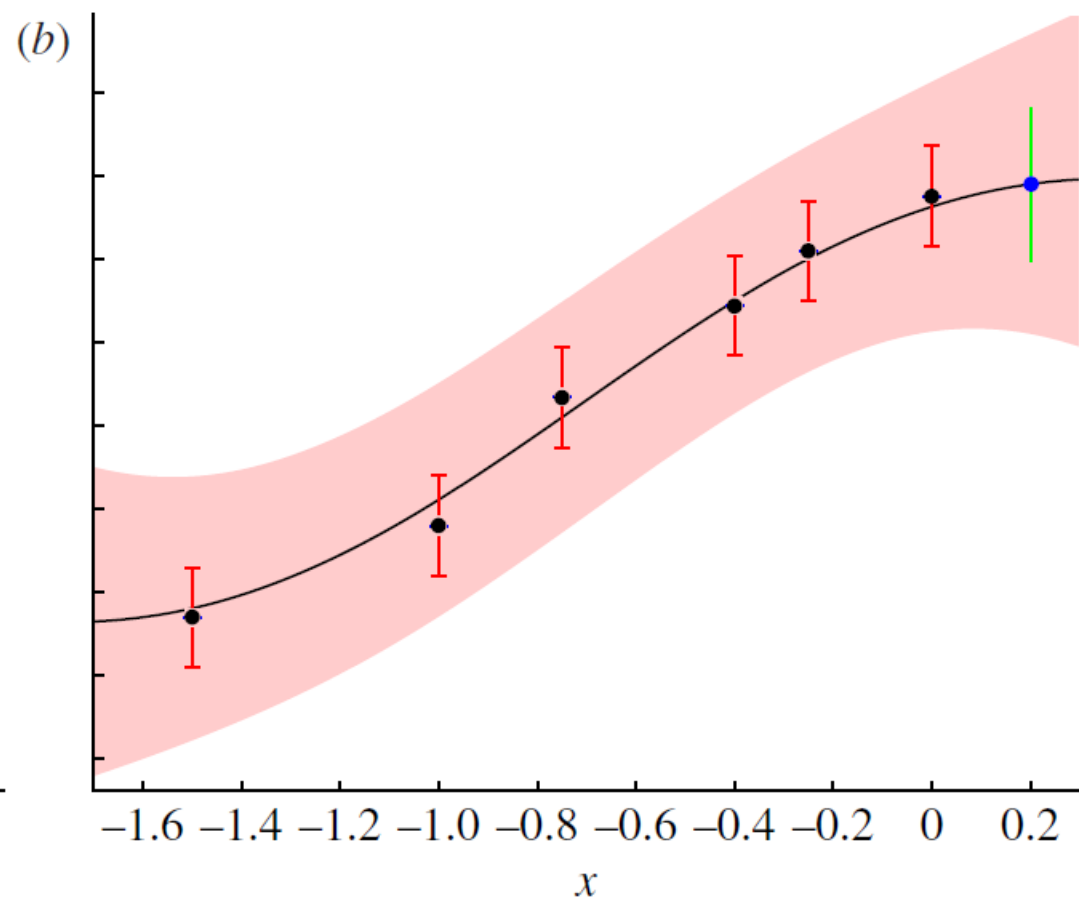
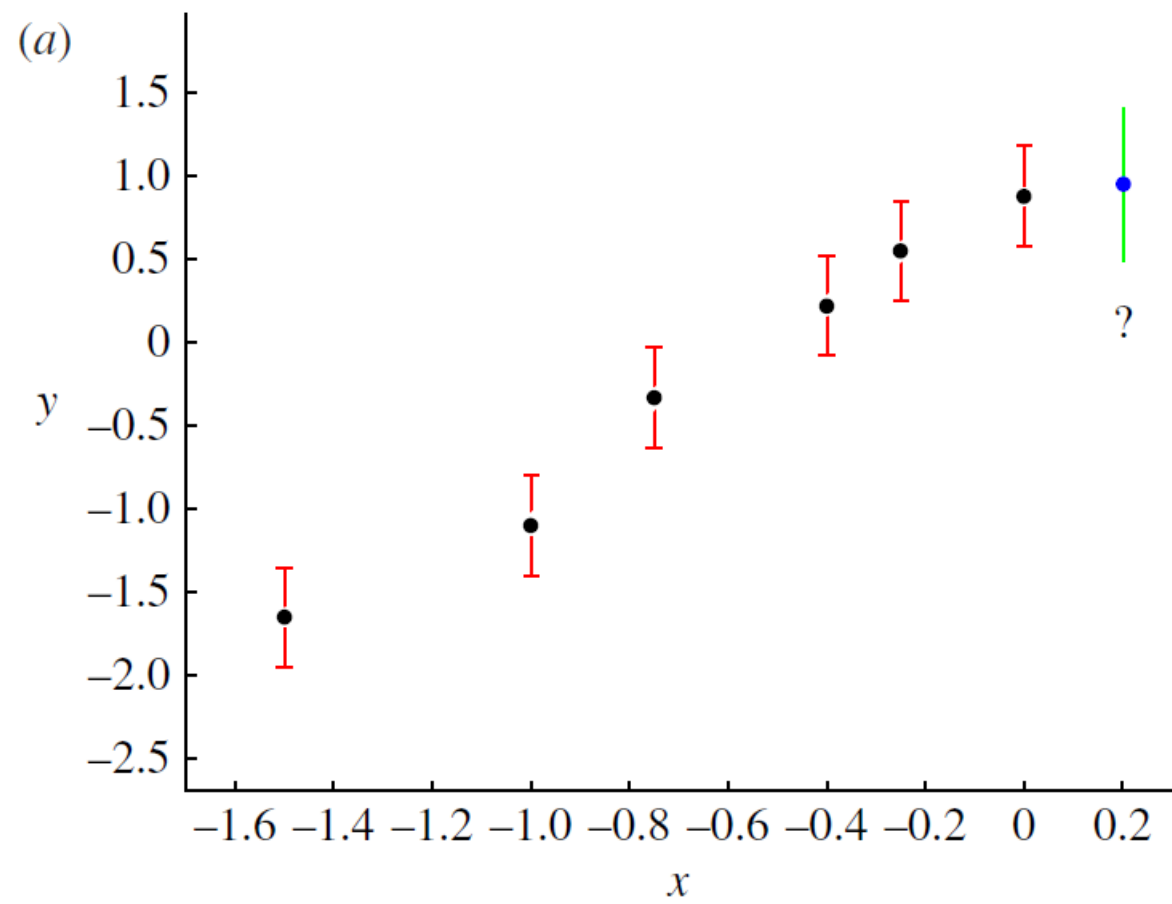
## How does it work?

- We choose priors  $\mu(x), \mu(x_*)$ . For  $x$  could be observations, for  $x_*$  could be some ‘rough’ average of previous observations.
- We choose a series of additive kernels to model our covariance.
- Choosing  $\phi(a, b) = h^2 e^{-\left(\frac{a-b}{\lambda}\right)^2}$  we can start to model the covariance by proximity

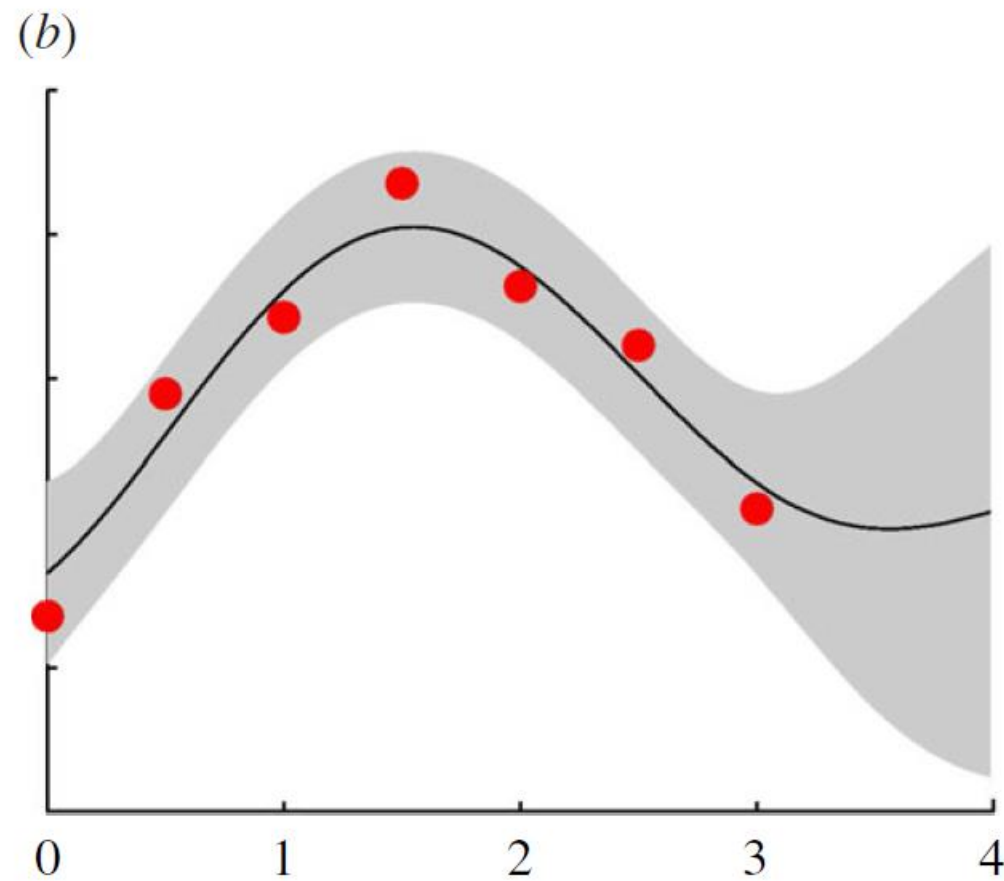
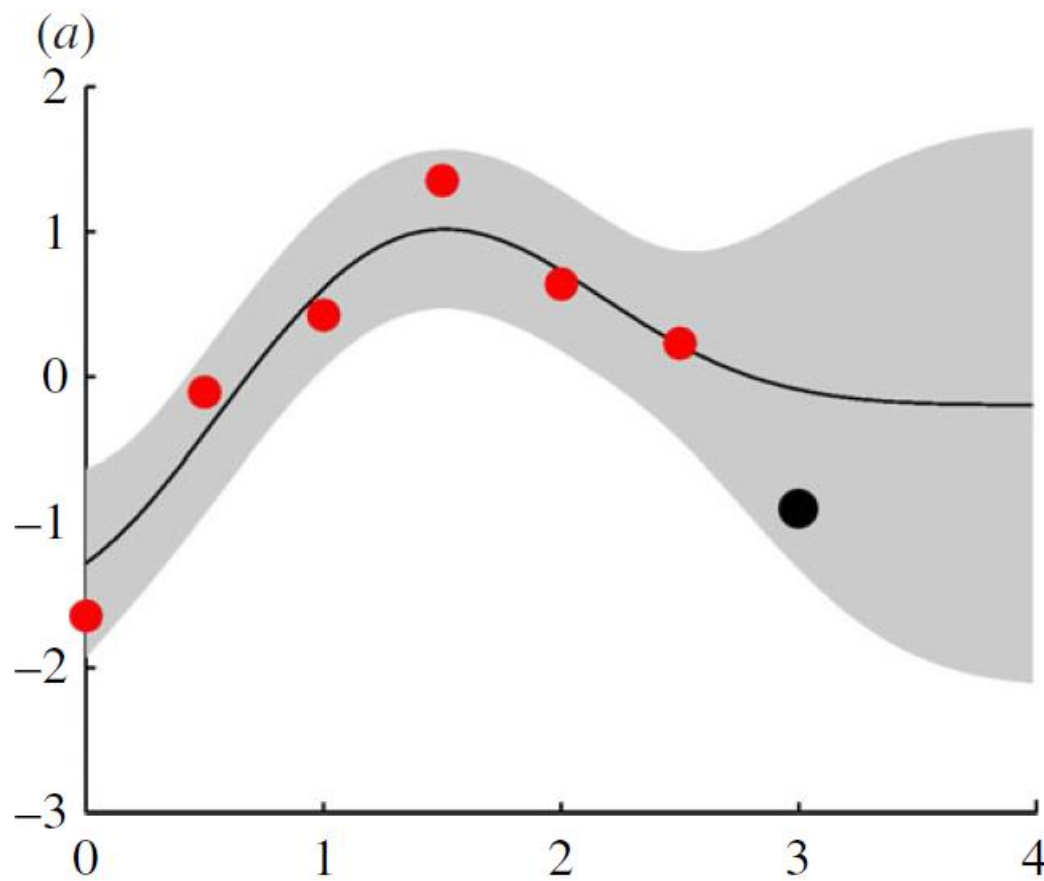
$$\text{cov}(y_i, y_j) = \phi(x_i, x_j), \quad 1 \leq i, j \leq n.$$

- But we need to account for noise! A popular choice is  $\text{cov}(y_i, y_j) = \phi(x_i, x_j) + \sigma^2 \delta_{i,j}$ , where one assumes that  $\sigma^2$  is the noise variance in the problem and that the noise is uncorrelated from sample to sample.
- The hyper parameters  $h, \lambda, \sigma$  are learnt through the process.





# Sequential forecasting



# Adapting to non-stationary noise I

- Suppose  $y$  is a vector of hourly average revenue generated from each visitor
- Each hour there is a different number of visitors  $n_i, i = 1, \dots, k$ .
- Surely, we have more confidence in averages computed from more hourly visitors!
- So we can adapt the noise kernel component from  $\phi_{noise} = \sigma^2 I$  to  $\phi_{noise}(x_i, x_i) = w(x_i)\sigma^2, i = 1, \dots, k$ , where the weight  $w(x_i)$  is a multiple of

$$\frac{n_i^{-1}}{\sum_{j=1}^k n_j^{-1}}$$

## Adapting to non-stationary noise II

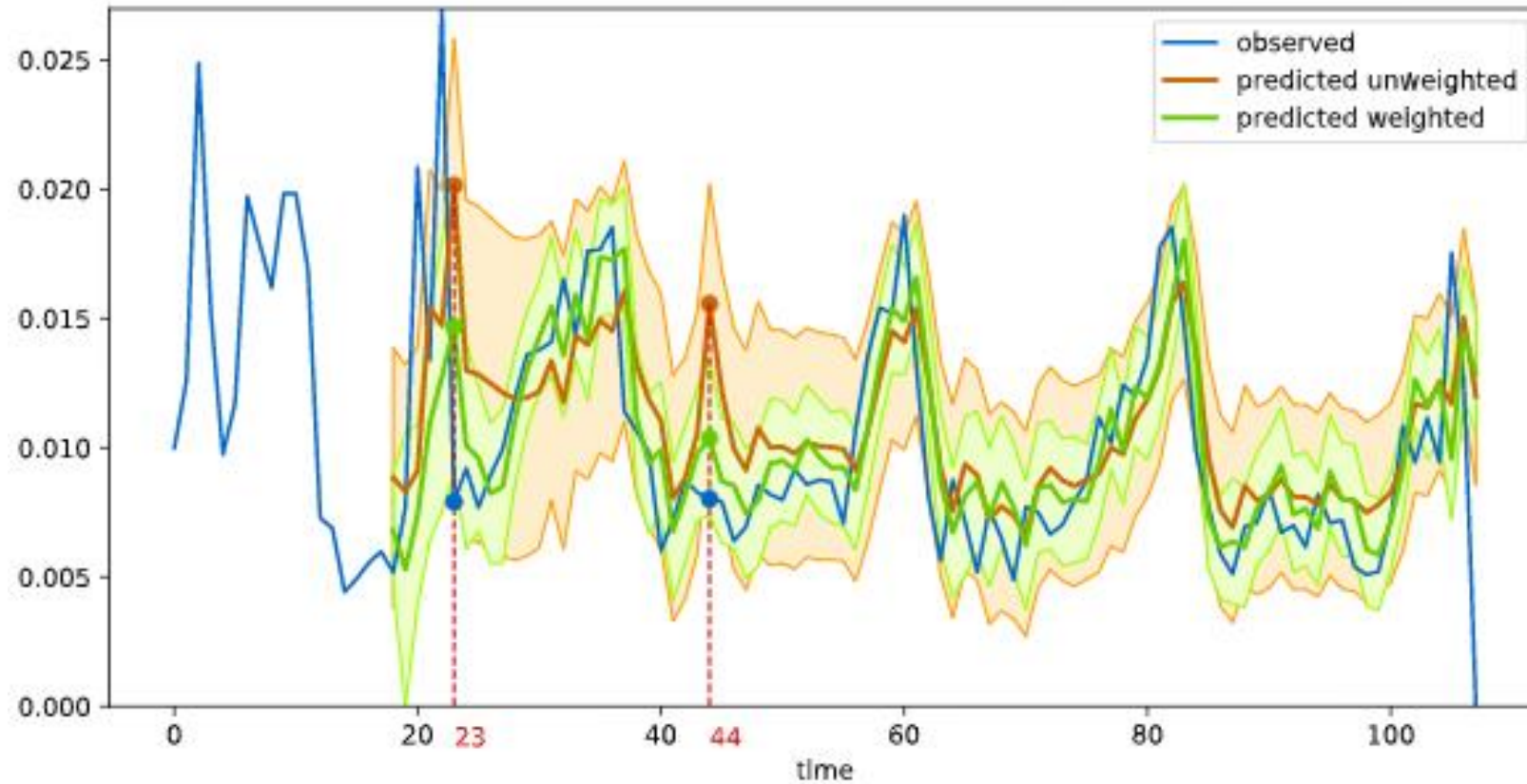
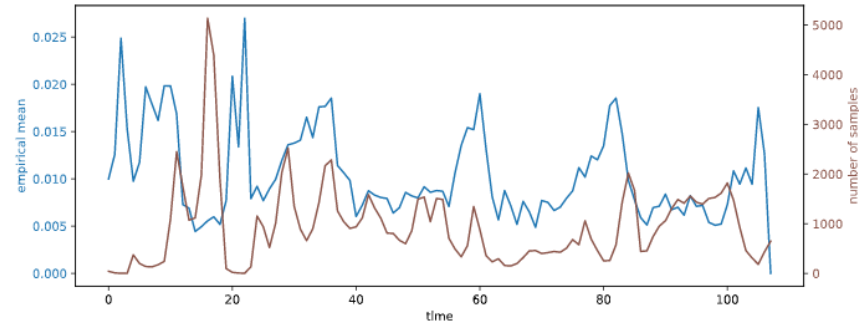
We can then model the noise component at the forecasted points by

$$\phi_{noise}(x_i, x_i) = w_+ \sigma^2, \quad i = k + 1, \dots, n,$$

where  $w_+$  is the harmonic mean

$$w_+ = \left( \frac{1}{k} \sum_{j=1}^k (w(x_j))^{-1} \right)^{-1}$$

# Adapting to non-stationary noise III



# How to model seasonality?

As an example, we can use a quasi-periodical kernel

$$k(a,b) = h^2 e^{-\left(\frac{a-b}{\lambda}\right)^2 - \left(\frac{\sin(\pi(a-b)/T)}{\omega}\right)^2}$$

In sequential mode, all hyper-parameters are optimized as part of the process.


In a nutshell, this has to do with maximizing the likelihood of the parameters with respect to the observed data.

**Generalizations through kernel  
functions...  
input feature vectors...  
more...**

# Code

Tree: 7813f7efb5 [scikit-learn](#) / [sklearn](#) / [gaussian\\_process](#) / [gpr.py](#)

[Find file](#) [Copy path](#)



 [aditya1702](#) MAINT Replace absolute imports with relative imports (#13653)

301076e on Apr 17

16 contributors



477 lines (401 sloc) | 20.4 KB

[Raw](#) [Blame](#) [History](#)   

```
1  """Gaussian processes regression. """
2
3  # Authors: Jan Hendrik Metzen <jhm@informatik.uni-bremen.de>
4  #
5  # License: BSD 3 clause
6
7  import warnings
8  from operator import itemgetter
9
10 import numpy as np
11 from scipy.linalg import cholesky, cho_solve, solve_triangular
12 from scipy.optimize import fmin_l_bfgs_b
13
14 from ..base import BaseEstimator, RegressorMixin, clone
15 from ..base import MultiOutputMixin
16 from .kernels import RBF, ConstantKernel as C
17 from ..utils import check_random_state
18 from ..utils.validation import check_X_y, check_array
19 from ..exceptions import ConvergenceWarning
20
21
22 class GaussianProcessRegressor(BaseEstimator, RegressorMixin,
23                               MultiOutputMixin):
24     """Gaussian process regression (GPR).
```