Gaussian Process for Noisy Time Series forcasting

Multivariate Normal Distribution

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

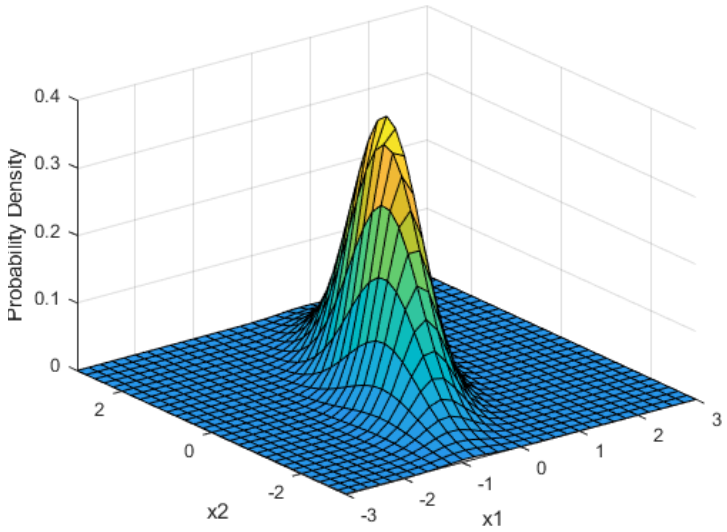
$$y \sim \mathcal{N}(\mu, \Sigma) .$$

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

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

Density function

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



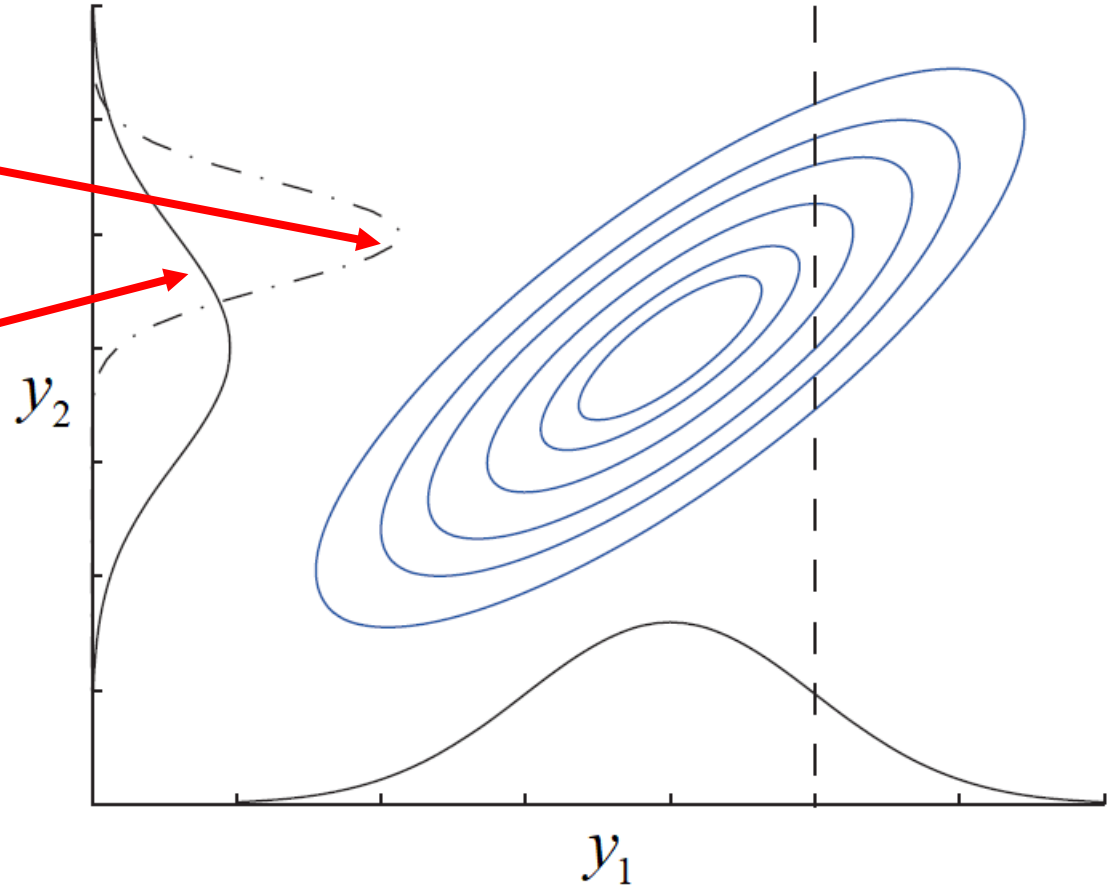
Marginal/Conditional distributions

Conditional

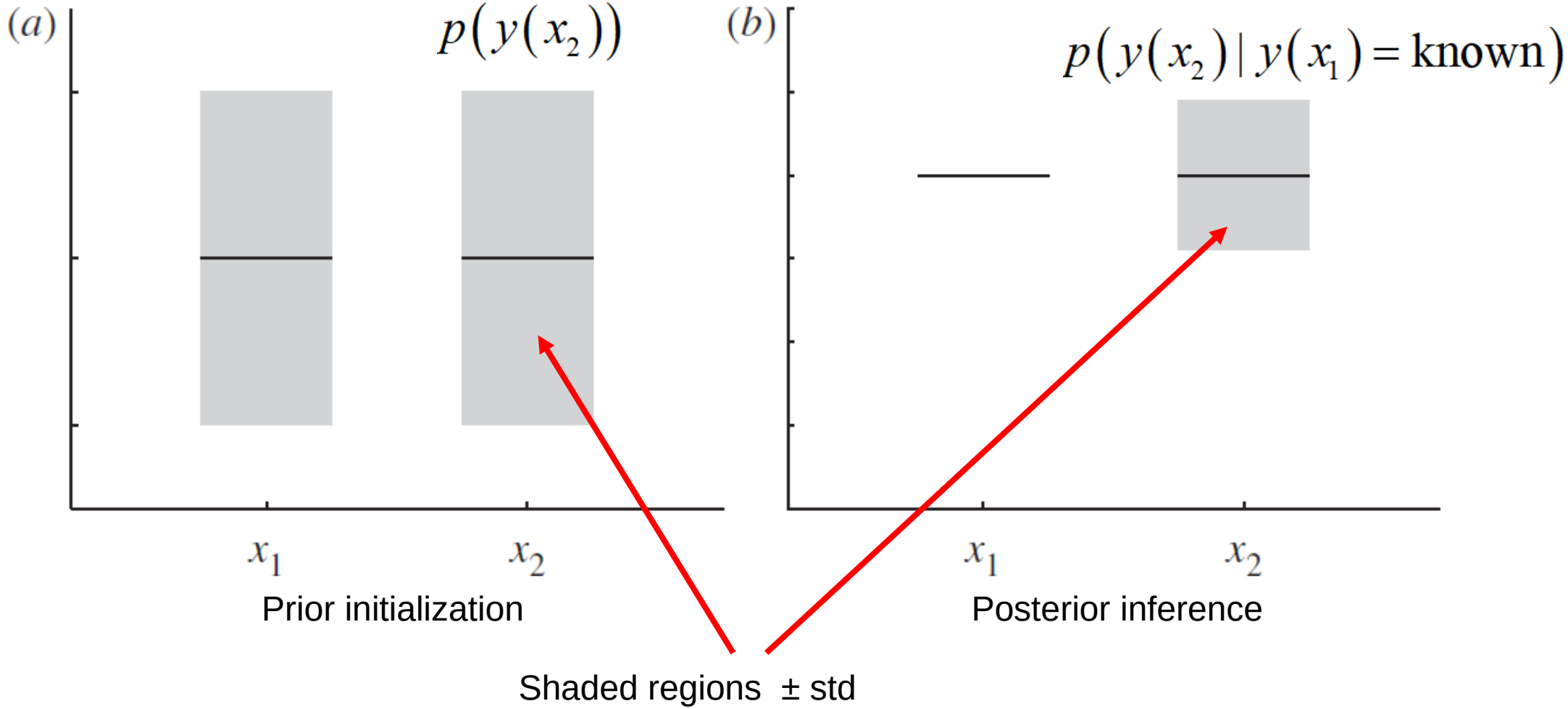
$p(y_2 \mid y_1 = \text{known})$

Marginal

$p(y_2)$



GP – same example as a time-series

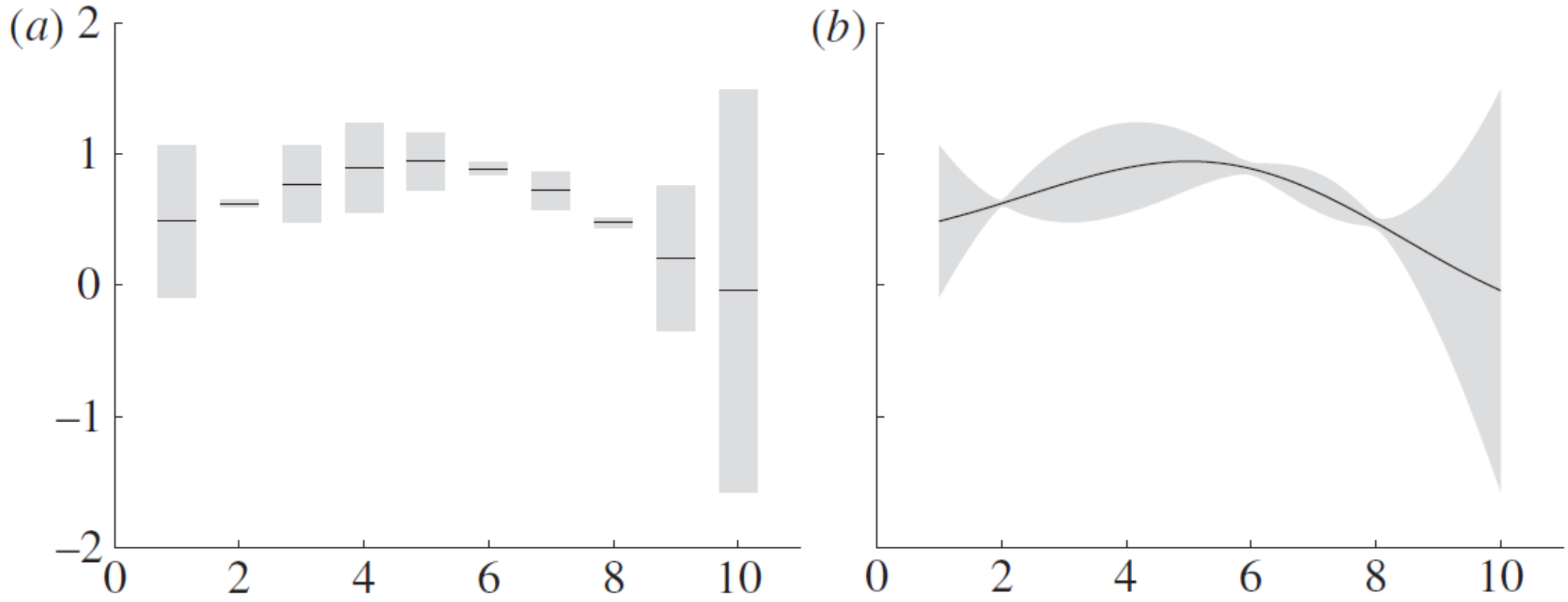


Prior initialization

Posterior inference

Shaded regions ± std

Qualitative view with more time samples…



Posterior after observations at x=2,6,8

How does it work?

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

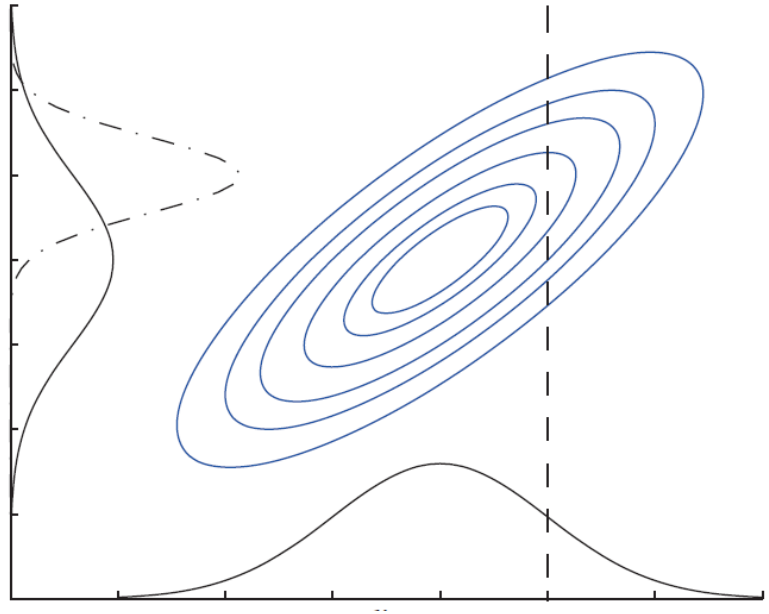
We assume the normal distribution

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

and then extract the conditional (posterior) distribution

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

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



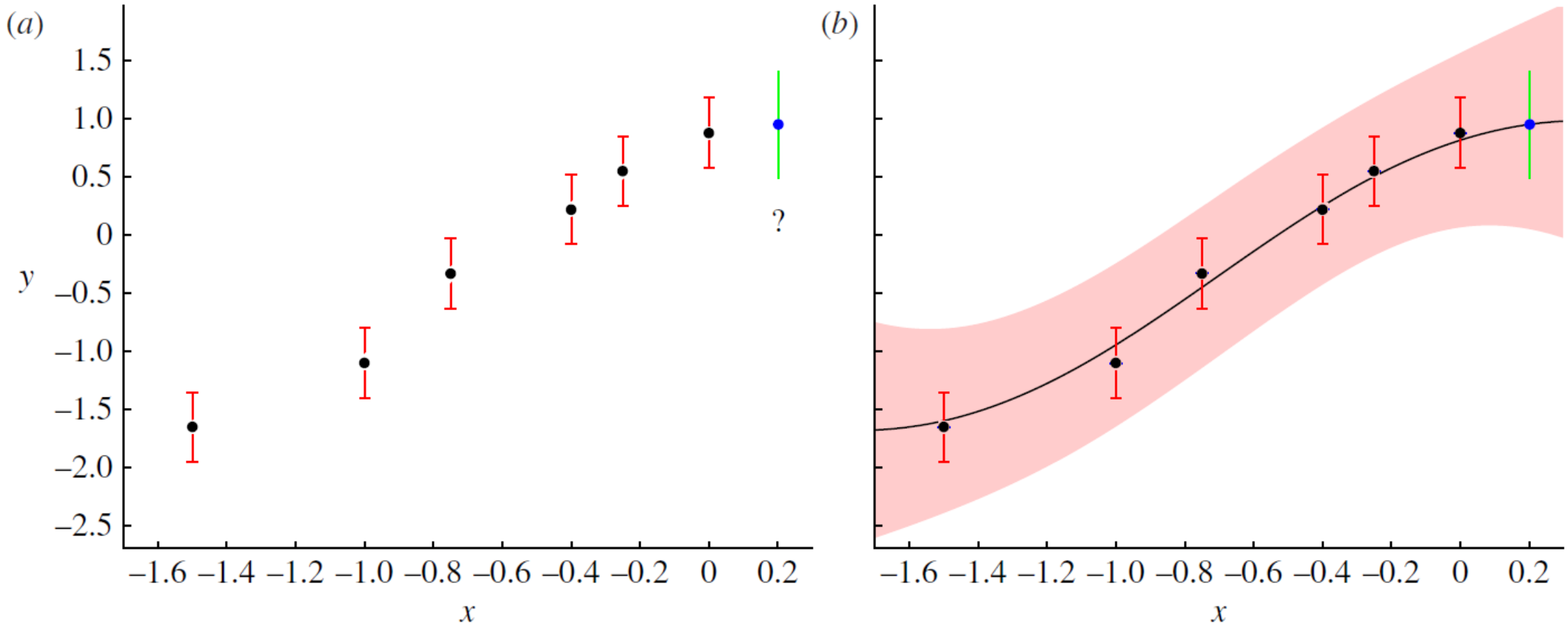
How does it work?

- We choose priors $\mu(x), \mu(x_*)$.
- We choose a series of additive kernels to model our covariance.
- For example, by choosing $\phi(a,b) = h^2 e^{-\left(\frac{a-b}{\lambda}\right)^2}$ we can start to model the covariance by <u>proximity</u>

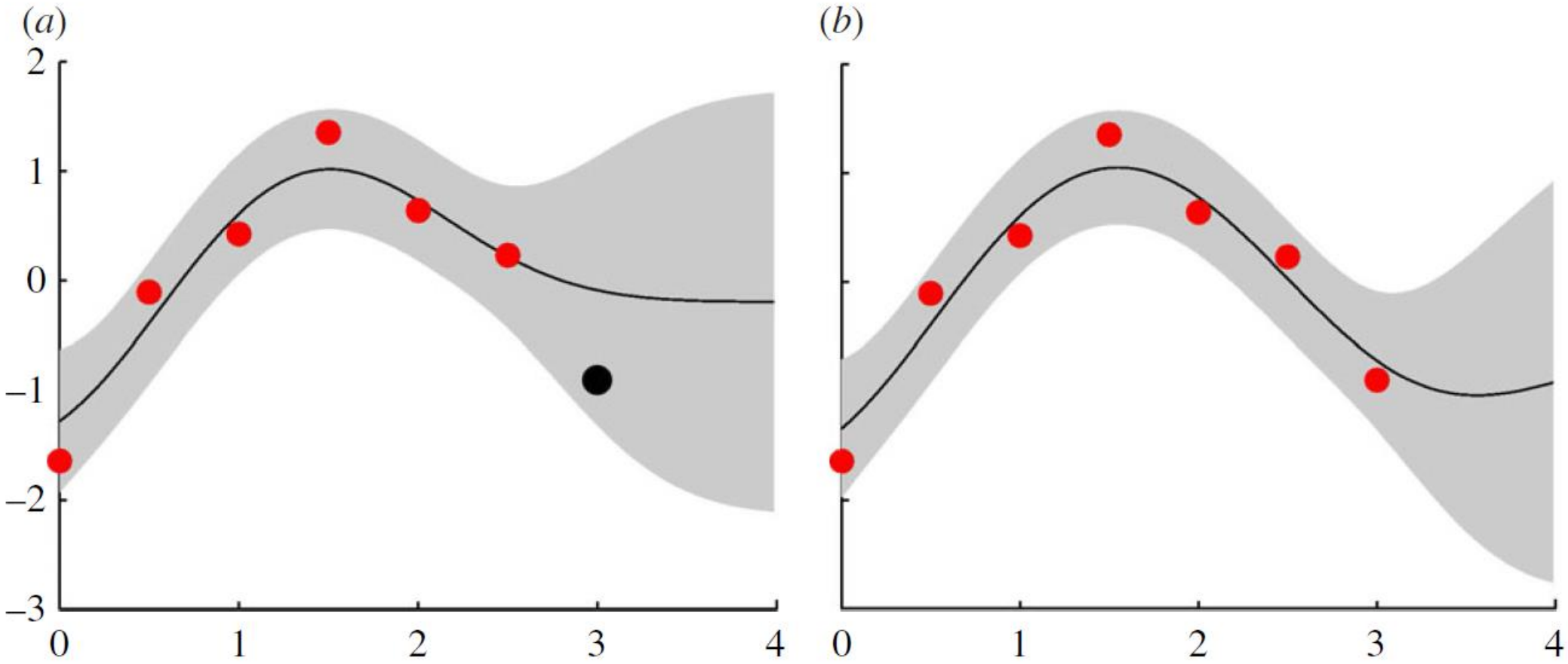
$$\Sigma(x,x) = \phi(x,x) = \left(\phi\left(x_i, x_j\right)\right)_{i,j=1}^{k}.$$

- But we need to account for noise! A popular choice is $\Sigma(x,x) = \phi(x,x) + \sigma^2 I$, where one assumes that σ^2 is the noise variance in the problem and that the noise is uncorrelated from sample to sample.

- The hyper parameters h, λ, σ are learnt through the process.


(a)
1.5
1.0
0.5
0
–0.5
–1.0
–1.5
–2.0
–2.5
y
?
–1.6 –1.4 –1.2 –1.0 –0.8 –0.6 –0.4 –0.2 0 0.2
x
(b)
–1.6 –1.4 –1.2 –1.0 –0.8 –0.6 –0.4 –0.2 0 0.2
x

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, $i = 1, \ldots, k$.
- 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 $\phi_{noise}(x_i, x_i) = w(x_i)\sigma^2$, $i = 1, \ldots, k$, 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

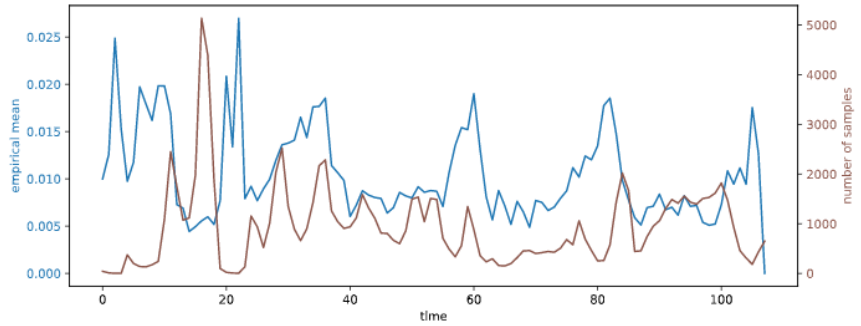
We can then model the noise component at the forecasted points by

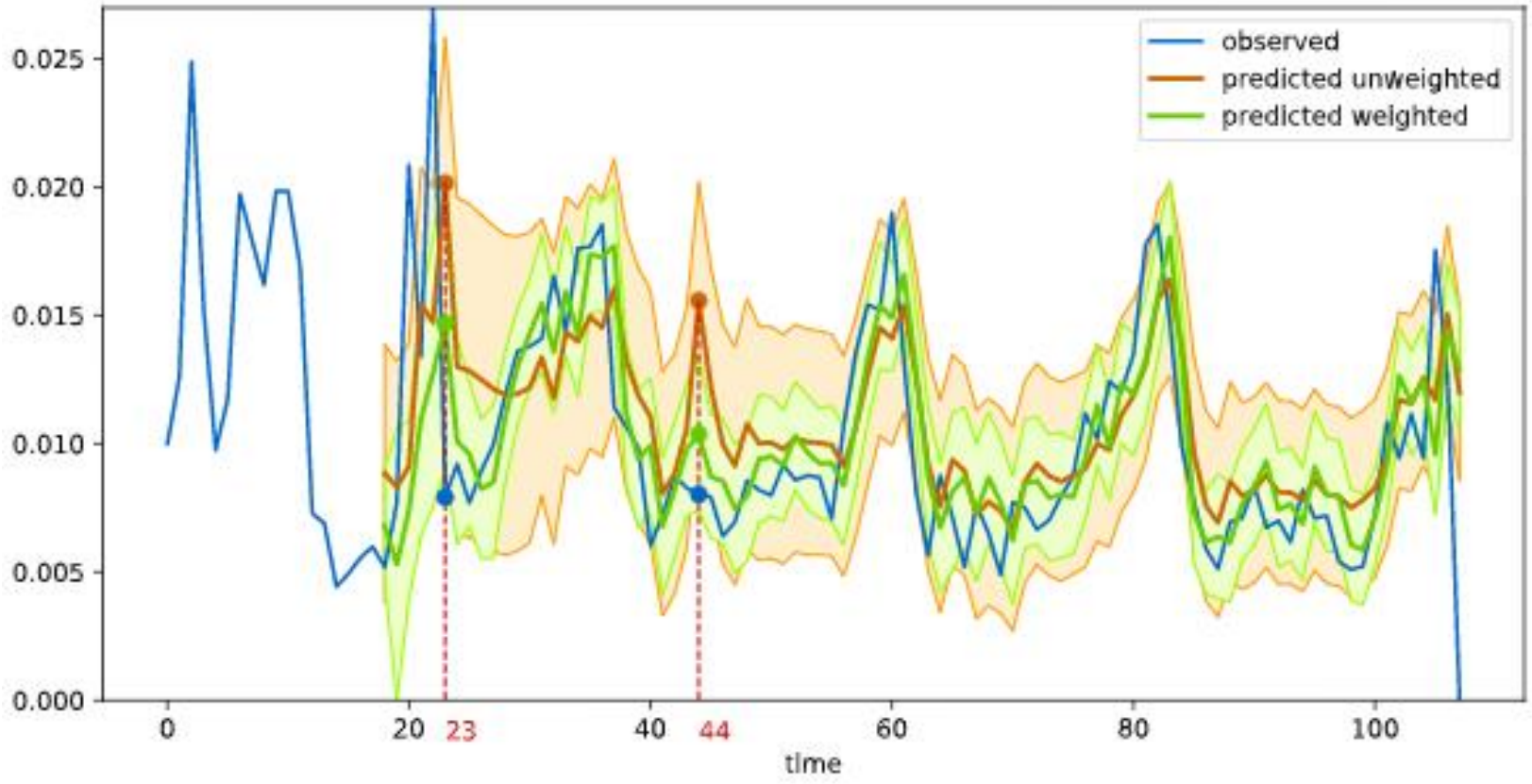
$$\phi_{noise}(x_i, x_i) = w_+ \sigma^2,\ i = k+1, \ldots, n,$$

where w_+ is the harmonic mean

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

Adapting to non-stationary noise III





How to model seasonality?

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

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

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

```
"""Gaussian processes regression. """

# Authors: Jan Hendrik Metzen <jhm@informatik.uni-bremen.de>
#
# License: BSD 3 clause

import warnings
from operator import itemgetter

import numpy as np
from scipy.linalg import cholesky, cho_solve, solve_triangular
from scipy.optimize import fmin_l_bfgs_b

from ..base import BaseEstimator, RegressorMixin, clone
from ..base import MultiOutputMixin
from .kernels import RBF, ConstantKernel as C
from ..utils import check_random_state
from ..utils.validation import check_X_y, check_array
from ..exceptions import ConvergenceWarning


class GaussianProcessRegressor(BaseEstimator, RegressorMixin,
                               MultiOutputMixin):
    """Gaussian process regression (GPR).
```