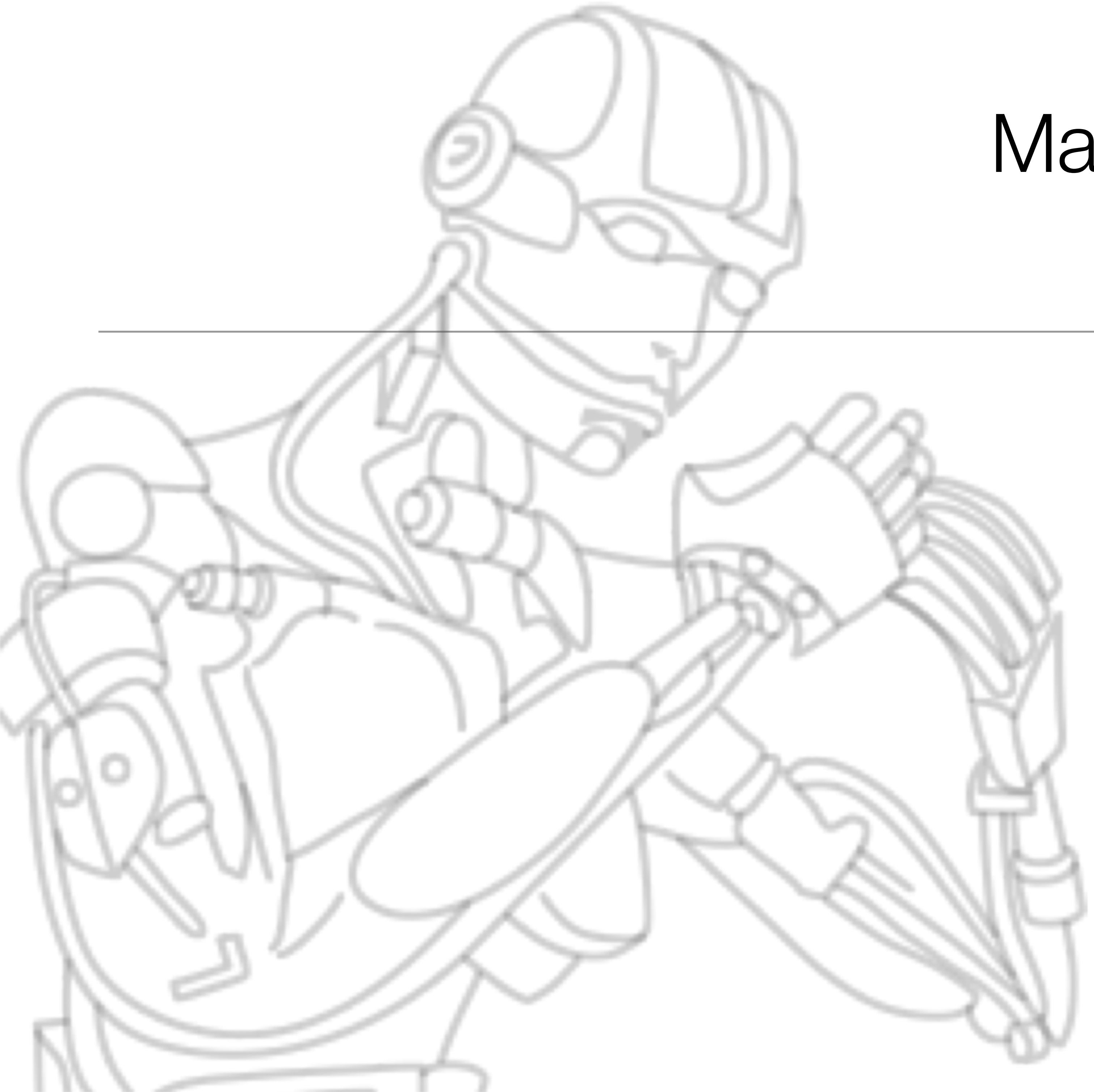


Machine Learning – Part2 in a few Minutes

Jan Peters
Gerhard Neumann

Purpose of this Lecture

- How can we define such features for general machine learning problems?
- Can we avoid or automate the feature specification?
- Familiarize you with non-parametric models

Content of this Lecture

Constructing Basis Functions

- Radial Basis Function Networks

Non-Parametric Approaches

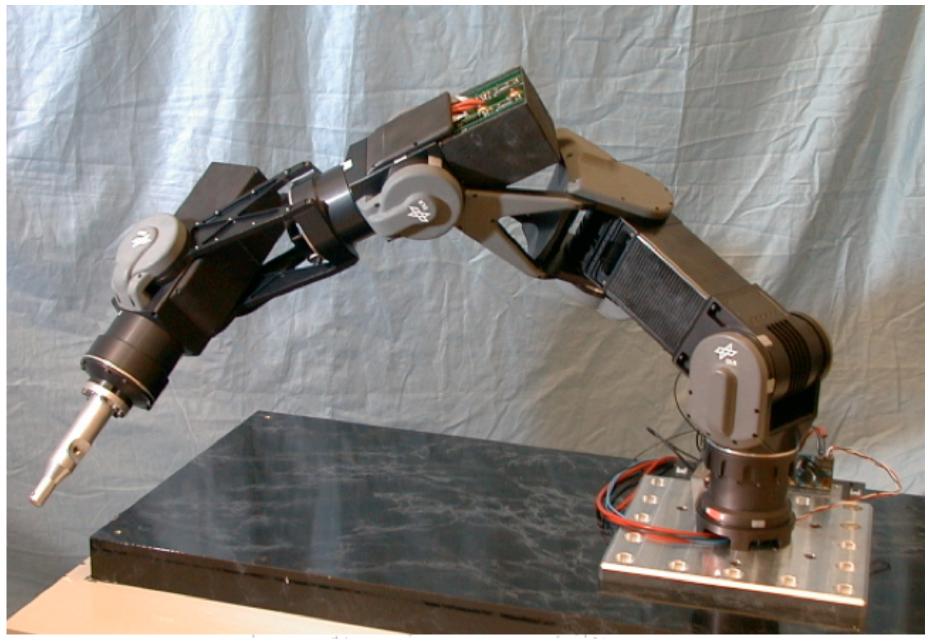
- Locally Weighted Regression
- Kernel Methods

What we did so far…

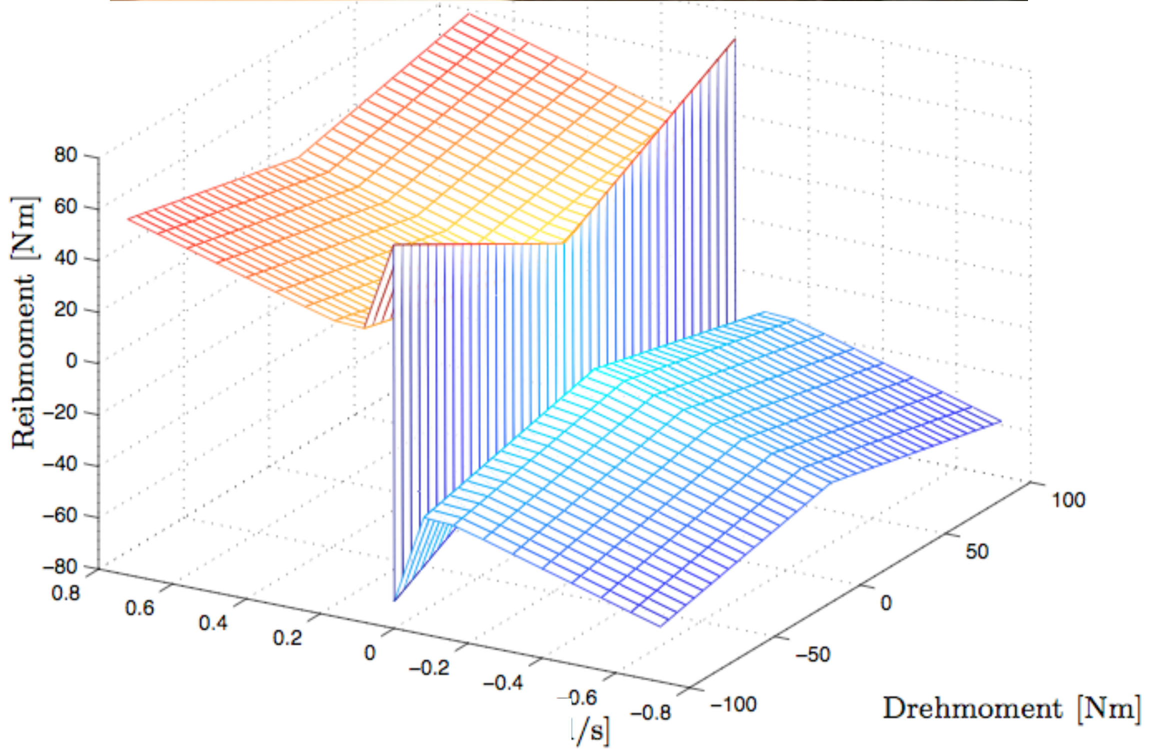
- Models that are linear in the parameters: $y = \boldsymbol{\phi}(\boldsymbol{x})^T \boldsymbol{\theta}$
- **Parameter Estimation:** Frequentist vs. Bayesian
    - **Least Squares** ~ Maximum Likelihood estimation (ML)
    - **Ridge Regression** ~ Maximum a Posteriori estimation (MAP)
- **Bayesian Regression** integrates out the parameters when predicting
    - State dependent uncertainty

However, for most problems good features are not easy to find

What to do when you don’t know the features?



- In most real applications, we know **some** good features.
- However, we almost certainly **don’t know all** features we need.
- ***Example***: Rigid body dynamics
  - Friction has no good features
  - Unknown dynamics causes huge problems (requires more state variables).
- There may also be way too many features!



Hand-crafted features are almost never enough...

Can we avoid having to find good features?

Yes, we can!

We need to find machine learning approaches that **generate the features automatically** from data.

- **Type 1**: *Automatic Basis Function Construction* constructs basis functions from data.
- **Type 2**: *Non-Parametric Regression* look at data locally and interpolate with similar data.
- **Type 3**: *Kernel Regression* finds the features implicitly by going into *function space* using a *kernel*

Type 1: Construct Basis Functions from Data

Classical idea behind “neural networks”

- Multi-Layer Perceptrons (see ML 1)
- Radial Basis Function Networks

Radial Basis Function Network

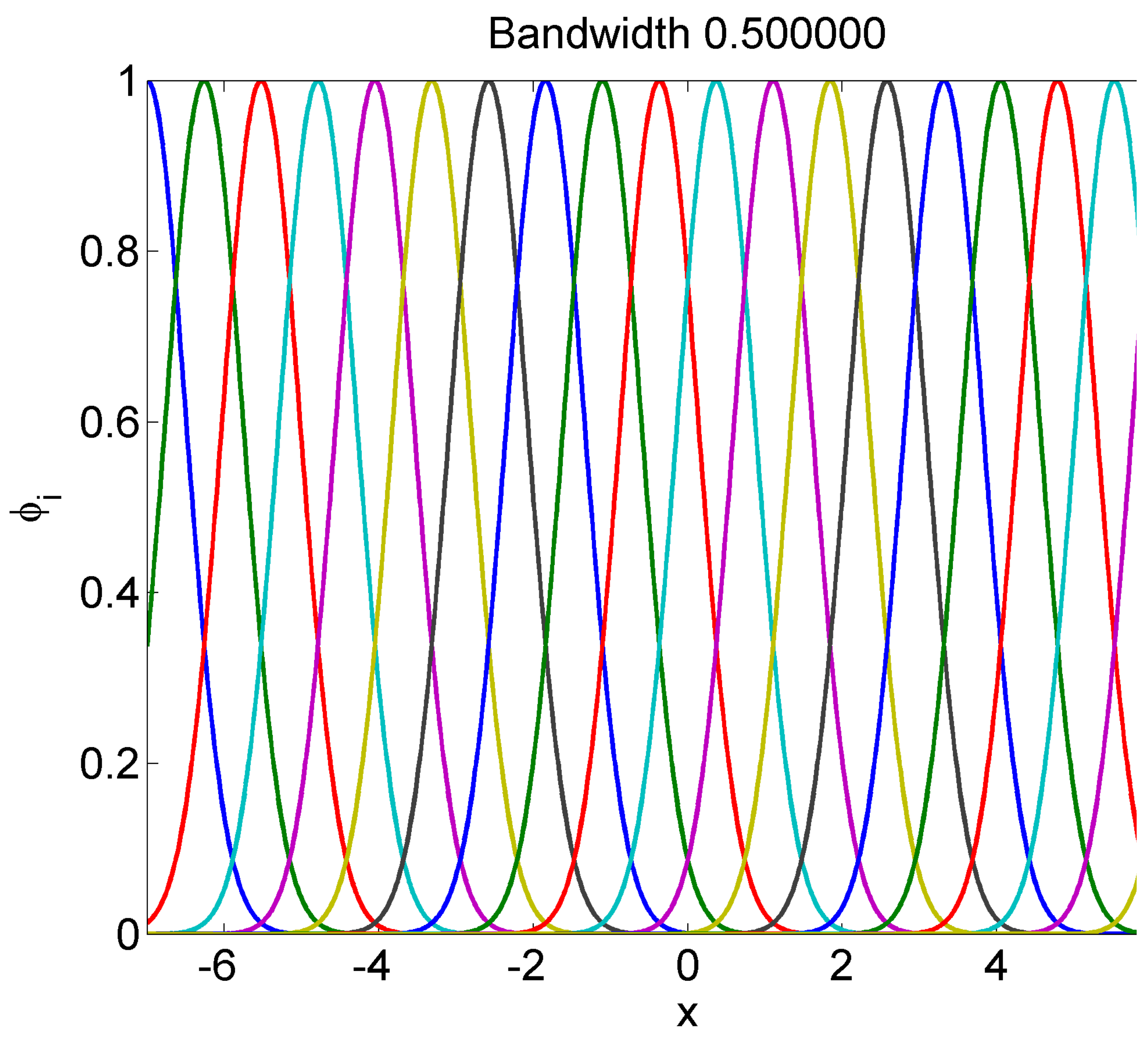
Assume a smoothness prior and obtain the cost function

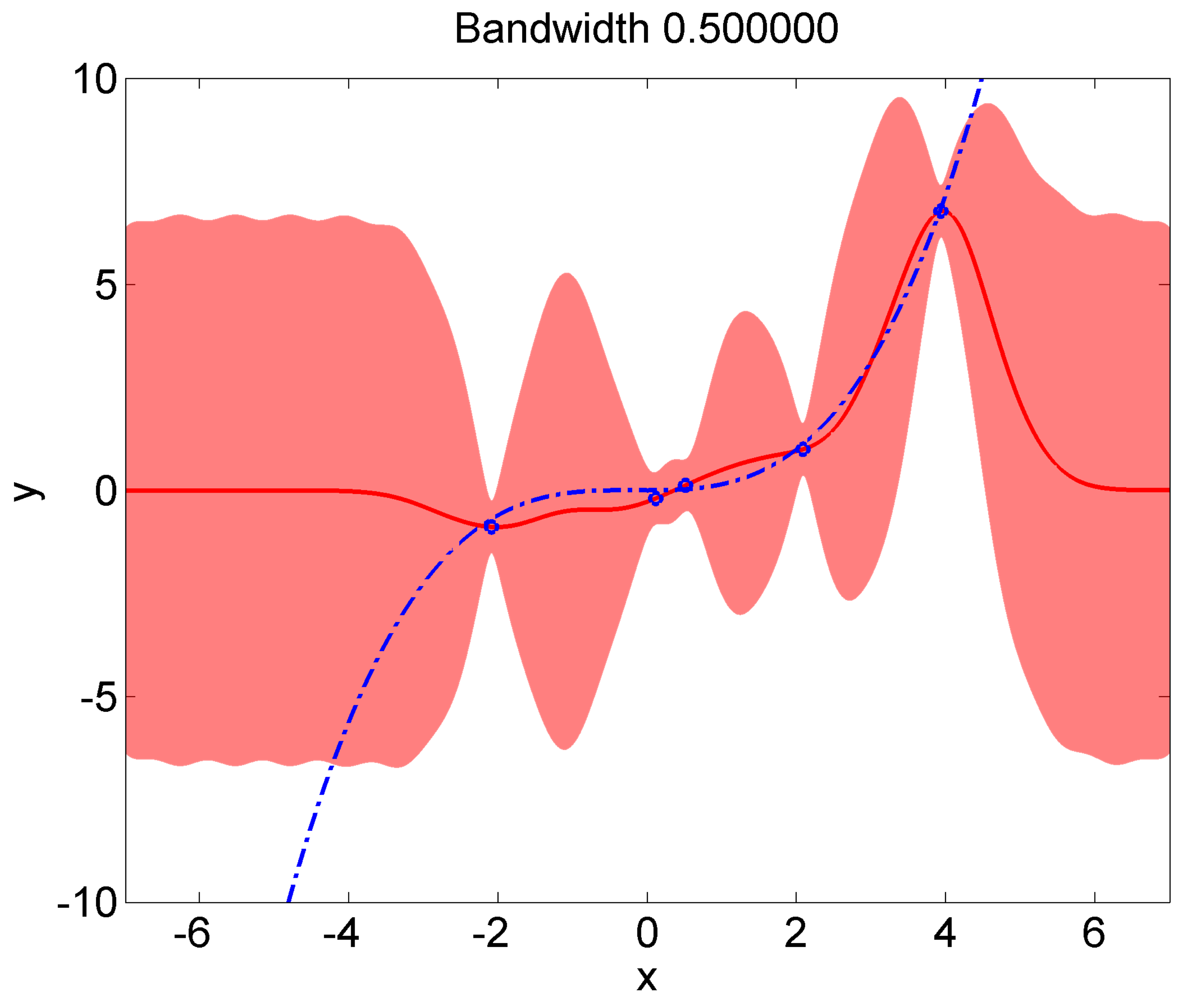
$$J = \frac{1}{2}\sum_{i=1}^{N}\left[(y_i - \mathbf{f}_\theta(\mathbf{x}_i))^2 + \left\|\frac{d^2}{dx^2}\mathbf{f}_\theta(\mathbf{x}_i)\right\|^2\right]$$

This prior yields radial basis functions as features:

$$\begin{aligned} f_{\boldsymbol{\theta}}(\boldsymbol{x}) &= \sum_i \theta_i \exp\left(\frac{-||\boldsymbol{x} - \boldsymbol{\mu}_i||^2}{2l^2}\right) \\ &= \boldsymbol{\phi}(\boldsymbol{x})^T\boldsymbol{\theta}, \quad \text{with } \phi_i(\boldsymbol{x}) = \exp\left(\frac{-||\boldsymbol{x} - \boldsymbol{\mu}_i||^2}{2l^2}\right) \end{aligned}$$

Example: Radial Basis Function Features





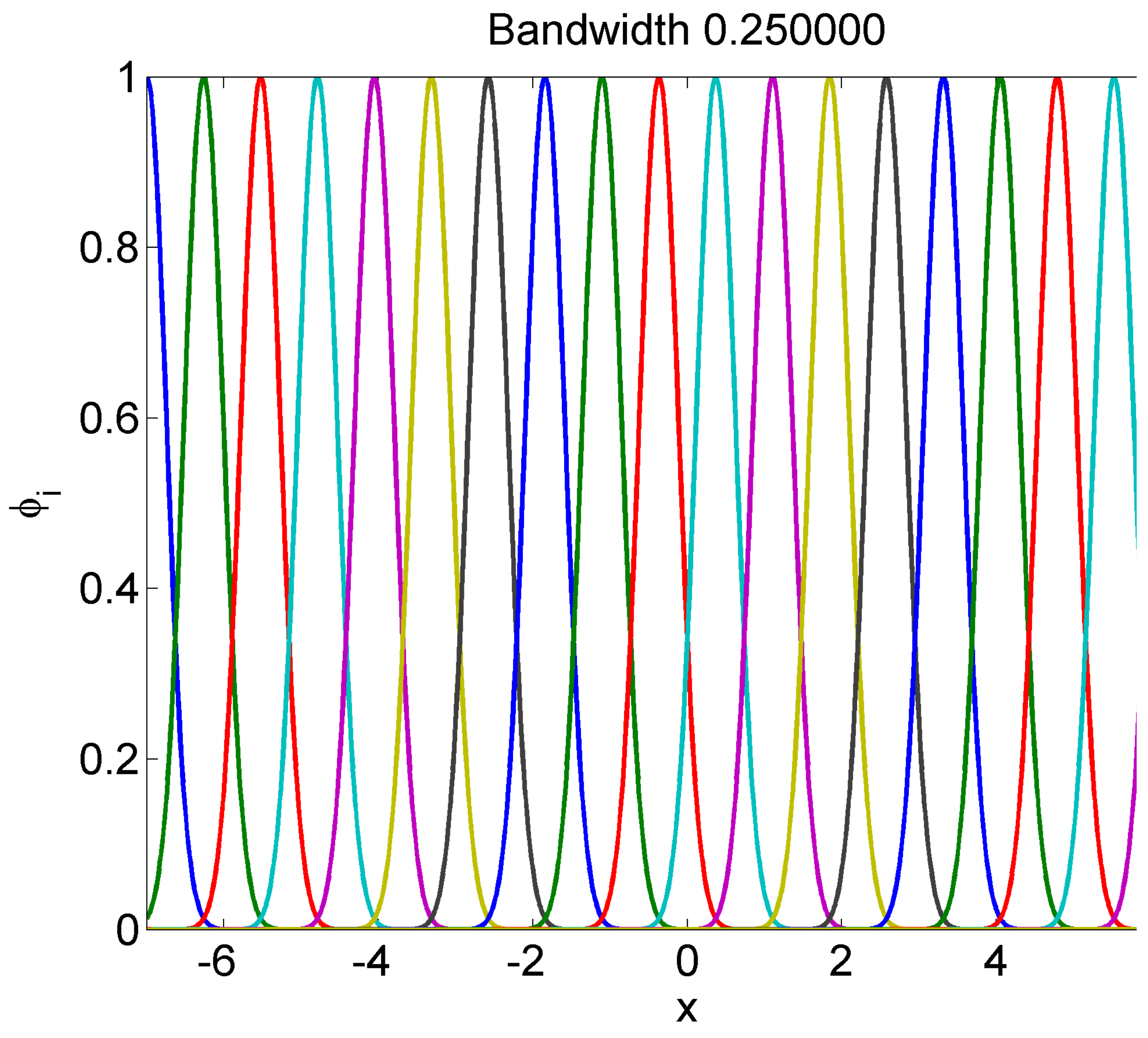
Radial Basis Functions Hyperparameters

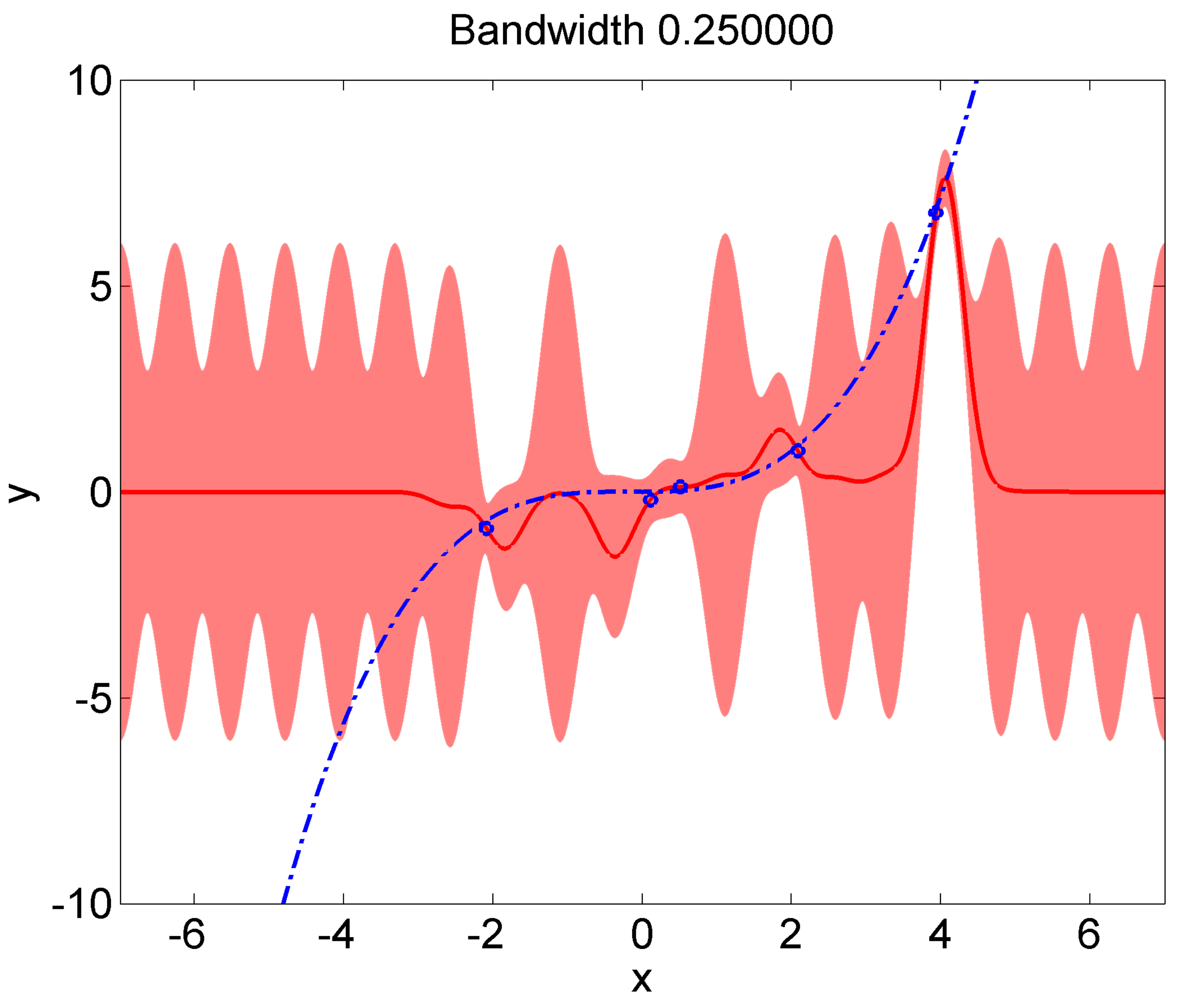
Let's look again at

$$\phi_i(\boldsymbol{x}) = \exp\left(\frac{-||\boldsymbol{x} - \boldsymbol{\mu}_i||^2}{2l^2}\right)$$

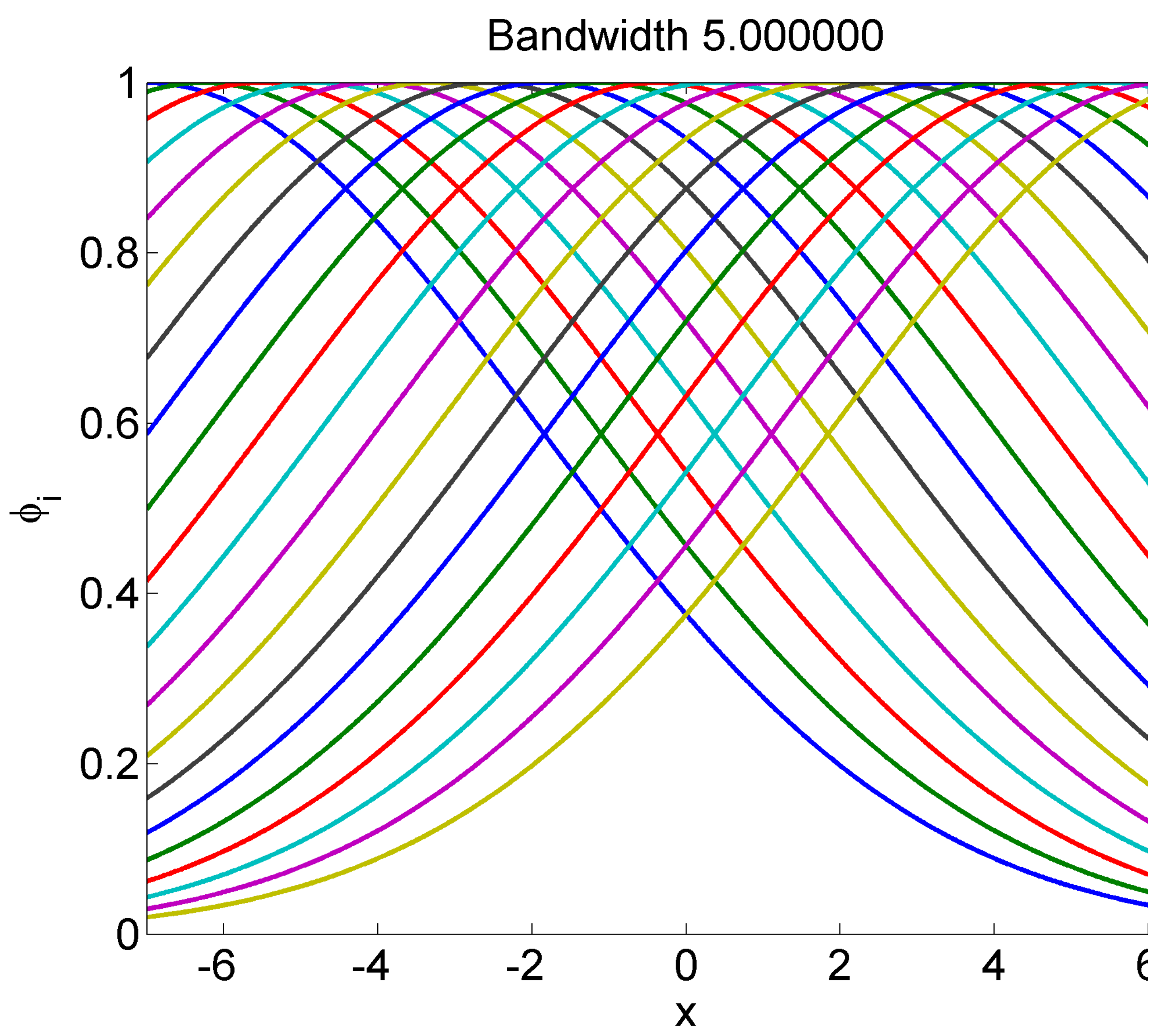
- How do I find the width *l* of the basis functions or the centers $\boldsymbol{\mu}_i$?
- Linear regression? Nope: not linear in *l or mu* !
- We need to optimize this width on the training set
- We can do that by gradient descent: Write down a loss function, take the derivative w.r.t. *l,* and use an algorithm for non-convex optimization

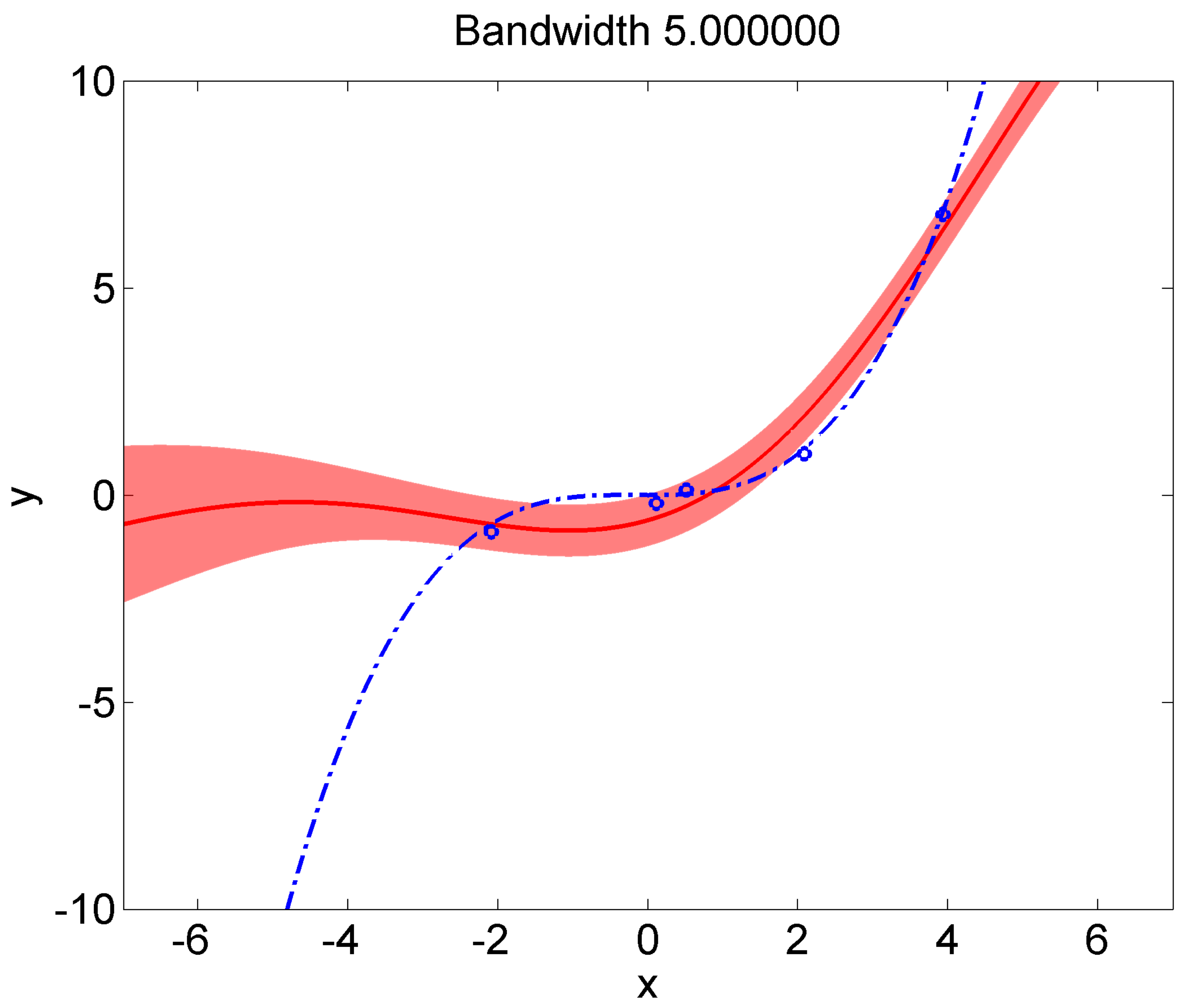
Example: Bandwidth too small





Example: Bandwidth too large





Content of this Lecture

Constructing Basis Functions

- ➡ Radial Basis Function Networks

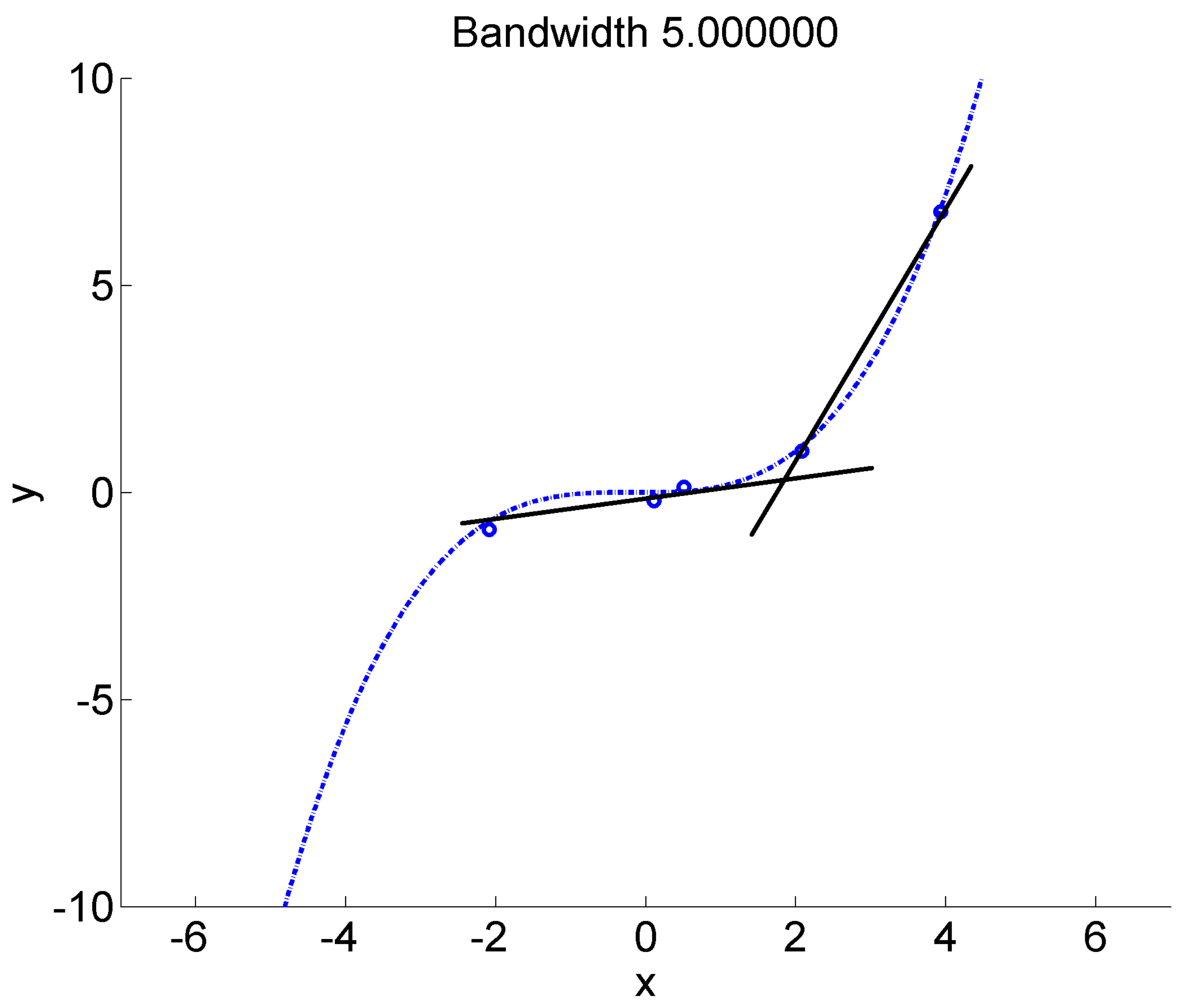
Non-Parametric Approaches

- ➡ Locally Weighted Regression
- ➡ Kernel Methods

Type 2: Non-Parametric Regression

- If you choose to have one feature/basis function per sample, you have a "**non-parametric method**" $\Rightarrow$ Don't need to select the number of bases
- Non-parametric means
  - infinitely many parameters not no parameters
  - expressiveness of the model depends on the number of data points
  - No predetermined "parametric" form necessary
    - (e.g., "5th-degree polynomial")
- One of them is **locally-weighted linear regression**...

Example: Locally Linear Solutions



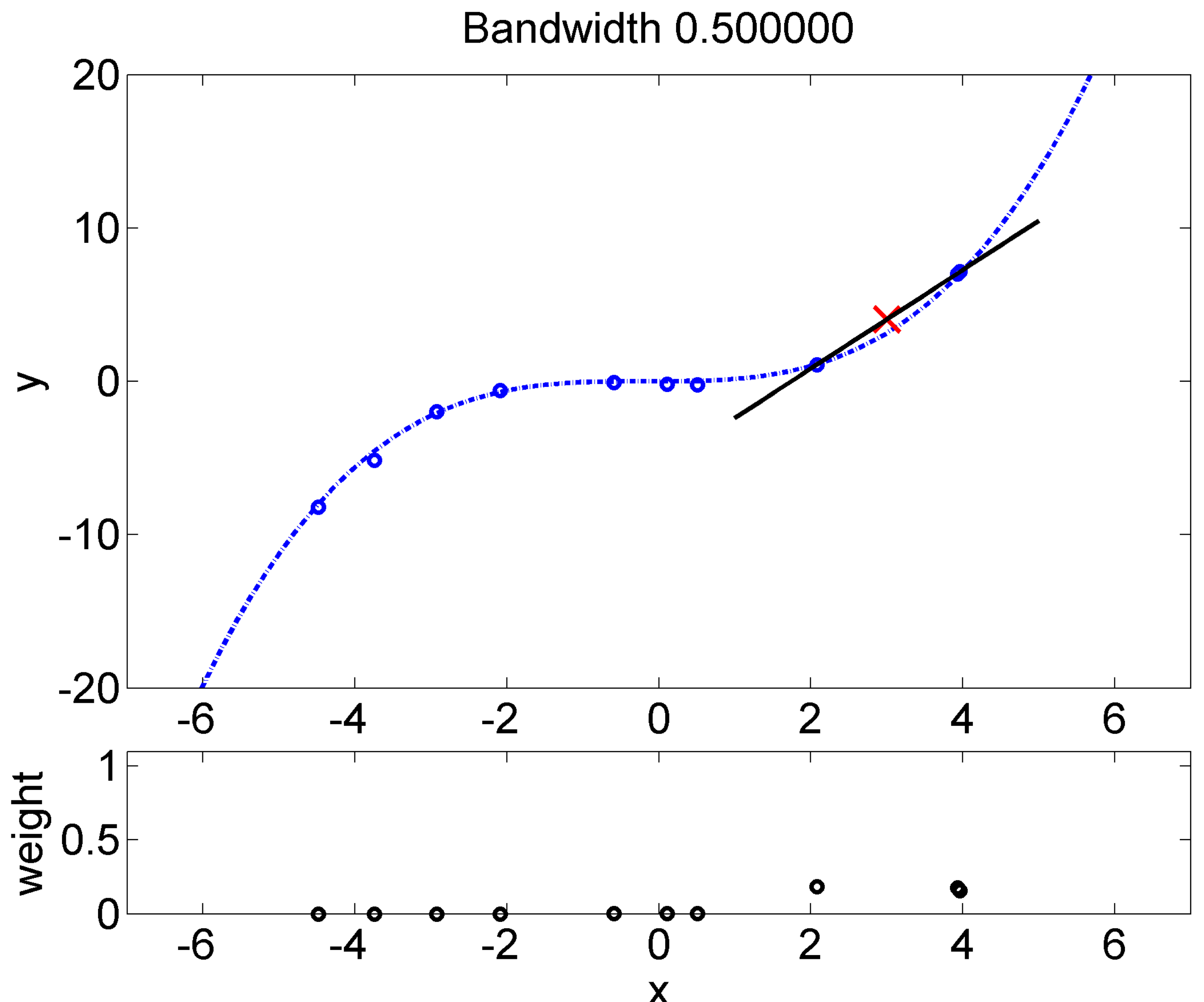
Locally all data is linear!

Locally all data is linear ...

so why don't we take the neighboring data points to predict the solution?

- Use higher importance or weighting of neighboring data points
- For each query point $\boldsymbol{x}$, weight training points $\boldsymbol{x}_i$ by

$$w_i(\boldsymbol{x}) = \exp\left(-\frac{||\boldsymbol{x}-\boldsymbol{x}_i||^2}{2l^2}\right) \ldots \text{squarred exponential weighting}$$



Weighted Linear Regression

Weighted cost function

$$J = \frac{1}{2}\sum_{i=1}^{N} w_i(\mathbf{x})(y_i - \mathbf{f}_\theta(\mathbf{x}_i))^2, \qquad w_i(\boldsymbol{x}) = \exp\left(-\frac{||\boldsymbol{x} - \boldsymbol{x}_i||^2}{2l^2}\right)$$

The function is linear in x

$$f_{\boldsymbol{\theta}}(\boldsymbol{x}) = \boldsymbol{\theta}^T \begin{bmatrix} 1 \\ \boldsymbol{x} \end{bmatrix} = \boldsymbol{\theta}^T \tilde{\boldsymbol{x}}$$

In matrix form with $W = \mathrm{diag}(w_1, w_2, w_3, \ldots, w_n)$:

$$J = \frac{1}{2}(\tilde{\boldsymbol{X}}\boldsymbol{\theta} - \boldsymbol{y})^T \boldsymbol{W} (\tilde{\boldsymbol{X}}\boldsymbol{\theta} - \boldsymbol{y})$$

Weighted Linear Regression

The solution to this problem: weighted pseudo inverse

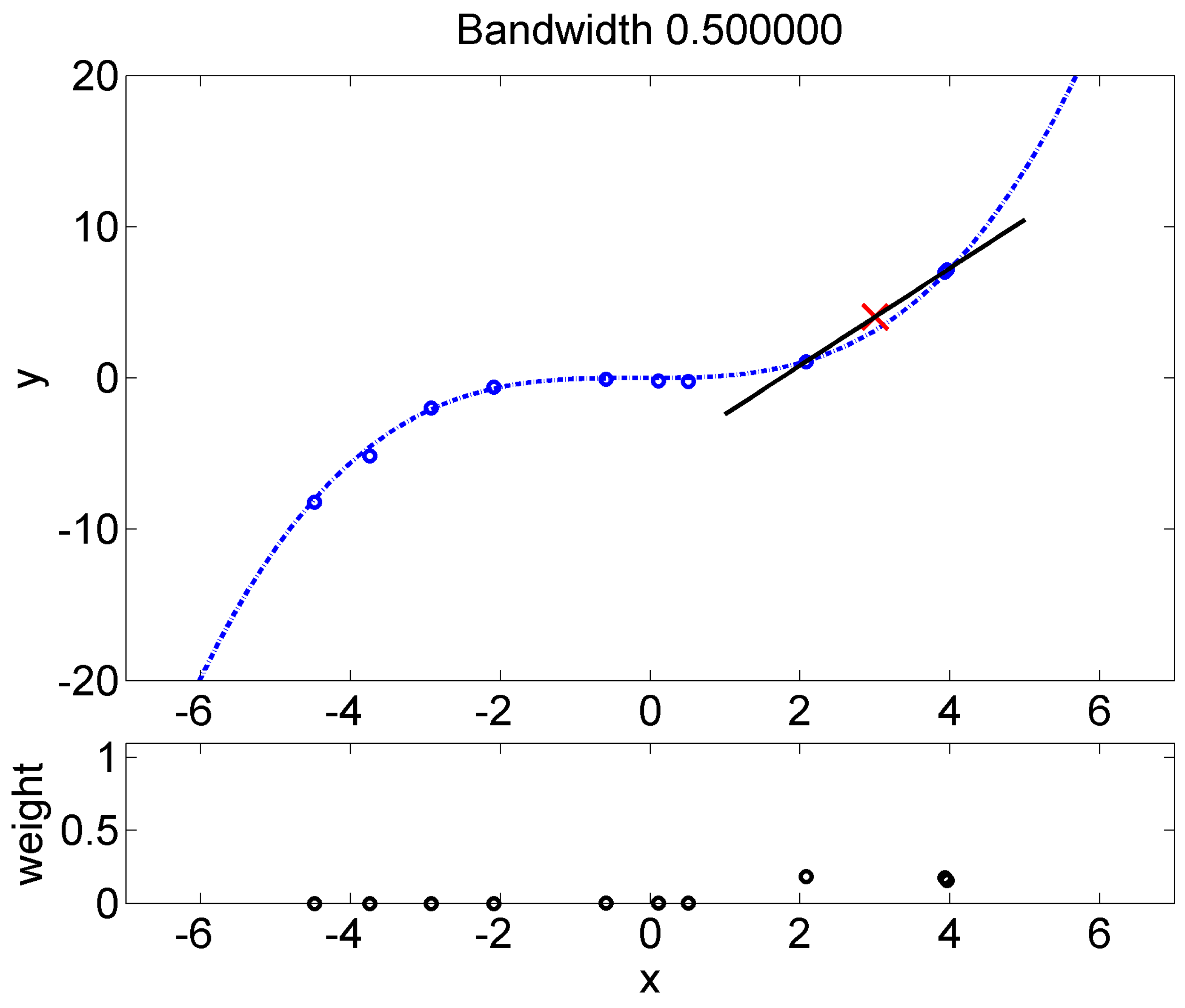
$$\boldsymbol{\theta} = (\tilde{\boldsymbol{X}}^T \boldsymbol{W} \tilde{\boldsymbol{X}})^{-1} \tilde{\boldsymbol{X}}^T \boldsymbol{W} \boldsymbol{y}$$

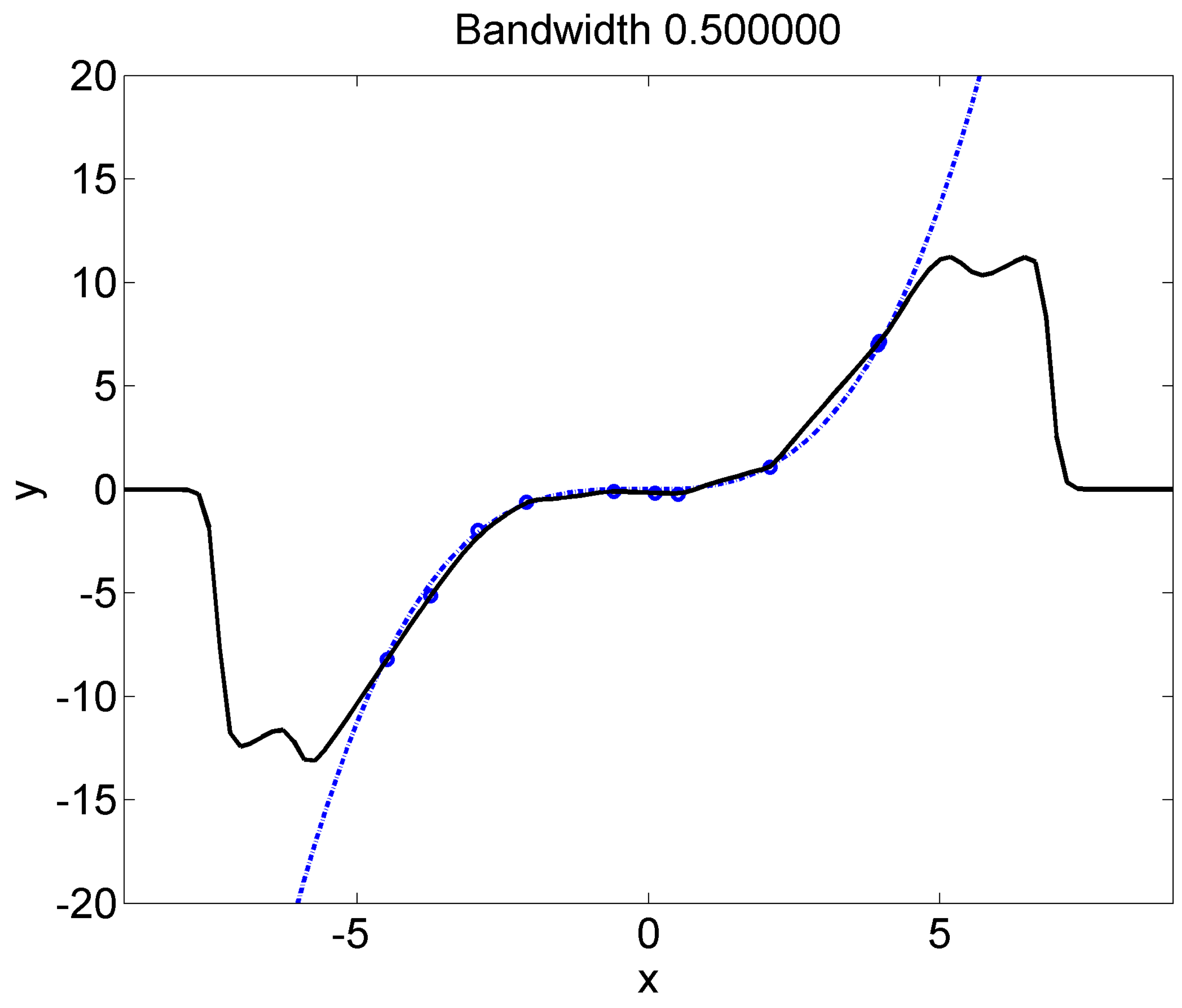
- **W** can be large - don’t implement it like this...
- Dismiss data points with small weights / use bsxfun

Local Ridge Regression:

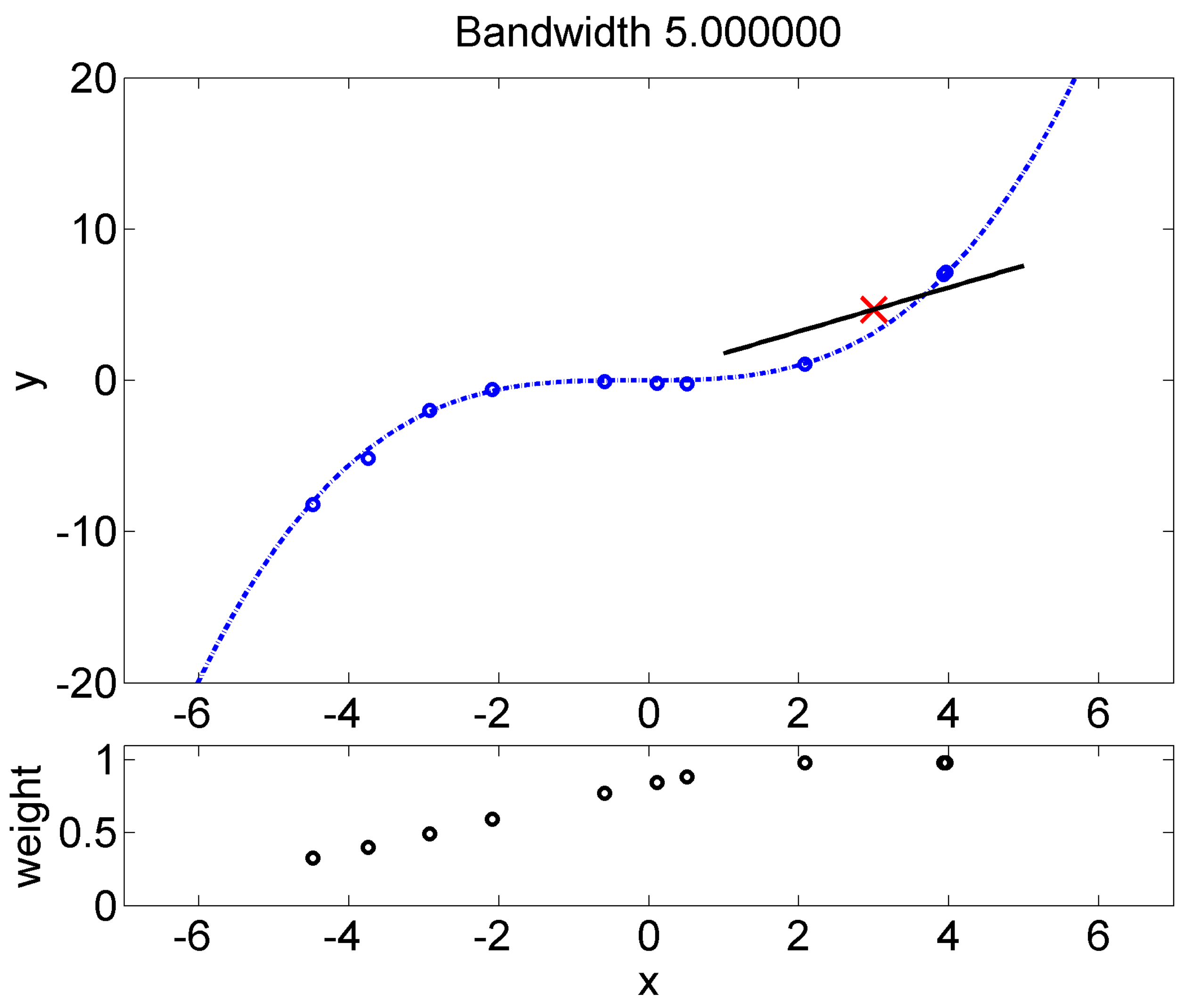
$$\boldsymbol{\theta} = (\tilde{\boldsymbol{X}}^T \boldsymbol{W} \tilde{\boldsymbol{X}} + \sigma^2 \boldsymbol{I})^{-1} \tilde{\boldsymbol{X}}^T \boldsymbol{W} \boldsymbol{y}$$

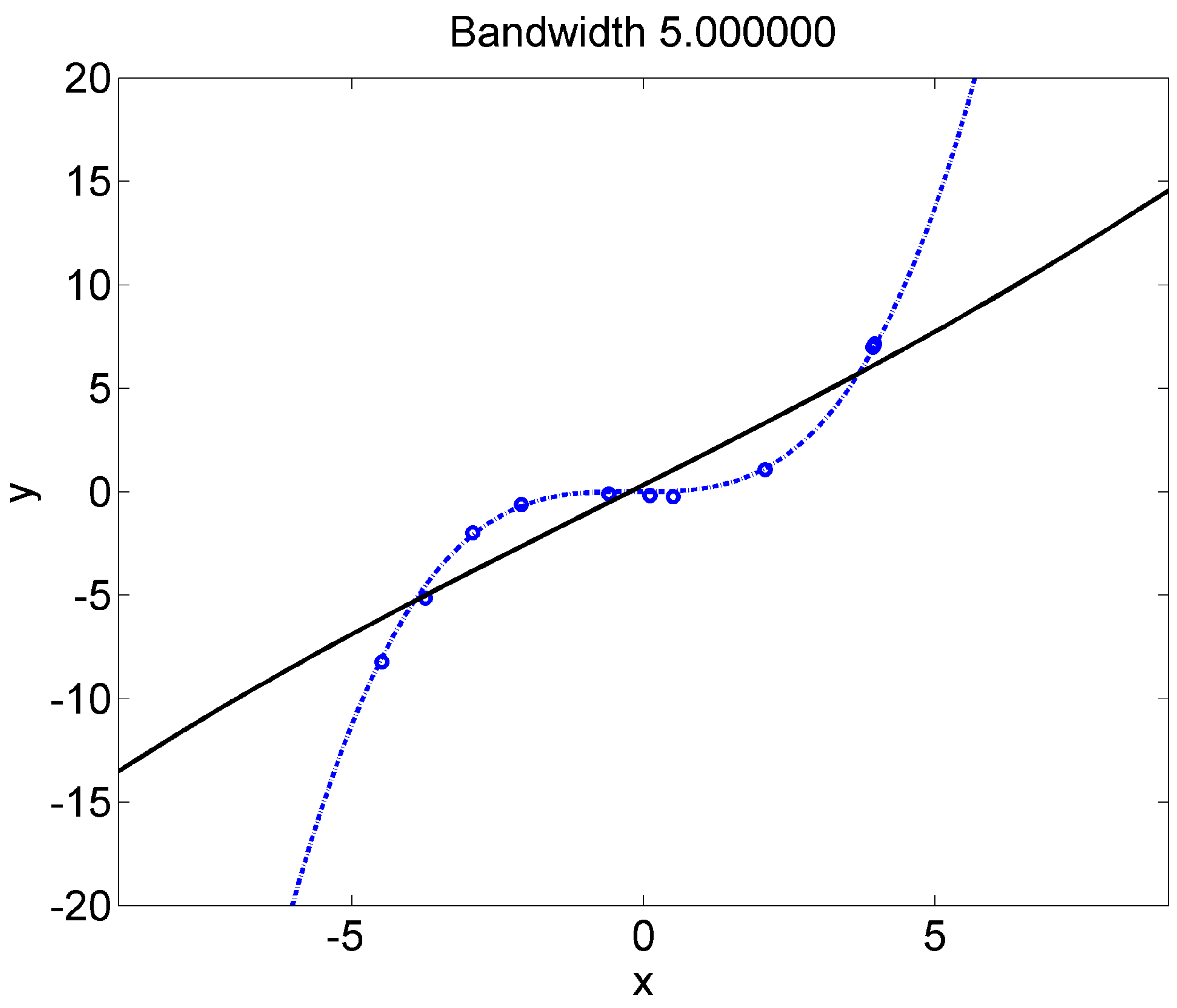
Solution with Locally-Weighted Regression



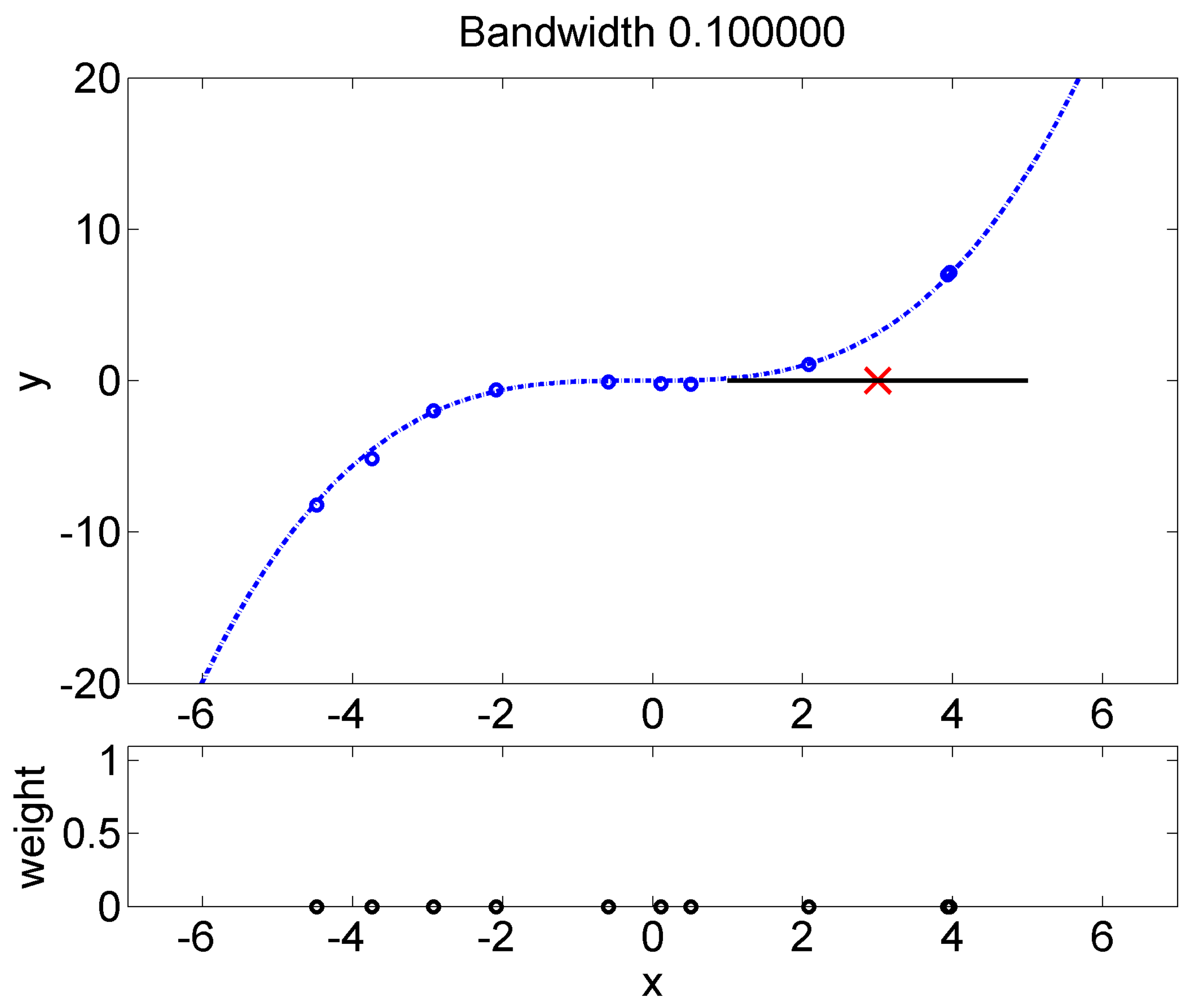


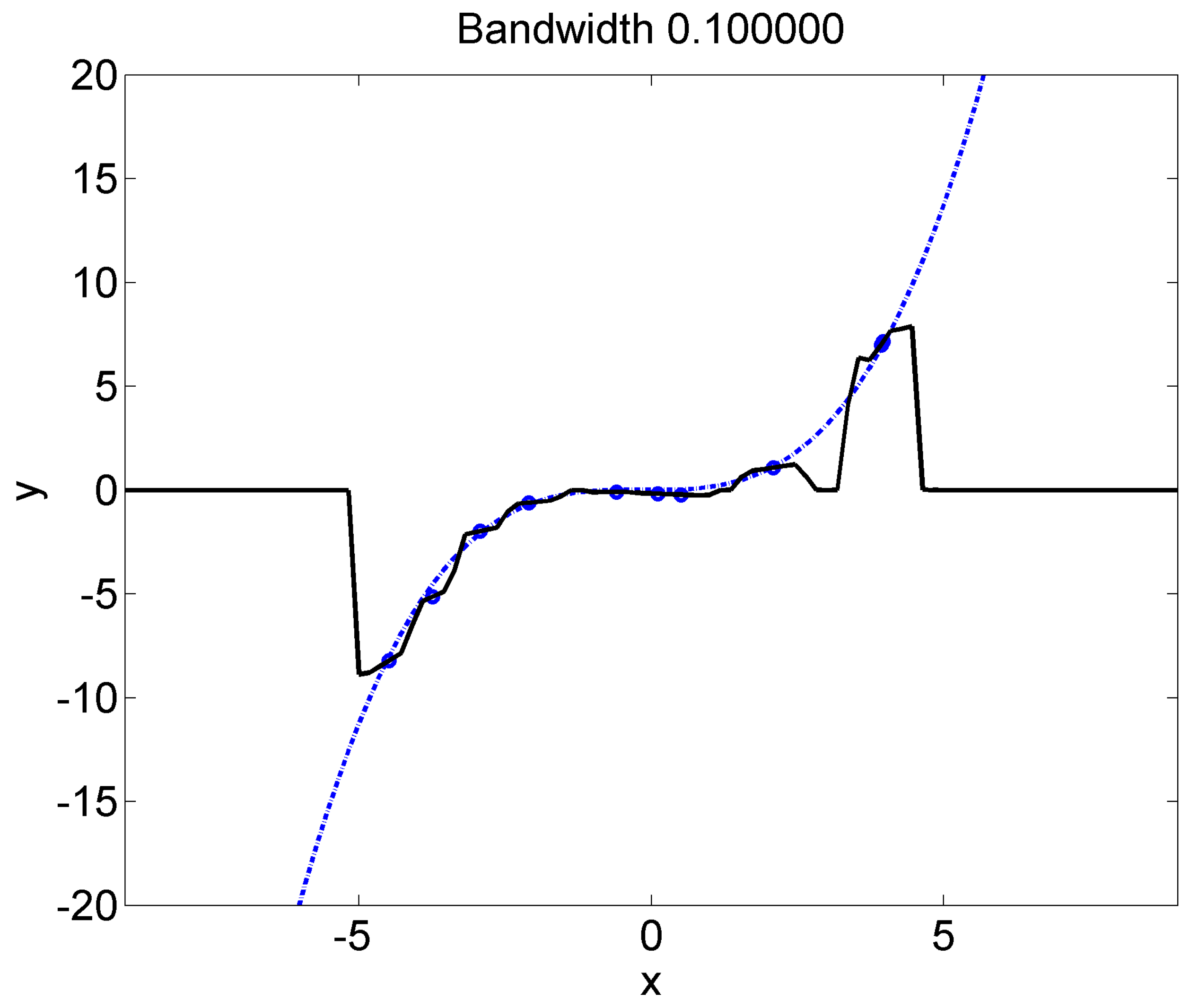
Solution with Locally-Weighted Regression





Solution with Locally-Weighted Regression



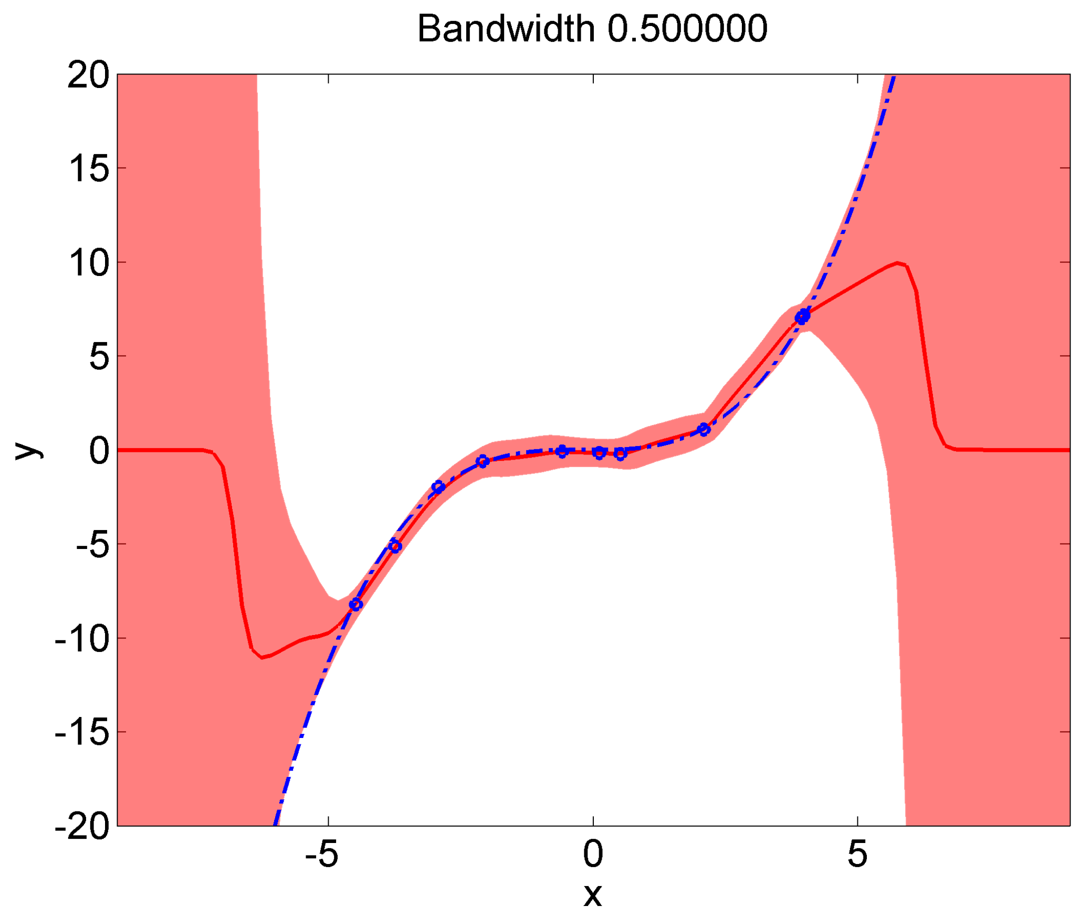


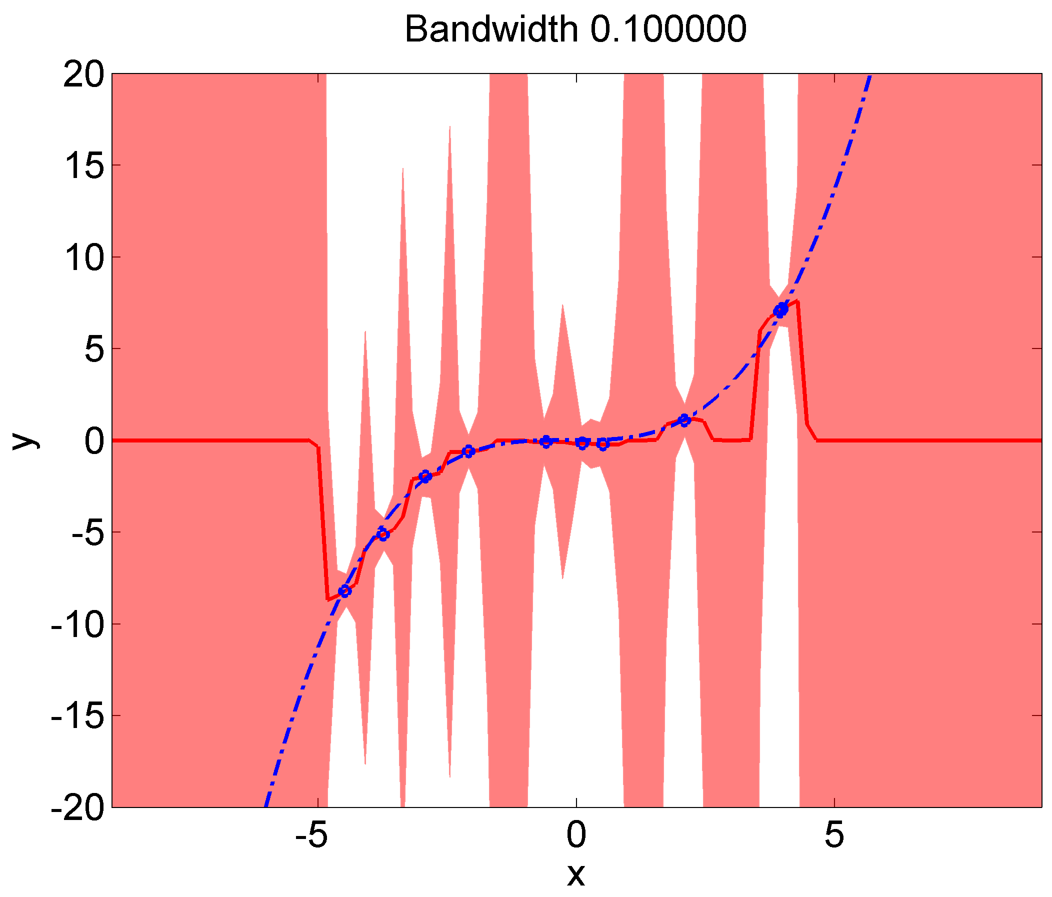
Weighted Linear Regression

Locally Weighted Bayesian Linear Regression

$$p(\boldsymbol{\theta}|\boldsymbol{y}, \boldsymbol{X}, \boldsymbol{W}) = \mathcal{N}(\boldsymbol{\theta}|\boldsymbol{\mu}_N, \boldsymbol{\Sigma}_N)$$

$$\boldsymbol{\Sigma}_N = (\tilde{\boldsymbol{X}}^T \boldsymbol{W} \tilde{\boldsymbol{X}} + \sigma^2 \lambda \boldsymbol{I})^{-1} \qquad \boldsymbol{\mu}_N = \boldsymbol{\Sigma}_N \tilde{\boldsymbol{X}} \boldsymbol{W} \boldsymbol{y}$$





Content of this Lecture

Constructing Basis Functions

- Radial Basis Function Networks

Non-Parametric Approaches

- Locally Weighted Regression
- Kernel Methods

Type 3: Kernel Methods

Kernel methods rely on the 'kernel trick'

➡ It is sufficient to evaluate the scalar product between two samples in feature space, called kernel

$$k(\boldsymbol{x}_1, \boldsymbol{x}_2) = \boldsymbol{\phi}(\boldsymbol{x}_1)^T \boldsymbol{\phi}(\boldsymbol{x}_2)$$

Why is this useful?

- Kernels are easier to design than features
- The feature space can be possibly infinite dimensional.
- We just need to be able to compute the scalar product

Type 3: Kernel Methods

Example: One RBF feature at every position c

$$k(\boldsymbol{x}_1, \boldsymbol{x}_2) = \boldsymbol{\phi}(\boldsymbol{x}_1)^T \boldsymbol{\phi}(\boldsymbol{x}_2) = \int \exp\left(-\frac{||\boldsymbol{x}_1 - \boldsymbol{c}||^2}{2l^2}\right) \exp\left(-\frac{||\boldsymbol{x}_2 - \boldsymbol{c}||^2}{2l^2}\right) d\boldsymbol{c}$$

$$= \exp\left(-\frac{||\boldsymbol{x}_1 - \boldsymbol{x}_2||^2}{4l^2}\right)$$

➡ Reduces to an RBF feature at each sample

General conditions for kernels

- symmetric: $k(\boldsymbol{x}_1, \boldsymbol{x}_2) = k(\boldsymbol{x}_2, \boldsymbol{x}_1)$
- positive definite…

Bayesian Linear Regression revisited

We have:

Data-Likelihood: $p(\boldsymbol{y}|\boldsymbol{\theta}, \boldsymbol{X}) = \mathcal{N}(\boldsymbol{y}|\boldsymbol{\Phi}\boldsymbol{\theta}, \sigma^2\boldsymbol{I})$

Prior: $p(\boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{\theta}|\boldsymbol{0}, \lambda^{-1}\boldsymbol{I})$

If we integrate out the weights, we get

$$p(\boldsymbol{y}|\boldsymbol{X}) = \int \mathcal{N}(\boldsymbol{y}|\boldsymbol{\Phi}\boldsymbol{\theta}, \sigma^2\boldsymbol{I})\mathcal{N}(\boldsymbol{\theta}|\boldsymbol{0}, \lambda^{-1}\boldsymbol{I})d\boldsymbol{\theta}$$
$$= \mathcal{N}(\boldsymbol{y}|\boldsymbol{0}, \sigma^2\boldsymbol{I} + \lambda^{-1}\boldsymbol{\Phi}\boldsymbol{\Phi}^T)$$

Defines a multivariate Gaussian distribution over the samples

➡ Samples are correlated as the marginalized weight vector is the same for each sample

Type 3: Kernel Methods

- Look at the predictions with the MAP/RR estimator (linear regression) again:

$$y(\boldsymbol{x}_*) = \phi(\boldsymbol{x}_*)^T\boldsymbol{\theta} = \phi(\boldsymbol{x}_*)^T(\boldsymbol{\Phi}^T\boldsymbol{\Phi} + \lambda\boldsymbol{I})^{-1}\boldsymbol{\Phi}^T\boldsymbol{y}$$

- Even more general, the **Woodbury identity for matrix inversion** yields

$$\boxed{(\boldsymbol{\Phi}^T\boldsymbol{\Phi} + \lambda\boldsymbol{I}_D)^{-1}\boldsymbol{\Phi}^T} = \boxed{\boldsymbol{\Phi}^T(\boldsymbol{\Phi}\boldsymbol{\Phi}^T + \lambda\boldsymbol{I}_N)^{-1}}$$

- This yields

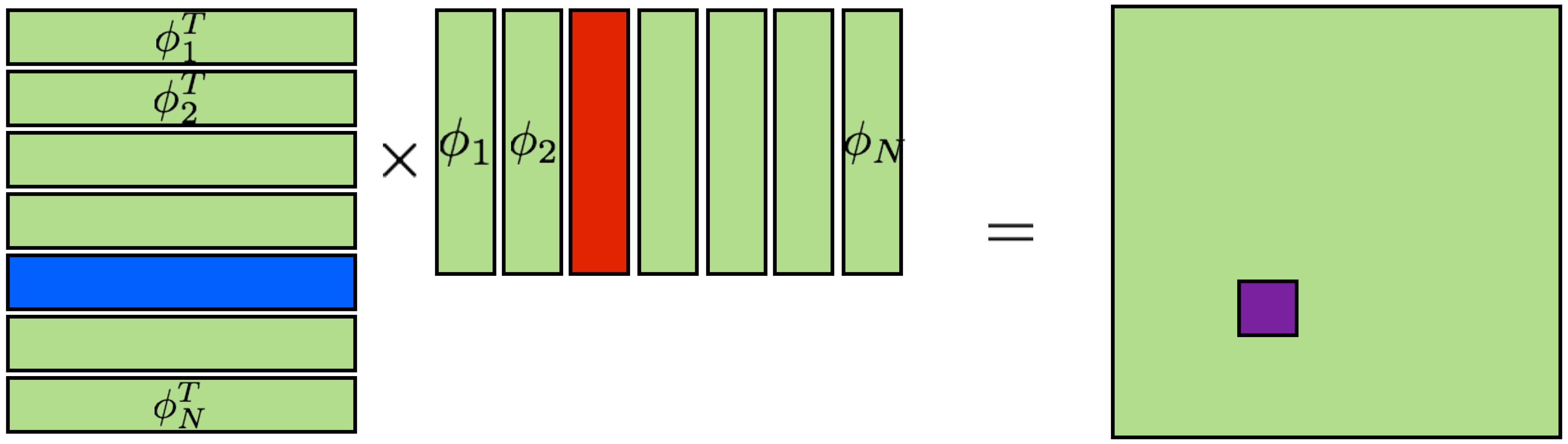
$$y(\boldsymbol{x}_*) = \phi(\boldsymbol{x}_*)^T \boxed{\underbrace{(\boldsymbol{\Phi}^T\boldsymbol{\Phi} + \lambda\boldsymbol{I}_D)^{-1}}_{D\times D}\boldsymbol{\Phi}^T}\boldsymbol{y}$$

$$= \phi(\boldsymbol{x}_*)^T \boxed{\boldsymbol{\Phi}^T\underbrace{(\boldsymbol{\Phi}\boldsymbol{\Phi}^T + \lambda\boldsymbol{I}_N)^{-1}}_{N\times N}}\boldsymbol{y}$$

Equivalent solution to ridge regression
Why is this potentially useful?

Gaussian Processes

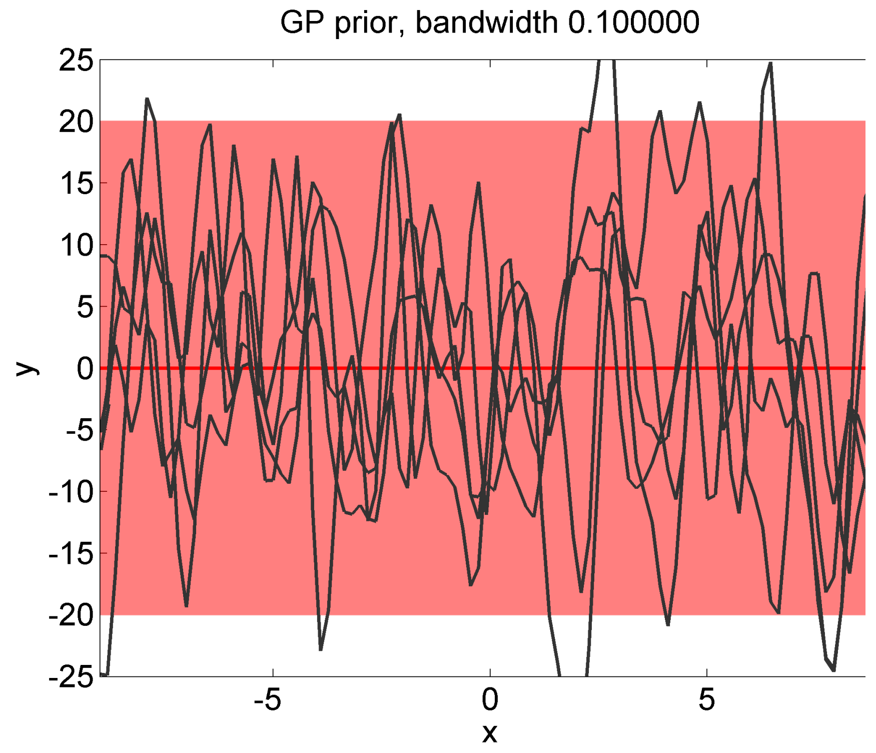
$$p(\boldsymbol{y}|\boldsymbol{X}) = \mathcal{N}(\boldsymbol{y}|\mathbf{0}, \sigma^2\boldsymbol{I} + \boldsymbol{K}) \text{ with } \boldsymbol{K} = \lambda^{-1}\boldsymbol{\Phi}\boldsymbol{\Phi}^T$$

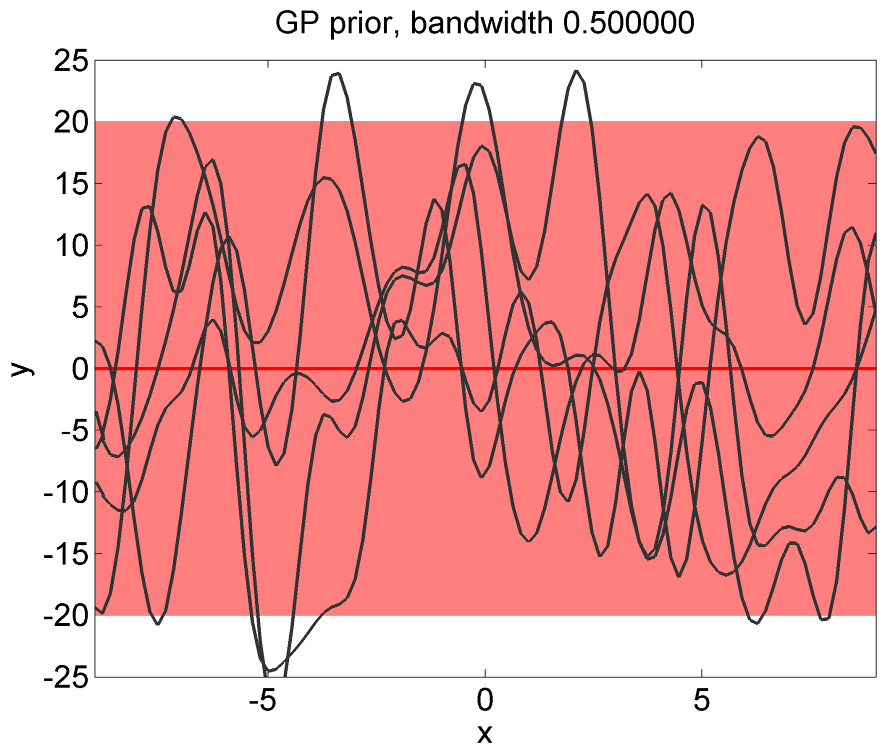
$\boldsymbol{K} =$ 

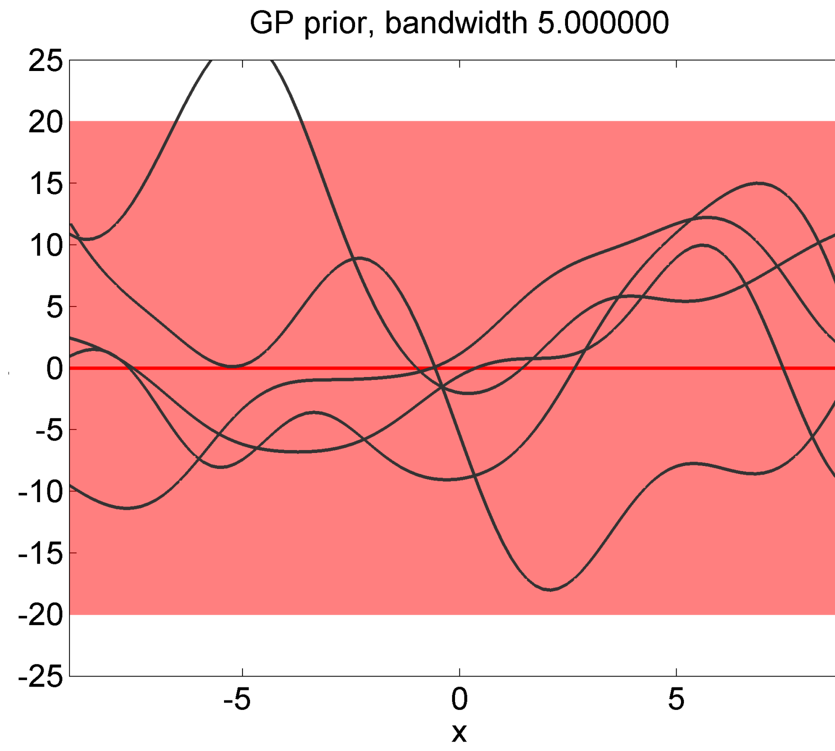
$$K_{ij} = \lambda^{-1}\boldsymbol{\phi}(\boldsymbol{x}_i)^T\boldsymbol{\phi}(\boldsymbol{x}_j) = k(\boldsymbol{x}_1, \boldsymbol{x}_2)$$ scalar products in feature space

This is called a Gaussian Process $\mathcal{GP}(\mathbf{0}, \boldsymbol{K})$
with covariance function k

Sampling from the GP-Prior







The bandwidth is a prior on the smoothness on the function

GP-Posterior

Now we observe a data set given by$\boldsymbol{y}$ and $\boldsymbol{X}$ and we want to predict y^* for $\boldsymbol{x}^*$

- We can write down the GP prior for the concatenated data

$$p\left(\begin{bmatrix} \boldsymbol{y} \\ y_* \end{bmatrix} \middle| \begin{bmatrix} \boldsymbol{X} \\ \boldsymbol{x}_* \end{bmatrix}\right) = \mathcal{N}\left(\begin{bmatrix} \boldsymbol{y} \\ y_* \end{bmatrix} \middle| \boldsymbol{0}, \begin{bmatrix} \boldsymbol{K} & \boldsymbol{k}(\boldsymbol{X}, \boldsymbol{x}_*) \\ \boldsymbol{k}(\boldsymbol{x}_*, \boldsymbol{X}) & k(\boldsymbol{x}_*, \boldsymbol{x}_*) \end{bmatrix} + \sigma^2 \boldsymbol{I}\right)$$

- We get the GP-posterior by Gaussian conditioning (see refresher)

$$p(y_* | \boldsymbol{y}, \boldsymbol{X}, \boldsymbol{x}_*) = \mathcal{N}(\boldsymbol{y}_* | \mu(\boldsymbol{x}_*), \sigma^2(\boldsymbol{x}_*))$$

GP-Posterior

$$p(y_*|\boldsymbol{y}, \boldsymbol{X}, \boldsymbol{x}_*) = \mathcal{N}(\boldsymbol{y}_*|\mu(\boldsymbol{x}_*), \sigma^2(\boldsymbol{x}_*))$$

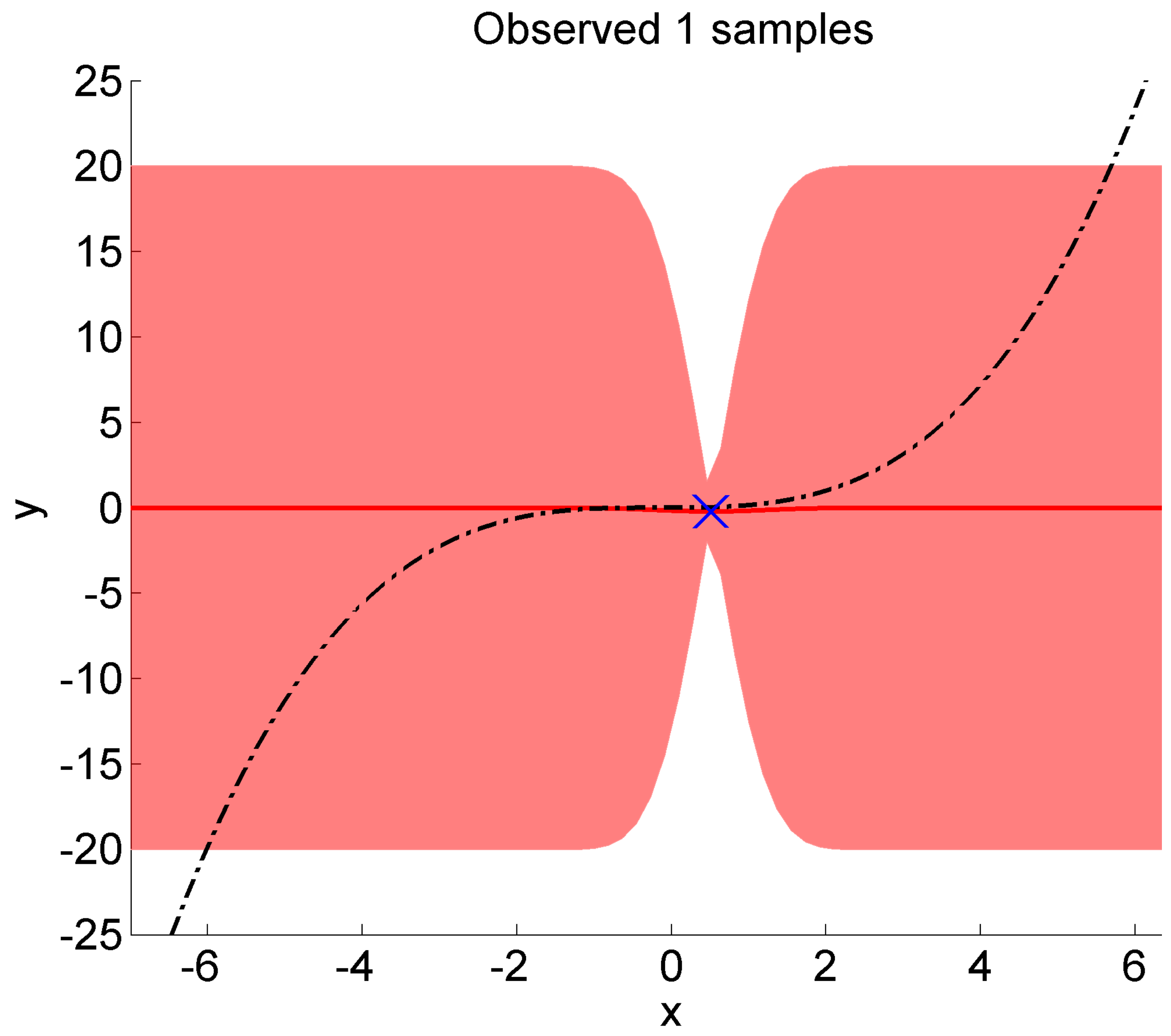
Predictive mean

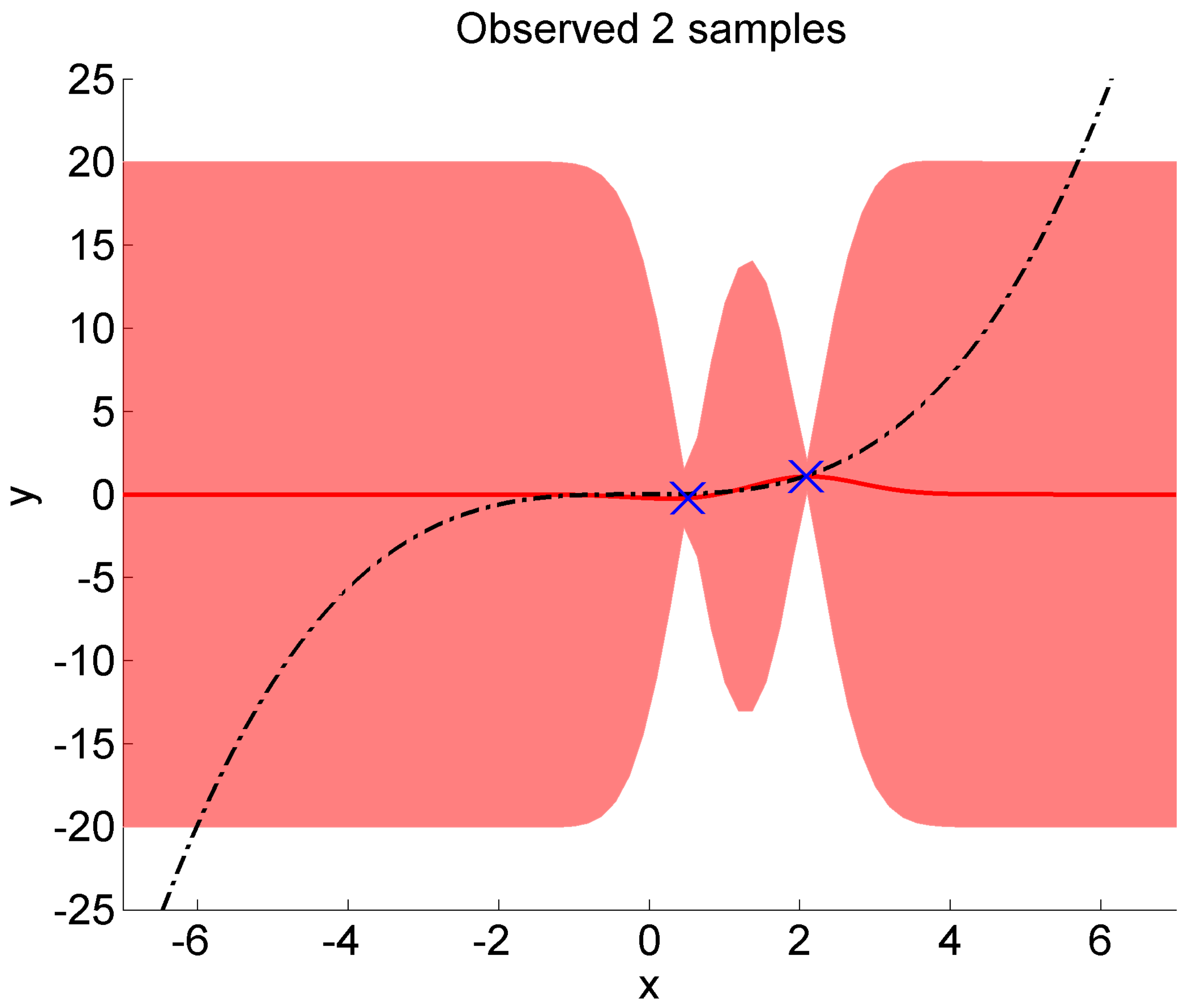
$$\mu(\boldsymbol{x}_*) = k(\boldsymbol{x}_*, \boldsymbol{X})(\boldsymbol{K} + \sigma^2\boldsymbol{I})^{-1}\boldsymbol{y}$$

Predictive variance

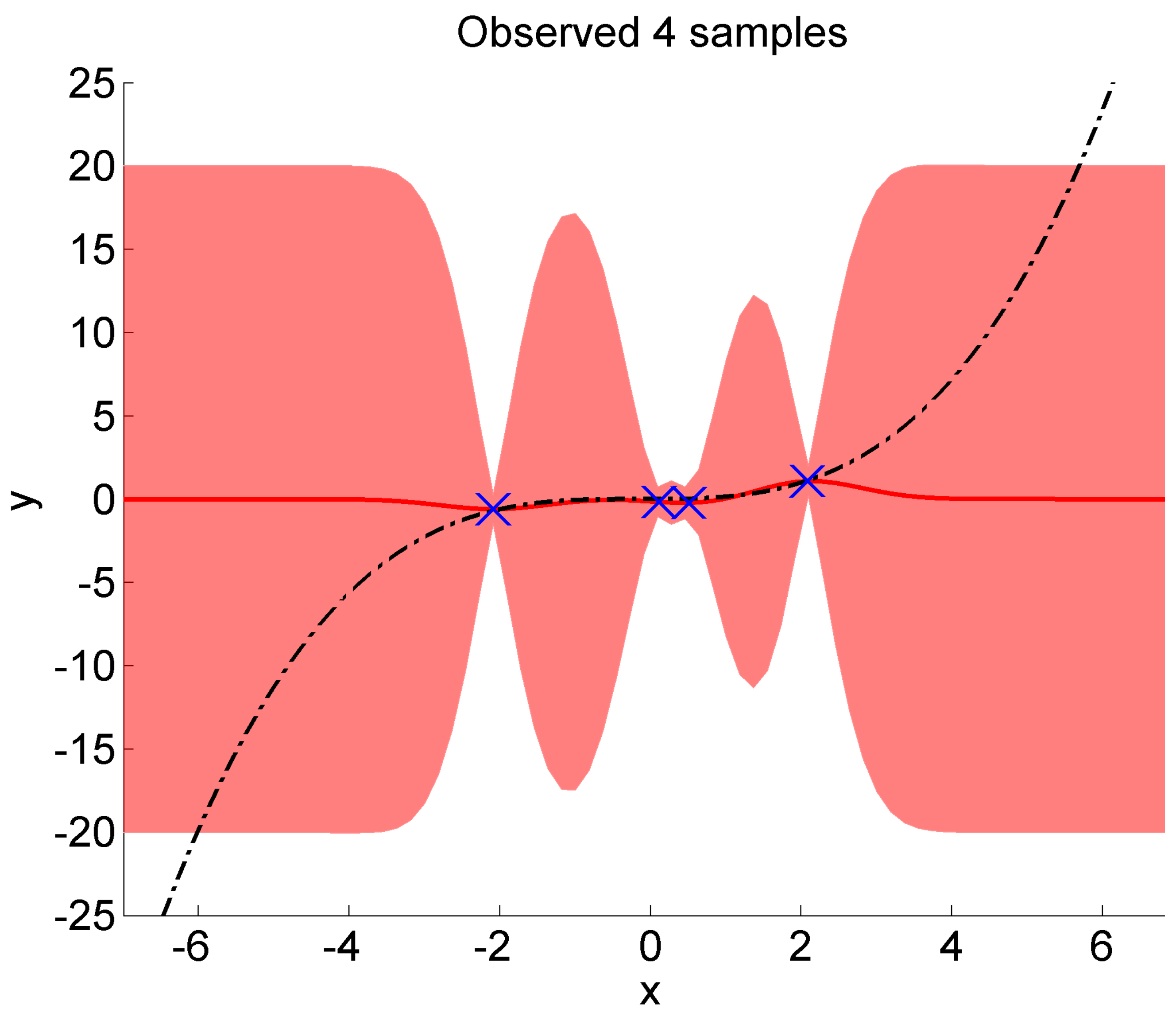
$$\begin{aligned}\sigma^2(\boldsymbol{x}_*) =& k(\boldsymbol{x}_*, \boldsymbol{x}_*) + \sigma^2 \\ & - \boldsymbol{k}(\boldsymbol{x}_*, \boldsymbol{X})(\boldsymbol{K} + \sigma^2\boldsymbol{I})^{-1}\boldsymbol{k}(\boldsymbol{X}, \boldsymbol{x}_*)\end{aligned}$$

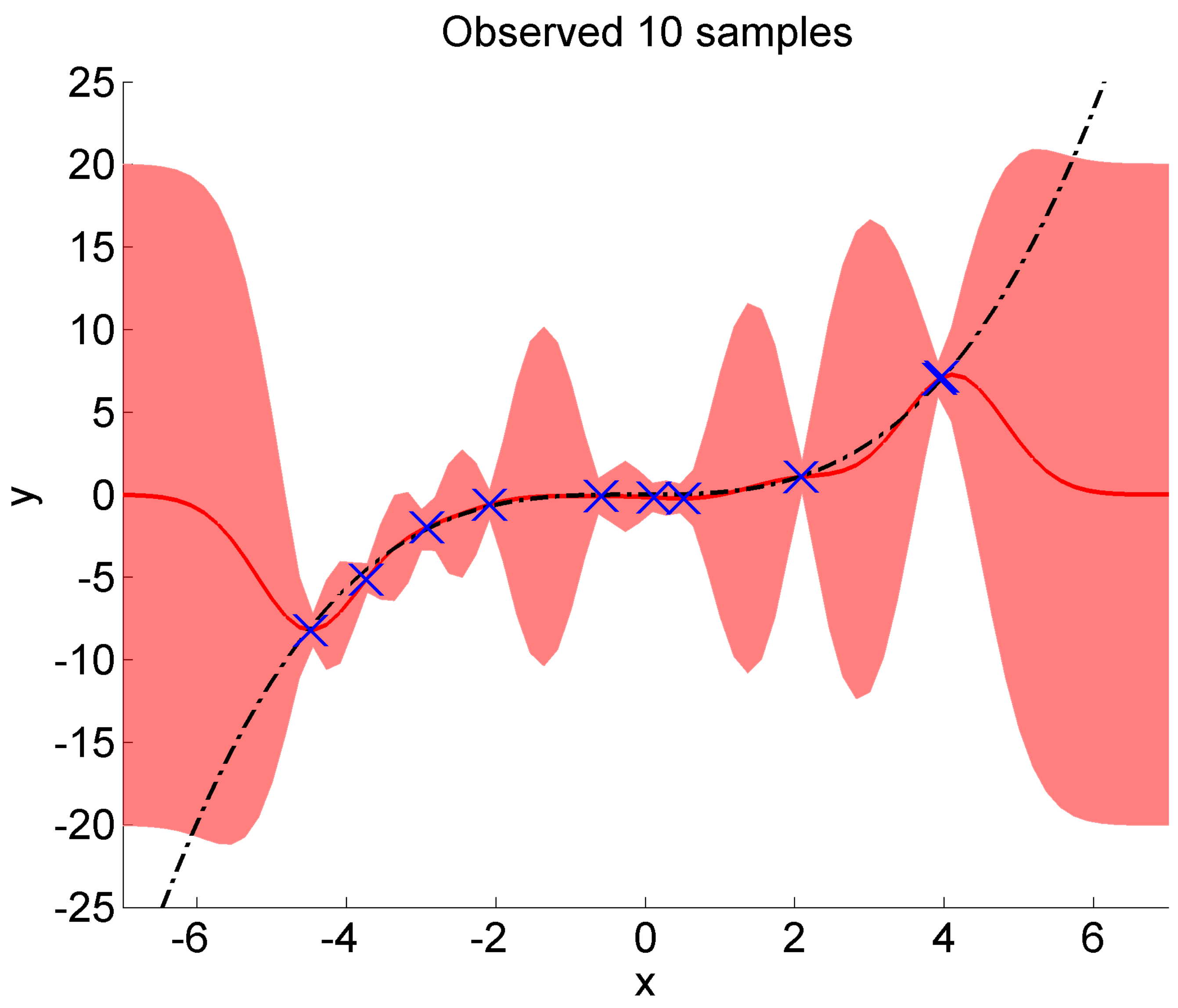
GP-Posterior





GP-Posterior





Gaussian Processes

Optimization of Hyper-Parameters

- The parameters of the kernel are called hyper-parameters
- Cross validation or maximization of marginal log-likelihood

GPs vs. Bayesian Linear Regression:

- GPs are the kernelized version
- Kernels are easier to use than features!

GPs are currently the gold standard for regression!

… if you do not have too many data points!

Summary

- You should have a really good overview of machine learning by now.
- You should remember the following **regression methods**
  - Least-Squares Regression / Ridge Regression
  - Bayesian Regression
  - Radial-Basis Function Regression
  - Locally-Weighted Linear Regression
  - GPs
- You should know how to **choose the right method** for a regression problem