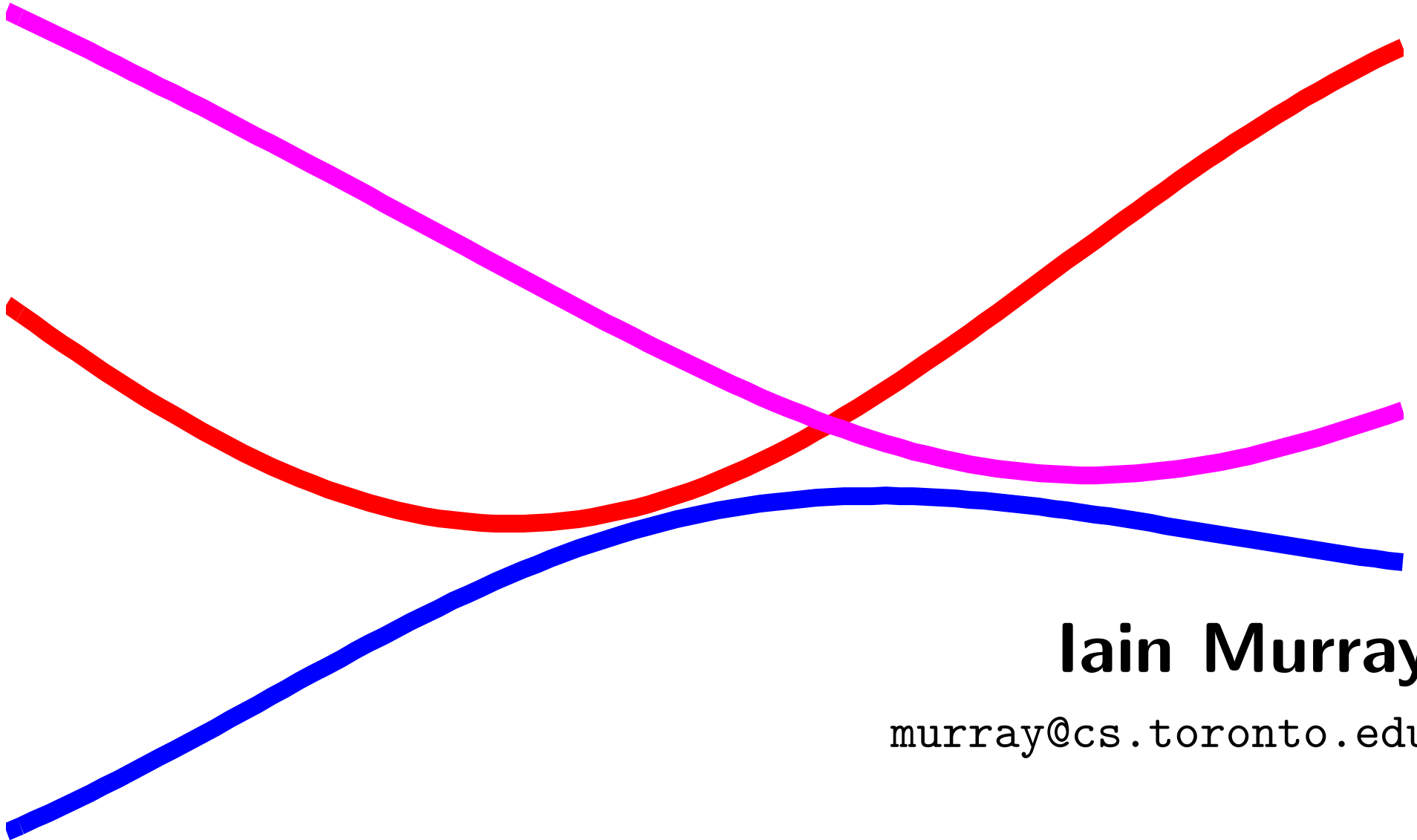


Introduction to Gaussian Processes



Iain Murray

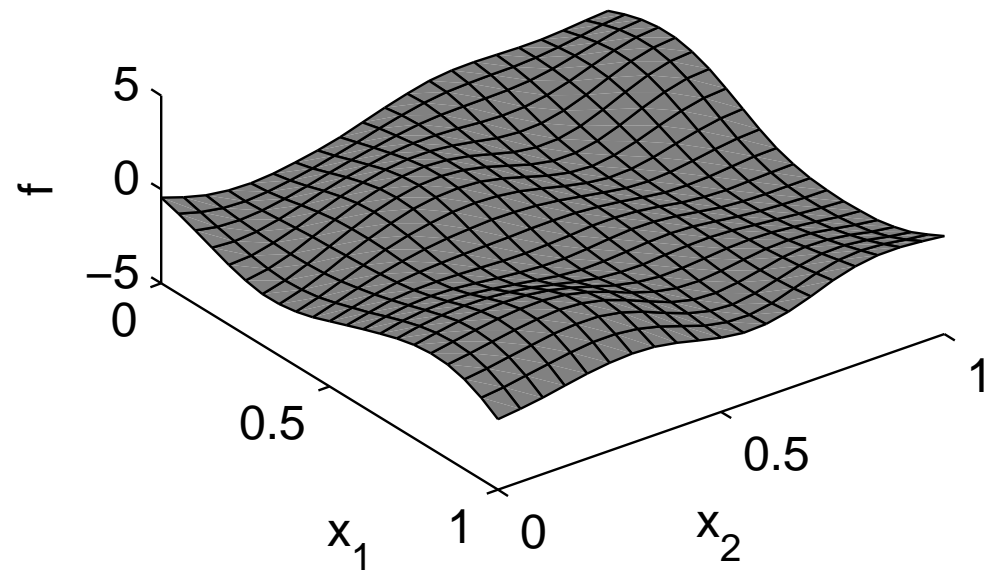
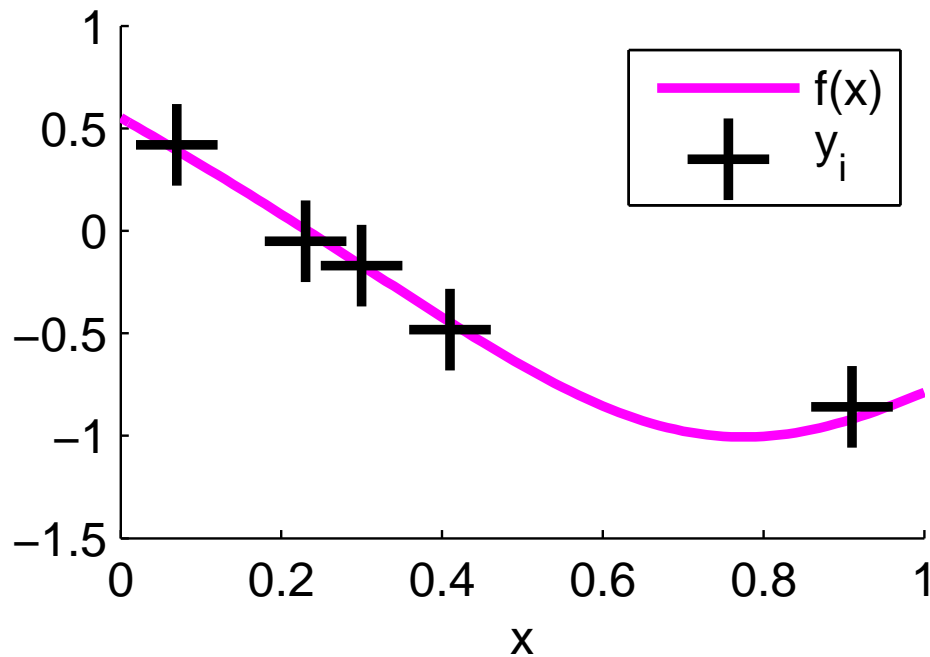
`murray@cs.toronto.edu`

CSC2515, Introduction to Machine Learning, Fall 2008

Dept. Computer Science, University of Toronto

The problem

Learn scalar function of vector values $f(\mathbf{x})$



We have (possibly noisy) observations $\{\mathbf{x}_i, y_i\}_{i=1}^n$

Example Applications

Real-valued regression:

- Robotics: target state \rightarrow required torque
- Process engineering: predicting yield
- Surrogate surfaces for optimization or simulation

Classification:

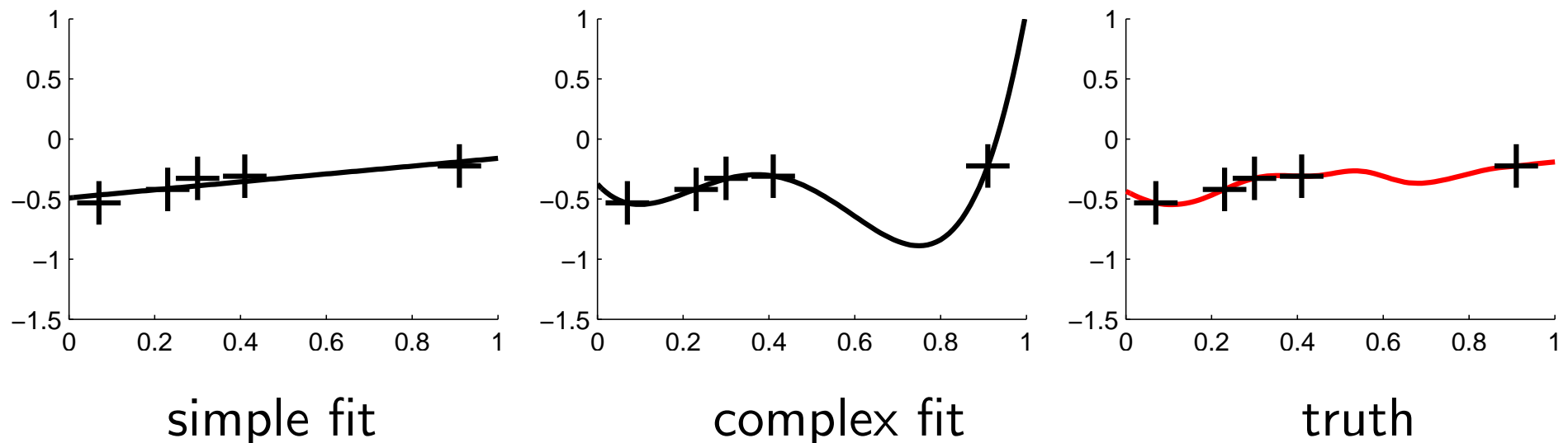
- Recognition: e.g. handwritten digits on cheques
- Filtering: fraud, interesting science, disease screening

Ordinal regression:

- User ratings (e.g. movies or restaurants)
- Disease screening (e.g. predicting Gleason score)

Model complexity

The world is often complicated:



Problems:

- Fitting complicated models can be hard
- How do we find an appropriate model?
- How do we avoid over-fitting some aspects of model?

Predicting yield

Factory settings $x_1 \rightarrow$ profit of 32 ± 5 monetary units

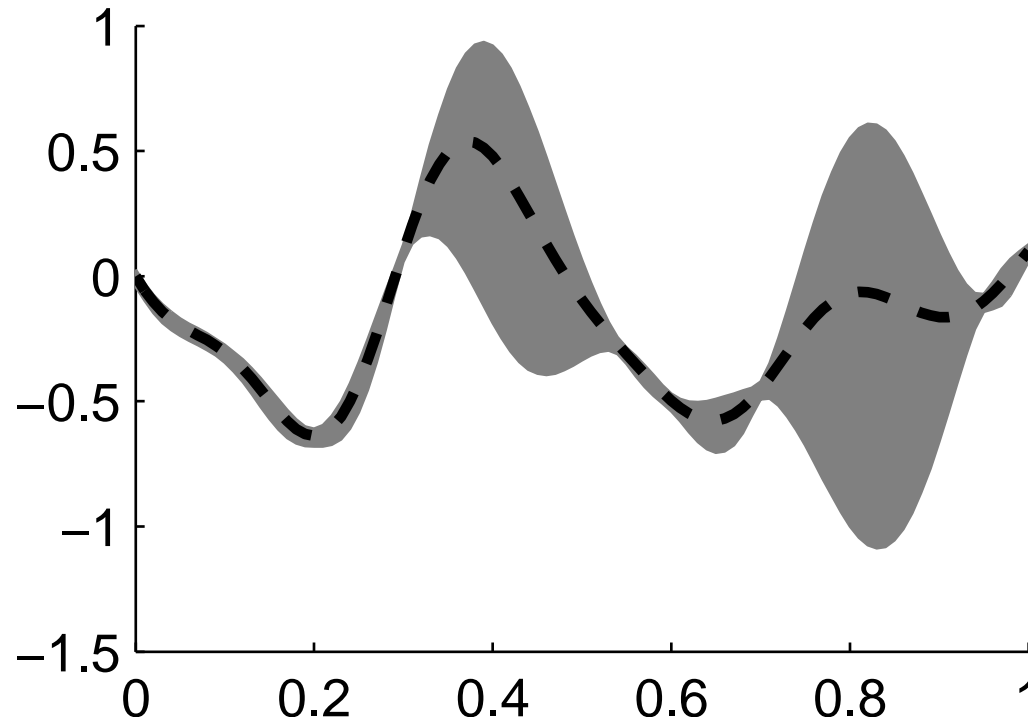
Factory settings $x_2 \rightarrow$ profit of 100 ± 200 monetary units

Which are the best settings x_1 or x_2 ?

Knowing the error bars can be very important.

Optimization

In high dimensions it takes many function evaluations to be certain everywhere. Costly if experiments are involved.

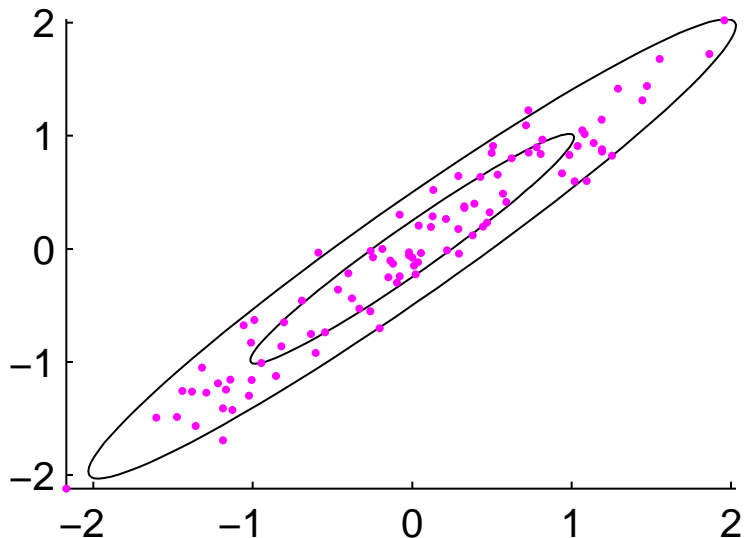


Error bars are needed to see if a region is still promising.

Bayesian modelling

If we come up with a parametric family of functions, $f(\mathbf{x}; \theta)$ and define a prior over θ , probability theory tells us how to make predictions given data. For flexible models, this usually involves intractable integrals over θ .

We're really good at integrating Gaussians though



Can we really solve significant machine learning problems with a simple multivariate Gaussian distribution?

Gaussian distributions

Completely described by parameters μ and Σ :

$$P(\mathbf{f}|\Sigma, \mu) = |2\pi\Sigma|^{-\frac{1}{2}} \exp\left(-\frac{1}{2}(\mathbf{f} - \mu)^T \Sigma^{-1}(\mathbf{f} - \mu)\right)$$

μ and Σ are the mean and covariance of the distribution.

For example:

$$\Sigma_{ij} = \langle f_i f_j \rangle - \mu_i \mu_j$$

If we know a distribution is Gaussian and know its mean and covariances, we know its density function.

Marginal of Gaussian

The marginal of a Gaussian distribution is Gaussian.

$$P(\mathbf{f}, \mathbf{g}) = \mathcal{N} \left(\begin{bmatrix} \mathbf{a} \\ \mathbf{b} \end{bmatrix}, \begin{bmatrix} A & C \\ C^\top & B \end{bmatrix} \right)$$

As soon as you convince yourself that the marginal

$$P(\mathbf{f}) = \int d\mathbf{g} P(\mathbf{f}, \mathbf{g})$$

is Gaussian, you already know the means and covariances:

$$P(\mathbf{f}) = \mathcal{N}(\mathbf{a}, A).$$

Conditional of Gaussian

Any conditional of a Gaussian distribution is also Gaussian:

$$P(\mathbf{f}, \mathbf{g}) = \mathcal{N} \left(\begin{bmatrix} \mathbf{a} \\ \mathbf{b} \end{bmatrix}, \begin{bmatrix} A & C \\ C^\top & B \end{bmatrix} \right)$$

$$P(\mathbf{f}|\mathbf{g}) = \mathcal{N}(\mathbf{a} + CB^{-1}(\mathbf{y} - \mathbf{b}), A - CB^{-1}C^\top)$$

Showing this is not completely straightforward.
But it is a standard result, easily looked up.

Noisy observations

Previously we inferred \mathbf{f} given \mathbf{g} .

What if we only saw a noisy observation, $\mathbf{y} \sim \mathcal{N}(\mathbf{g}, S)$?

$P(\mathbf{f}, \mathbf{g}, \mathbf{y}) = P(\mathbf{f}, \mathbf{g})P(\mathbf{y}|\mathbf{g})$ is Gaussian distributed; still a quadratic form inside the exponential after multiplying.

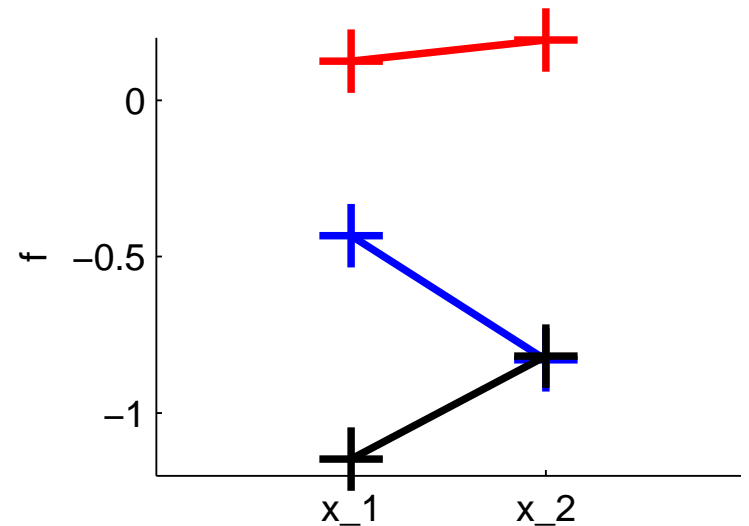
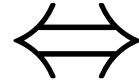
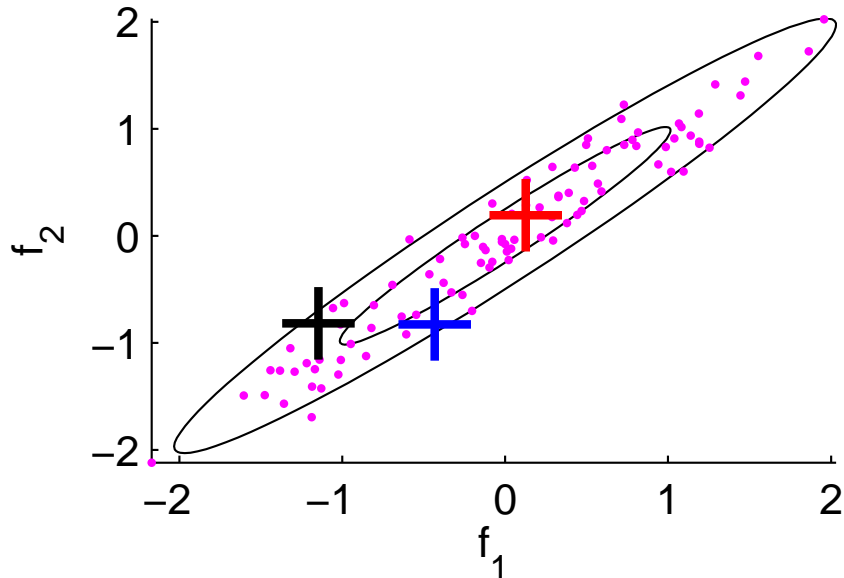
Our posterior over \mathbf{f} is still Gaussian:

$$P(\mathbf{f}|\mathbf{y}) \propto \int d\mathbf{g} P(\mathbf{f}, \mathbf{g}, \mathbf{y})$$

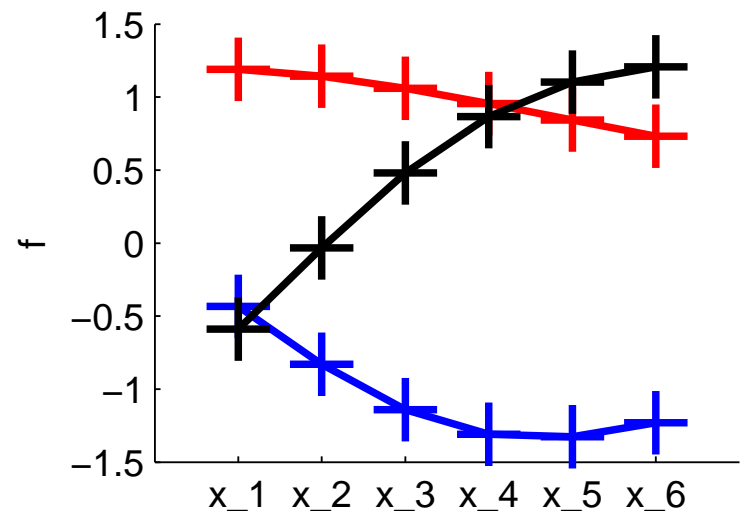
(RHS is Gaussian after marginalizing, so still a quadratic form in \mathbf{f} inside an exponential.)

Laying out Gaussians

A way of visualizing draws from a 2D Gaussian:

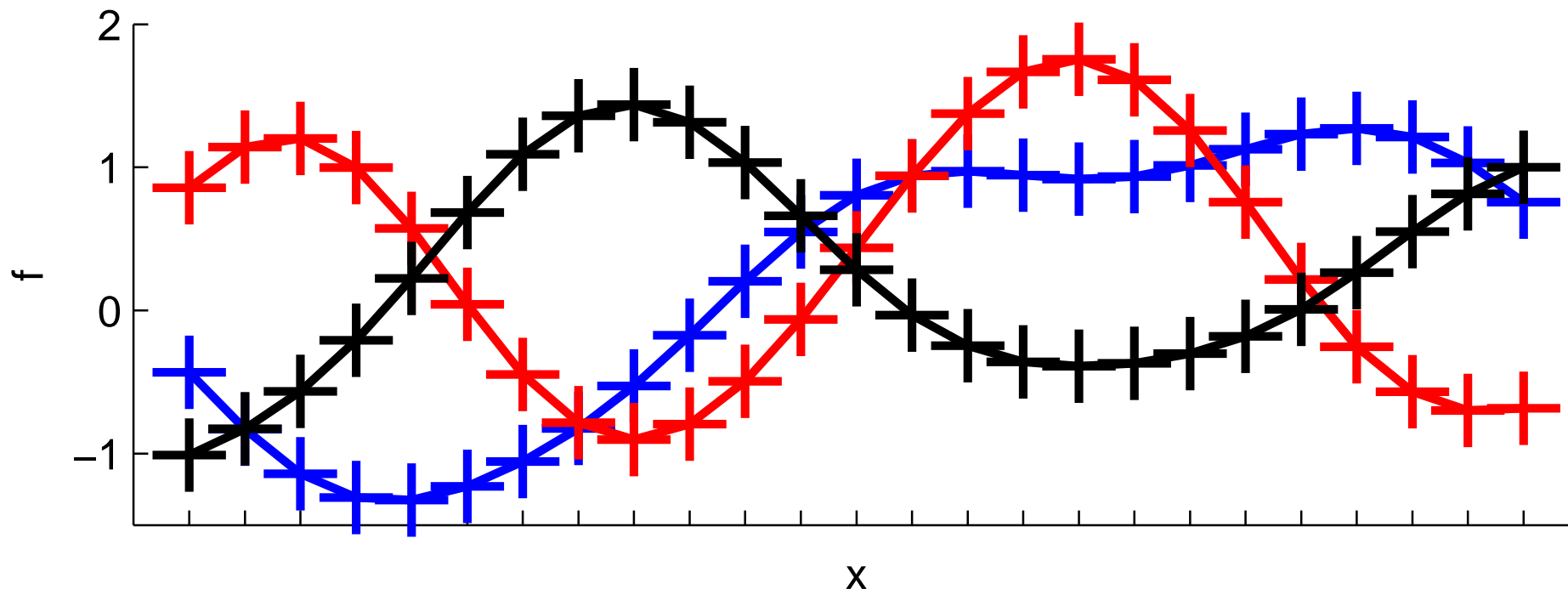


Now it's easy to show three draws from a 6D Gaussian:



Building large Gaussians

Three draws from a 25D Gaussian:



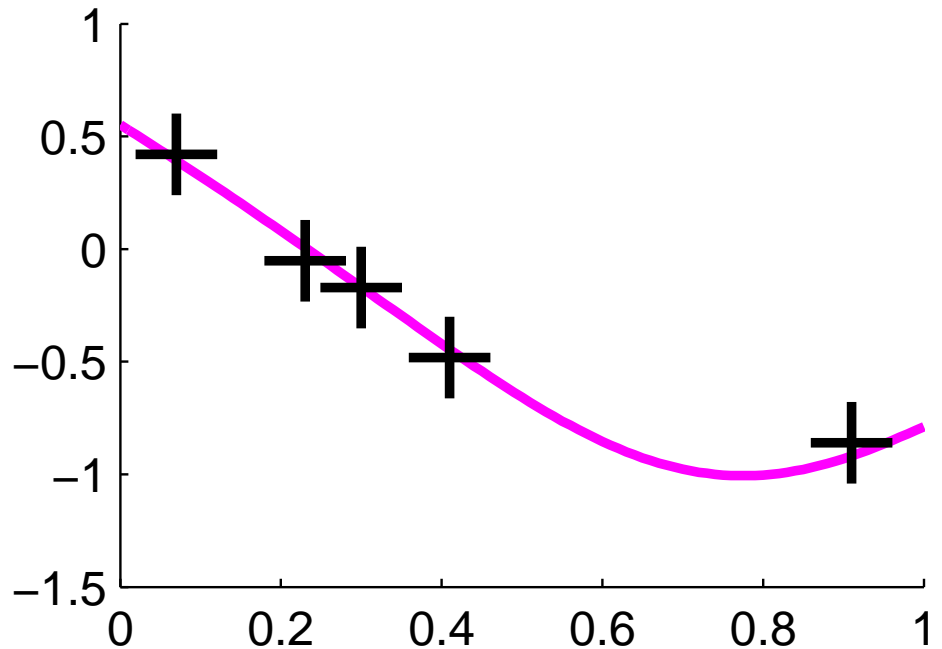
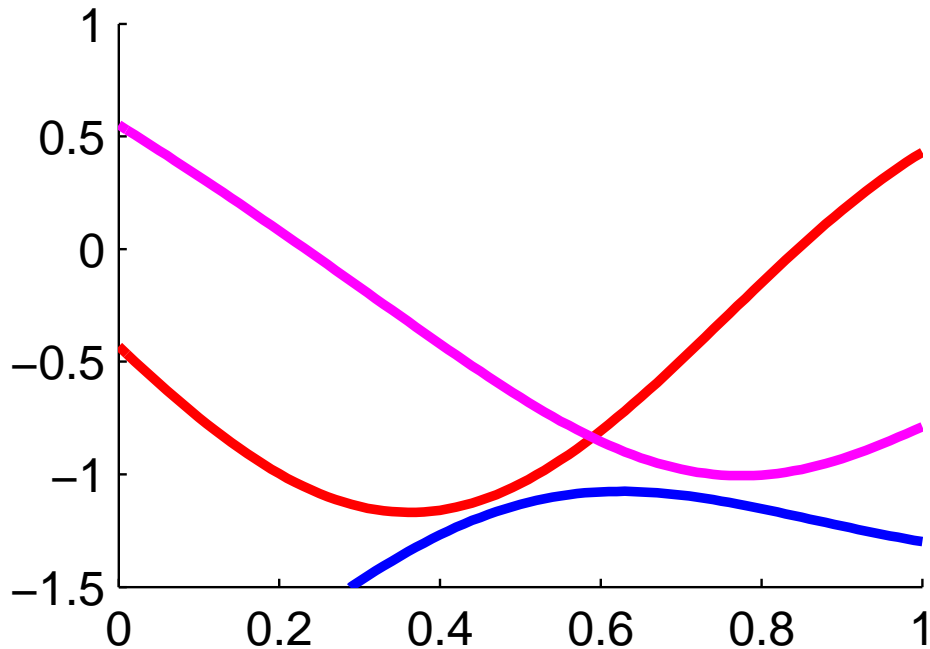
To produce this, we needed a mean: I used `zeros(25, 1)`

The covariances were set using a *kernel* function: $\Sigma_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$.

The \mathbf{x} 's are the positions that I planted the ticks on the axis.

Later we'll find k 's that ensure Σ is always positive semi-definite.

GP regression model



$$f \sim \mathcal{GP}$$

$$\mathbf{f} \sim \mathcal{N}(\mathbf{0}, K), \quad K_{ij} = k(x_i, x_j)$$

where $f_i = f(x_i)$

Noisy observations:

$$y_i | f_i \sim \mathcal{N}(f_i, \sigma_n^2)$$

GP Posterior

Our prior over observations and targets is Gaussian:

$$P\left(\begin{bmatrix} \mathbf{y} \\ \mathbf{f}_* \end{bmatrix}\right) = \mathcal{N}\left(\mathbf{0}, \begin{bmatrix} K(X, X) + \sigma_n^2 \mathbb{I} & K(X, X_*) \\ K(X_*, X) & K(X_*, X_*) \end{bmatrix}\right)$$

Using the rule for conditionals, $P(\mathbf{f}_*|\mathbf{y})$ is Gaussian with:

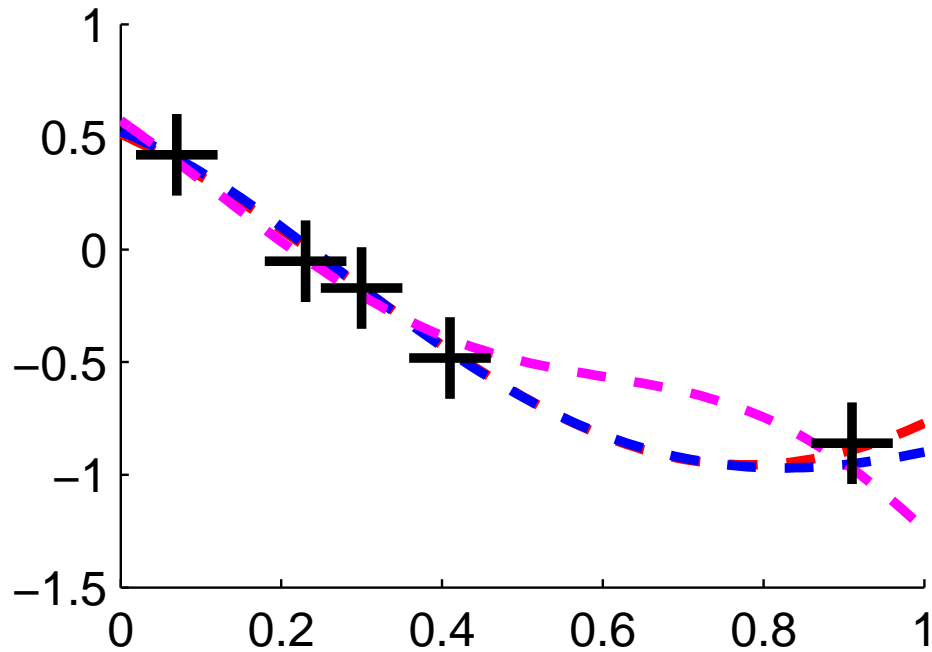
$$\text{mean, } \bar{\mathbf{f}}_* = K(X_*, X)(K(X, X) + \sigma_n^2 \mathbb{I})^{-1} \mathbf{y}$$

$$\text{cov}(\mathbf{f}_*) = K(X_*, X_*) - K(X_*, X)(K(X, X) + \sigma_n^2 \mathbb{I})^{-1} K(X, X_*)$$

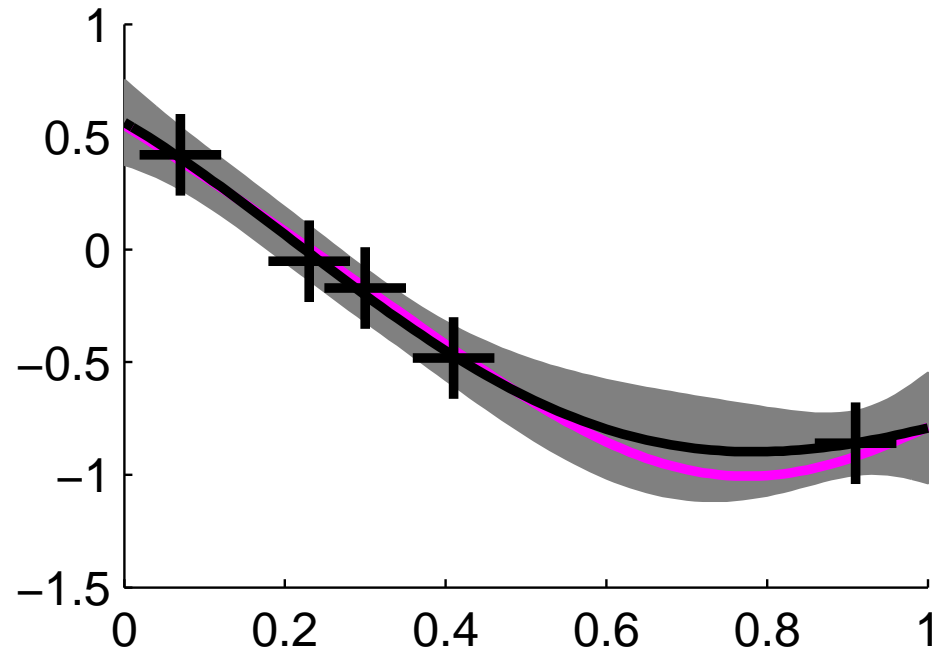
The posterior over functions is a Gaussian Process.

GP posterior

Two (incomplete) ways of visualizing what we know:



Draws $\sim p(\mathbf{f}|\text{data})$



Mean and error bars

Point predictions

Conditional at one point \mathbf{x}_* is a simple Gaussian:

$$p(f(\mathbf{x}_*)|\text{data}) = \mathcal{N}(m, s^2)$$

Need covariances:

$$K_{ij} = k(\mathbf{x}_i, \mathbf{x}_j), \quad (\mathbf{k}_*)_i = k(\mathbf{x}_*, \mathbf{x}_i)$$

Special case of joint posterior:

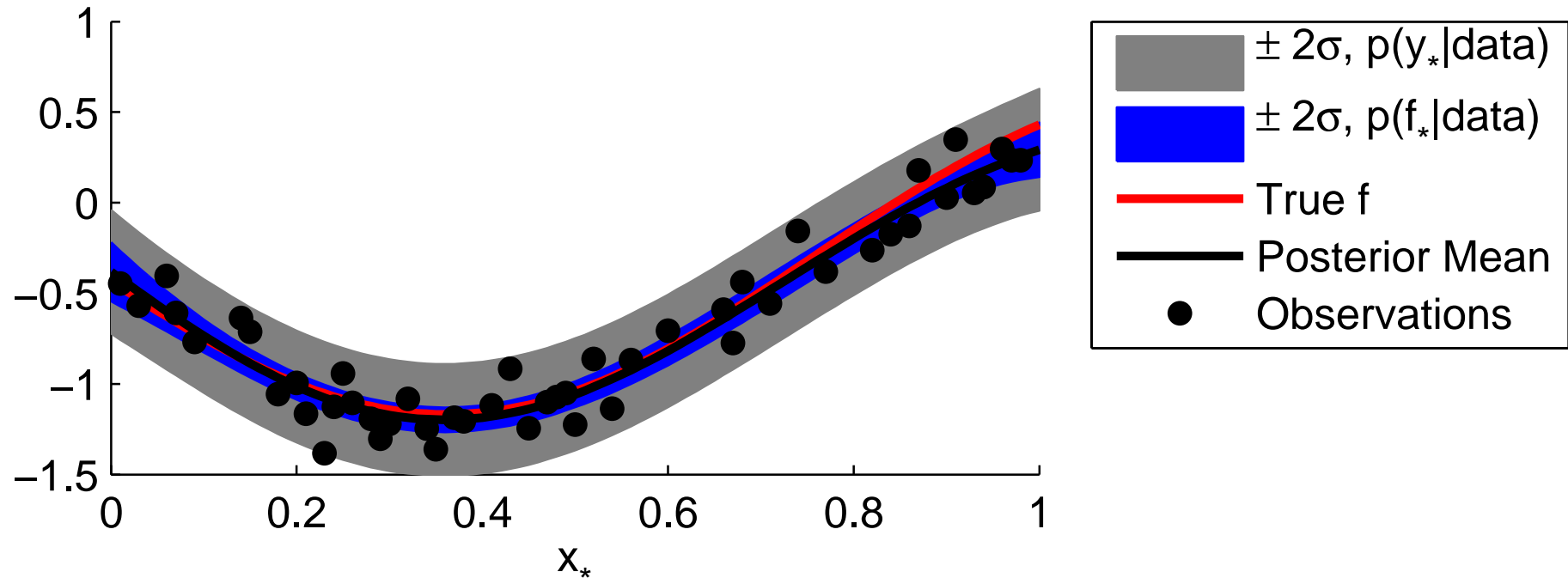
$$M = K + \sigma_n^2 \mathbb{I}$$

$$m = \mathbf{k}_*^\top M^{-1} \mathbf{y}$$

$$s^2 = k(\mathbf{x}_*, \mathbf{x}_*) - \underbrace{\mathbf{k}_*^\top M^{-1} \mathbf{k}_*}_{\text{positive}}$$

Discovery or prediction?

What should error-bars show?



$P(f_*|data) = \mathcal{N}(m, s^2)$ says what we know about the noiseless function.

$P(y_*|data) = \mathcal{N}(m, s^2 + \sigma_n^2)$ predicts what we'll see next.

Review so far

We can represent a function as a *big* vector \mathbf{f}

We assume that this unknown vector was drawn from a *big* correlated Gaussian distribution, a *Gaussian process*.

(This might upset some mathematicians, but for all practical machine learning and statistical problems, this is fine.)

Observing elements of the vector (optionally corrupted by Gaussian noise) creates a posterior distribution. This is also Gaussian: the posterior over functions is still a Gaussian process.

Because marginalization in Gaussians is trivial, we can easily ignore all of the positions \mathbf{x}_i that are neither observed nor queried.

Covariance functions

The main part that has been missing so far is where the covariance function $k(\mathbf{x}_i, \mathbf{x}_j)$ comes from.

Also, other than making nearby points covary, what can we express with covariance functions, and what do they mean?

Covariance functions

We can construct covariance functions from parametric models

Simplest example: **Bayesian linear regression:**

$$f(\mathbf{x}_i) = \mathbf{w}^\top \mathbf{x}_i + b, \quad \mathbf{w} \sim \mathcal{N}(0, \sigma_w^2 \mathbb{I}), \quad b \sim \mathcal{N}(0, \sigma_b^2)$$

$$\begin{aligned} \text{cov}(f_i, f_j) &= \langle f_i f_j \rangle - \cancel{\langle f_i \rangle} \cancel{\langle f_j \rangle} \\ &= \langle (\mathbf{w}^\top \mathbf{x}_i + b)^\top (\mathbf{w}^\top \mathbf{x}_j + b) \rangle \\ &= \sigma_w^2 \mathbf{x}_i^\top \mathbf{x}_j + \sigma_b^2 = k(\mathbf{x}_i, \mathbf{x}_j) \end{aligned}$$

Kernel parameters σ_w^2 and σ_b^2 are *hyper-parameters* in the Bayesian hierarchical model.

More interesting kernels come from models with a large or infinite feature space. Because feature weights \mathbf{w} are integrated out, this is computationally no more expensive.

Squared-exponential kernel

An ∞ number of radial-basis functions can give

$$k(\mathbf{x}_i, \mathbf{x}_j) = \sigma_f^2 \exp\left(-\frac{1}{2} \sum_{d=1}^D (x_{d,i} - x_{d,j})^2 / \ell_d^2\right),$$

the most commonly-used kernel in machine learning.

It looks like an (unnormalized) Gaussian, so is commonly called the Gaussian kernel. *Please* remember that this has nothing to do with it being a *Gaussian* process.

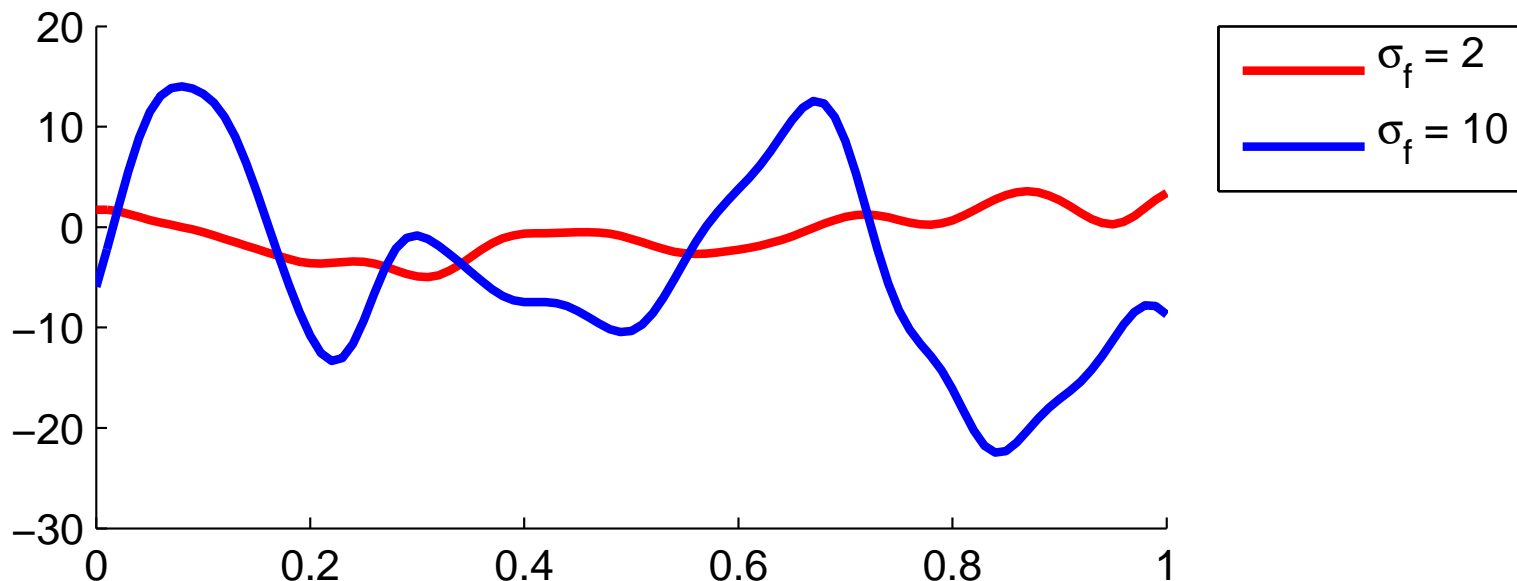
A Gaussian process need not use the “Gaussian” kernel. In fact, other choices will often be better.

Meaning of hyper-parameters

Many kernels have similar types of parameters:

$$k(\mathbf{x}_i, \mathbf{x}_j) = \sigma_f^2 \exp\left(-\frac{1}{2} \sum_{d=1}^D (x_{d,i} - x_{d,j})^2 / \ell_d^2\right),$$

Consider $\mathbf{x}_i = \mathbf{x}_j$, \Rightarrow marginal function variance is σ_f^2

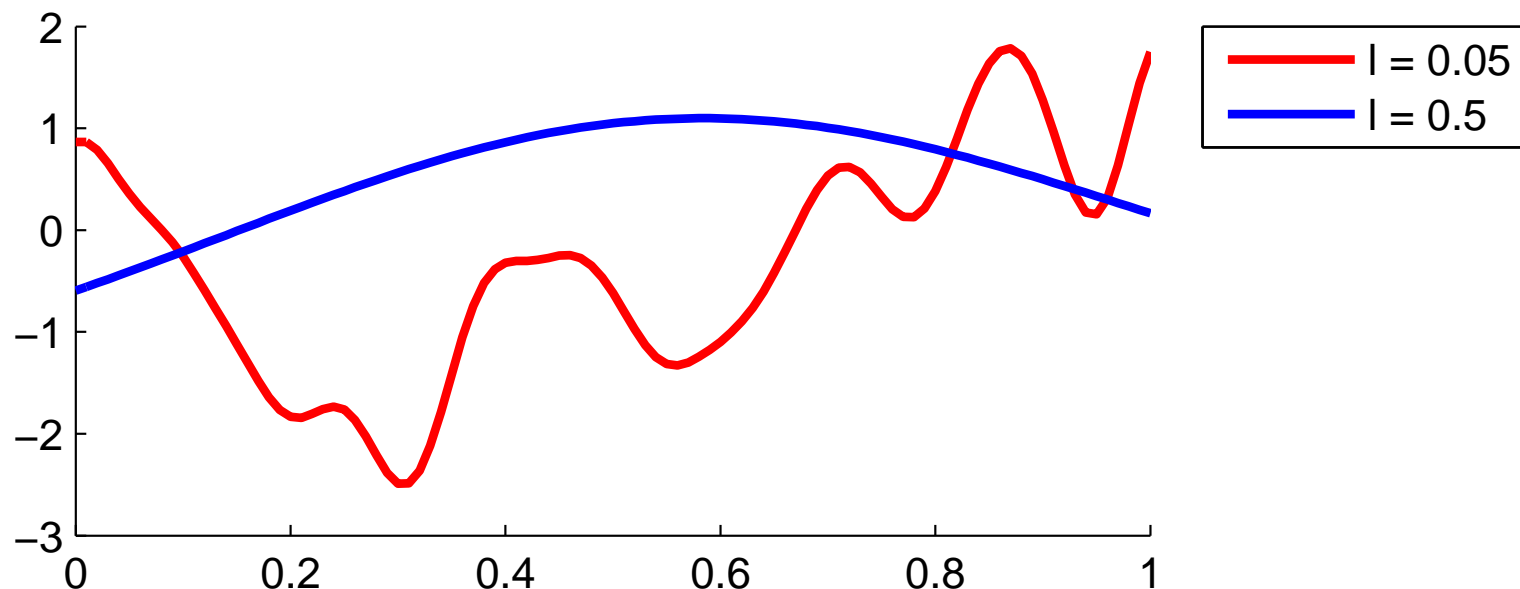


Meaning of hyper-parameters

The ℓ_d parameters give the overall lengthscale in dimension-d

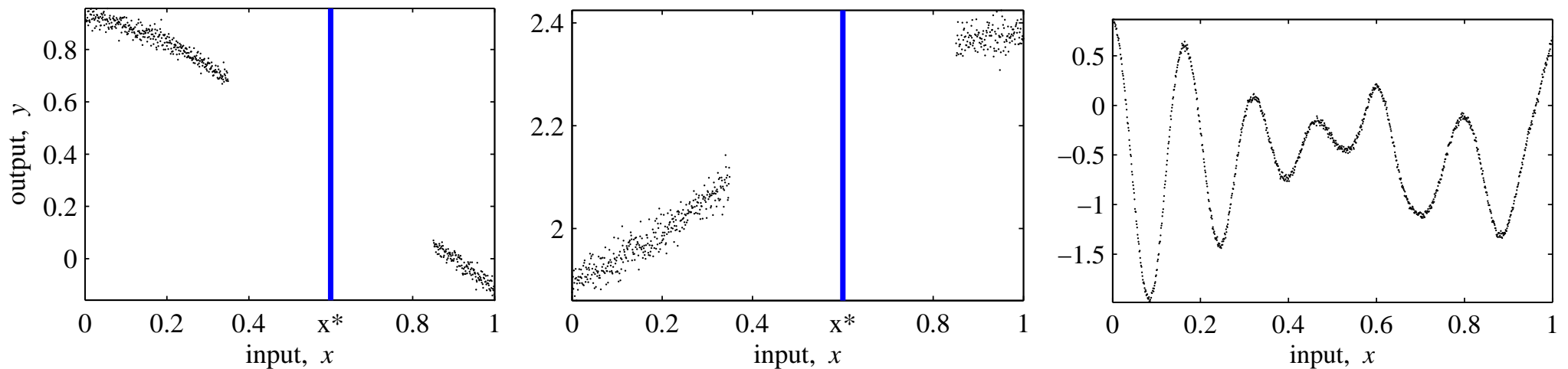
$$k(\mathbf{x}_i, \mathbf{x}_j) = \sigma_f^2 \exp\left(-\frac{1}{2} \sum_{d=1}^D (x_{d,i} - x_{d,j})^2 / \ell_d^2\right),$$

Typical distance between peaks $\approx \ell$



Typical GP lengthscales

What is the covariance matrix like? Consider 1D problems:



- Zeros in the covariance would \Rightarrow marginal independence
- Short length scales usually don't match my beliefs
- Empirically, I often learn $\ell \approx 1$ giving a dense K

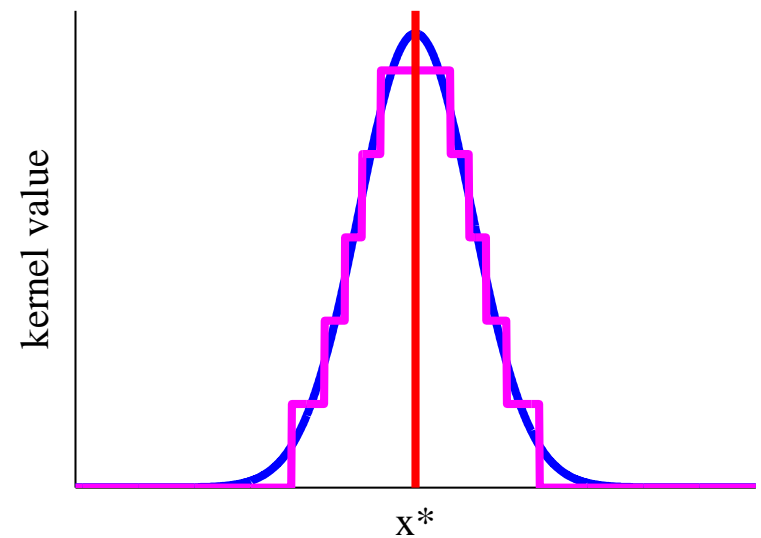
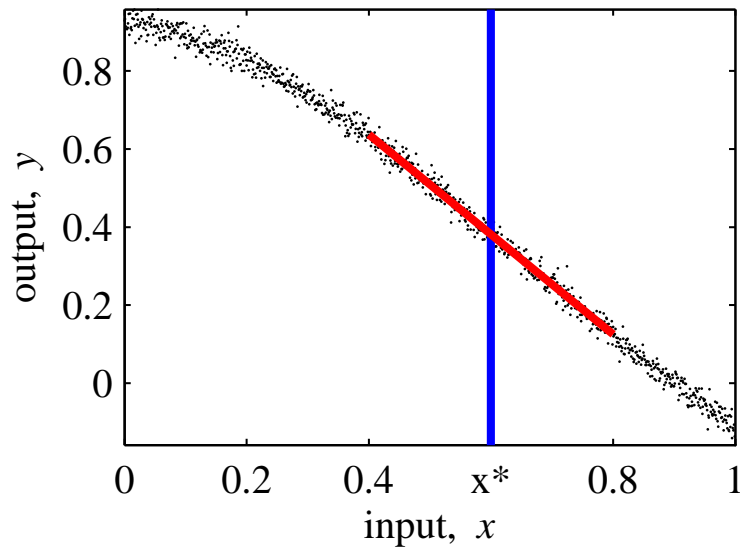
Common exceptions:

Time series data, ℓ small. Irrelevant dimensions, ℓ large.

In high dimensions, can have $K_{ij} \approx 0$ with $\ell \approx 1$.

What GPs are not

Locally-Weighted Regression weights points with a kernel before fitting a simple model

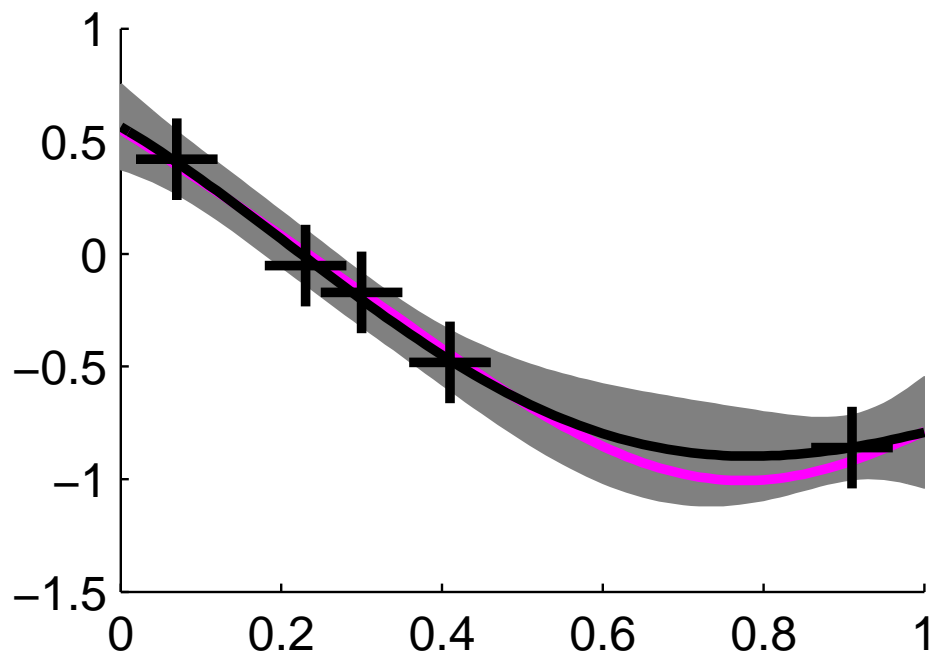


Meaning of kernel zero here: \approx *conditional* dependence.

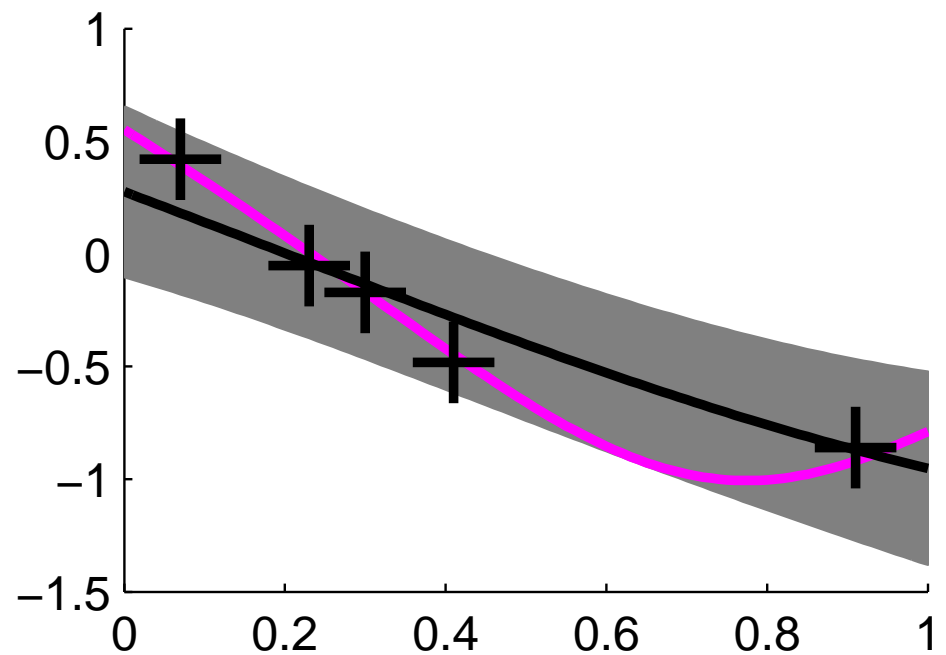
Unlike GP kernel: a) shrinks to small ℓ with many data points; b) does not need to be positive definite.

Effect of hyper-parameters

Different (SE) kernel parameters give different explanations of the data:



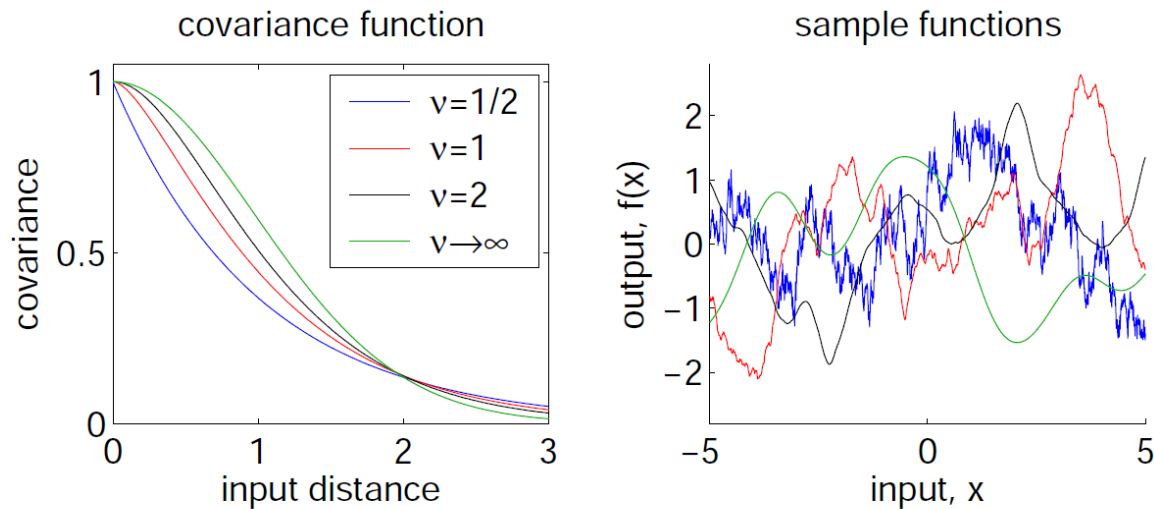
$$\ell = 0.5, \quad \sigma_n = 0.05$$



$$\ell = 1.5, \quad \sigma_n = 0.15$$

Other kernels

The squared-exponential kernel produces *very* smooth functions. For some problems the Matérn covariances functions may be more appropriate:



Periodic kernels are available, and some that vary their noise and lengthscales across space.

Kernels can be combined in many ways (Bishop p296). For example, add kernels with long and short lengthscales

The (marginal) likelihood

The probability of the data is just a Gaussian:

$$\log P(\mathbf{y}|X, \theta) = -\frac{1}{2}\mathbf{y}^\top M^{-1}\mathbf{y} - \frac{1}{2}\log |M| - \frac{n}{2}\log 2\pi$$

This is the likelihood of the kernel and its hyper-parameters, which are $\theta = \{\ell, \sigma_n, \dots\}$. This can be used to choose amongst kernels.

Gradients of the likelihood wrt the hyper-parameters can be computed to find (local) maximum likelihood fits.

Because the GP can be viewed as having an infinite number of weight parameters that have been integrated out, this is often called the *marginal* likelihood.

Learning hyper-parameters

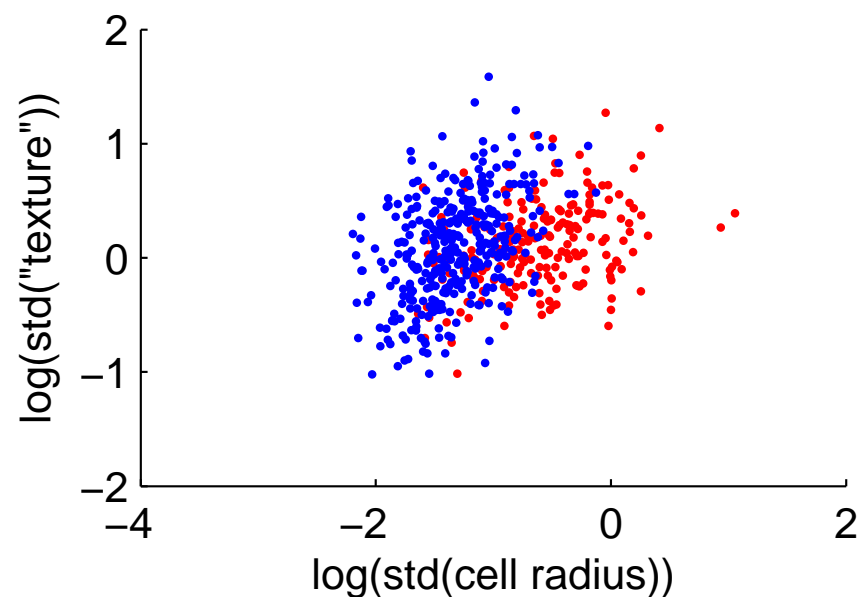
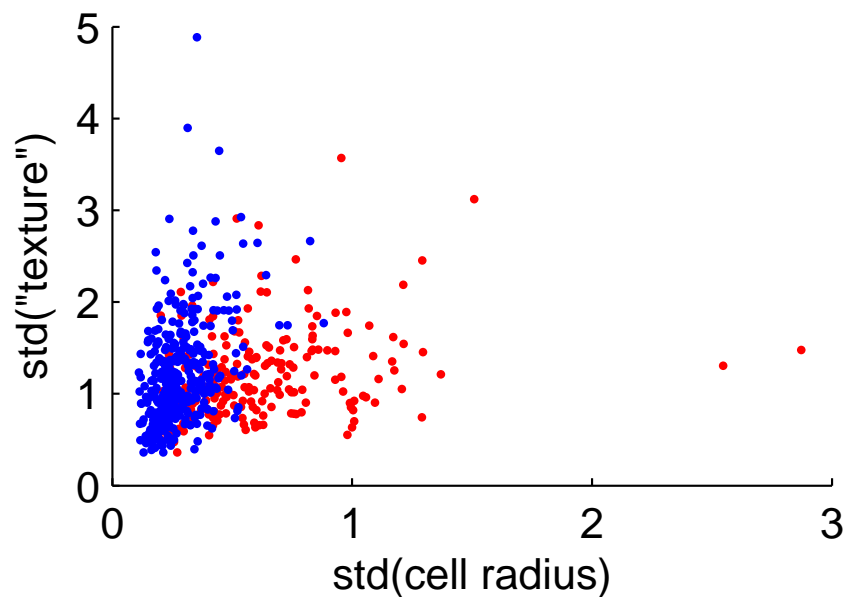
The **fully Bayesian solution** computes the function posterior:

$$P(f_*|\mathbf{y}, X) = \int d\theta P(f_*|\mathbf{y}, X, \theta)P(\theta|\mathbf{y}, X)$$

The first term in the integrand is tractable.

The second term is the posterior over hyper-parameters. This can be sampled using Markov chain Monte Carlo to **average predictions over plausible hyper-parameter settings**.

Log-transform +ve inputs



(Wisconsin breast cancer data from the UCI repository)

Positive quantities are often highly skewed

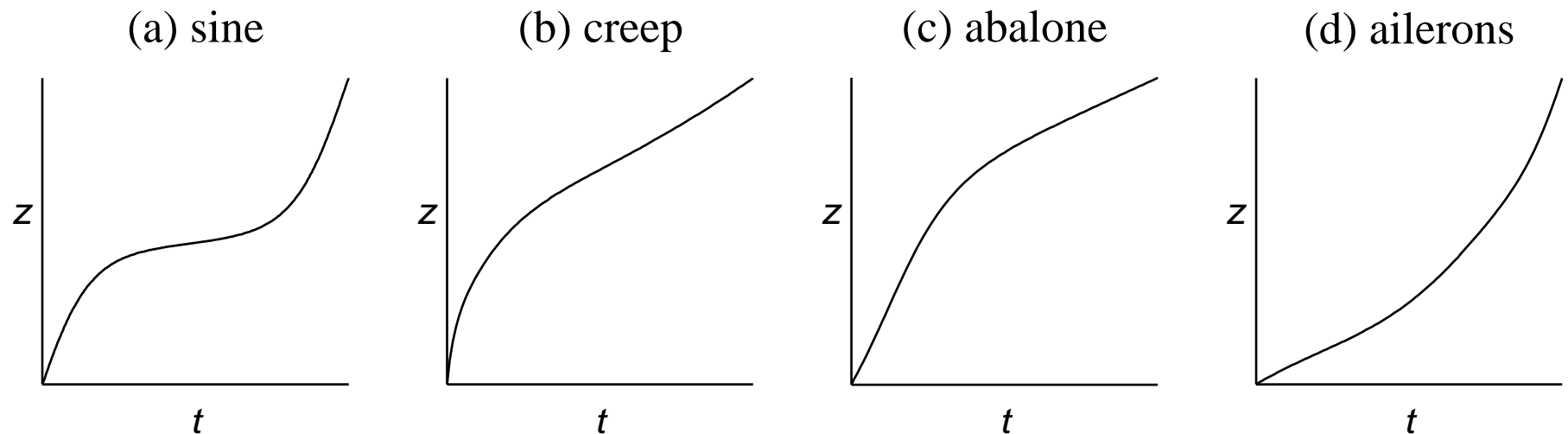
The log-domain is often a much more natural space

A better transformation could be learned:

Schmidt and O'Hagan, JRSSB, 65(3):743–758, (2003).

Log-transform +ve outputs

Warped Gaussian processes, Snelson et al. (2003)

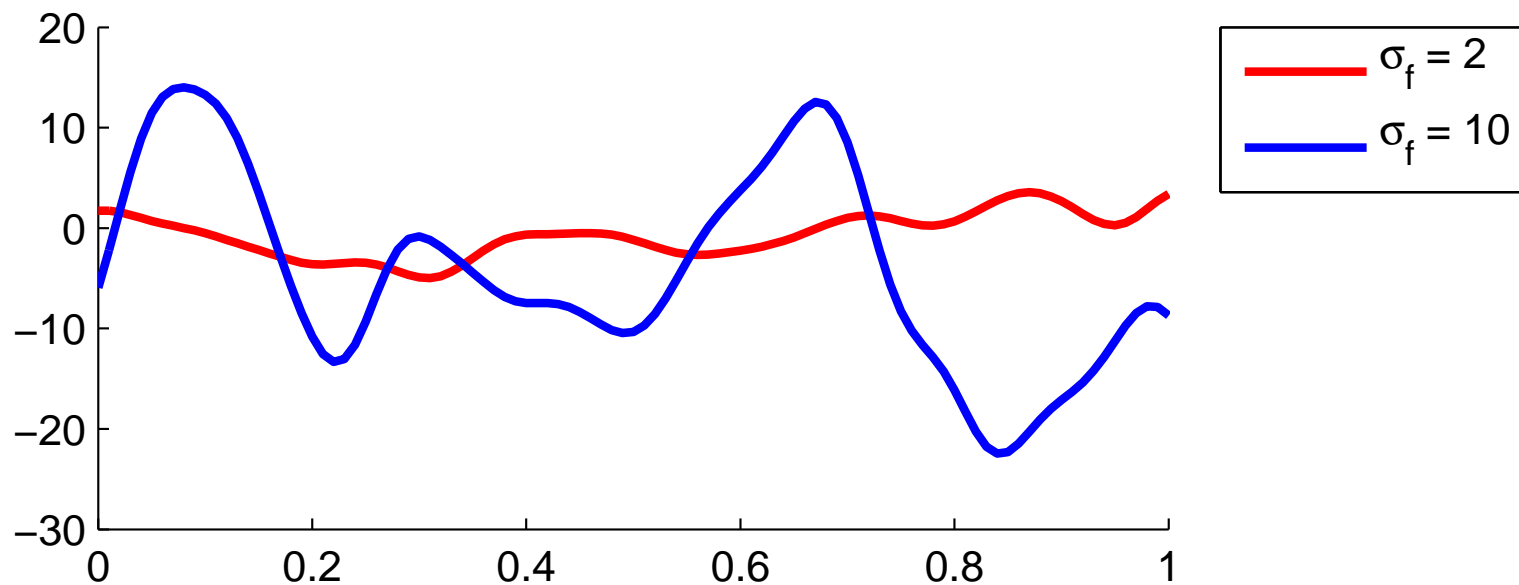


Learned transformations for positive data were log-like, so this is sometimes a good guess.

However, other transformations (or none at all) are sometimes the best option.

Mean function

Using $\mathbf{f} \sim \mathcal{N}(\mathbf{0}, K)$ is common



If your data is not zero-mean this is a poor model.
Center your data, or use a parametric mean function $m(\mathbf{x})$.

We can do this: the posterior is a GP with non-zero mean.

Other tricks

To set initial hyper-parameters, **use domain knowledge wherever possible**. Otherwise. . .

Standardize input data and set lengthscales to ~ 1 .

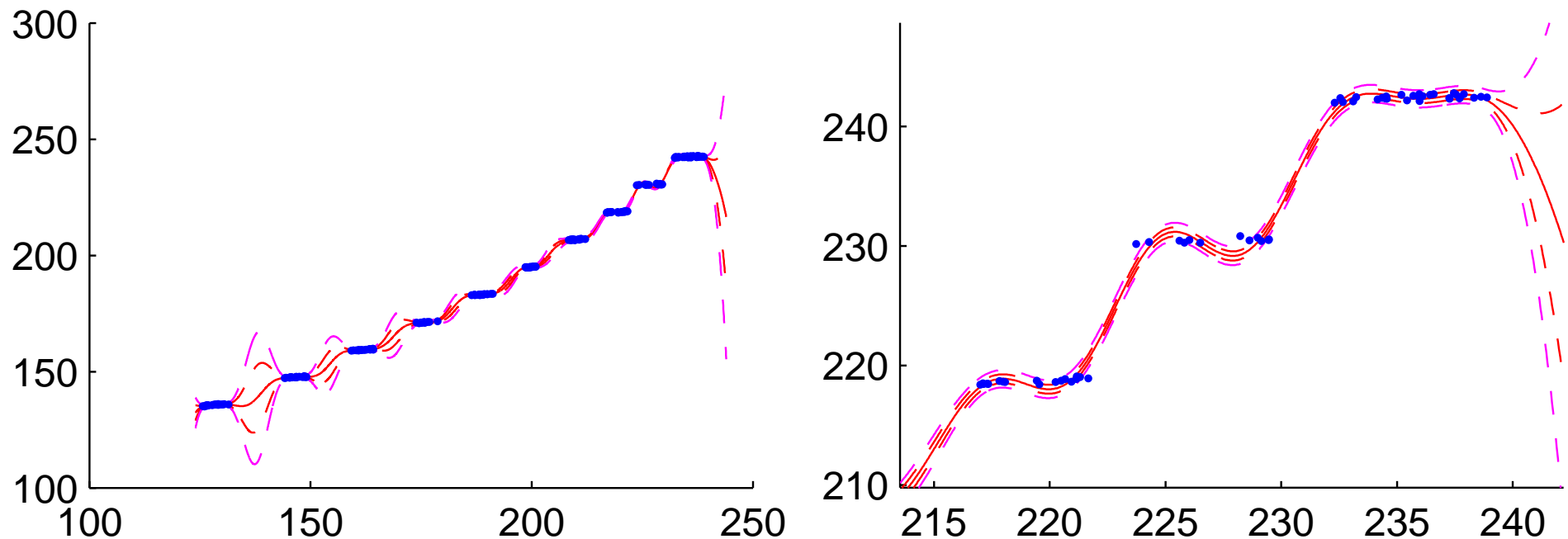
Standardize targets and set function variance to ~ 1 .

Often useful: **set initial noise level high**, even if you think your data have low noise. The optimization surface for your other parameters will be easier to move in.

If optimizing hyper-parameters, (as always) random restarts or other tricks to **avoid local optima** are advised.

Real data can be nasty

A projection of a robot arm problem:



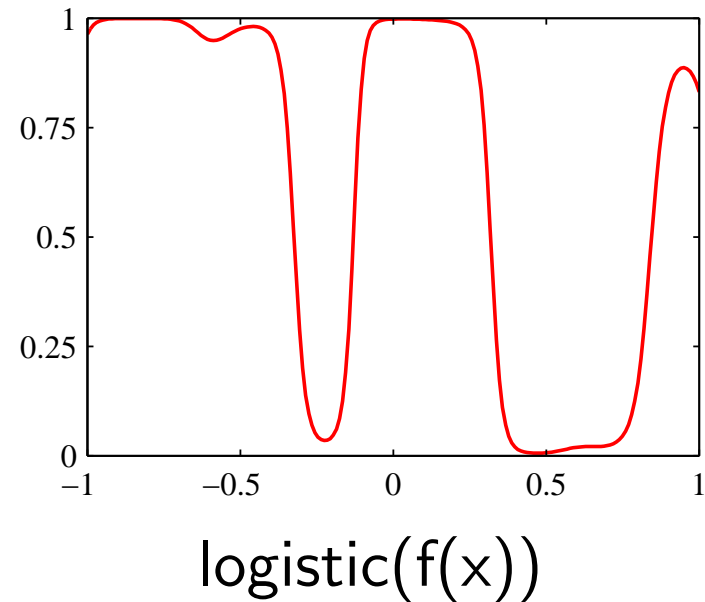
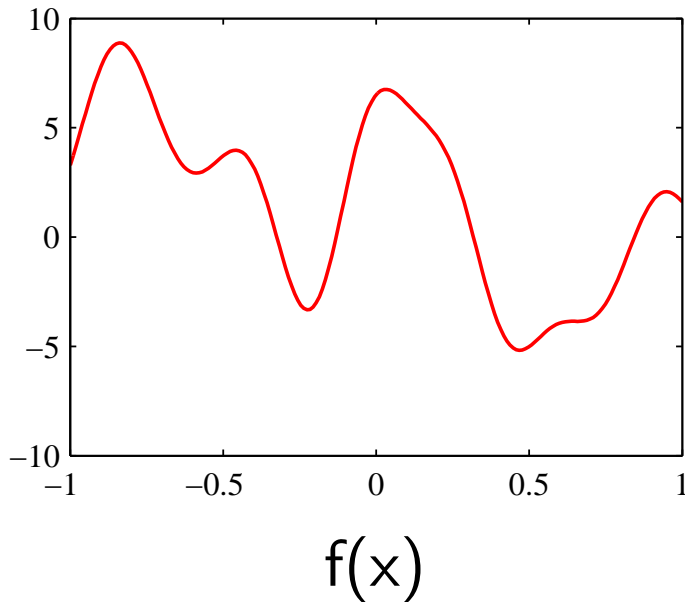
Common artifacts: thresholding, jumps, clumps, $k_i^n k_s$

How might we fix these problems? [discussion]

Classification

Special case of a non-Gaussian noise model

Assume $y_i \sim \text{Bernoulli}(\text{sigmoid}(f_i))$



MCMC can be used to sum over the latent function values.
EP (Expectation Propagation) also works very well.

Regressing on the labels

If we give up on a Bayesian modelling interpretation, **we could just apply standard GP regression code** on binary classification data with $y \in \{-1, +1\}$.

The sign of the mean function is a reasonable hard classifier. Asymptotically the posterior function will be peaked around $f(\mathbf{x}) = 2p(\mathbf{x}) - 1$.

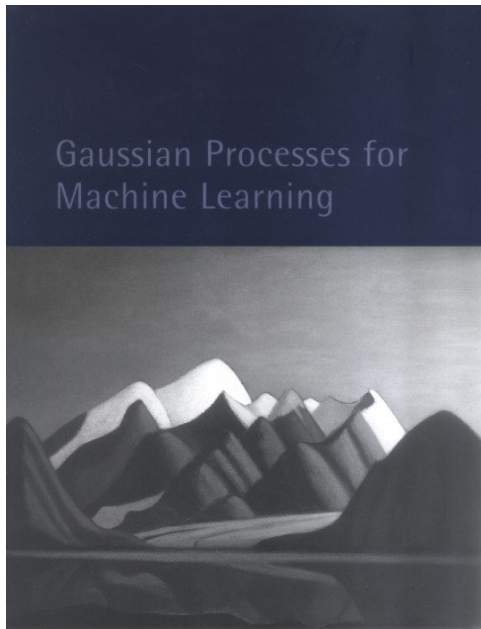
Multiway classification: regressing $y \in \{1, 2, \dots, C\}$ would be a bad idea. Instead, train C “one-against all” classifiers and pick class with largest mean function.

Not really Gaussian process modelling any more:
this is just regularized least squares fitting

Take-home messages

- **Simple to use:**
 - Just matrix operations (if likelihoods are Gaussian)
 - Few parameters: *relatively* easy to set or sample over
 - Predictions are often very good
- **No magic bullet:** best results need (at least) careful data scaling, which could be modelled or done by hand.
- **The need for approximate inference:**
 - Sometimes Gaussian likelihoods aren't enough
 - $\mathcal{O}(n^3)$ and $\mathcal{O}(n^2)$ costs are bad news for big problems

Further reading



Carl Edward Rasmussen and Christopher K. I. Williams

Many more topics and code:

<http://www.gaussianprocess.org/gpml/>

MCMC inference for GPs is implemented in FBM:

<http://www.cs.toronto.edu/~radford/fbm.software.html>

Gaussian processes for ordinal regression, Chu and Ghahramani, *JMLR*, 6:1019–1041, 2005.

Flexible and efficient Gaussian process models for machine learning, Edward L. Snelson, PhD thesis, UCL, 2007.