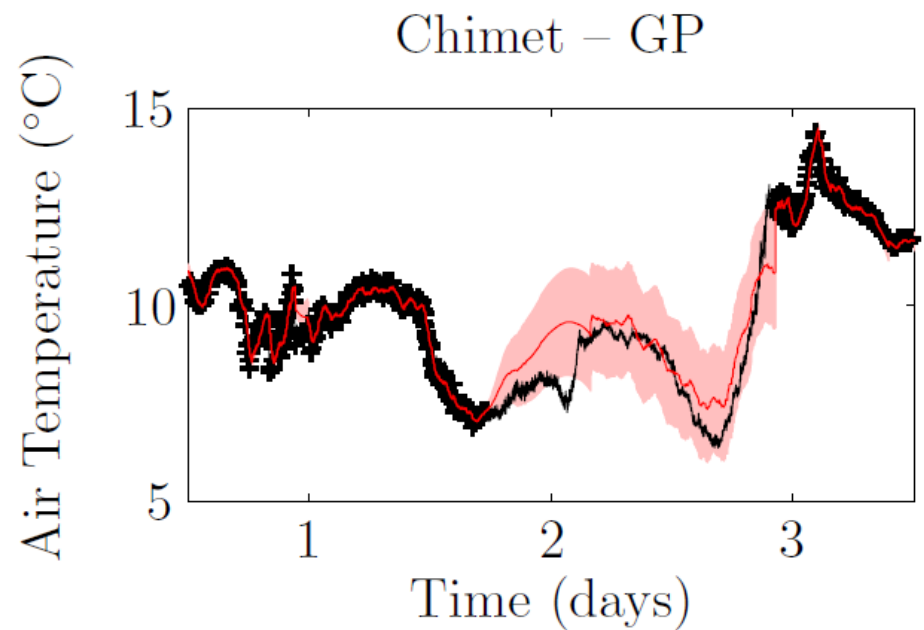
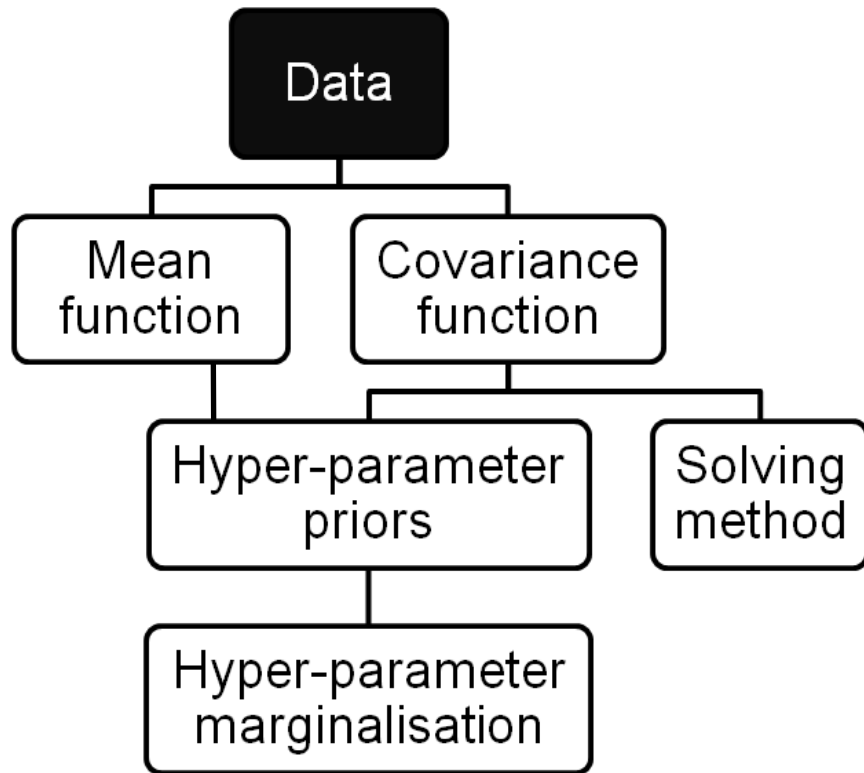


An Introduction to Fitting Gaussian Processes to Data

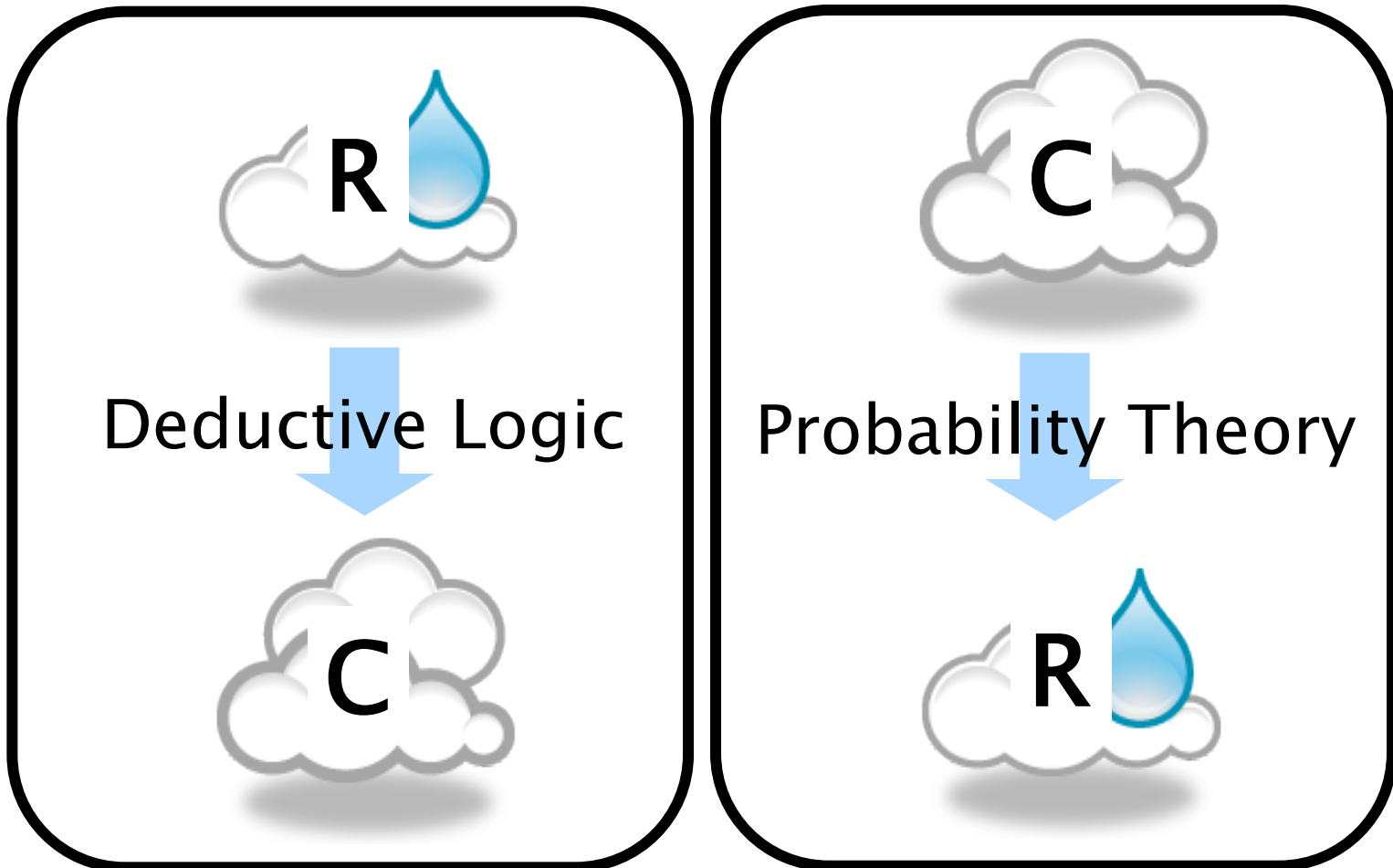
Michael Osborne

Pattern Analysis and Machine Learning Research Group
Department of Engineering
University of Oxford

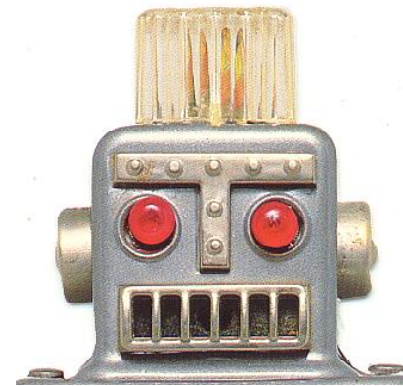
You will learn how to fit a **Gaussian process** to data.



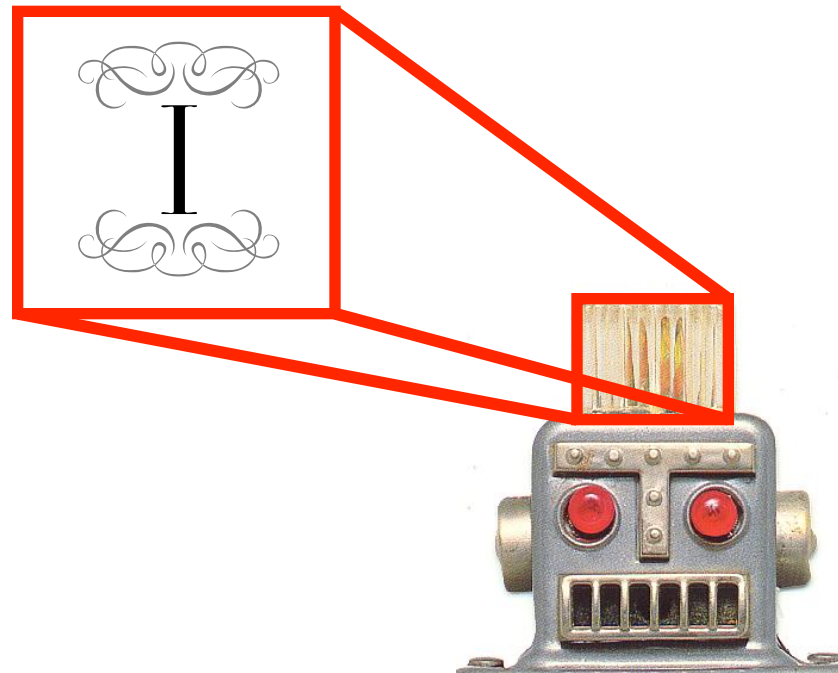
Probability theory represents an extension of traditional logic, allowing us to reason in the face of uncertainty.



A probability is a **degree of belief**. This might be held by any agent – a human, a robot, a sensor, etc.

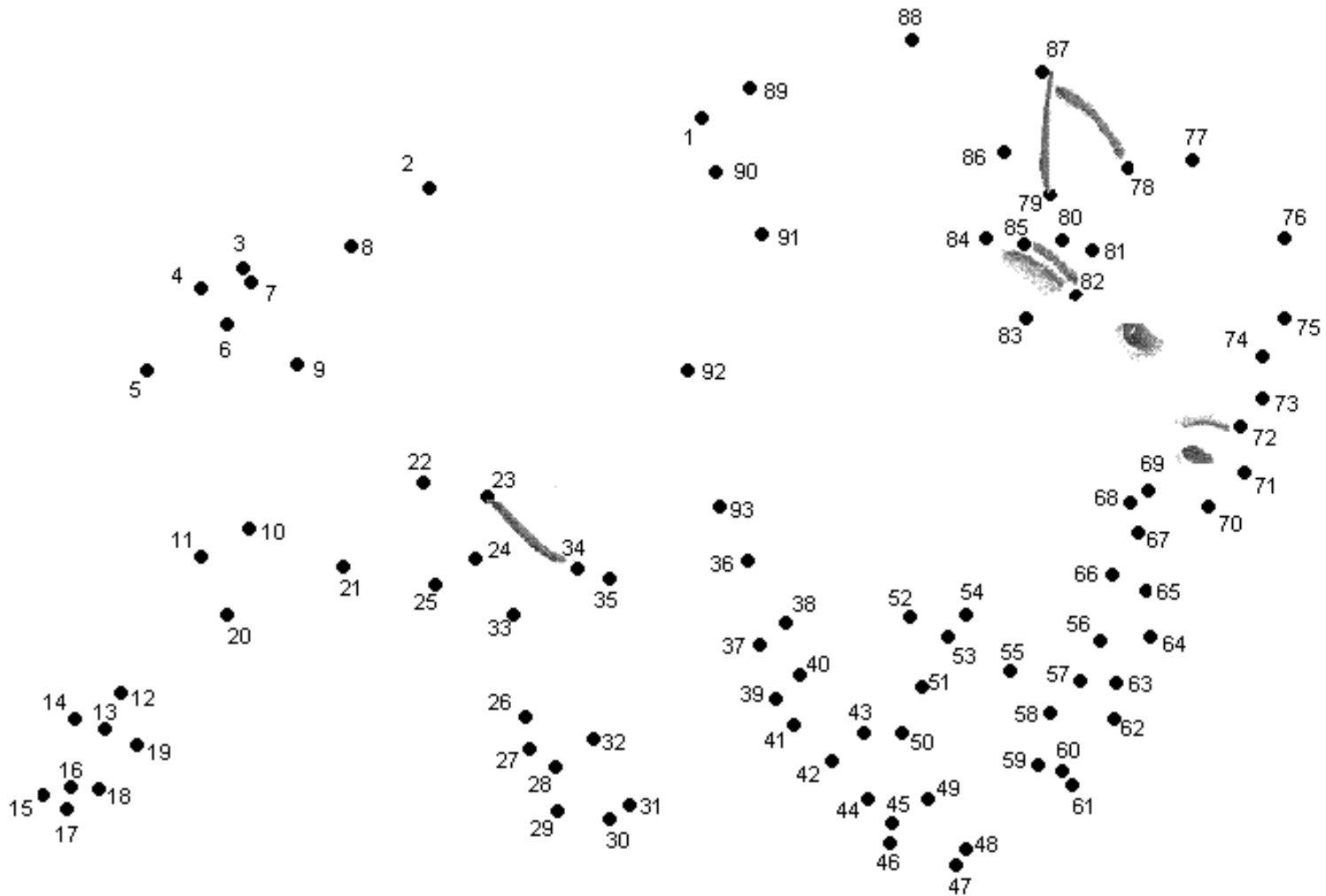


I is the totality of an agent's **prior information**.
An agent is defined by *I*.

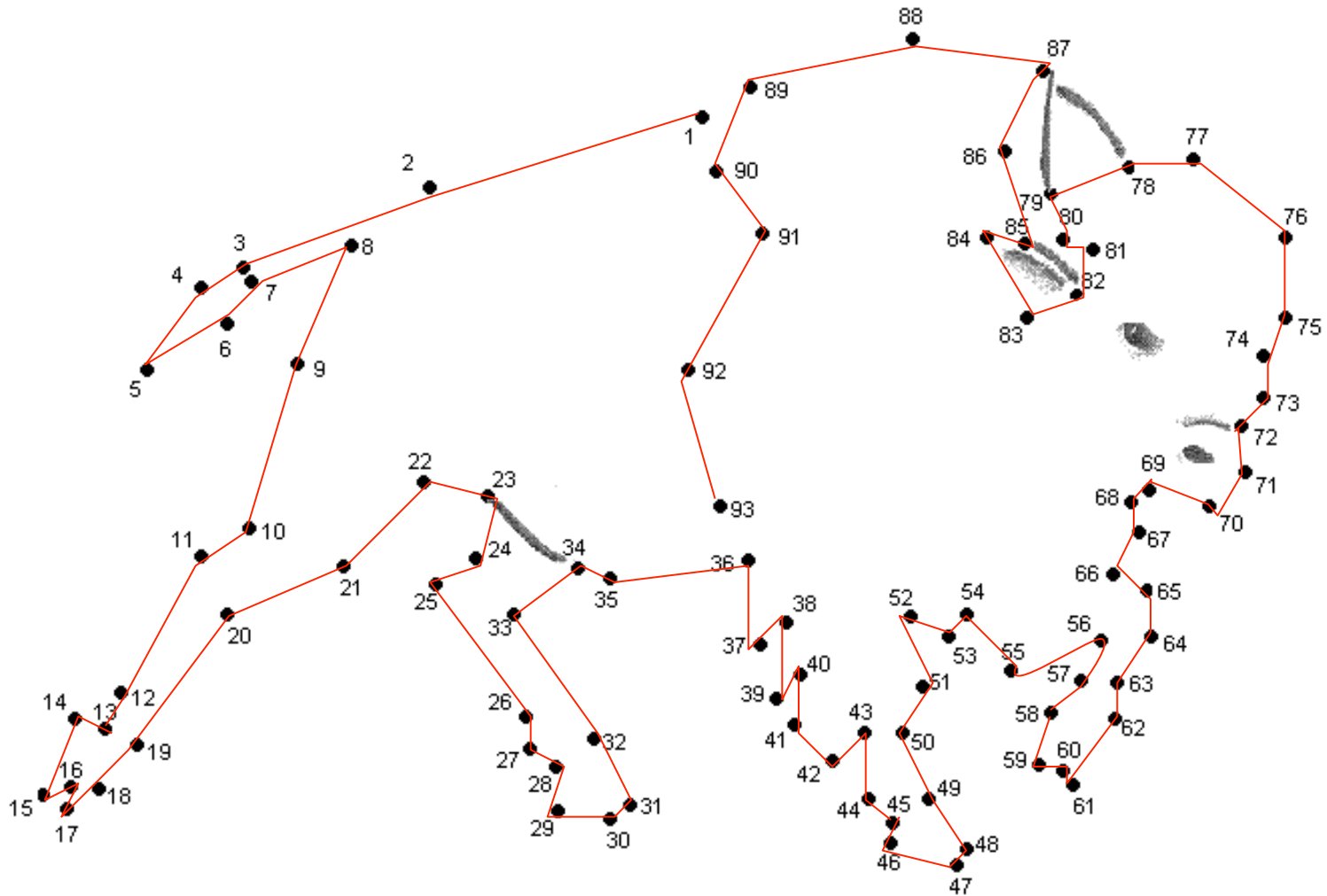


We define our agents so that they can perform difficult inference for us.

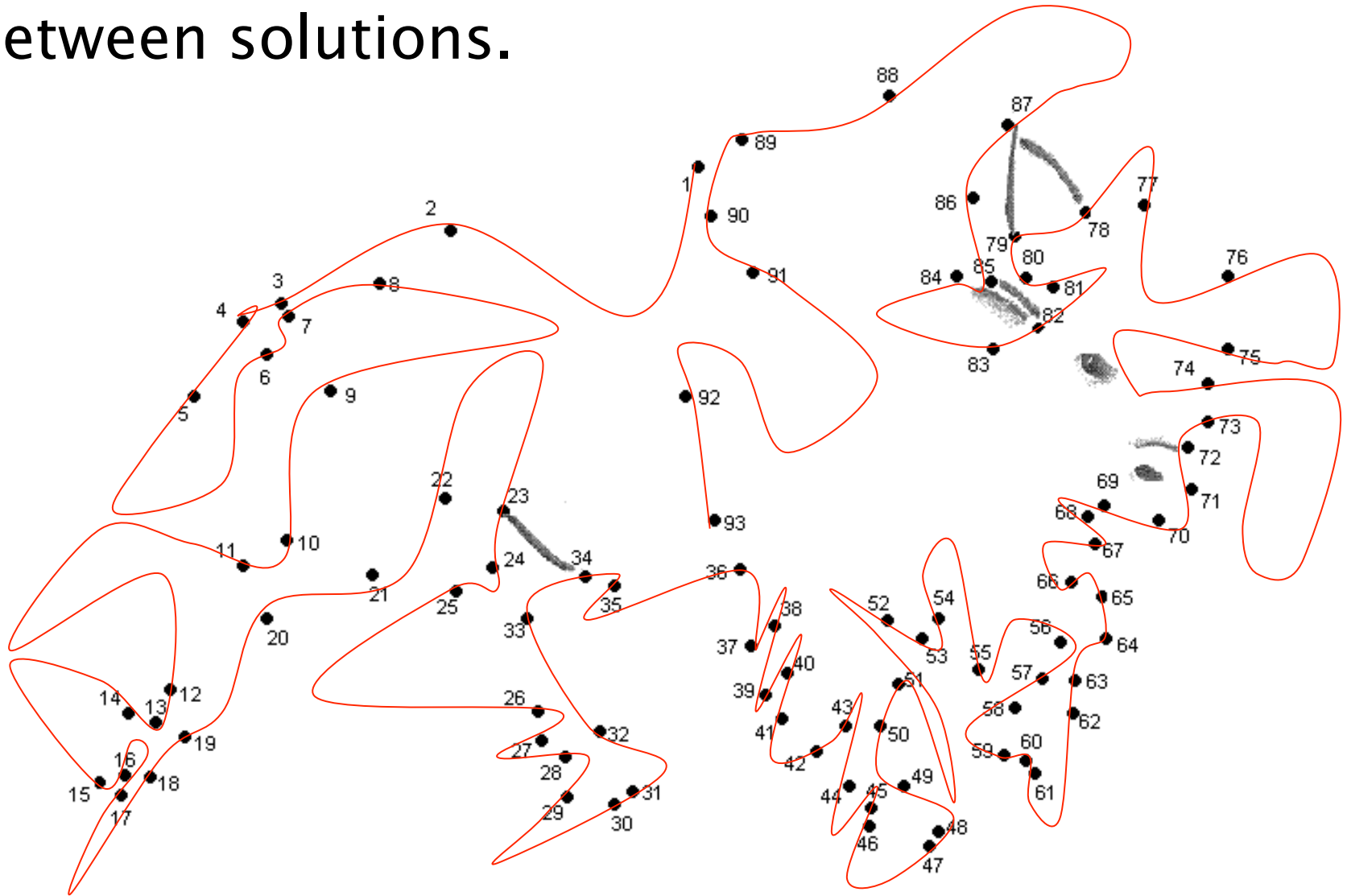
A **dot-to-dot** is an inference problem.



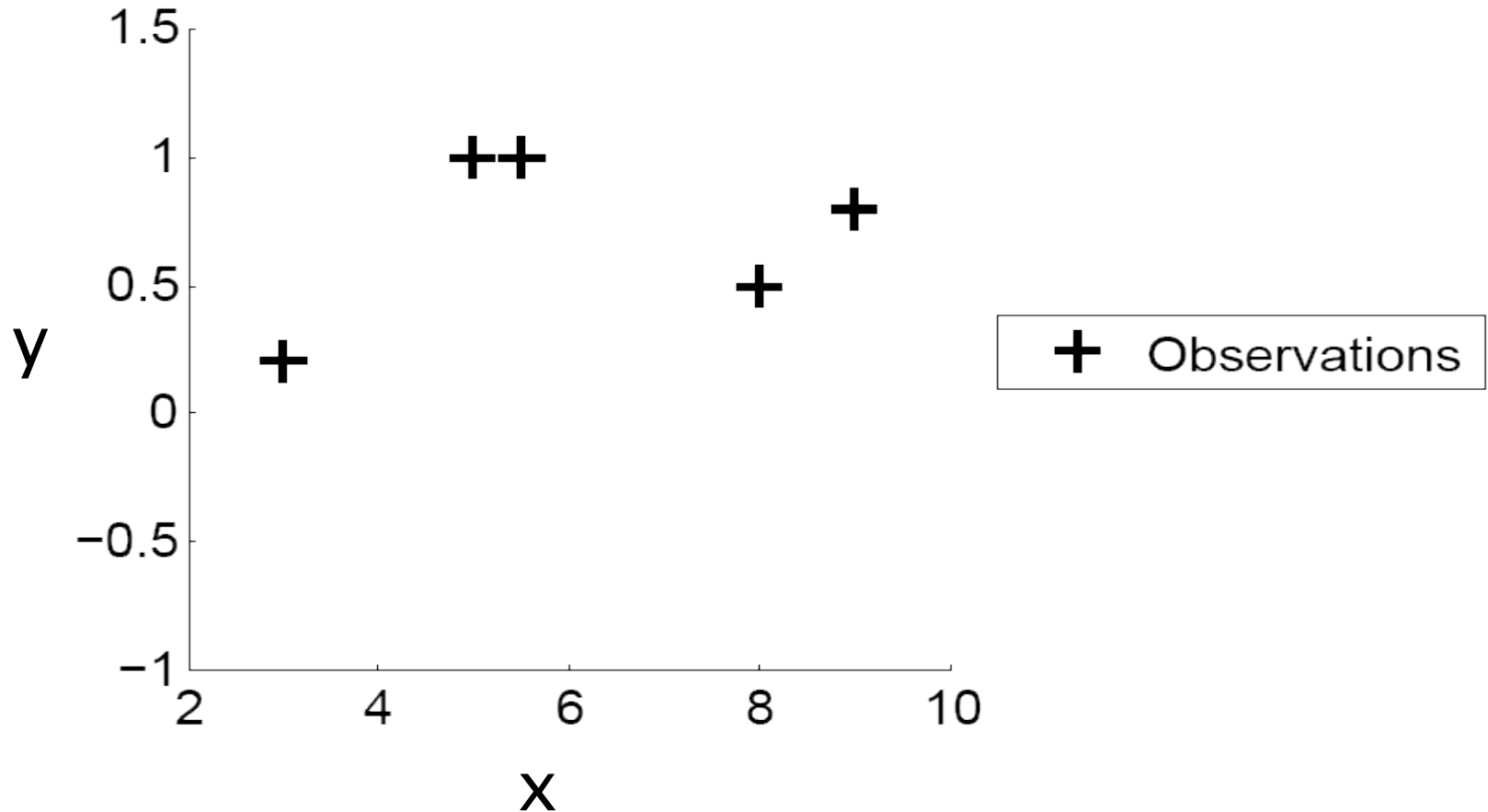
A dot-to-dot is a problem with **many possible solutions.**



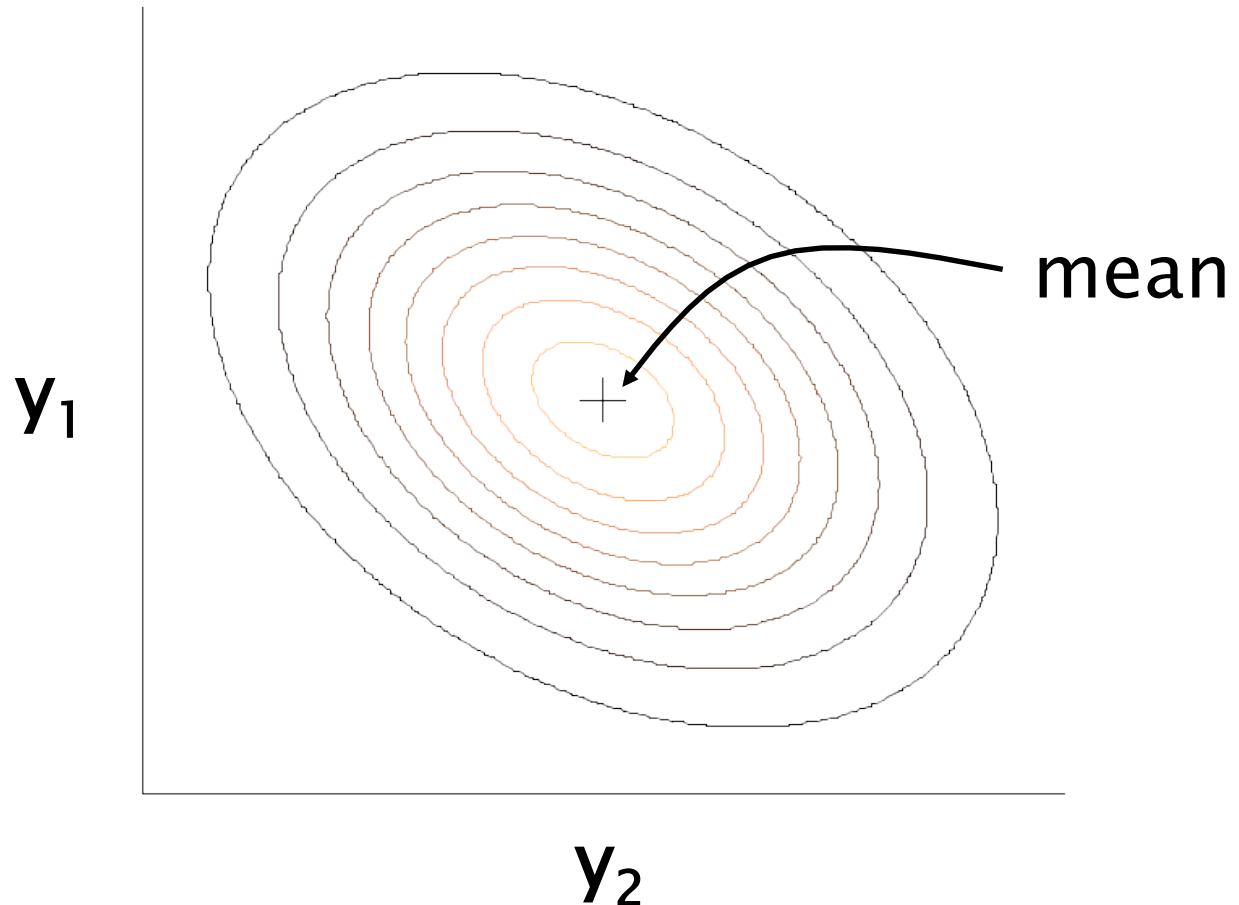
Our **prior** information allows us to **discriminate** between solutions.



A dot-to-dot requires us to do **inference** about **functions**, as can be seen more clearly in one dimension.



The multivariate **Gaussian** distribution is wonderful; it is defined by a mean vector (which simply gives the centrepoint) and covariance matrix.



The **covariance** K must be a **positive semi-definite matrix**; so for any vector x , $x^T K x \geq 0$.
This implies that:

K must be symmetric.

The diagonal of K
must be positive.

$K = R^T R$ for some
upper triangular R .

The eigenvalues of K
are all ≥ 0 .

$$\begin{pmatrix} 3 & 1 & 0 & 0 \\ 1 & 3 & 2 & 0 \\ 0 & 2 & 4 & -1 \\ 0 & 0 & -1 & 3 \end{pmatrix}$$

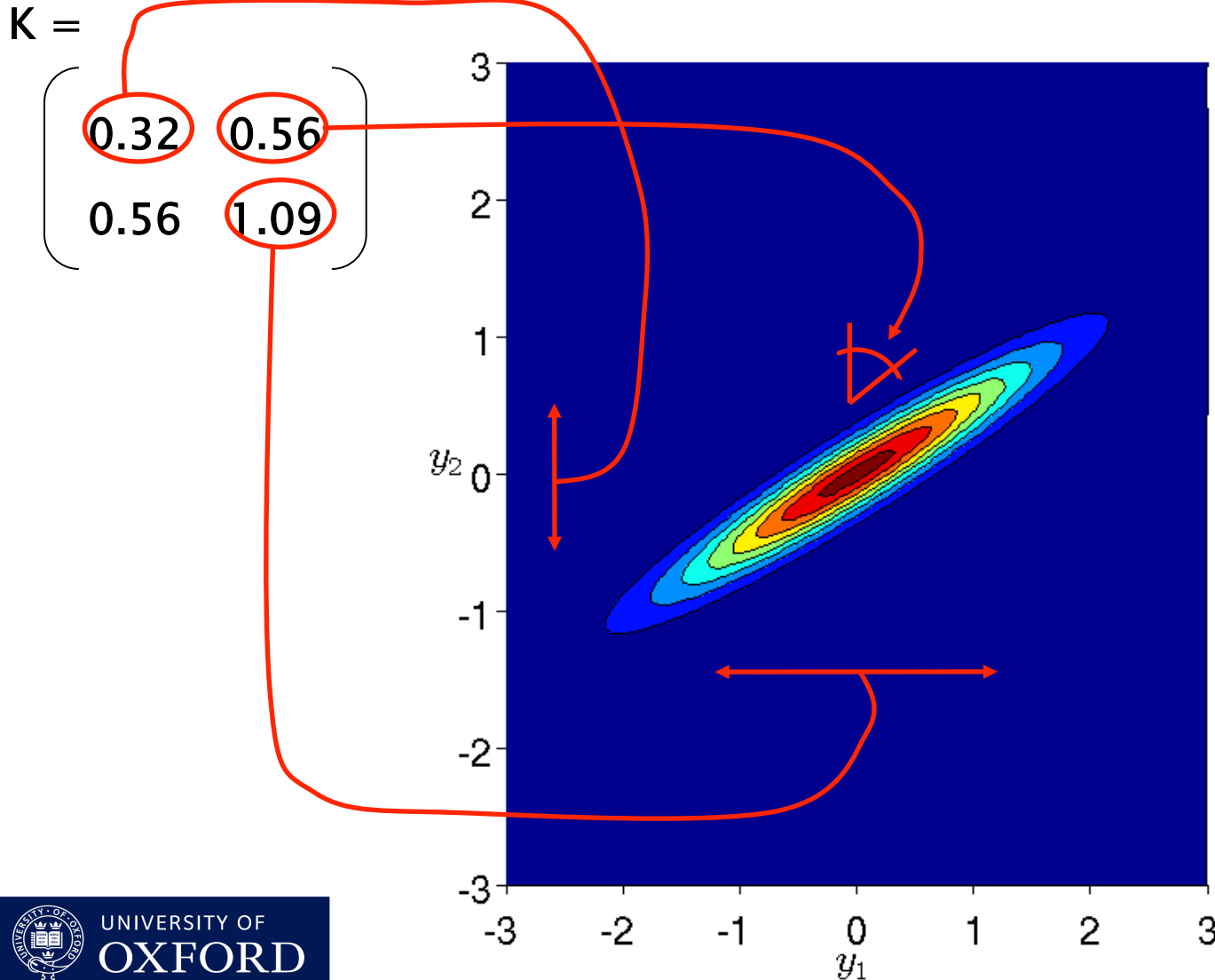
We can represent any covariance K using the **spherical parameterisation**.

$$K = R^T R$$

$$R = \begin{pmatrix} 1 & \cos(\theta_1) & \cos(\theta_2) & \cdots \\ 0 & \sin(\theta_1) & \sin(\theta_2)\cos(\theta_3) & \\ 0 & 0 & \sin(\theta_2)\sin(\theta_3) & \\ \vdots & & & \ddots \end{pmatrix} \begin{pmatrix} h_1 & 0 & 0 & \cdots \\ 0 & h_2 & 0 & \\ 0 & 0 & h_3 & \\ \vdots & & & \ddots \end{pmatrix}$$



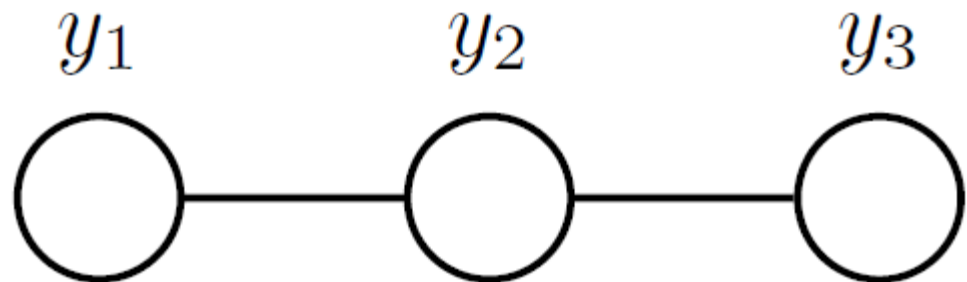
The $(i,j)^{\text{th}}$ element of the **covariance** expresses how variable i is **dependent** upon variable j .



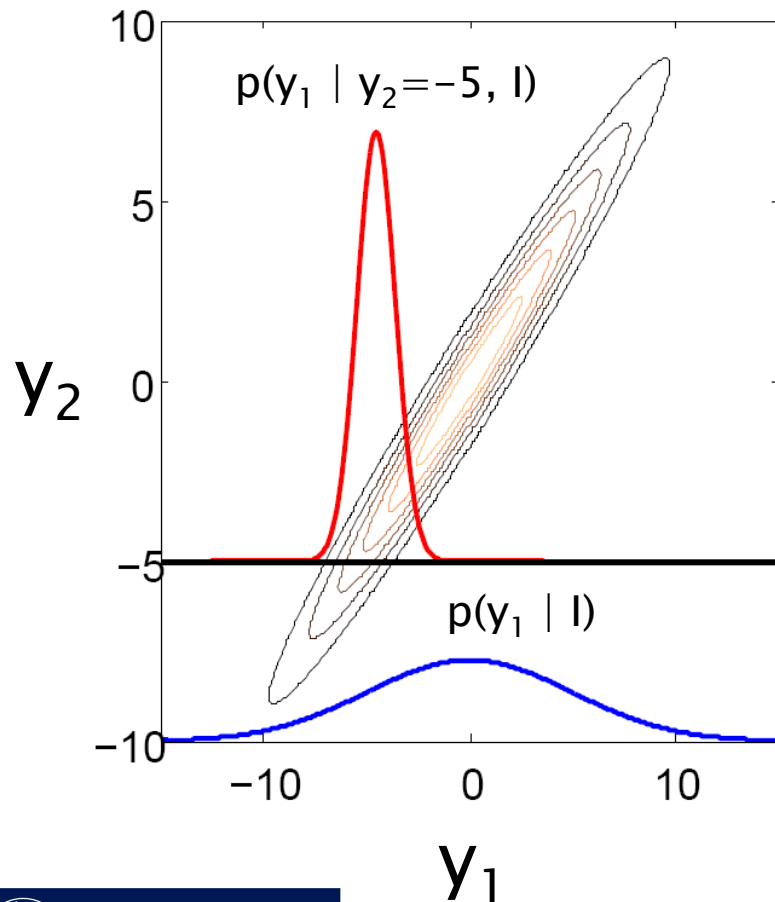
The $(i,j)^{\text{th}}$ element of the inverse covariance (*precision*) expresses how variable i is *dependent* upon variable j , conditioned on all other variables.

$$\mathbf{K} = \begin{pmatrix} 1.67 & -2 & 1.33 \\ -2 & 3 & -2 \\ 1.33 & -2 & 1.67 \end{pmatrix}$$

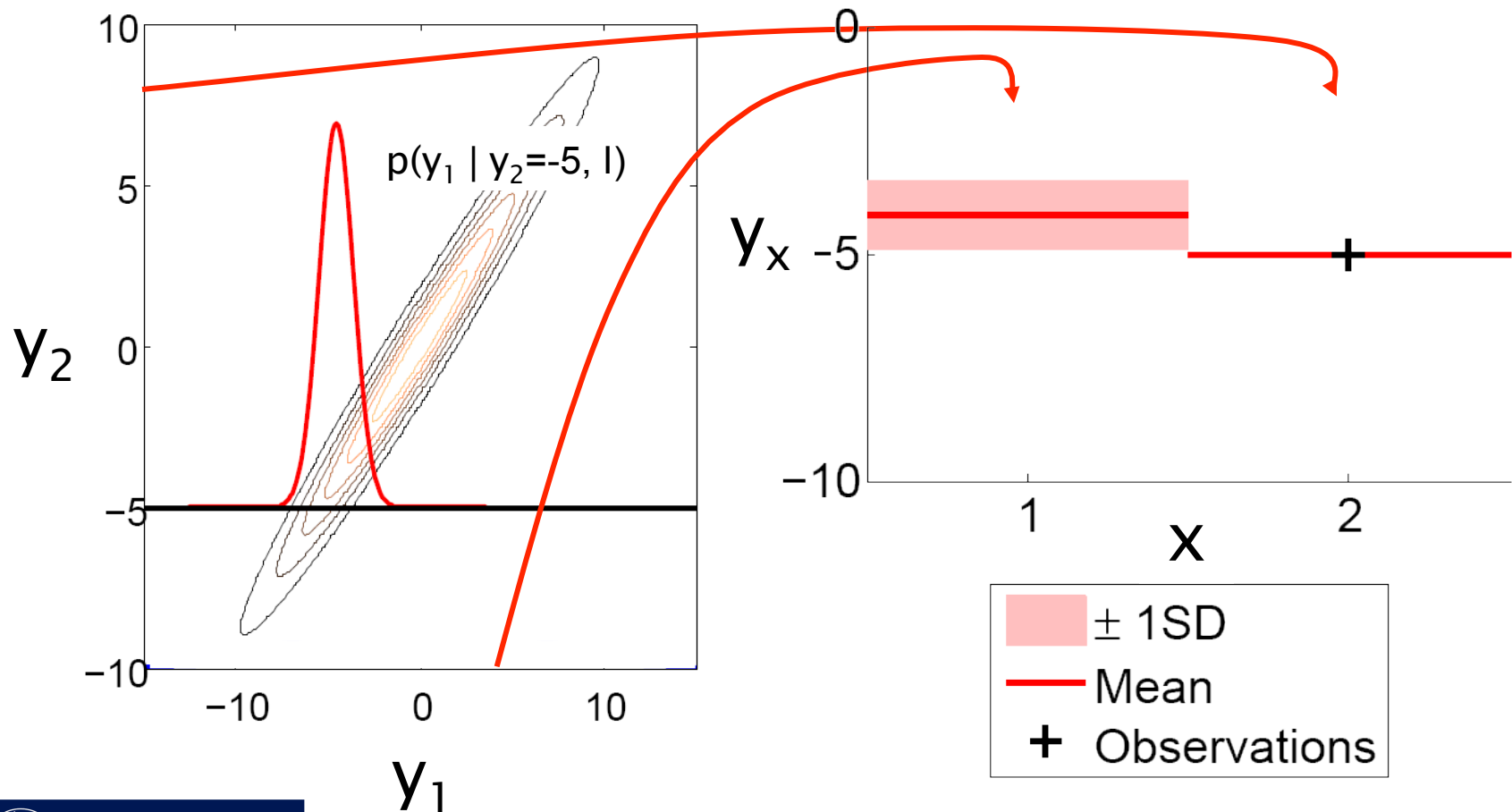
$$\mathbf{K}^{-1} = \begin{pmatrix} 3 & 2 & 0 \\ 2 & 3 & 2 \\ 0 & 2 & 3 \end{pmatrix}$$



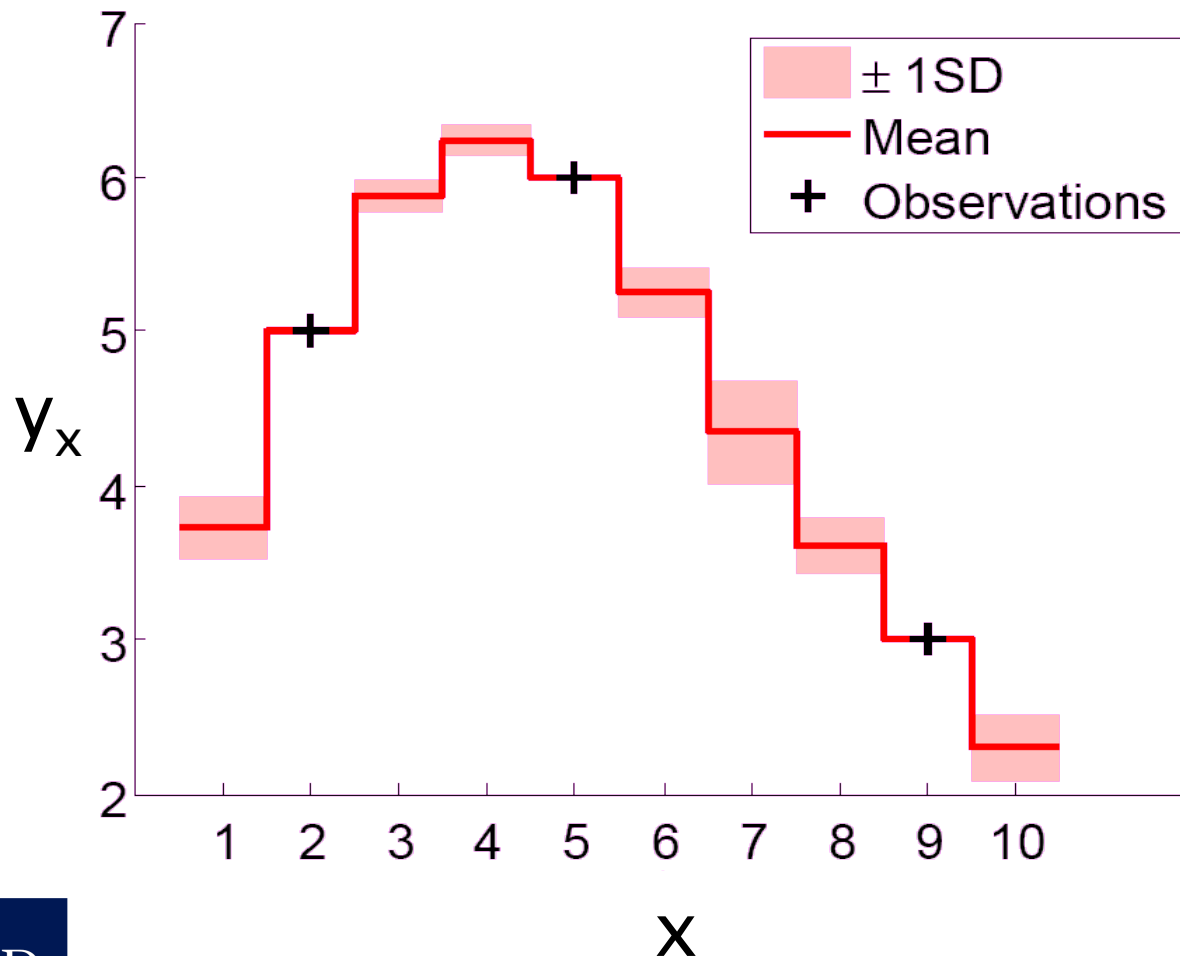
The multivariate **Gaussian** distribution allows us to produce distributions for variables conditioned on any other observed variables.



The **Gaussian** distribution allows us to produce distributions for variables conditioned on any other observed variables.



A **Gaussian process** is the generalisation of a multivariate Gaussian distribution to a potentially infinite number of variables.

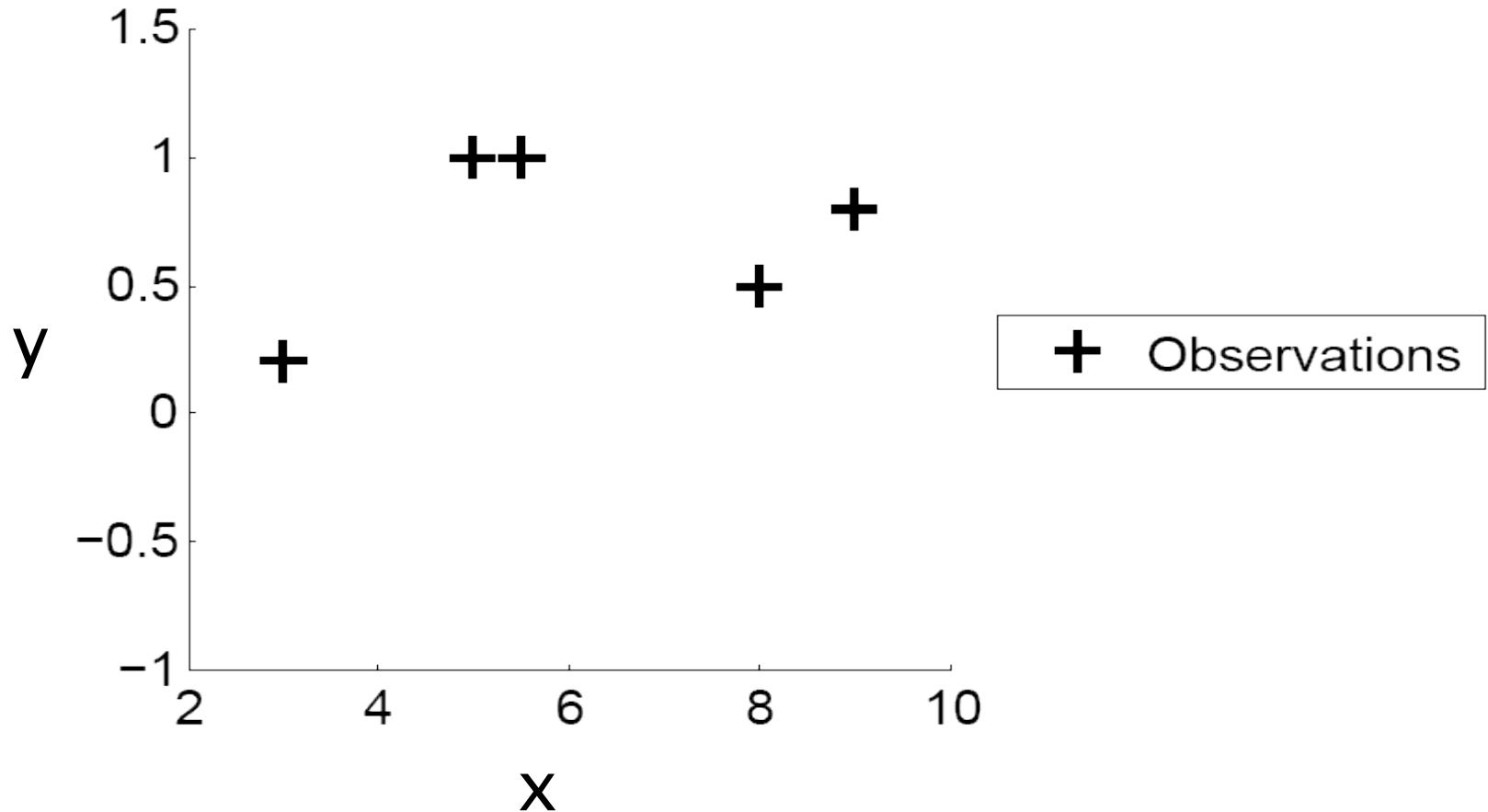


For a Gaussian process, we need to define mean and covariance **functions**, specified by hyperparameters ϕ .

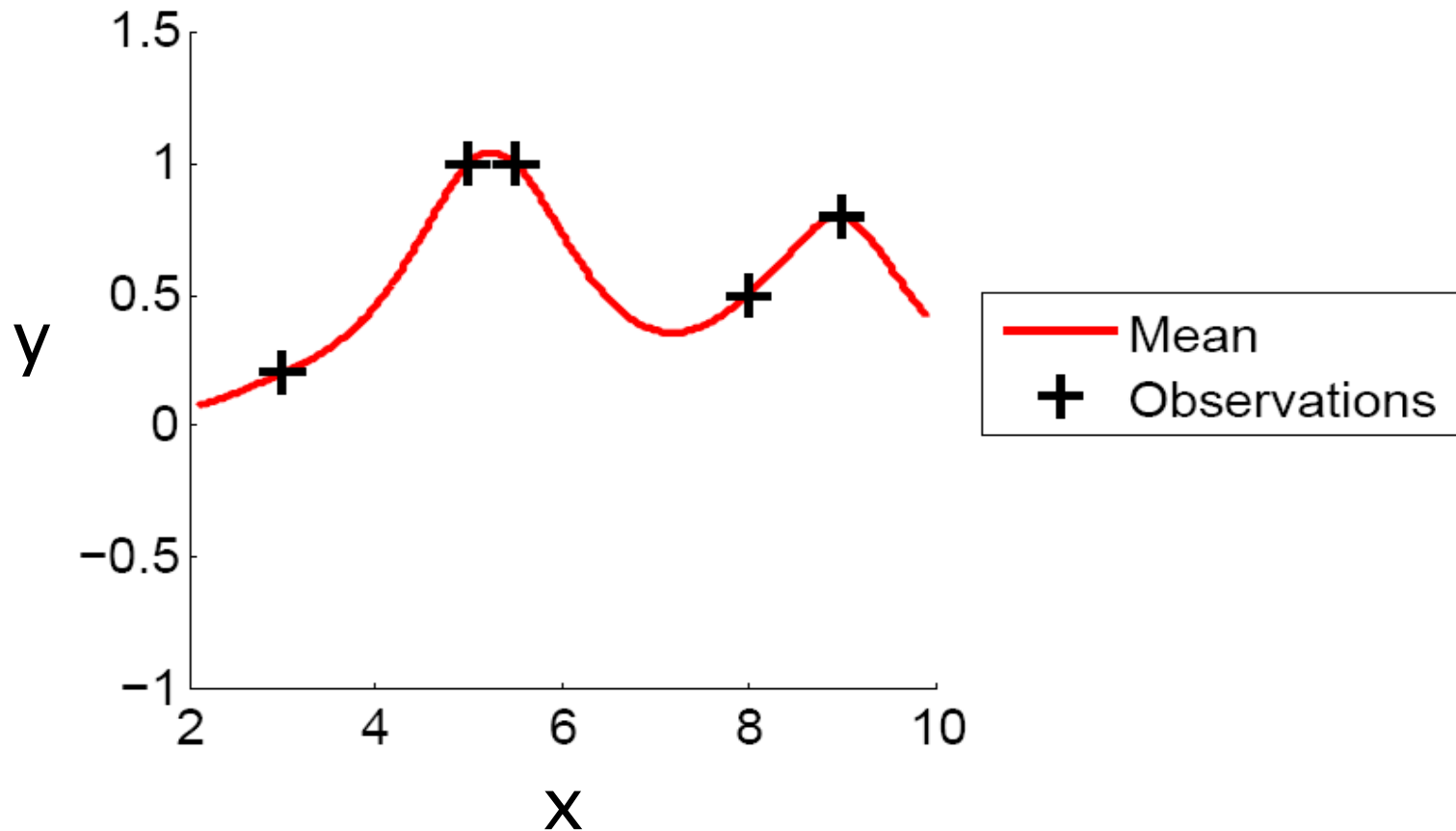
$$\begin{pmatrix} \mu_1 \\ \mu_2 \\ \mu_3 \\ \vdots \end{pmatrix} = \begin{pmatrix} \mu(x_1; \phi) \\ \mu(x_2; \phi) \\ \mu(x_3; \phi) \\ \vdots \end{pmatrix}$$

$$\begin{pmatrix} K_{11} & K_{21} & K_{13} & \cdots \\ K_{21} & K_{22} & K_{23} & \\ K_{31} & K_{32} & K_{33} & \\ \vdots & & & \ddots \end{pmatrix} = \begin{pmatrix} K(x_1, x_1; \phi) & K(x_1, x_2; \phi) & K(x_1, x_3; \phi) & \cdots \\ K(x_2, x_1; \phi) & K(x_2, x_2; \phi) & K(x_2, x_3; \phi) & \\ K(x_3, x_1; \phi) & K(x_3, x_2; \phi) & K(x_3, x_3; \phi) & \\ \vdots & & & \ddots \end{pmatrix}$$

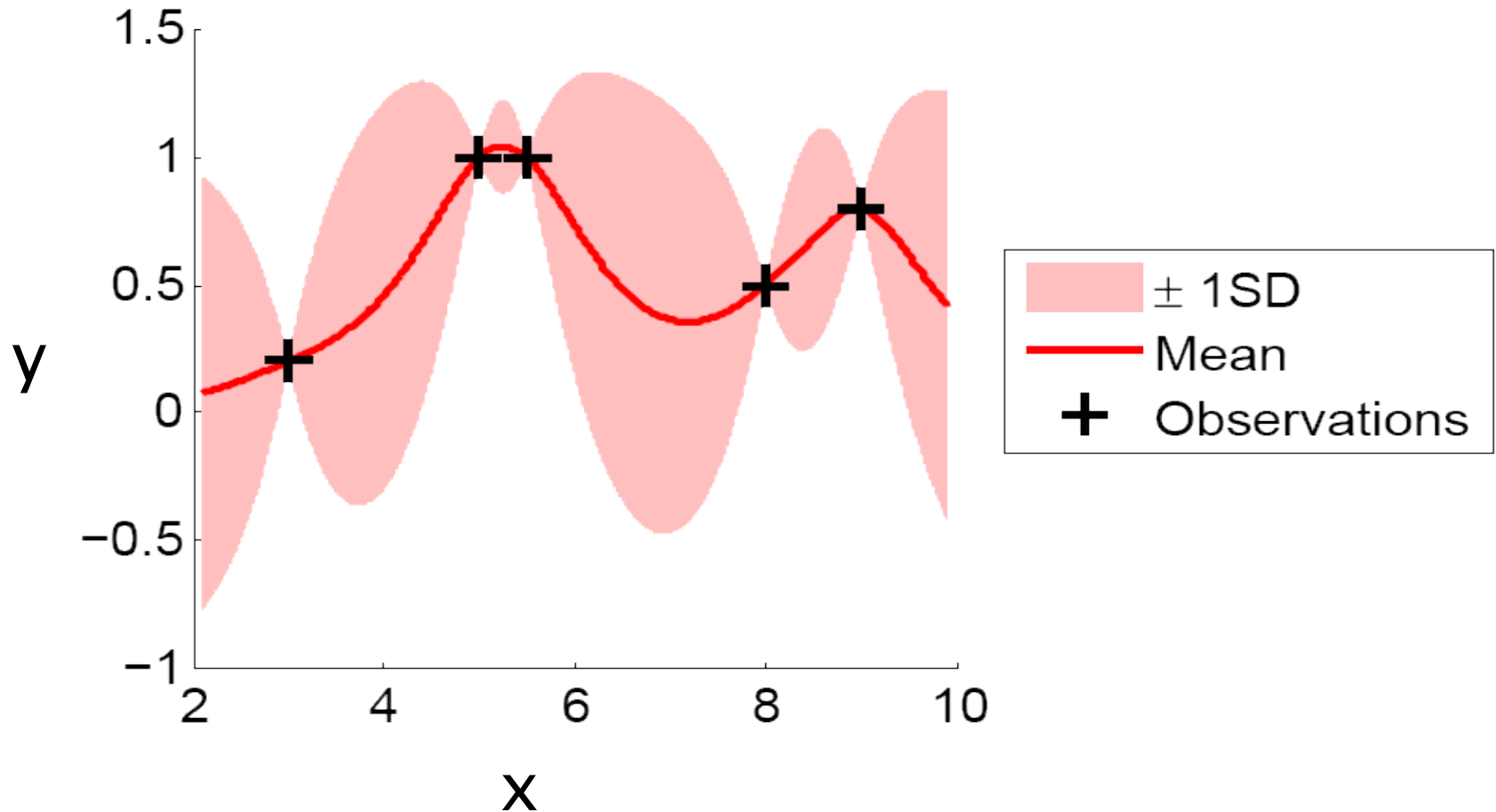
A Gaussian process represents a powerful way to perform Bayesian inference about **functions**.



A Gaussian process produces a **mean estimate**.



A Gaussian process produces a mean estimate along with an indication of the **uncertainty** in it.



The **posterior mean** and **covariance** equations follow simply from Gaussian identities.

$$y_* = y(x_*) \quad \text{Predictants}$$

$$y_d = y(x_d) \quad \text{Data}$$

$$p(y_*, y_d) = N\left(\begin{pmatrix} y_* \\ y_d \end{pmatrix}; \begin{pmatrix} \mu(x_*) \\ \mu(x_d) \end{pmatrix}, \begin{pmatrix} K(x_*, x_*) & K(x_*, x_d) \\ K(x_d, x_*) & K(x_d, x_d) \end{pmatrix}\right)$$

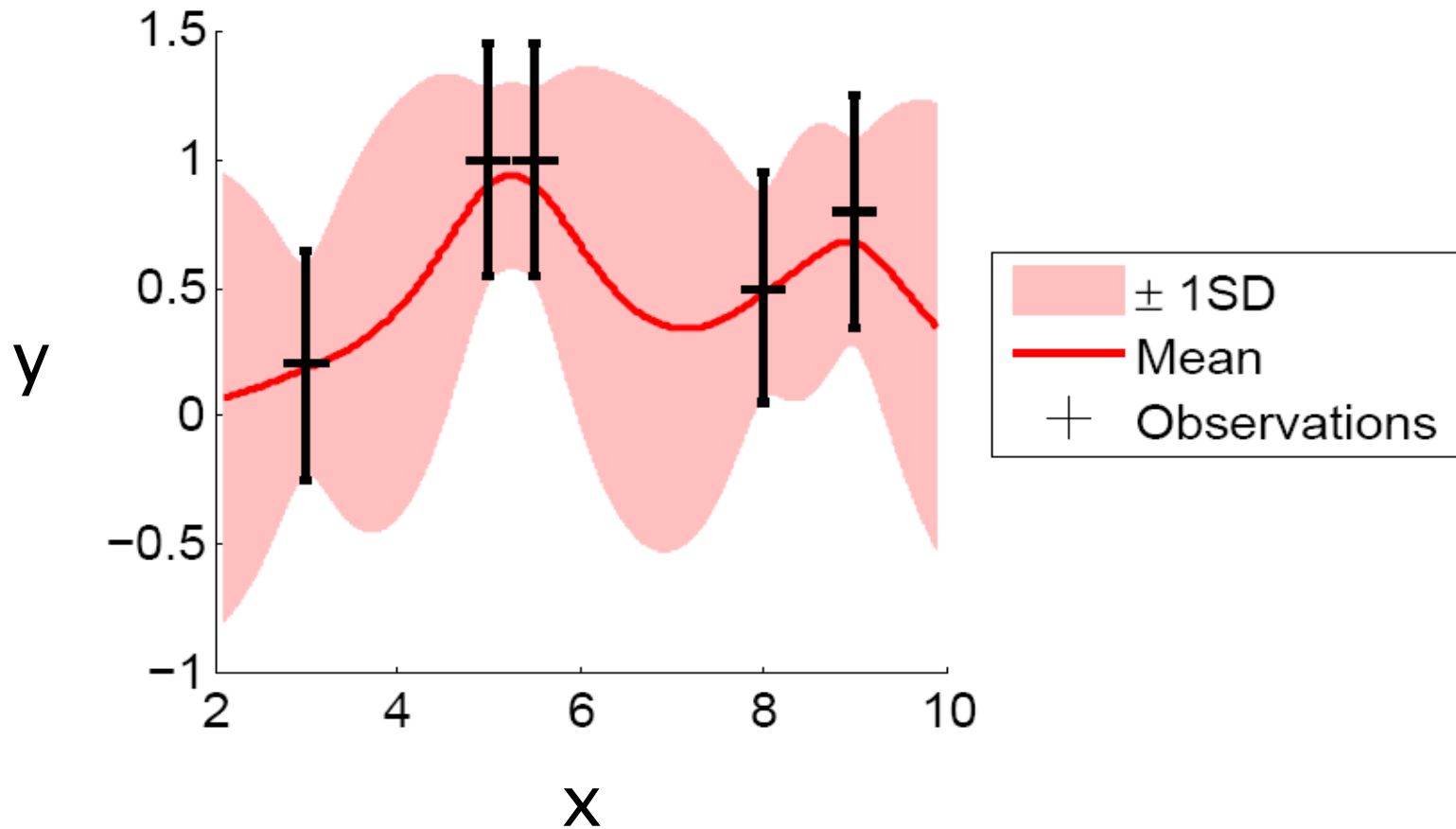
Mean $m(y_* | y_d) = \mu(x_*) + K(x_*, x_d)K(x_d, x_d)^{-1}(y_d - \mu(x_d))$

Cov. $C(y_* | y_d) = K(x_*, x_*) + K(x_*, x_d)K(x_d, x_d)^{-1}K(x_d, x_*)$

All functions here are dependent upon hyperparameters.



A Gaussian process can accommodate **noise**.



We usually consider making independent and identically distributed **(IID)** Gaussian noisy measurements z , of y ; giving

$$p(z_d | y_d) = N(z_d; y_d, \sigma^2 I_d) \quad \text{Identity matrix}$$

$$m(y_* | z_d) = \mu(x_*) + K(x_*, x_d) V(x_d, x_d)^{-1} (z_d - \mu(x_d))$$

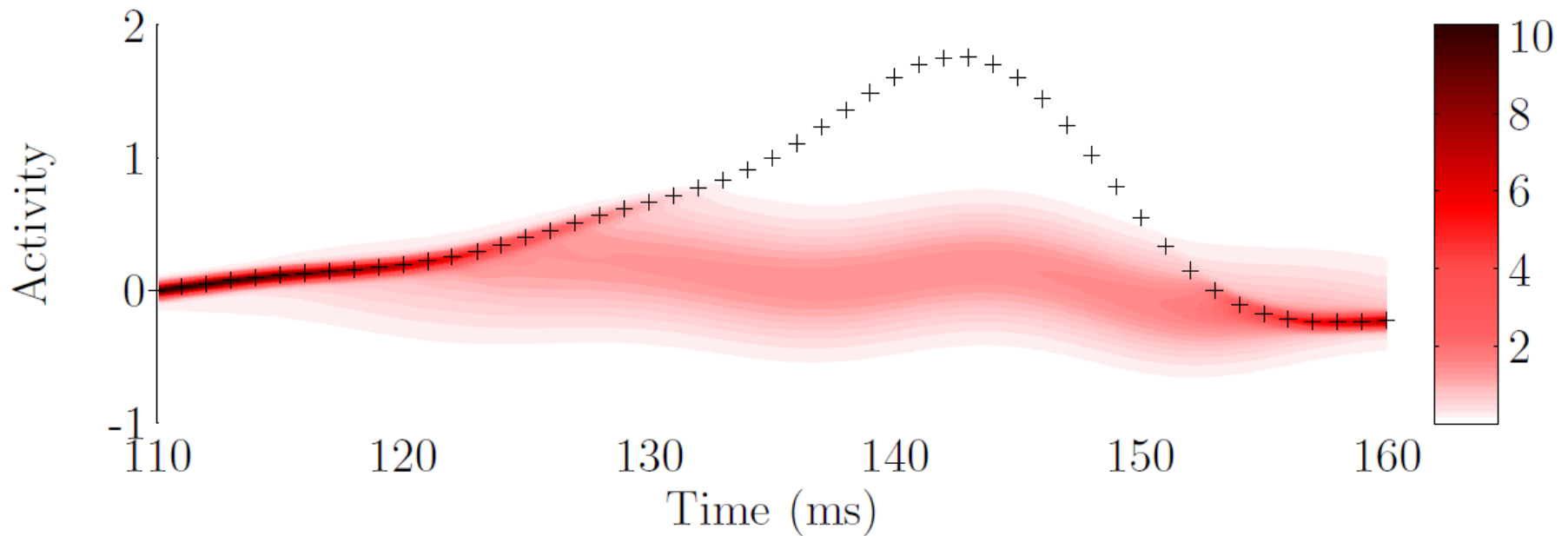
$$C(y_* | z_d) = K(x_*, x_*) + K(x_*, x_d) V(x_d, x_d)^{-1} K(x_d, x_*)$$

$$V(x_d, x_d) = K(x_d, x_d) + \sigma^2 I_d = \begin{pmatrix} K(x_1, x_1) + \sigma^2 & K(x_1, x_2) & \cdots \\ K(x_2, x_1) & K(x_2, x_2) + \sigma^2 & \\ \vdots & & \ddots \end{pmatrix}$$

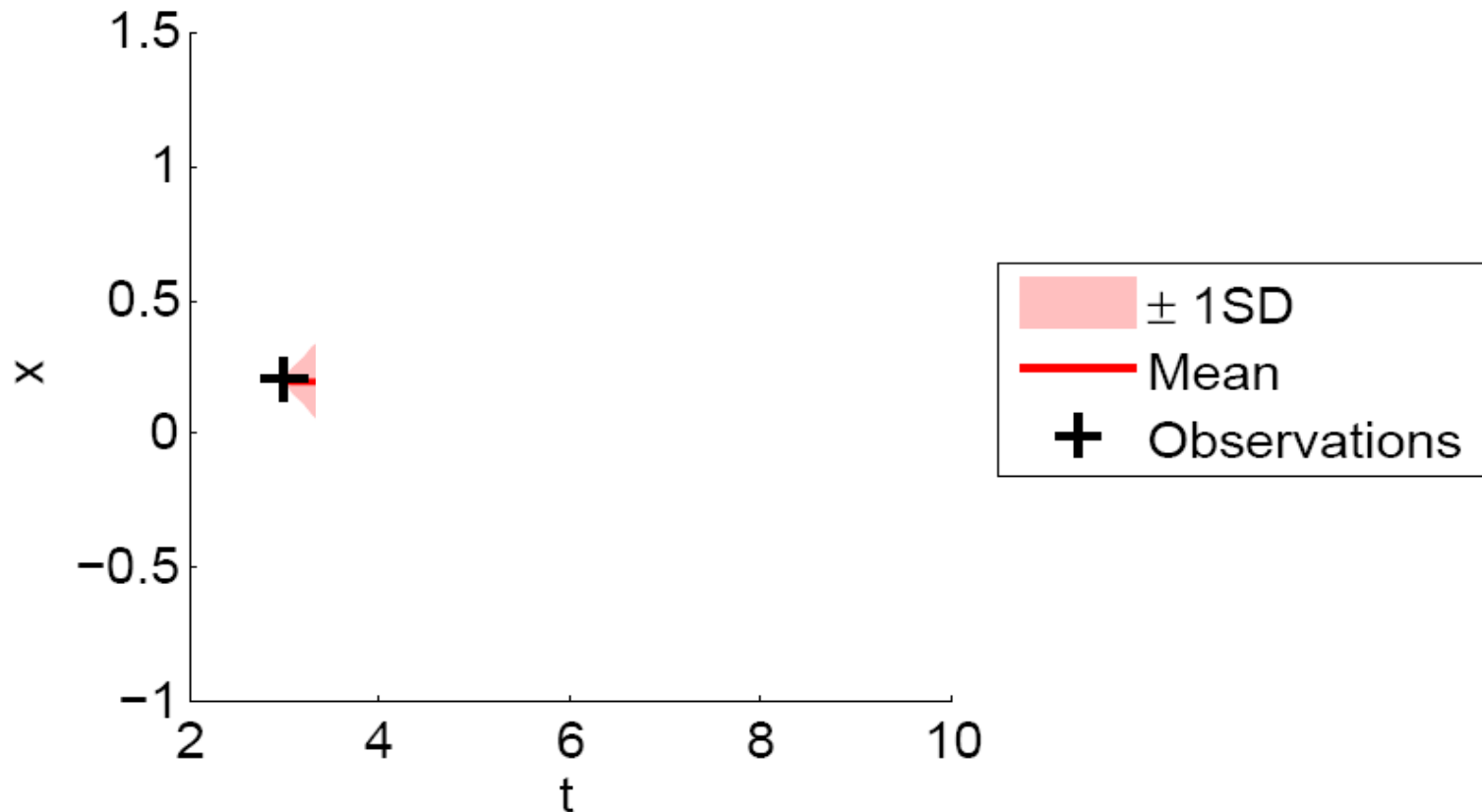


More generally, we could consider **correlated noise**, in which the noise contribution could itself be drawn from a GP.

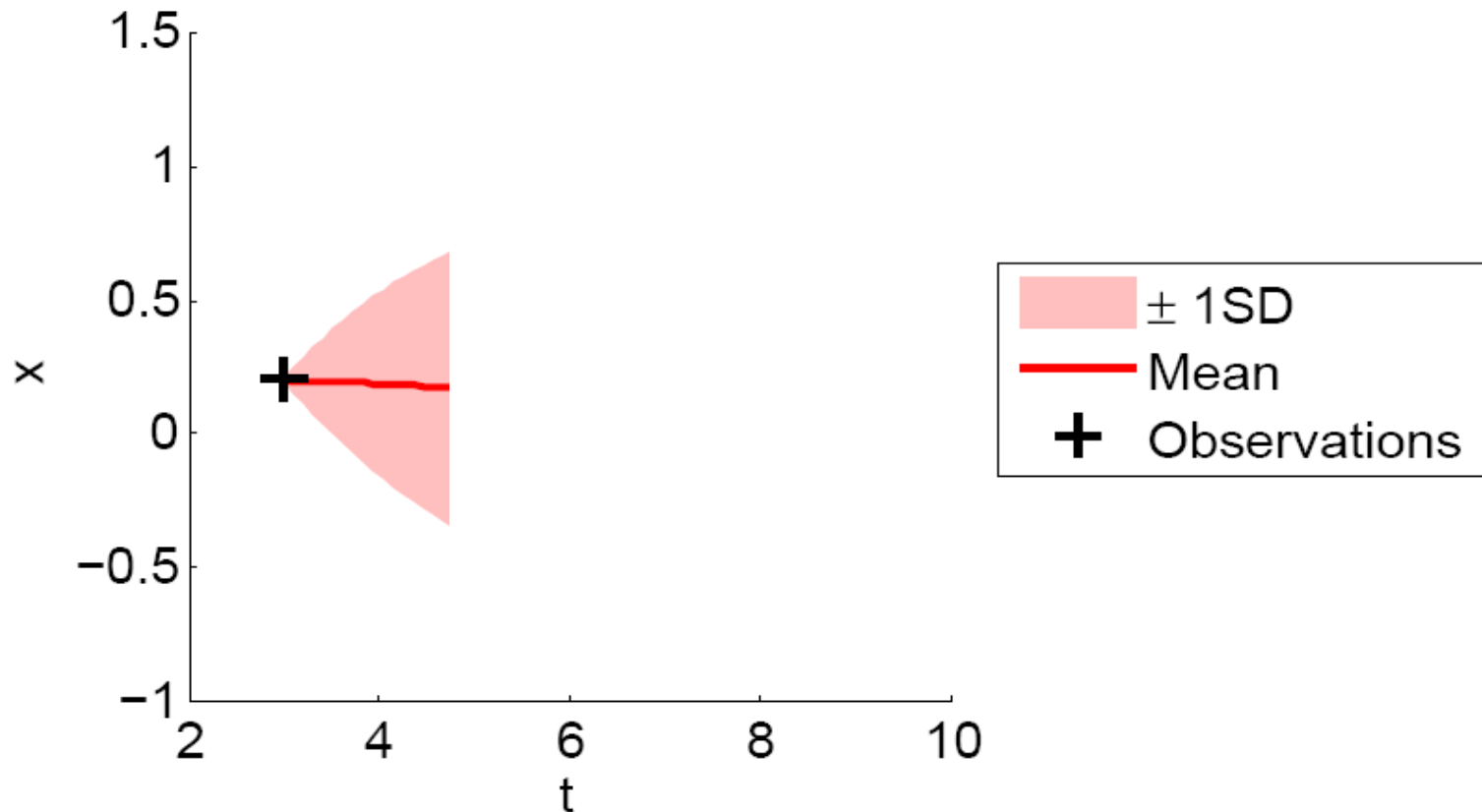
Full posterior for EEG data with saccade



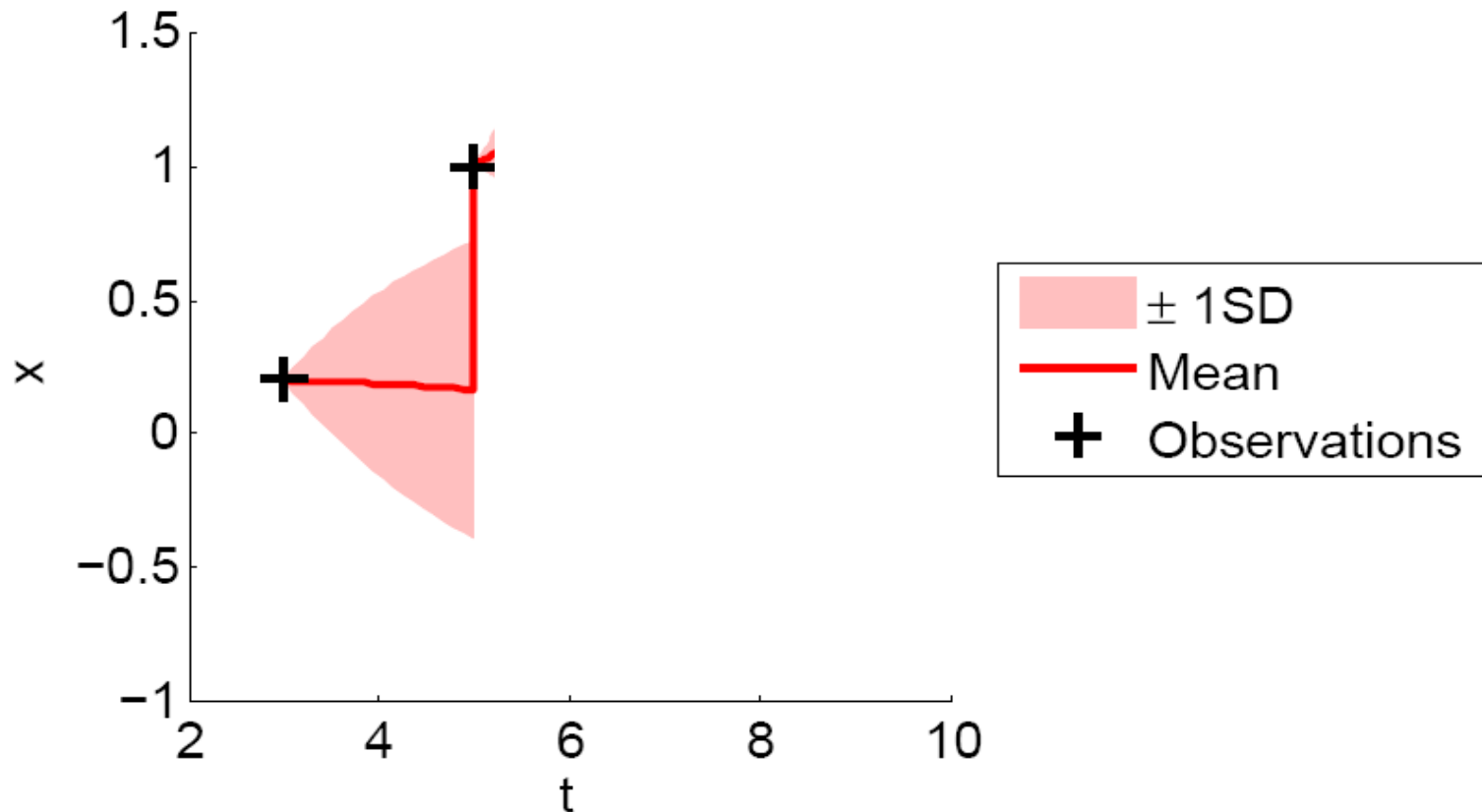
We often want to address functions of time, using Gaussian processes for **tracking**.



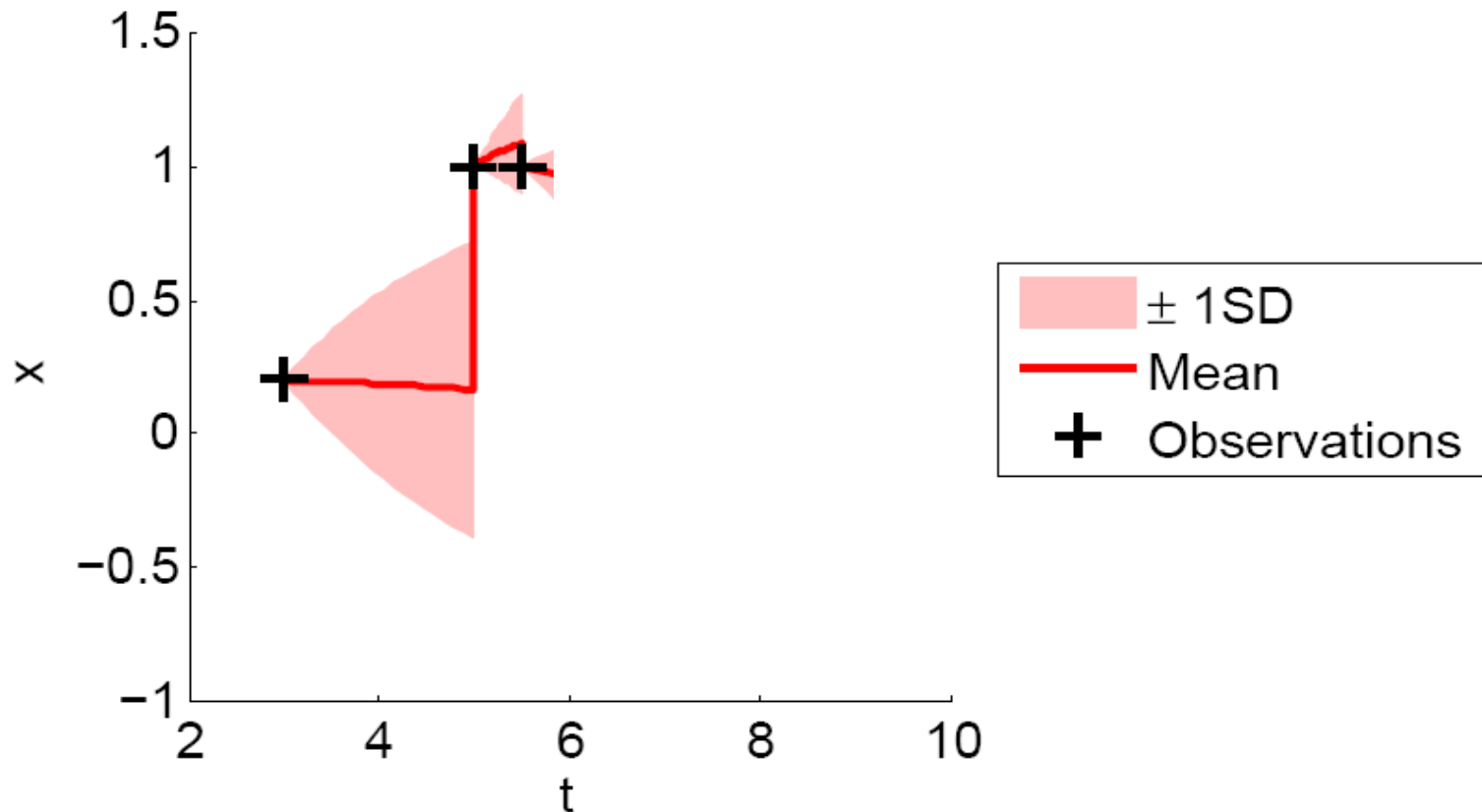
We often want to address functions of time, using Gaussian processes for **tracking**.



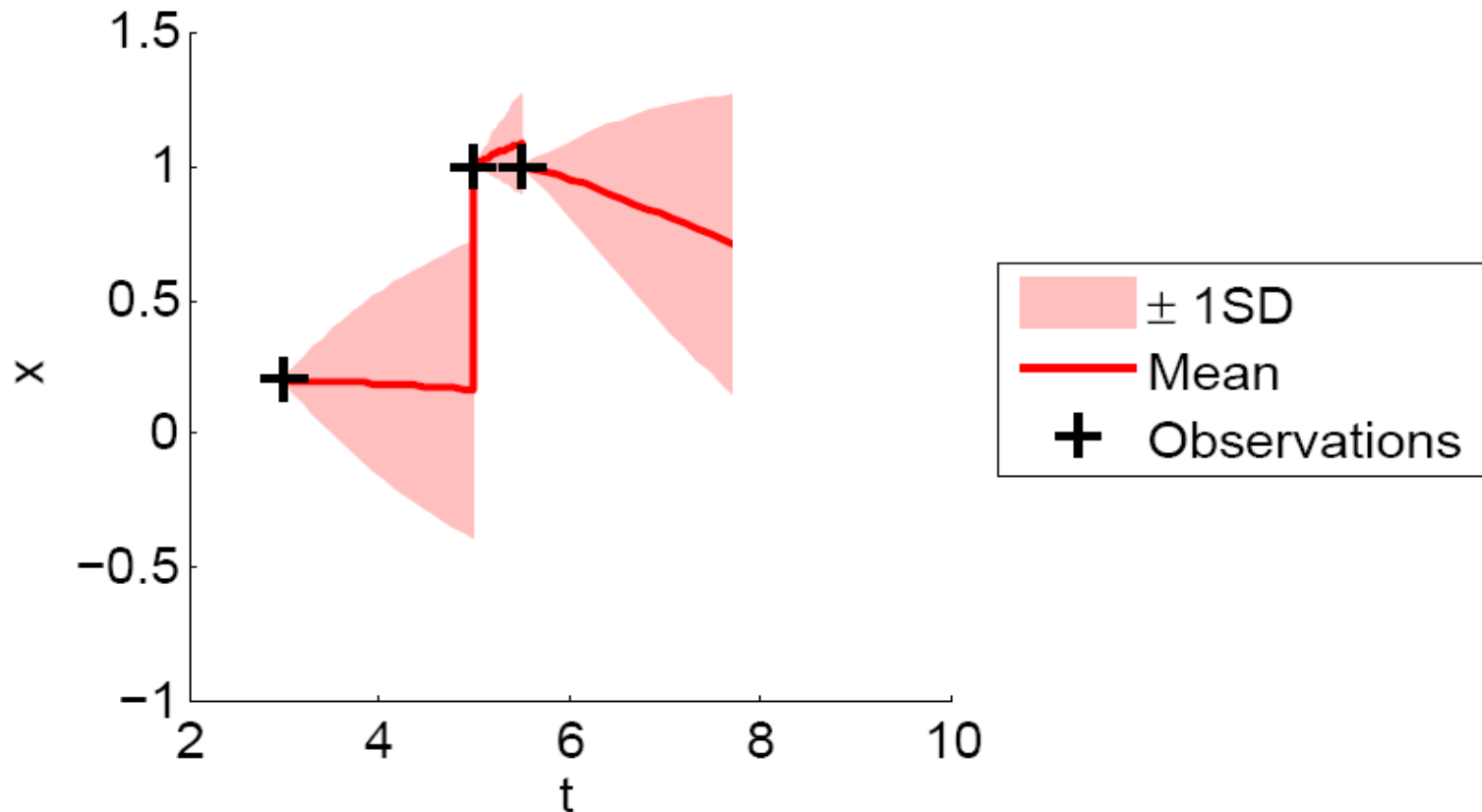
We often want to address functions of time, using Gaussian processes for **tracking**.



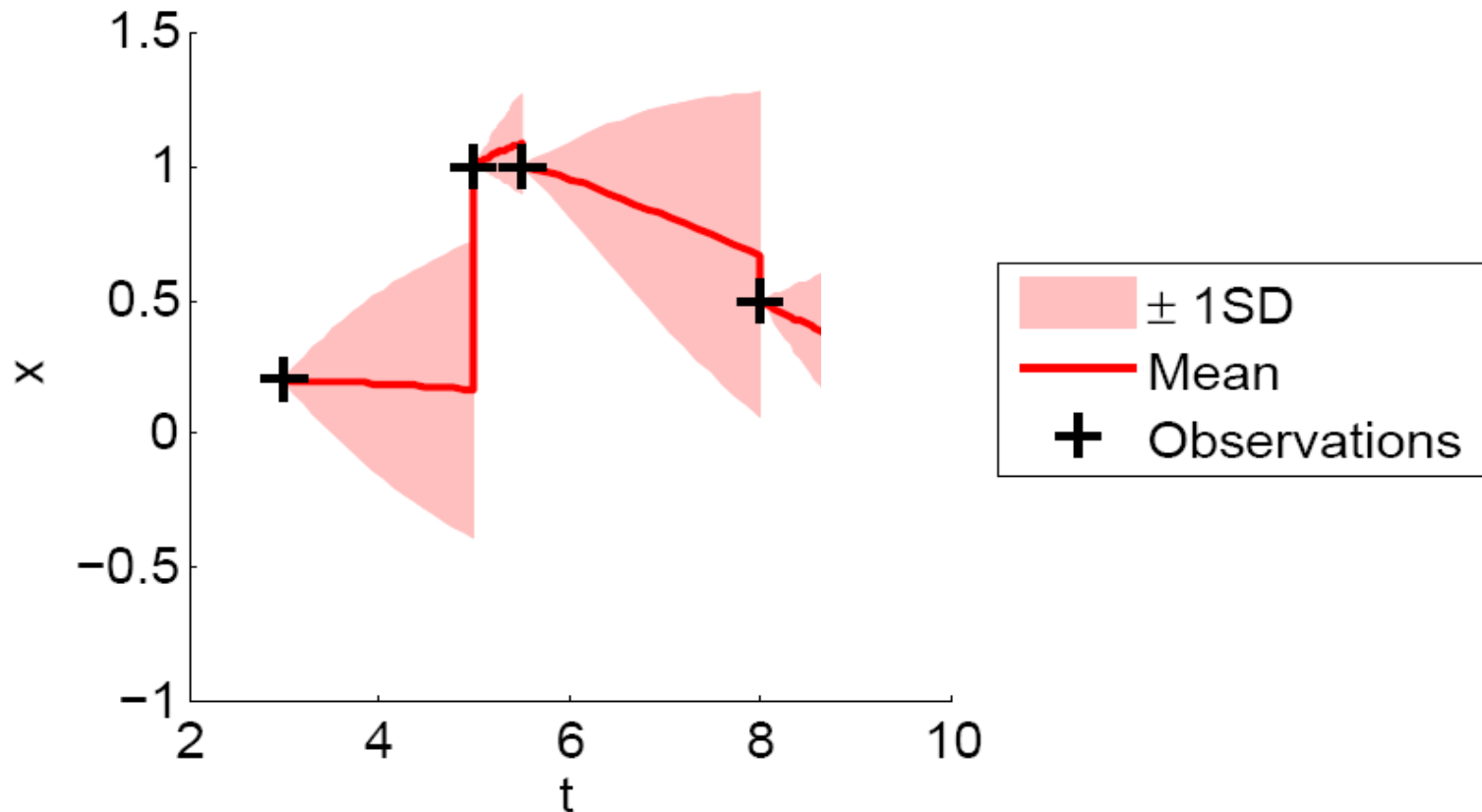
We often want to address functions of time, using Gaussian processes for **tracking**.



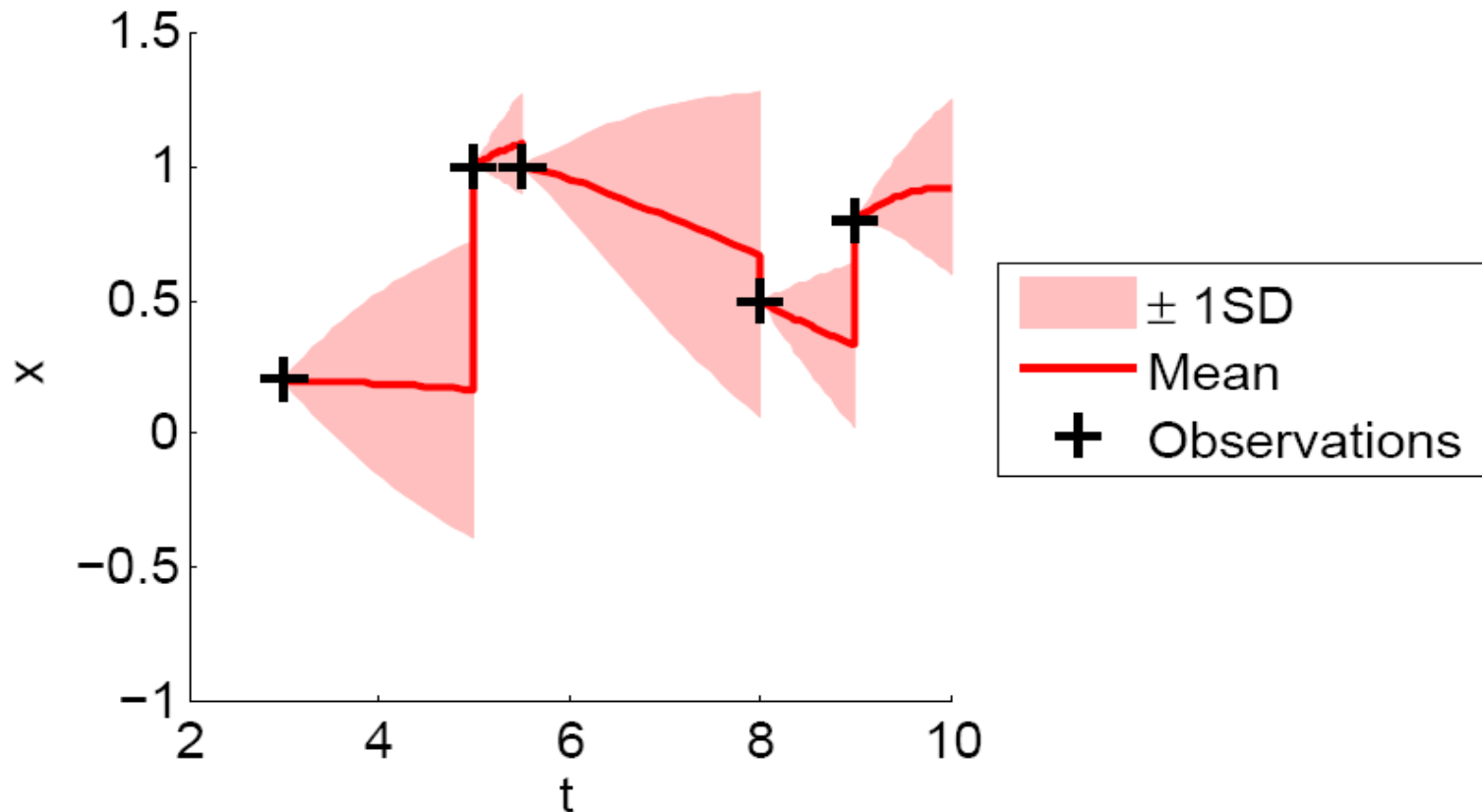
We often want to address functions of time, using Gaussian processes for **tracking**.



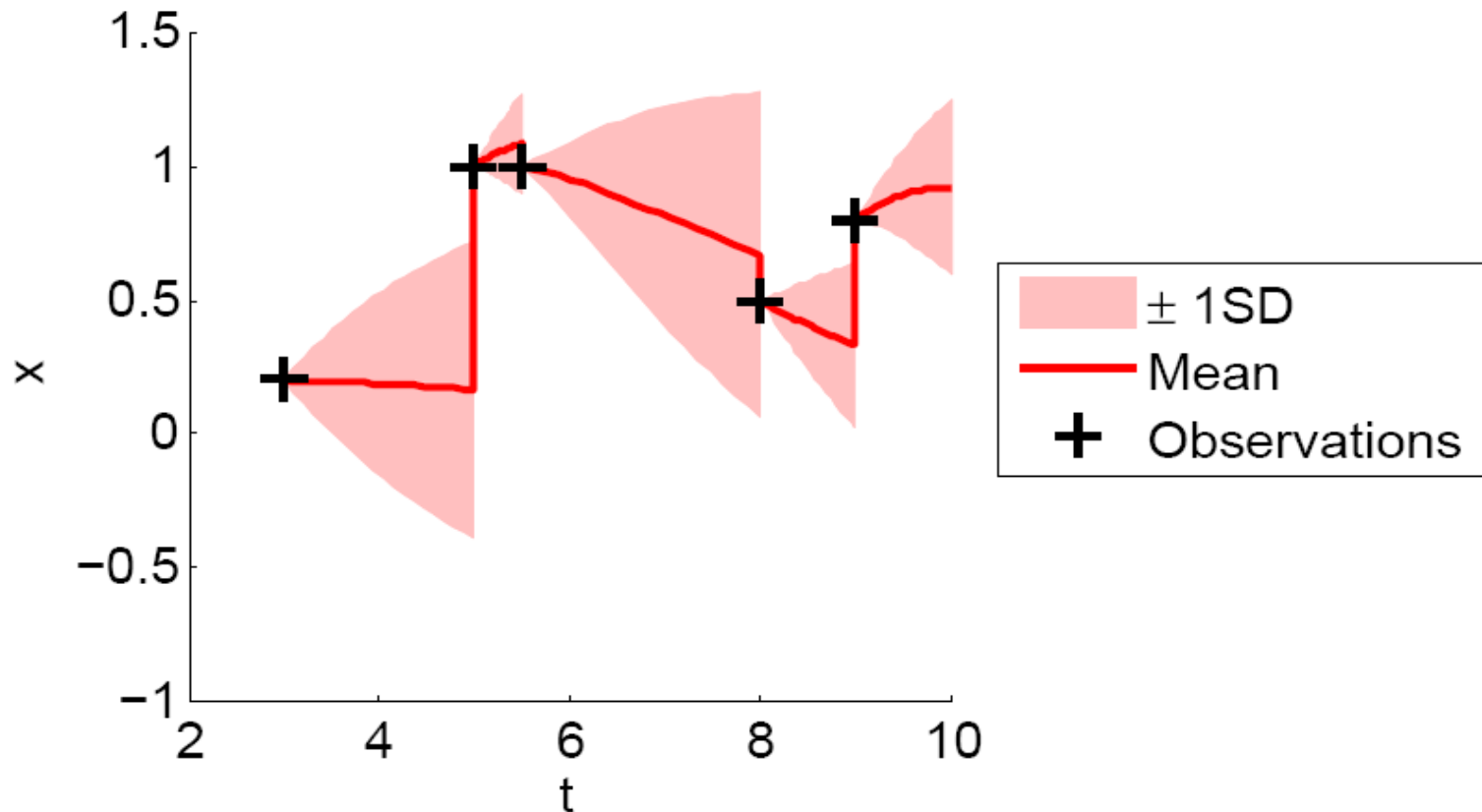
We often want to address functions of time, using Gaussian processes for **tracking**.



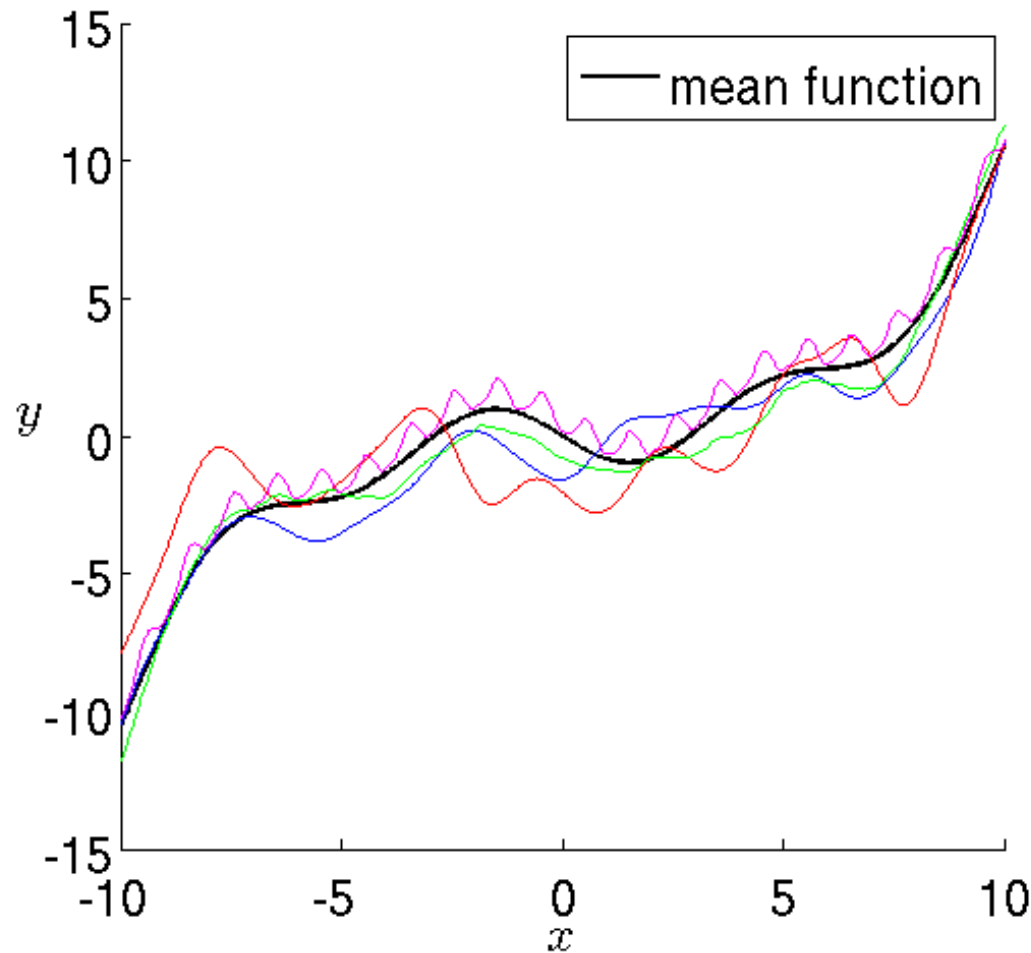
We often want to address functions of time, using Gaussian processes for **tracking**.



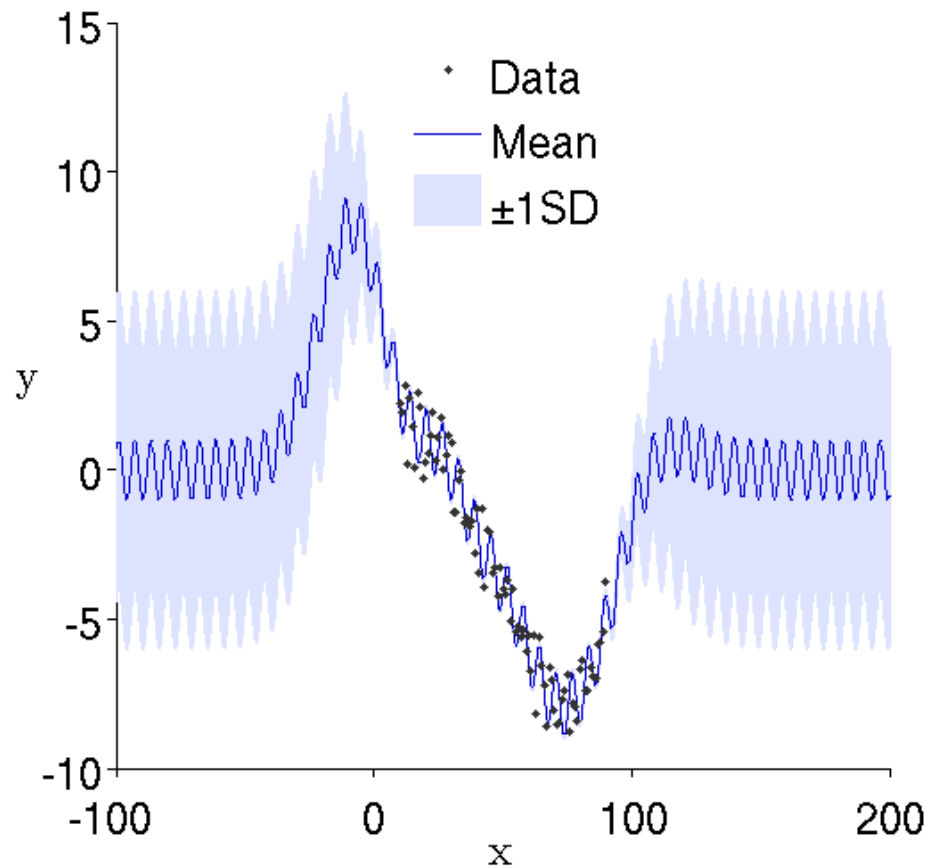
We often want to address functions of time, using Gaussian processes for **tracking**.



The prior **mean function** $\mu(x; \phi)$ should be our best guess (of any form) for the function $y(x)$ before any observations are made.



The prior mean function is the function our inference will default to **far from observations**.



It's rarely worth using a complicated **mean function** (with many hyperparameters), unless we're concerned with prediction far from our observations.

Predictions required	Mean function
Interpolation	$\mu(x; \phi) = \text{mean}(y_d)$.
Extrapolation	Bespoke model built using domain knowledge.

There are a huge number of **covariance functions** (in spite of the requirement that they be positive semi-definite) appropriate for modelling functions of different types.

Function type	Covariance function
Improbably smooth	Squared exponential.
Less smooth	Matérn.
Polynomial	Polynomial.

Many covariance functions (including the squared exponential and Matérn) are of the **metric** form

squared output scale

$$K(x_i, x_j; w) = \overbrace{h^2}^{\text{squared output scale}} \underbrace{\kappa}_{\text{distance function}}(\underbrace{d(x_i, x_j; w)}_{\text{input scale}})$$

decreases with
increasing d

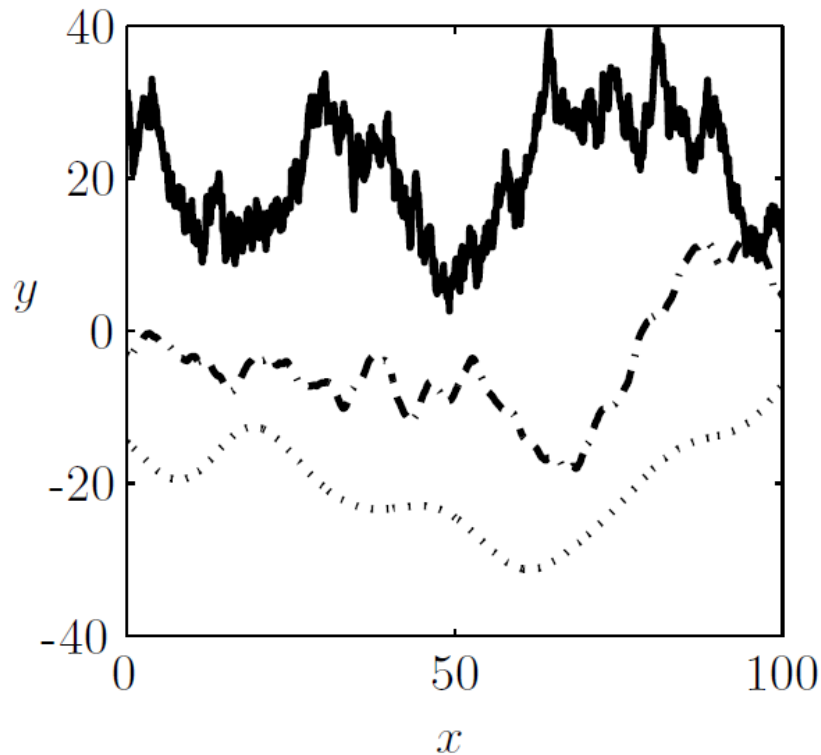
distance
function

input
scale

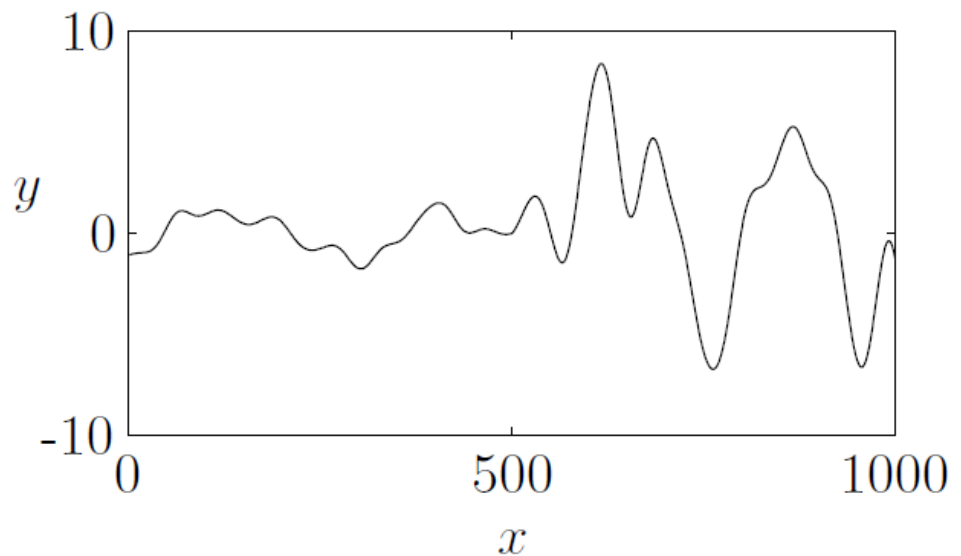
e.g. $d(x_i, x_j; w) = \left| \frac{x_i - x_j}{w} \right|$



We often want distances that are **stationary** (a function of $x_1 - x_2$), implying that the function looks similar throughout its domain.

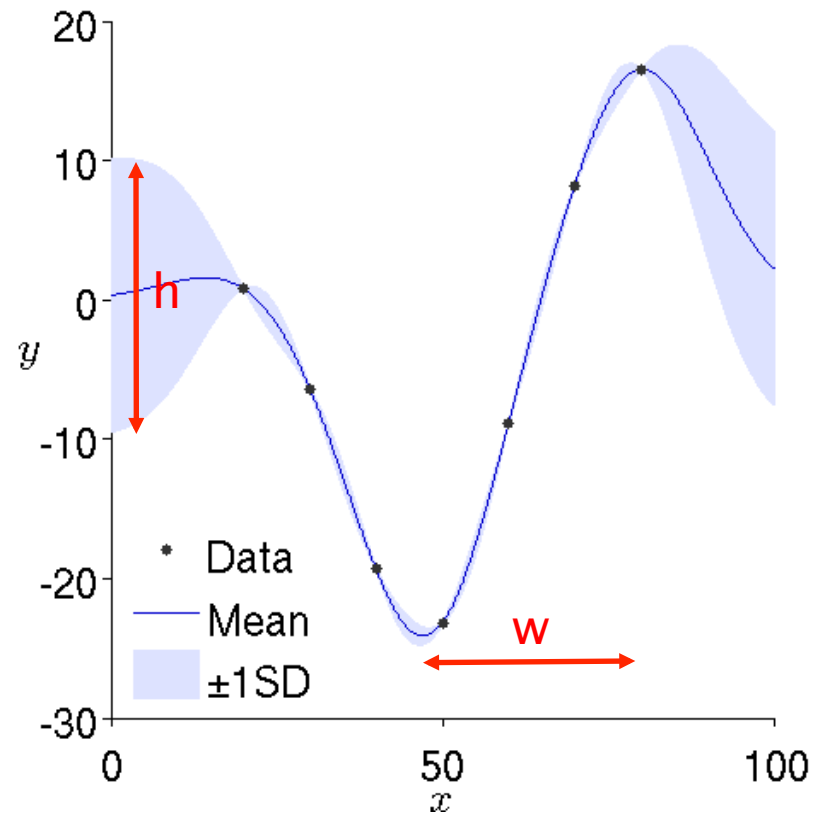
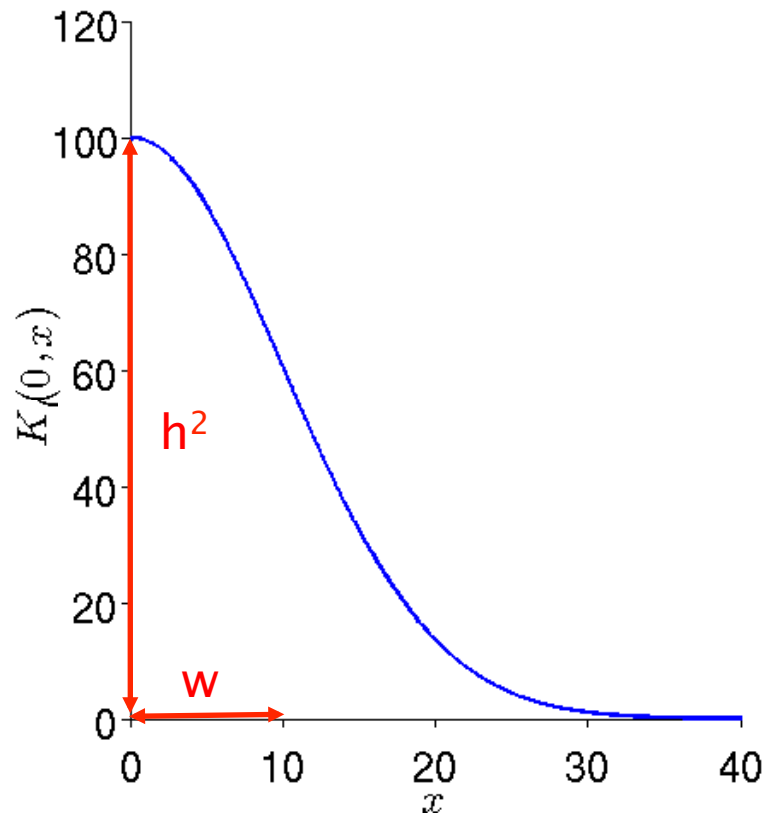


stationary functions

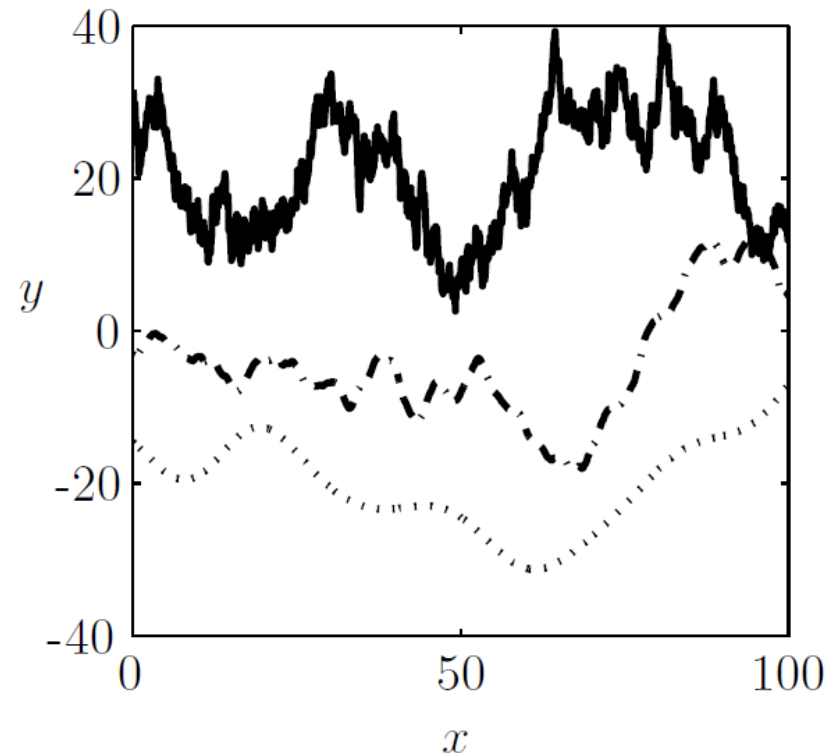
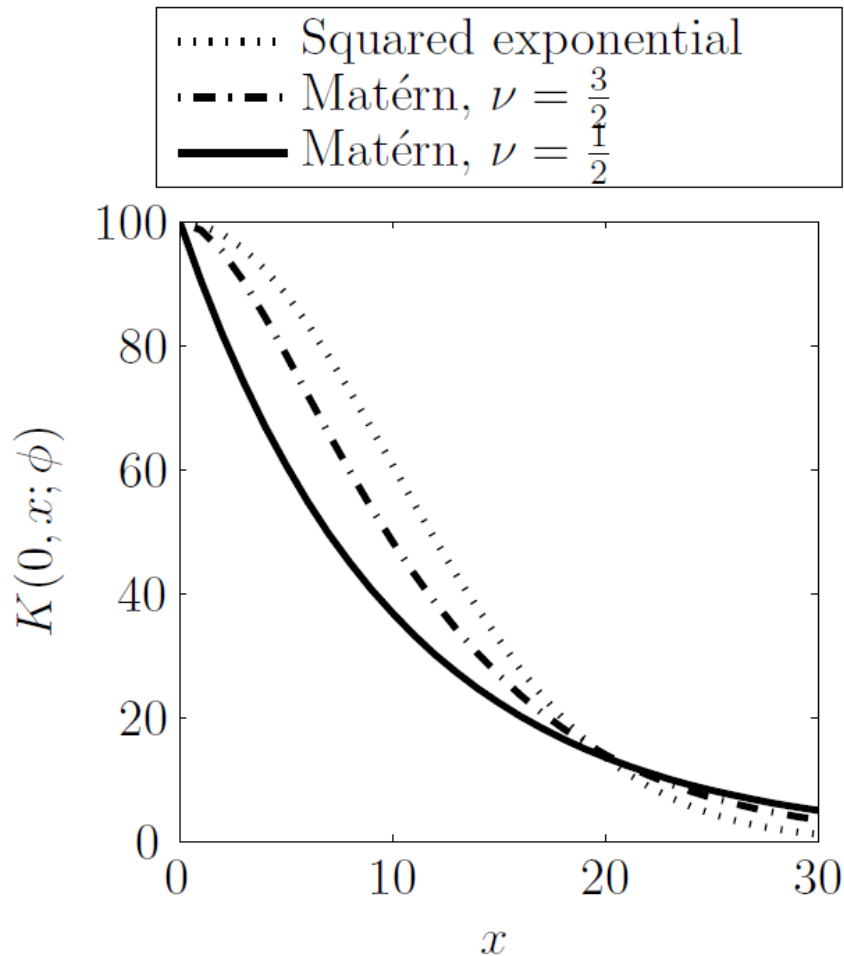


non-stationary
function

The hyperparameters h and w specify our **expected length scales** of the function in output ('height') and input ('width') spaces respectively.



The **squared exponential** and **Matérn** covariances allow us to model functions of various degrees of smoothness.

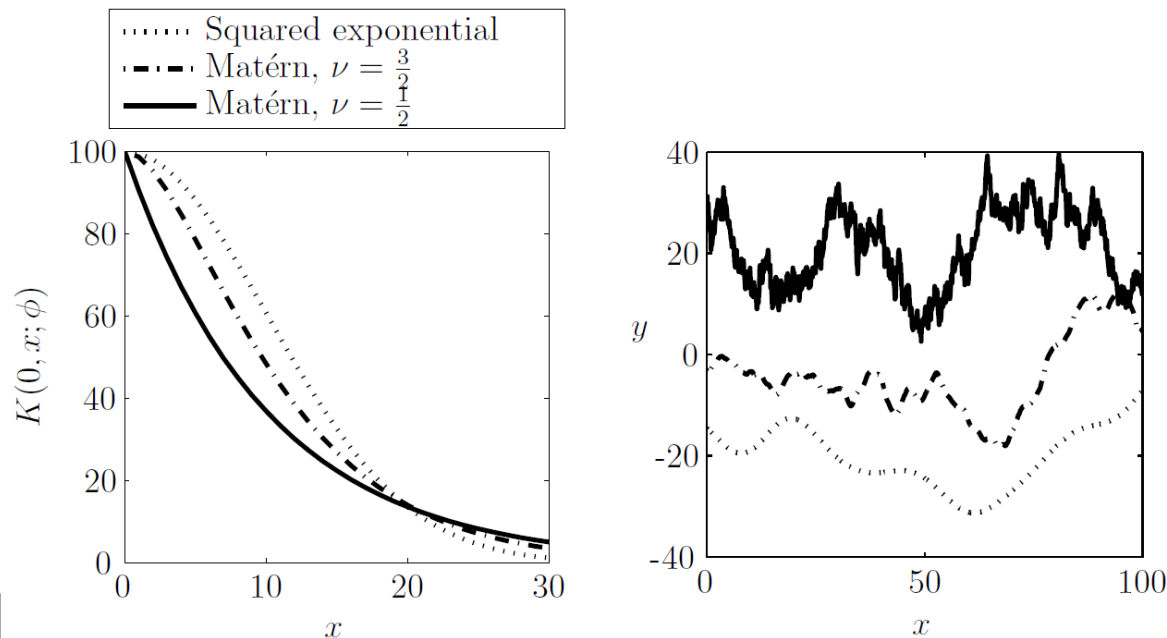


The **squared exponential** and **Matérn** covariances allow us to model functions of various degrees of smoothness.

$$K_{\text{SE}}(x_i, x_j; h, w) = h^2 \exp\left(-\frac{1}{2} d(x_i, x_j; w)^2\right)$$

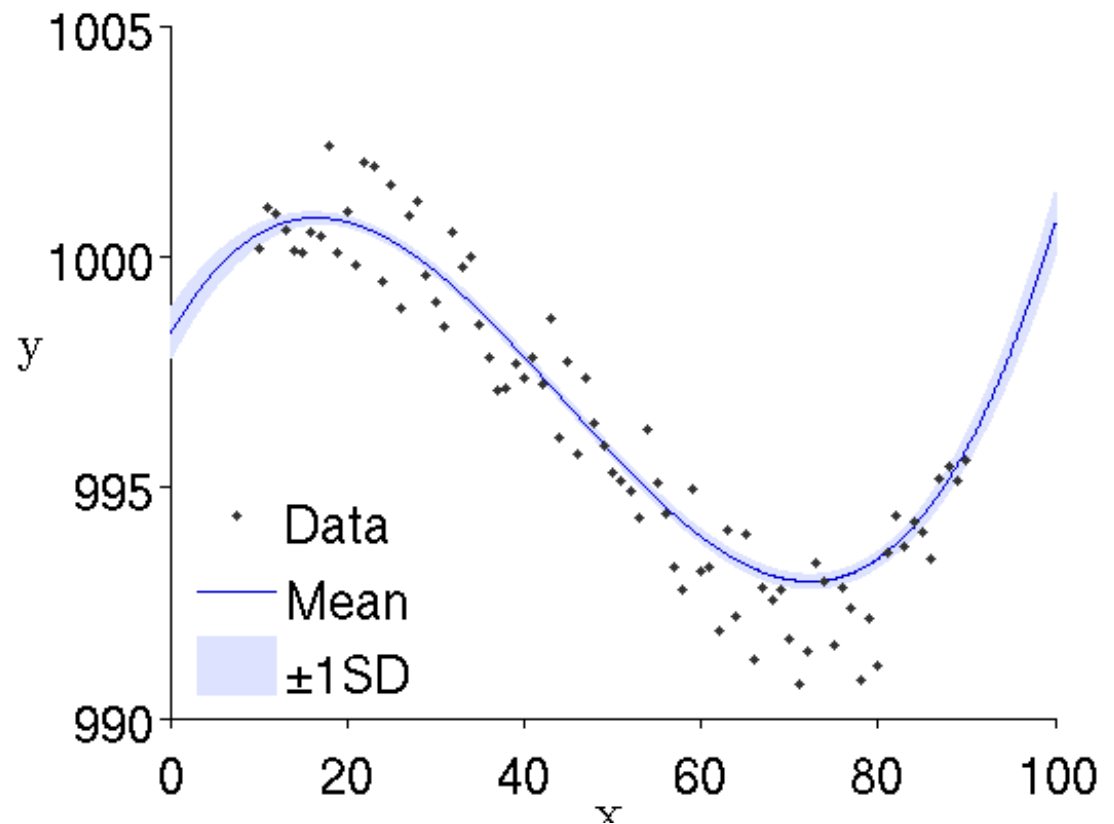
$$K_{\text{Mtn}}(x_i, x_j; h, w, \nu = \frac{3}{2}) = h^2 \left(1 + \sqrt{3} d(x_i, x_j; w)\right) \exp\left(-\sqrt{3} d(x_i, x_j; w)\right)$$

$$K_{\text{Mtn}}(x_i, x_j; h, w, \nu = \frac{1}{2}) = h^2 \exp\left(-d(x_i, x_j; w)\right)$$



Polynomial covariances exist to model functions that are known to be polynomial.

$$K_P(x_i, x_j; x, W) = (c^2 + x_i^T W x_j)^d$$

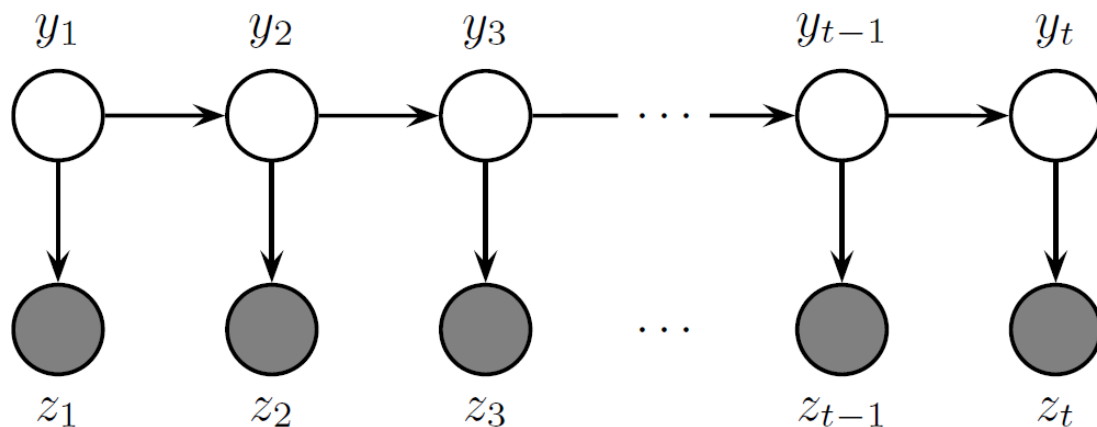


The **Kalman filter** is a Gaussian process with a special covariance function, one that gives a sparse precision matrix.

This allows efficient computation.

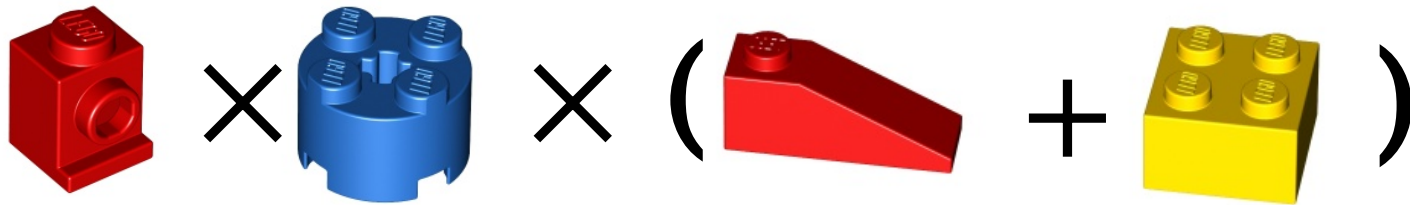
$K = (\text{ugly})$

$$K^{-1} = \begin{pmatrix} 2 & -1 & 0 & 0 & \dots \\ -1 & 2 & -1 & 0 & \\ 0 & -1 & 2 & -1 & \\ 0 & 0 & -1 & 2 & \\ \vdots & & & & \ddots \end{pmatrix}$$

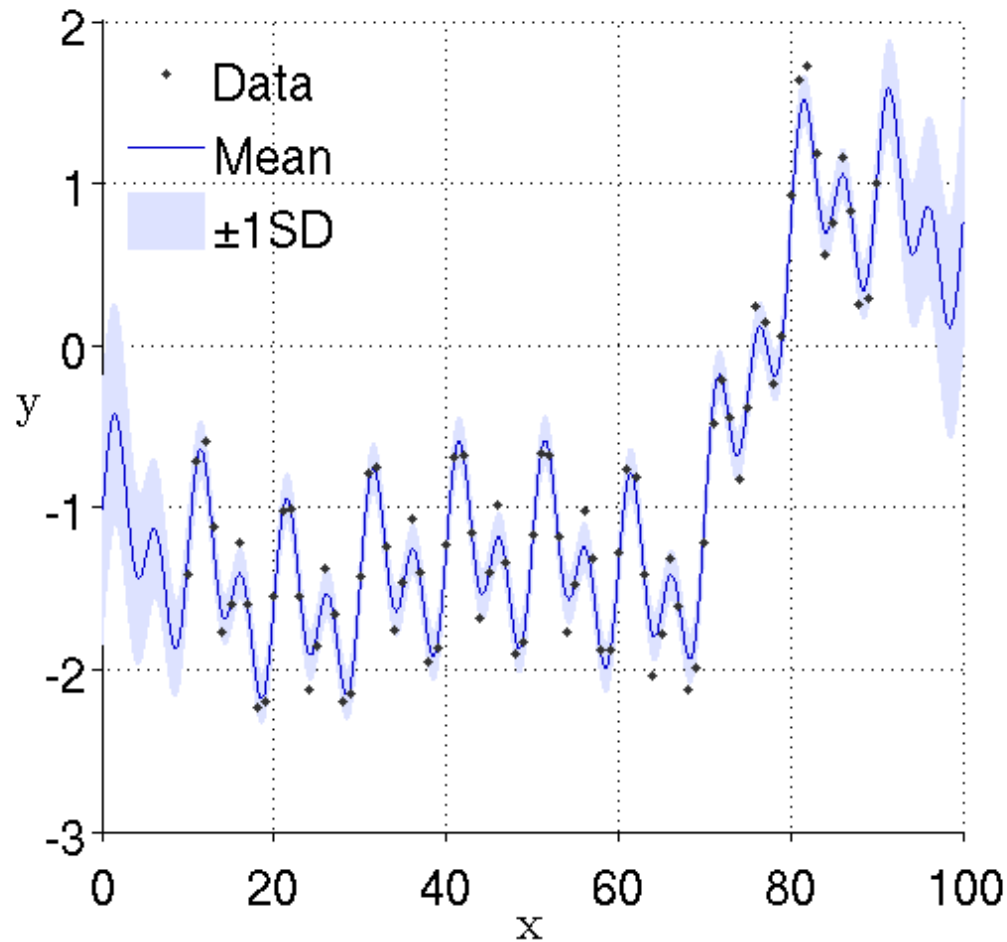


We can create new covariance functions by **adding** or **multiplying** other covariance functions.

e.g.



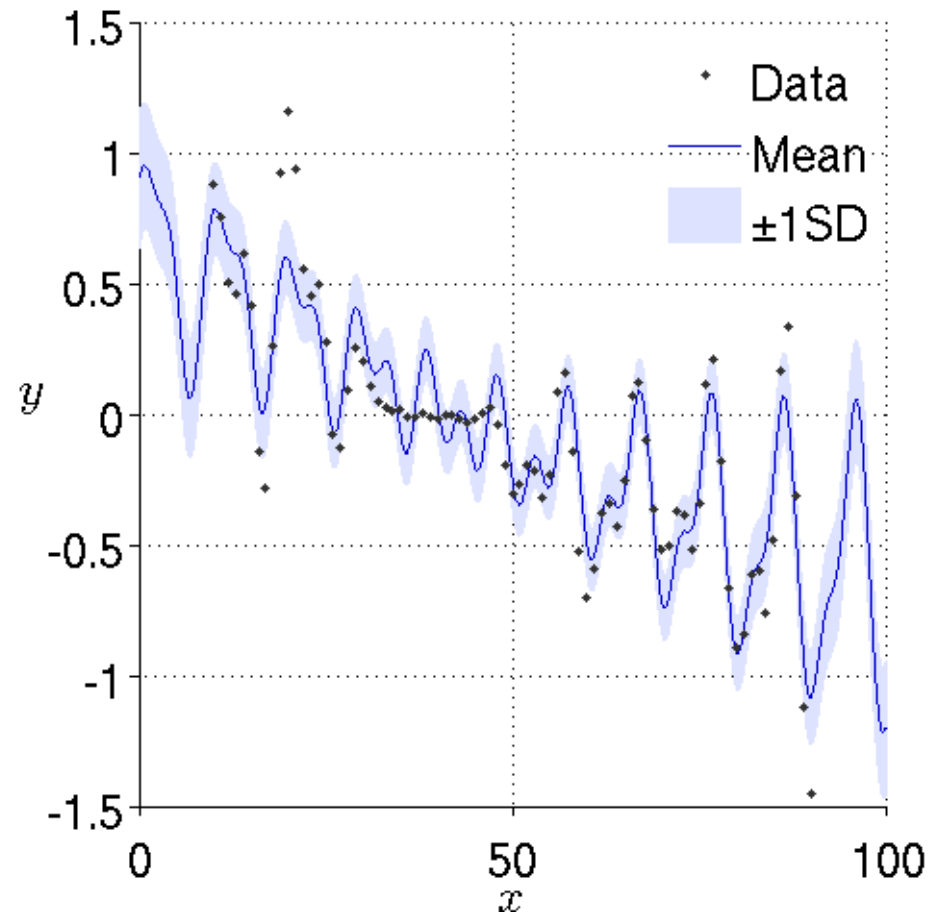
When a function is the **sum** of two independent functions, use a covariance that is the sum of the covariances for those two functions.



When a function is the **product** of two independent functions, use a covariance that is (almost) the product of the covariances for those two functions.

$$y(x) = a(x)b(x)$$

$$K_y(x_1, x_2) = K_a(x_1, x_2)K_b(x_1, x_2) + \\ K_a(x_1, x_2)\mu_b(x_1)\mu_b(x_2) + \\ K_b(x_1, x_2)\mu_a(x_1)\mu_a(x_2)$$



We can also **modify** covariance functions.

Inputs	Squared exponential	Matern	Polynomial
--------	------------------------	--------	------------

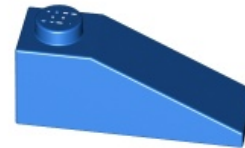
1-dim



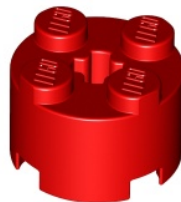
n-dim



derivative



periodic

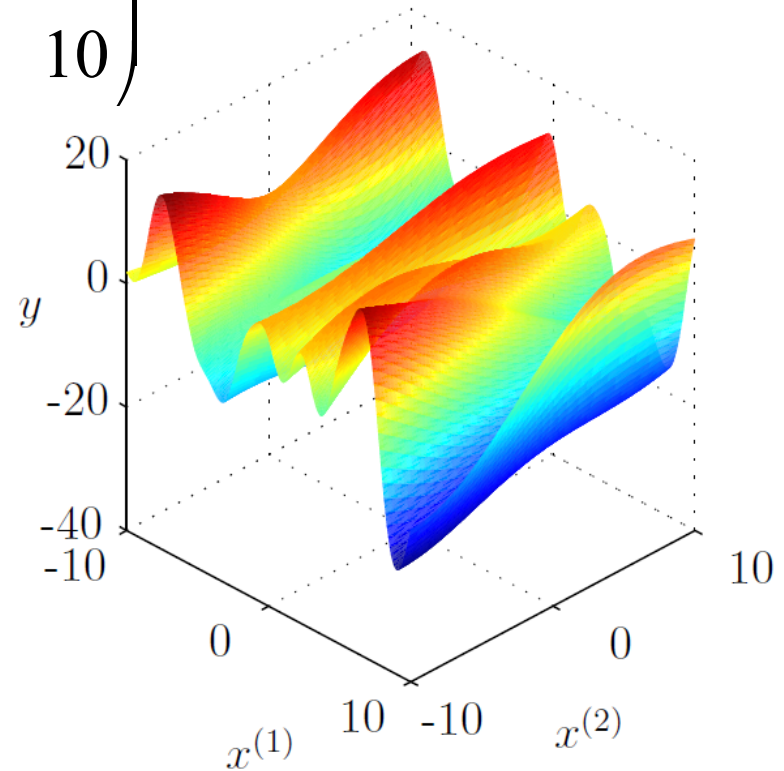
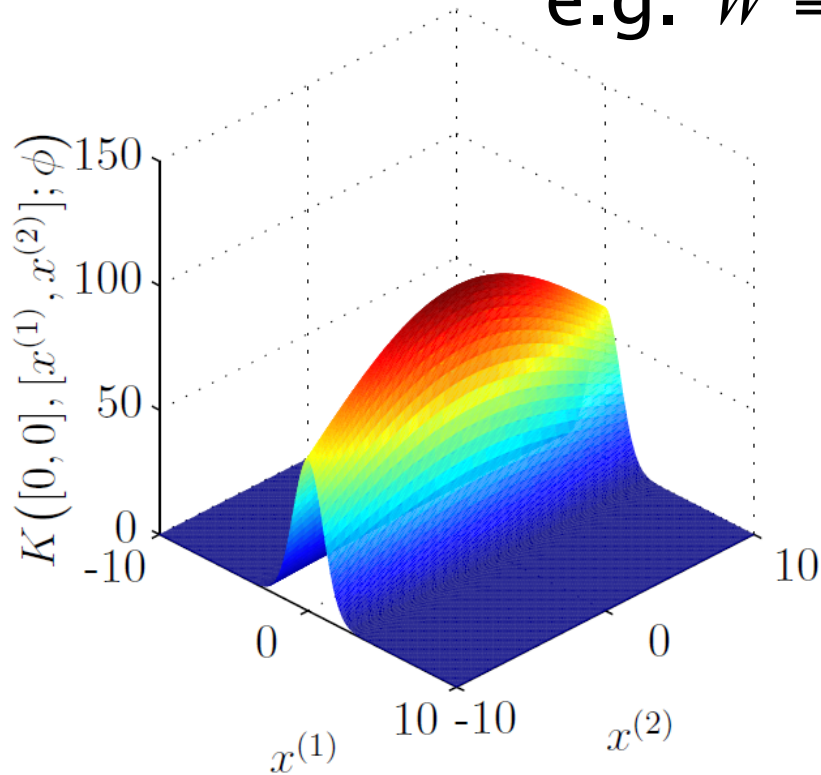


UNIVERSITY OF
OXFORD

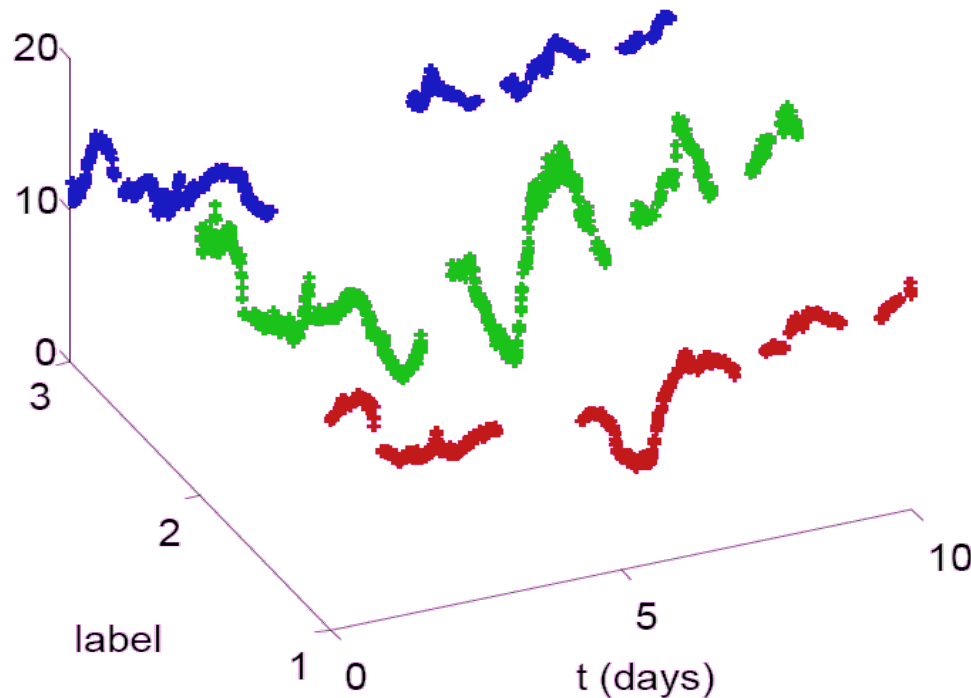
We can modify covariance functions to accommodate **multiple input dimensions**, using

$$d(x_i, x_j; W) = \sqrt{(x_i - x_j)^T W^{-1} (x_i - x_j)}$$

e.g. $W = \begin{pmatrix} 1 & 0 \\ 0 & 10 \end{pmatrix}$



If there are **multiple outputs**, reframe the problem as having a single output, and an additional *label* input specifying the output.



Hence we do not need simultaneous observations of all outputs.

If the inputs were previously x , and outputs were labelled by $l = 1, \dots, L$, we now need to specify a **covariance over both x and l** , e.g.

separable for
convenience

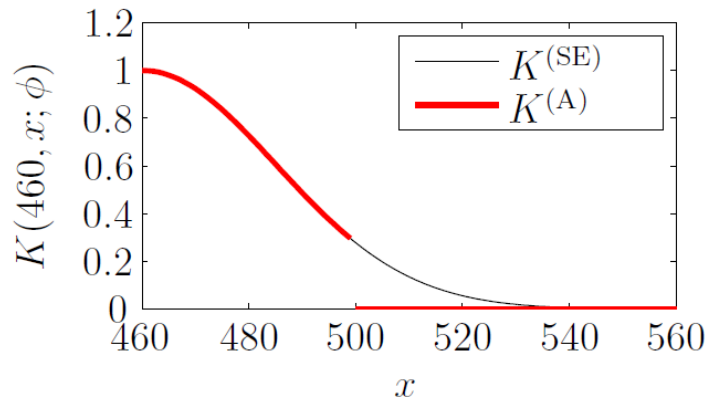
$$K((x_i, l_i), (x_j, l_j)) = \overbrace{K(x_i, x_j)} K(l_i, l_j)$$

If L is not too large, we could use the spherical parameterisation.

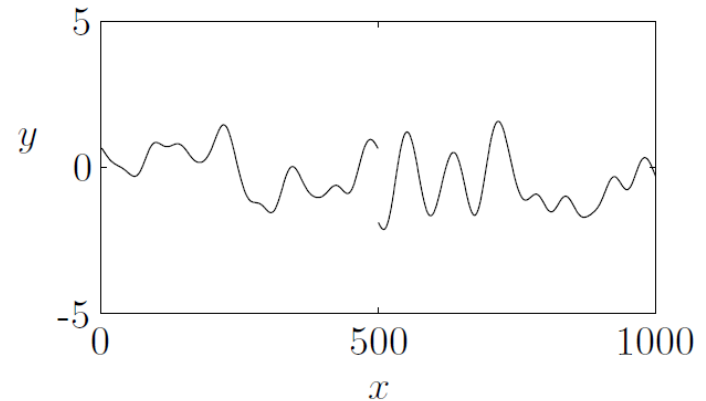


Many **other modifications** are possible, to build covariances allowing for e.g. changepoints, faults and sets.

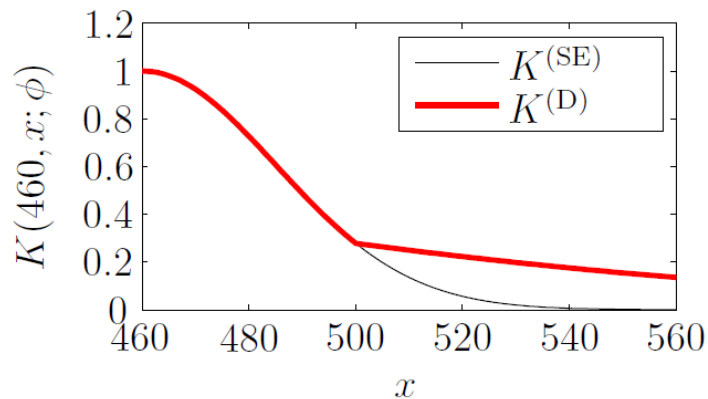
Drastic changepoint



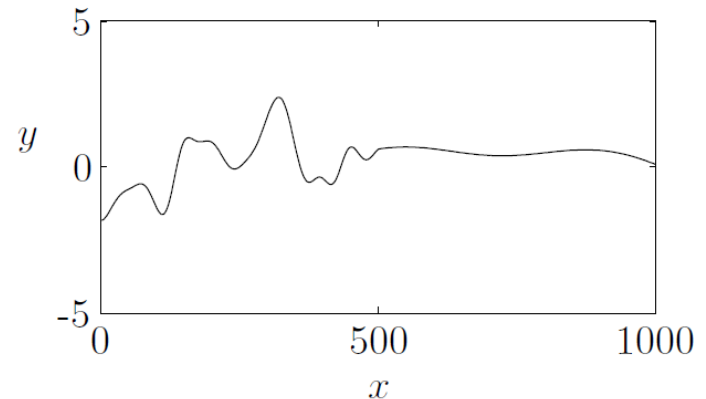
Drastic changepoint



Changepoint in input scale



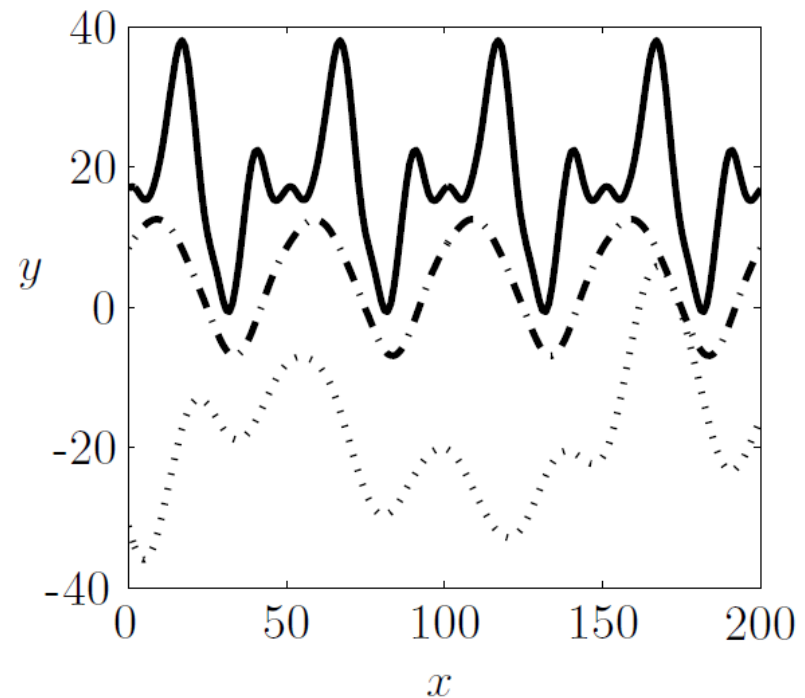
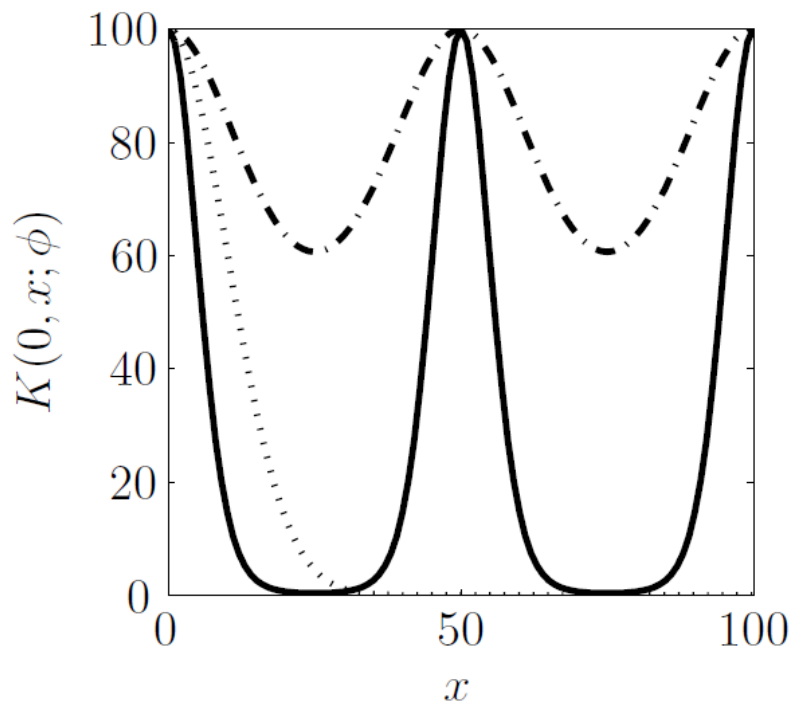
Changepoint in input scale



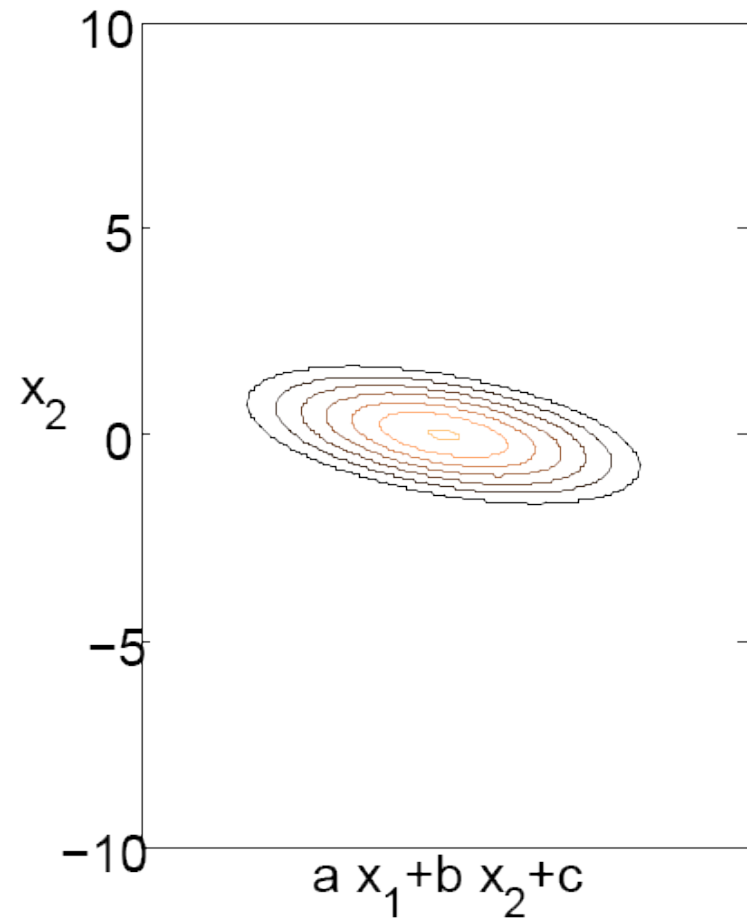
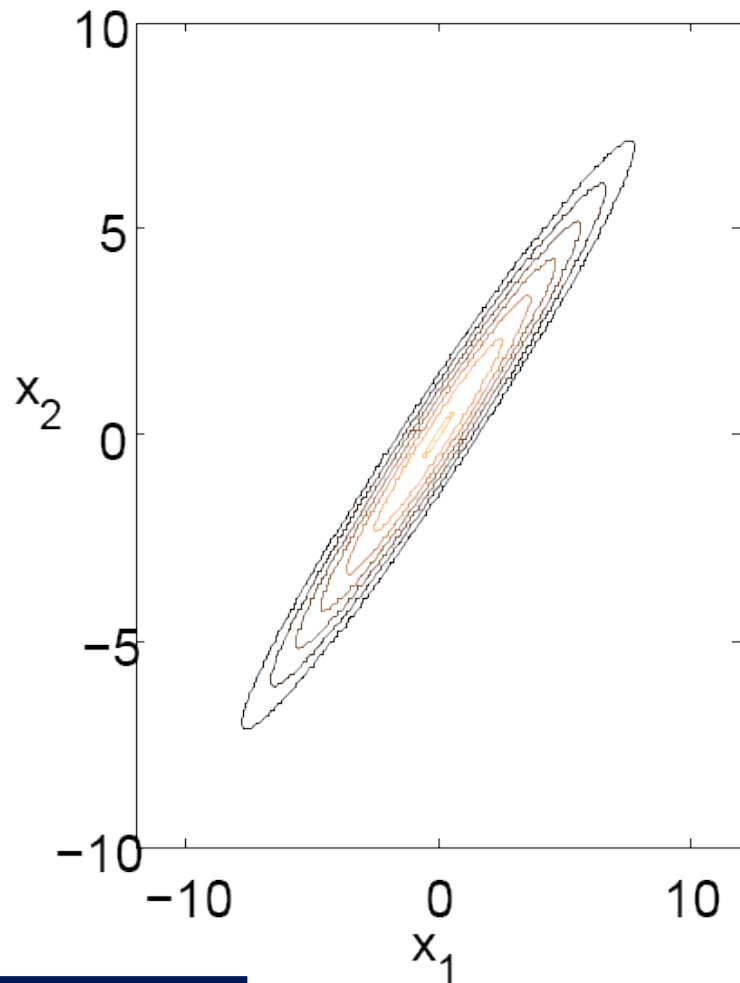
We can modify covariance functions for functions known to be **periodic**, by using the distance

$$d(x_i, x_j) = \frac{1}{w} \left| \sin \left(\pi \frac{x_i - x_j}{T} \right) \right|$$

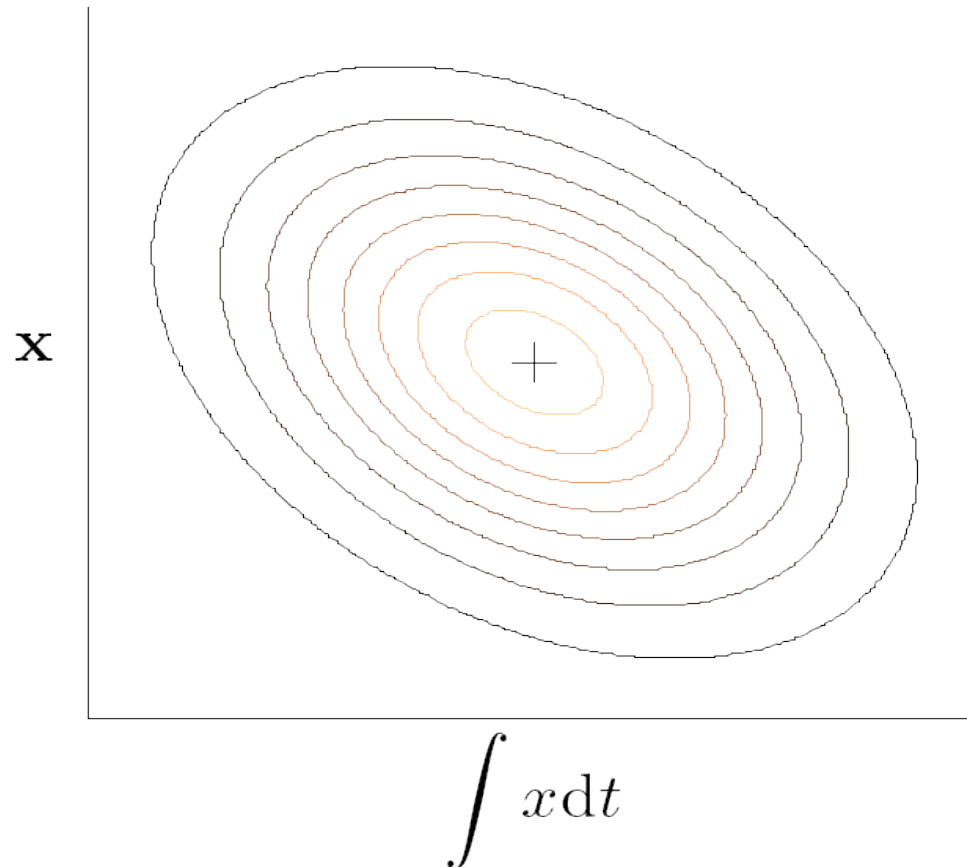
..... Squared exponential
 -.-.- Periodic sqd. exp., $w = 1$
 — Periodic sqd. exp., $w = 0.3$



Gaussian distributed variables are joint Gaussian with any **affine transform** of them.



A function over which we have a Gaussian process is joint Gaussian with any **integral** or **derivative** of it, as integration and differentiation are affine.



We can modify covariance functions to manage **derivative** or **integral** observations.

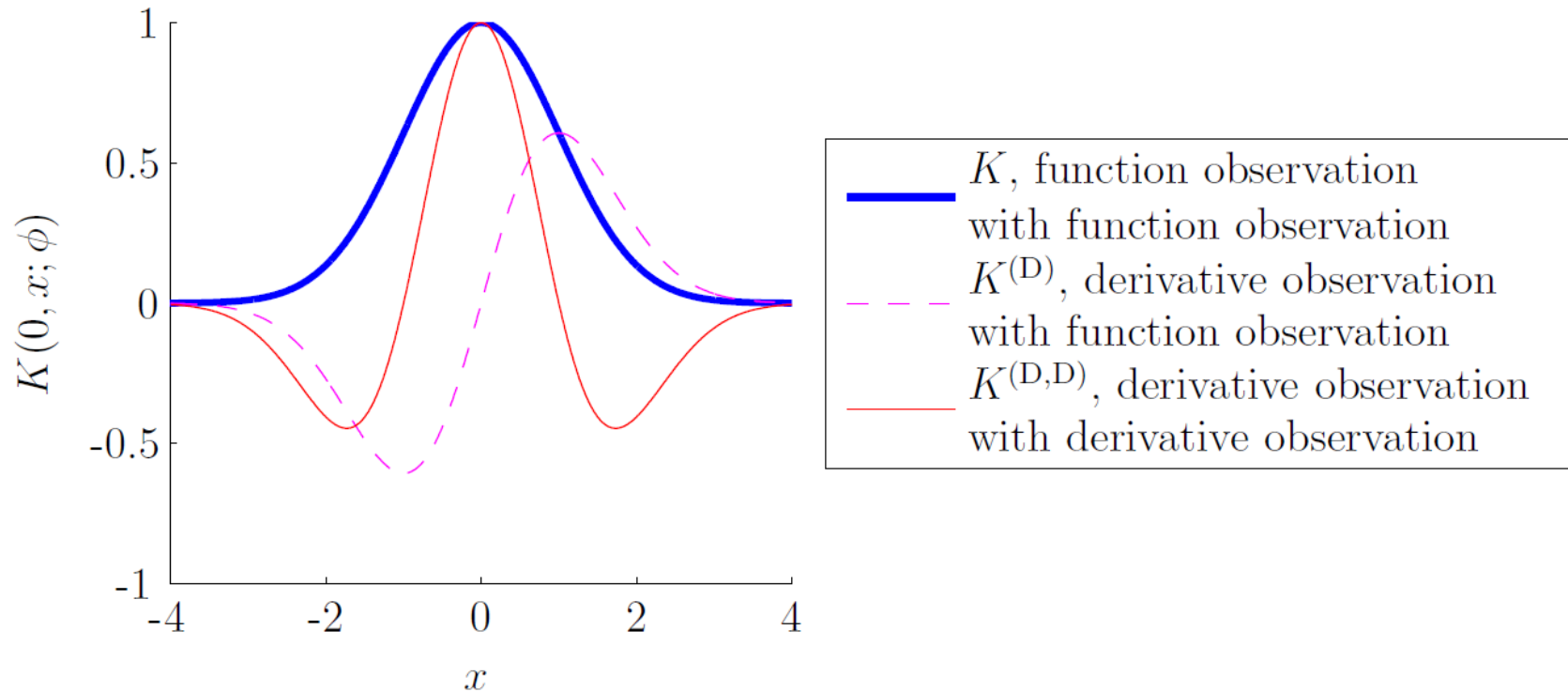
derivative
observation at
 x_i and function
observation at
 x_j .

$$K_D(x_i, x_j) = \left. \frac{\partial}{\partial x} K(x, x_j) \right|_{x=x_i}$$

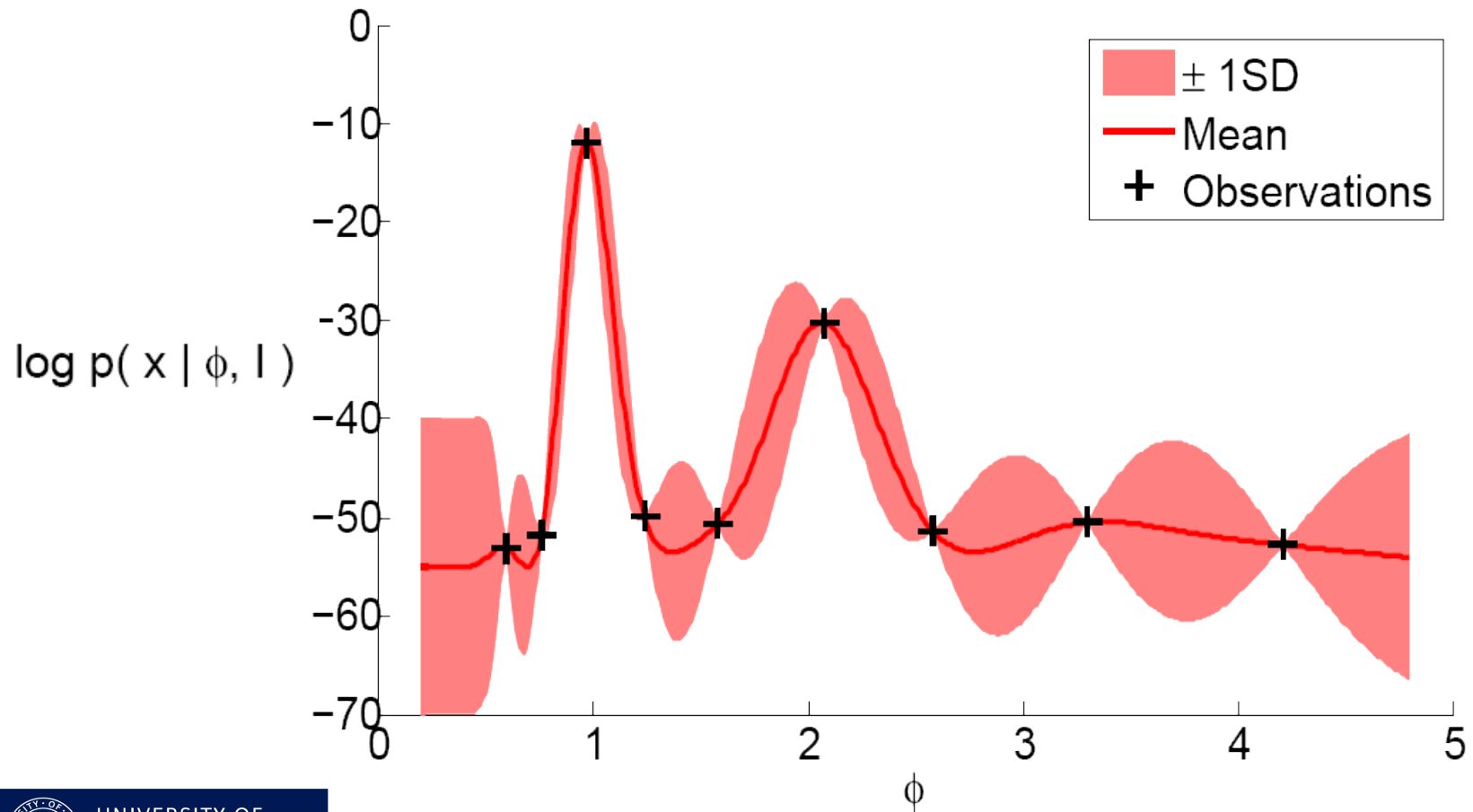
derivative
observation at
 x_i and derivative
observation at
 x_j .

$$K_{D,D}(x_i, x_j) = \left. \frac{\partial}{\partial x'} \frac{\partial}{\partial x} K(x, x') \right|_{x=x_i} \Big|_{x'=x_j}$$

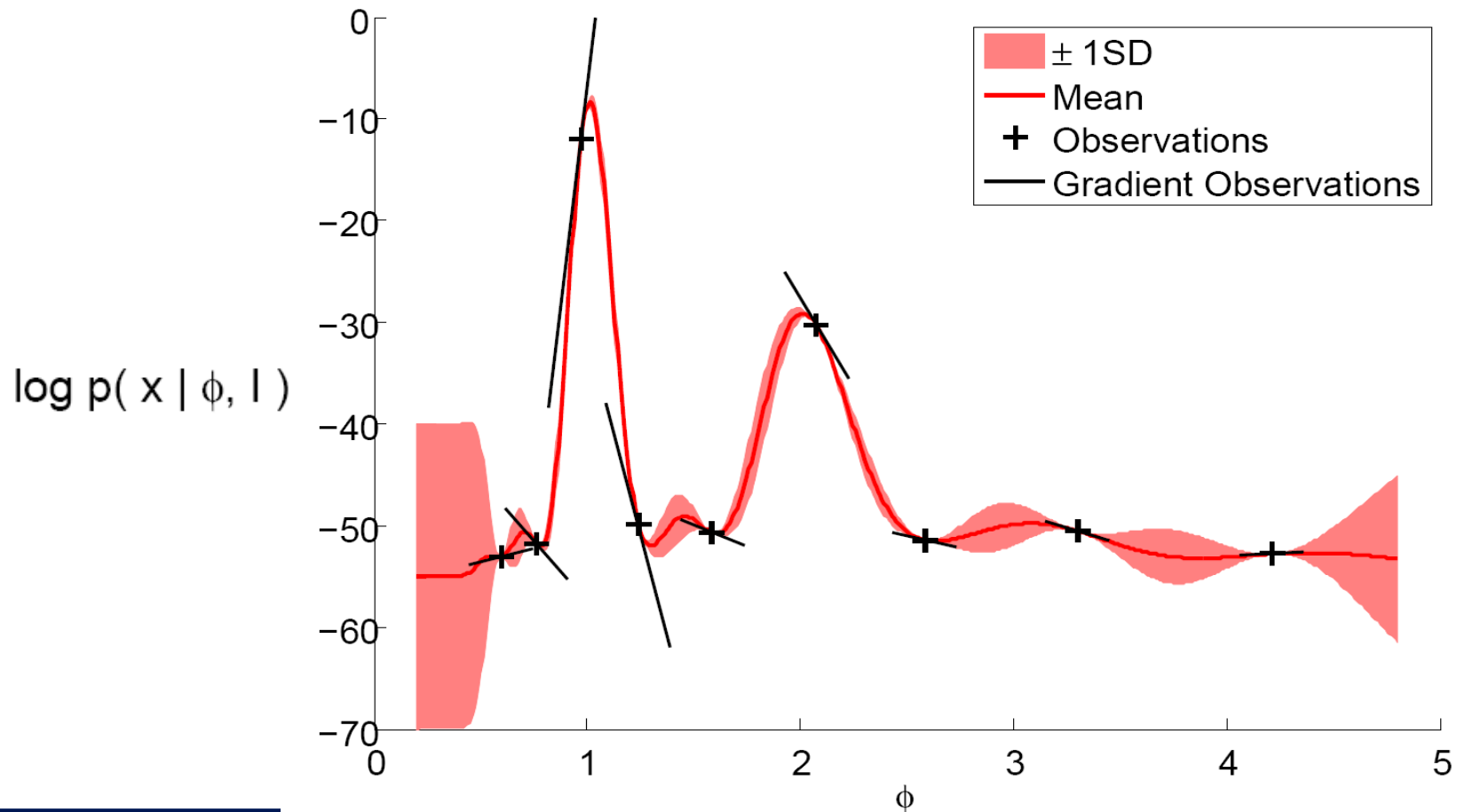
We can modify the squared exponential covariance to manage **derivative** observations.



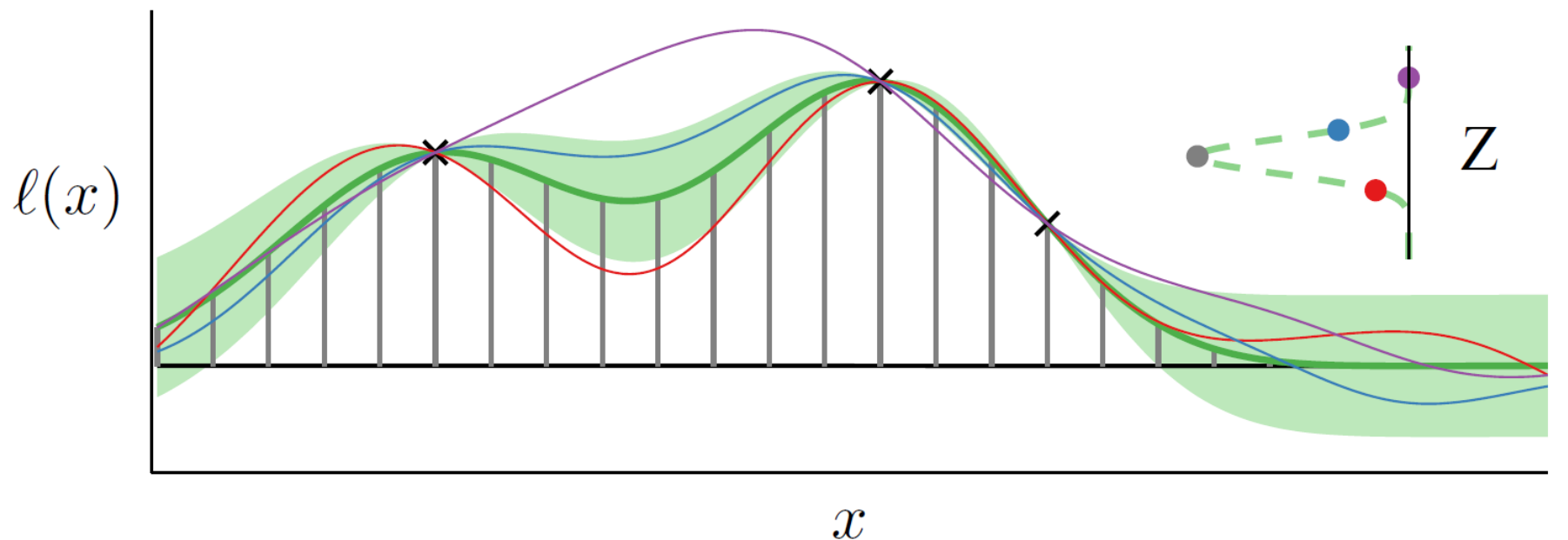
We can improve our inference by including observations of the **gradient** of a function.



We can improve our inference by including observations of the **gradient** of a function.



We can use observations of an integrand ℓ in order to perform inference for its **integral**, Z : this is known as **Bayesian Quadrature**.



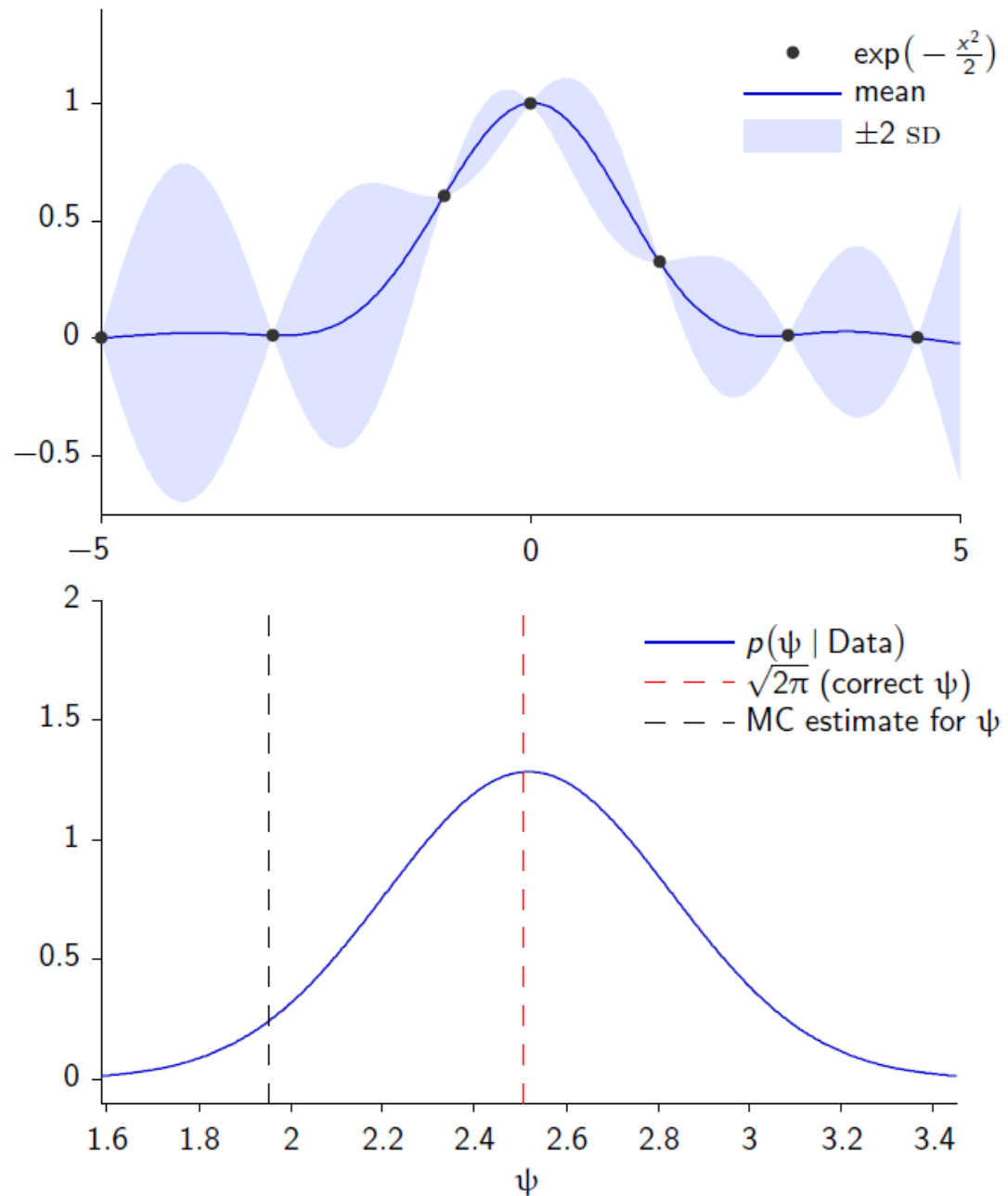
- × samples
- GP mean
- GP mean \pm SD
- expected Z
- - - $p(Z|\text{samples})$
- draw from GP
- draw from GP
- draw from GP



Consider the
integral

$$\psi = \int_{-5}^5 \exp\left(-\frac{x^2}{2}\right) dx.$$

Bayesian
quadrature
achieves **more
accurate** results
than Monte Carlo,
and provides an
estimate of our
uncertainty.



The enormous flexibility afforded by covariance functions comes at a price: **hyperparameters**, which must be **marginalised**.

$$p(y_{\star}|z_d) = \frac{\int p(y_{\star}|z_d, \phi) p(z_d|\phi) p(\phi) \, d\phi}{\int p(z_d|\phi) p(\phi) \, d\phi}$$

Unfortunately, these integrals are non-analytic.

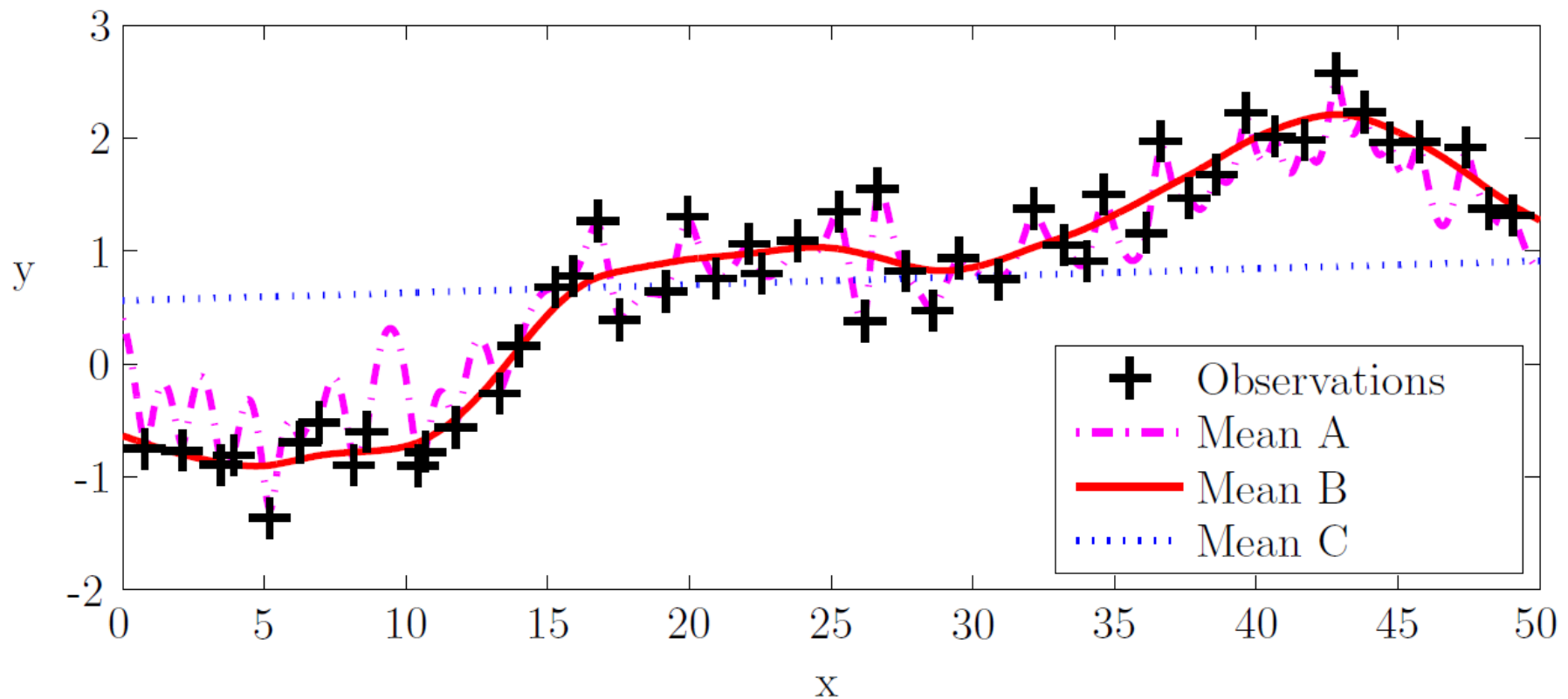


Given that we don't to fix y_* , the two important terms in our integrands are the **likelihood** and the **prior** (specifically, their product, proportional to the posterior for ϕ).

$$p(y_* | z_d) = \frac{\int \overbrace{p(y_* | z_d, \phi) p(z_d | \phi)}^{\text{likelihood prior}} p(\phi) \, d\phi}{\int p(z_d | \phi) p(\phi) \, d\phi}$$



The hyperparameter priors can have a **significant influence** on our inference. Prior A favours small input scale, prior C favours large input scale and prior B is uninformative.



Selecting priors is easy.

Prior

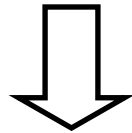
Use what you know.

If probability theory makes 'wrong' predictions,
then we have **learned something!**

Model (/)



Probability
theory



Predictions



?
≈

Our expectations

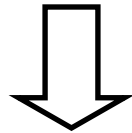


If probability theory makes 'wrong' predictions,
then we have **learned something!**

Model (I)



Probability
theory



Predictions



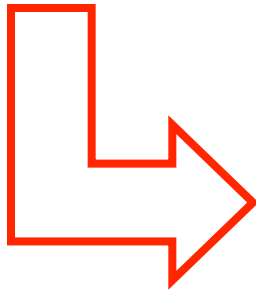
≠

One of these
two must be
wrong

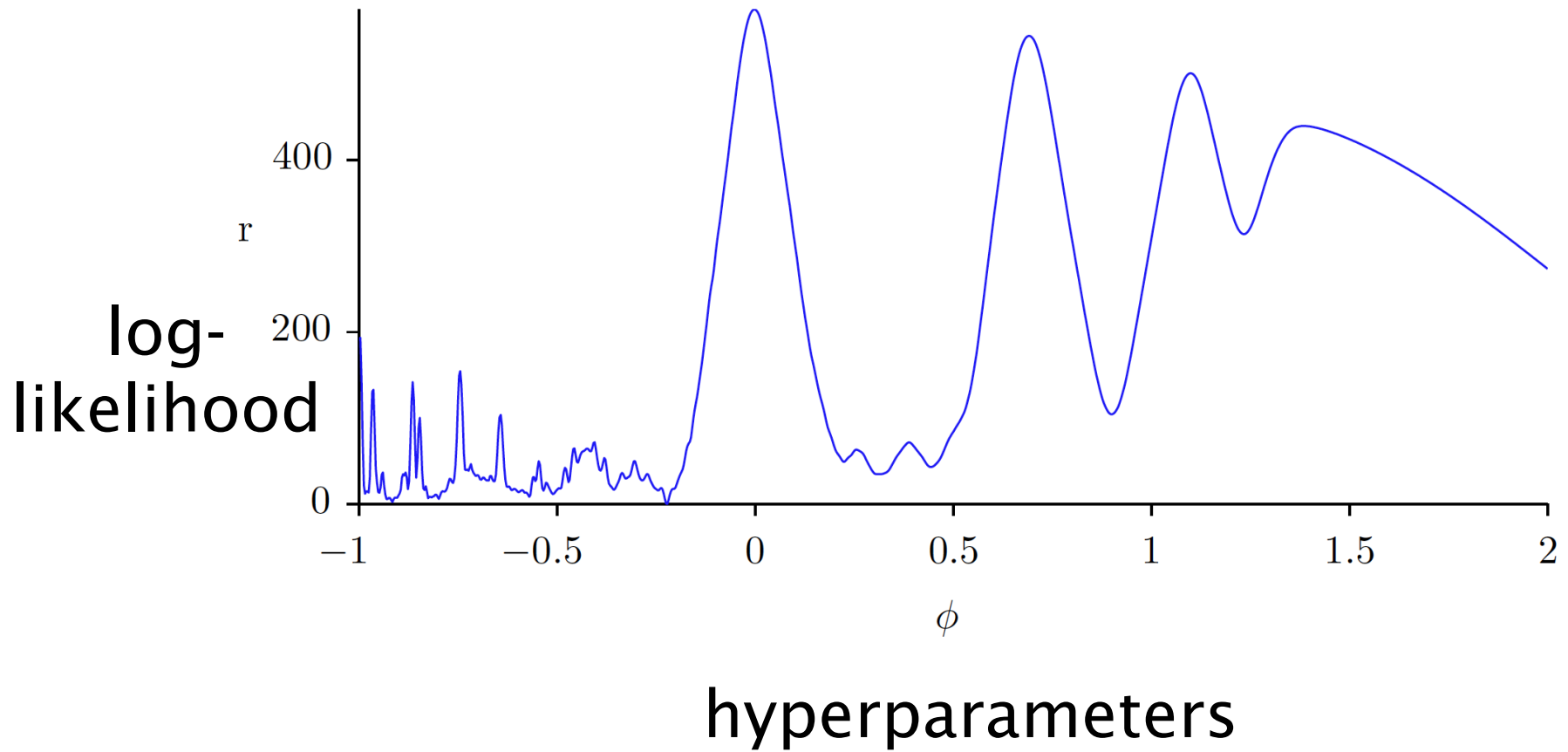
Our expectations



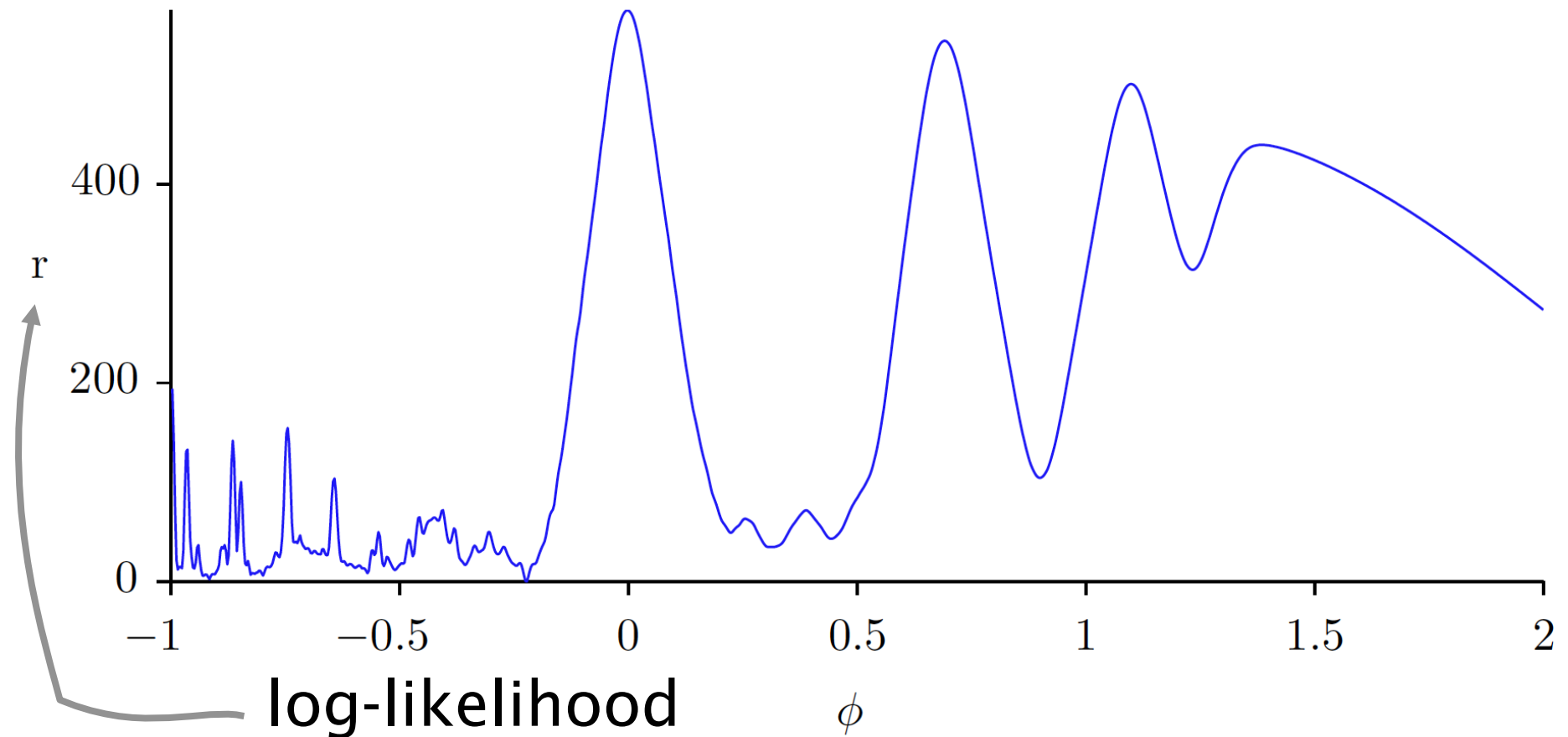
In this way, we are led to construct successively
better models.



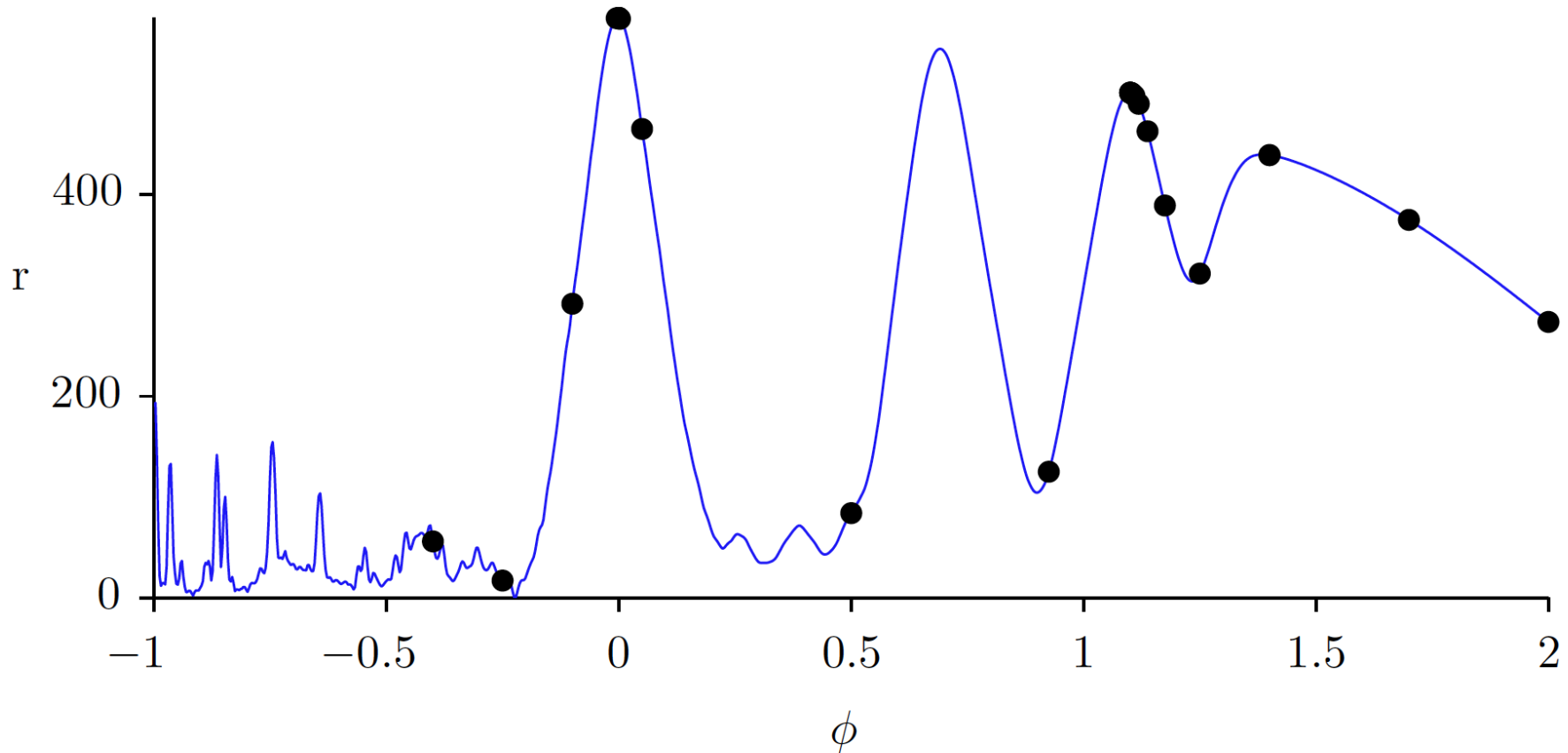
Marginalisation requires **quadrature**, which presents two challenges: integrand exploration, and integral estimation.



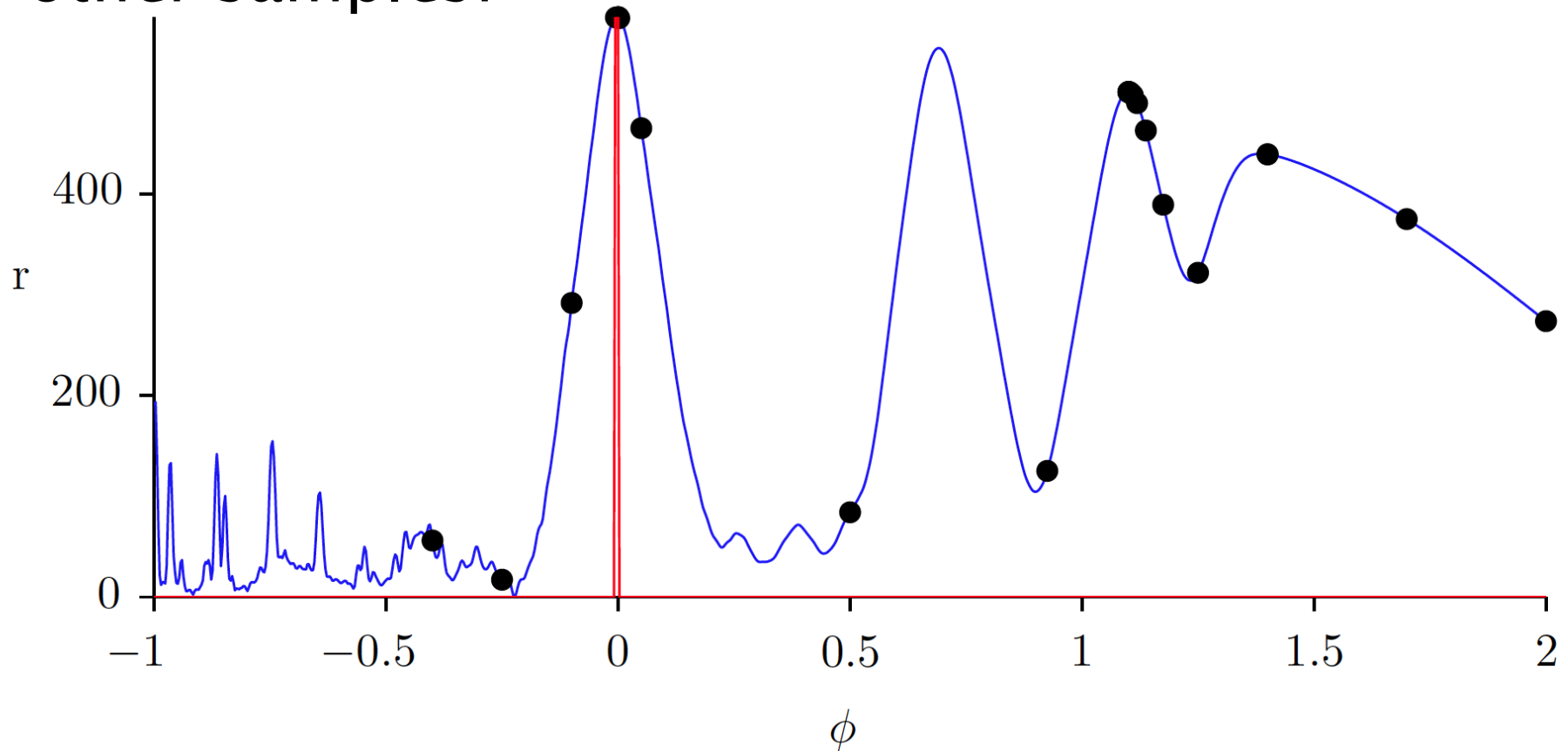
There are **many different** approaches to quadrature for probabilistic integrals; integrand estimation is usually undervalued.



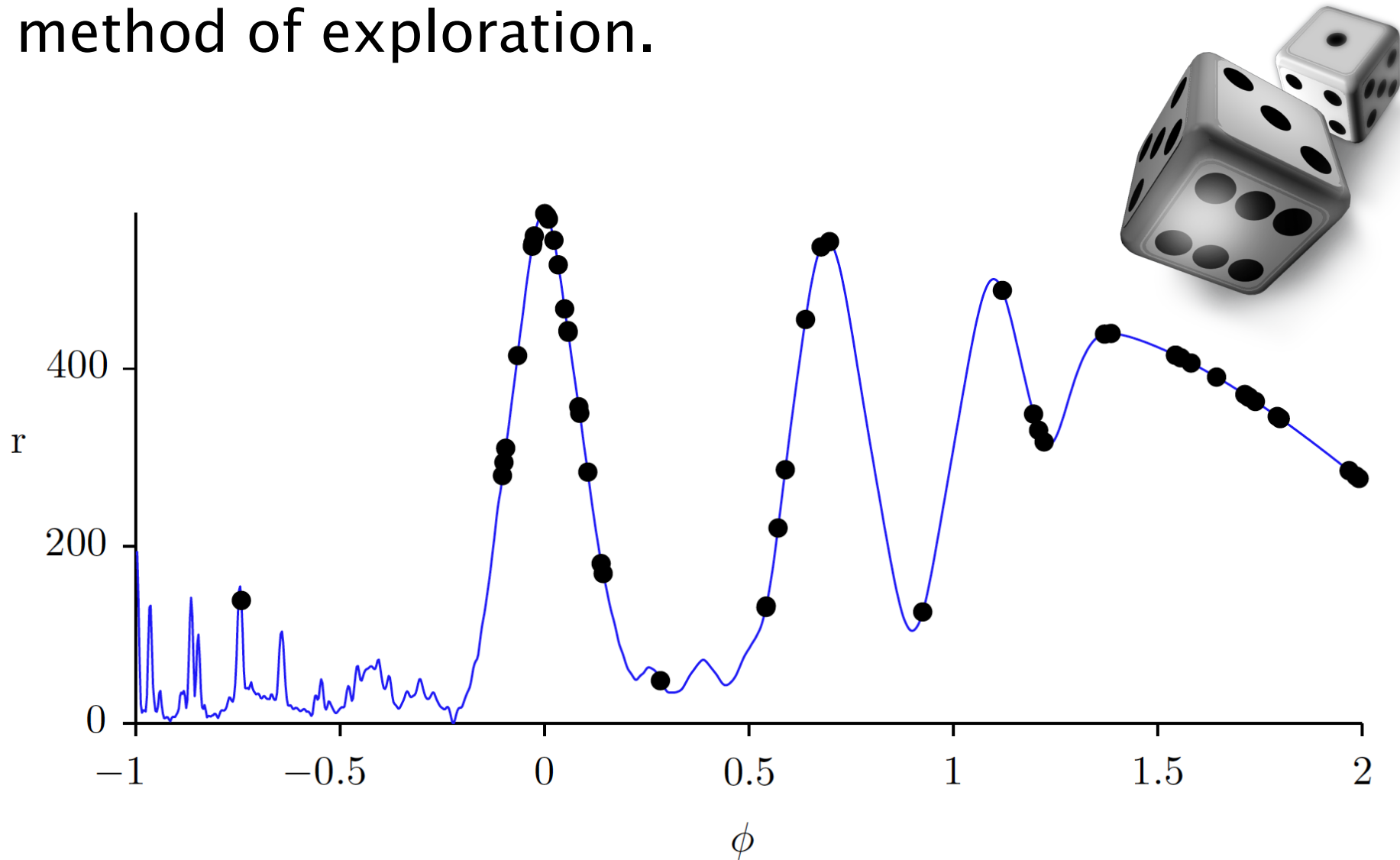
Optimisation (as in **maximum likelihood**), particularly using global optimisers, gives a reasonable heuristic for exploring the integrand.



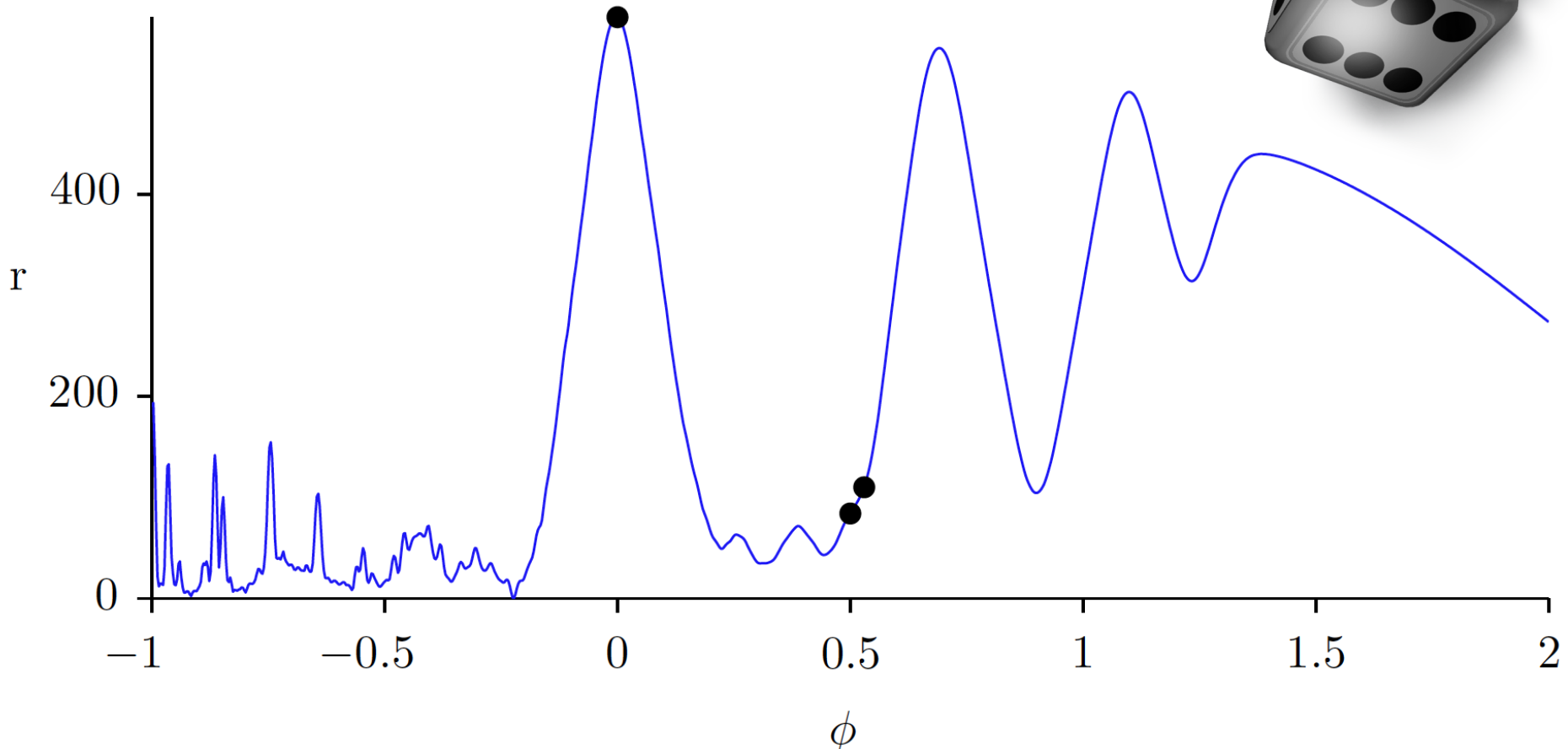
However, maximum likelihood is an **unreasonable** way of **estimating** a multi-modal likelihood integrand: why throw away all those other samples?



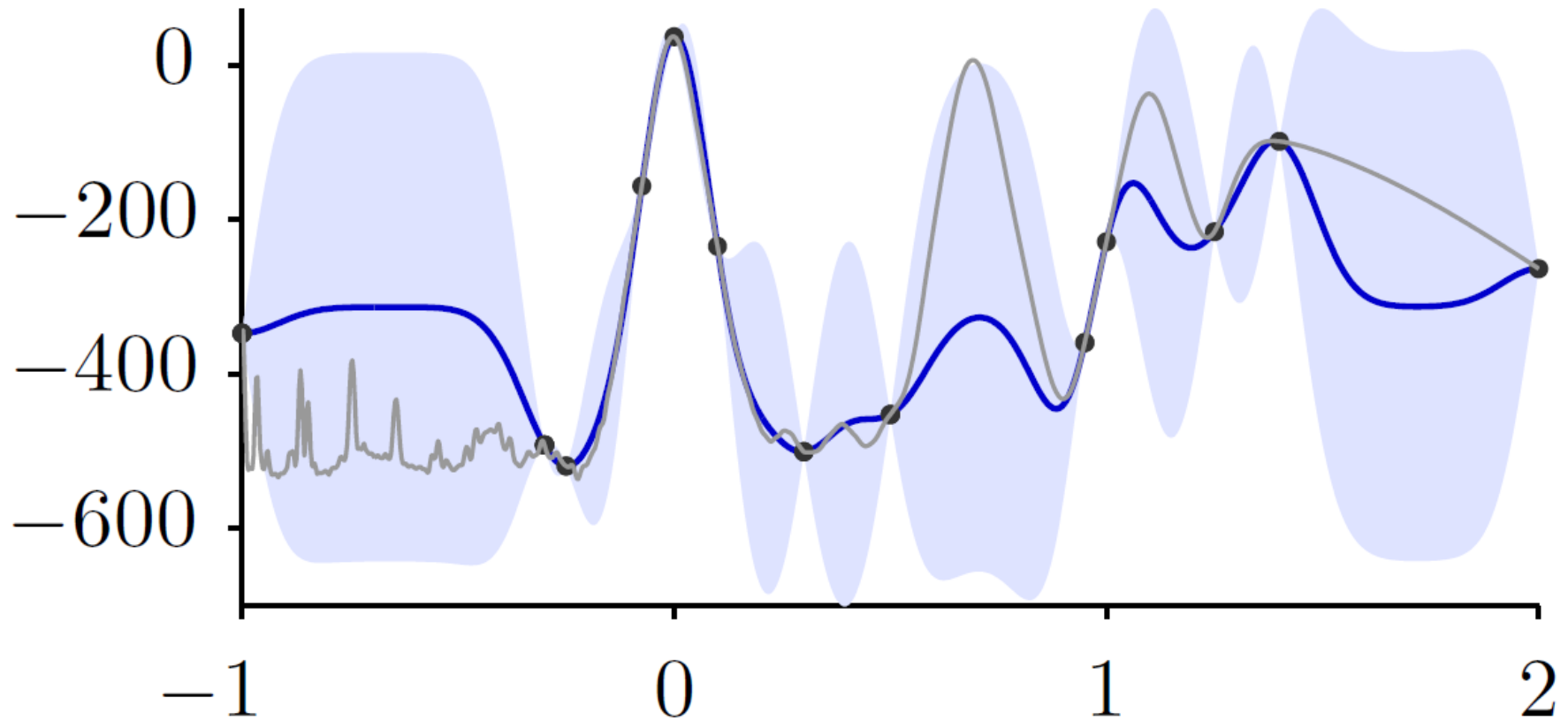
Monte Carlo schemes give a another reasonable method of exploration.



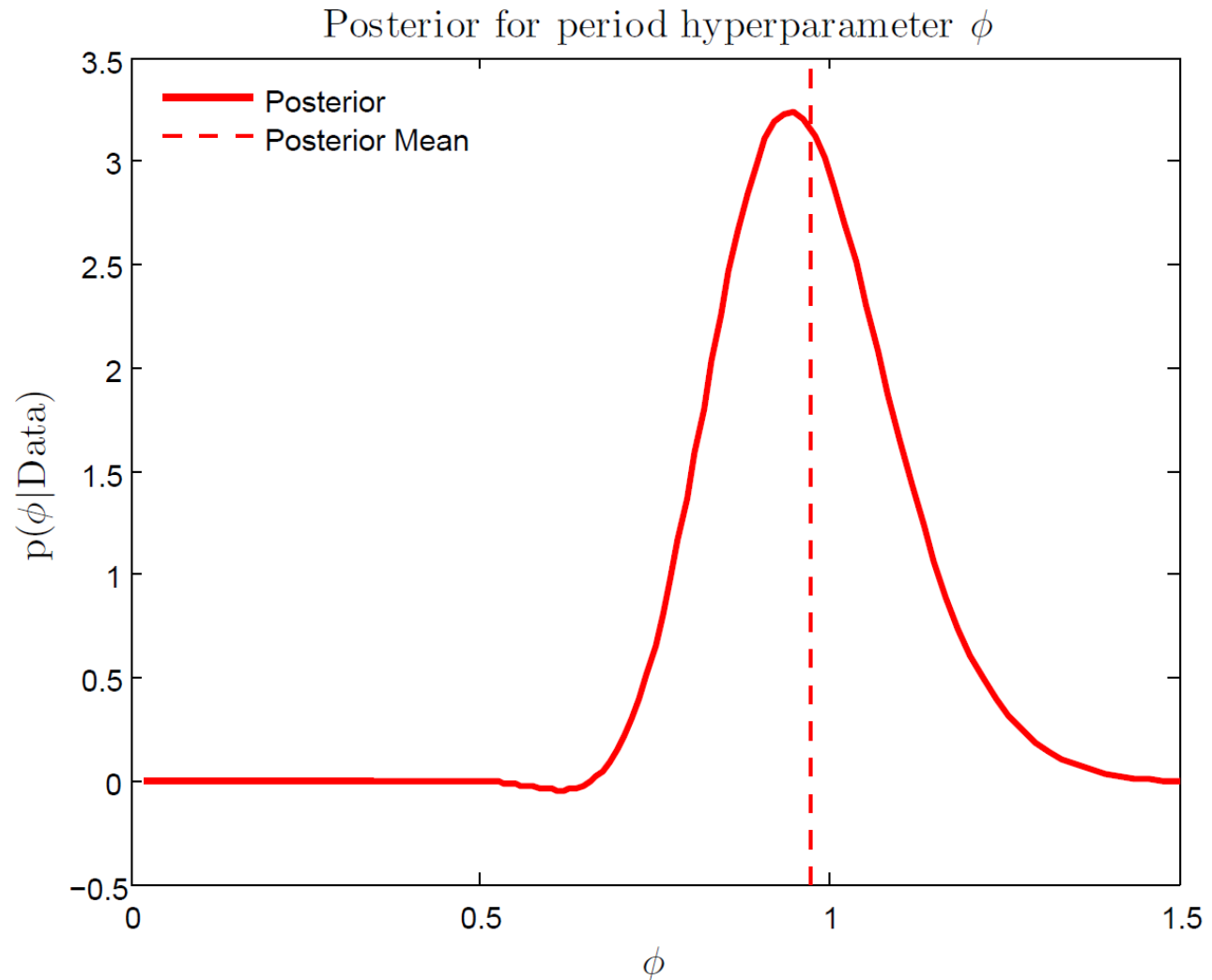
Monte Carlo schemes give a fairly reasonable method of exploration; but a less **reasonable** means of integrand estimation.



Bayesian Monte Carlo uses samples obtained via Monte Carlo within a Bayesian quadrature framework to give an estimate for the integral.



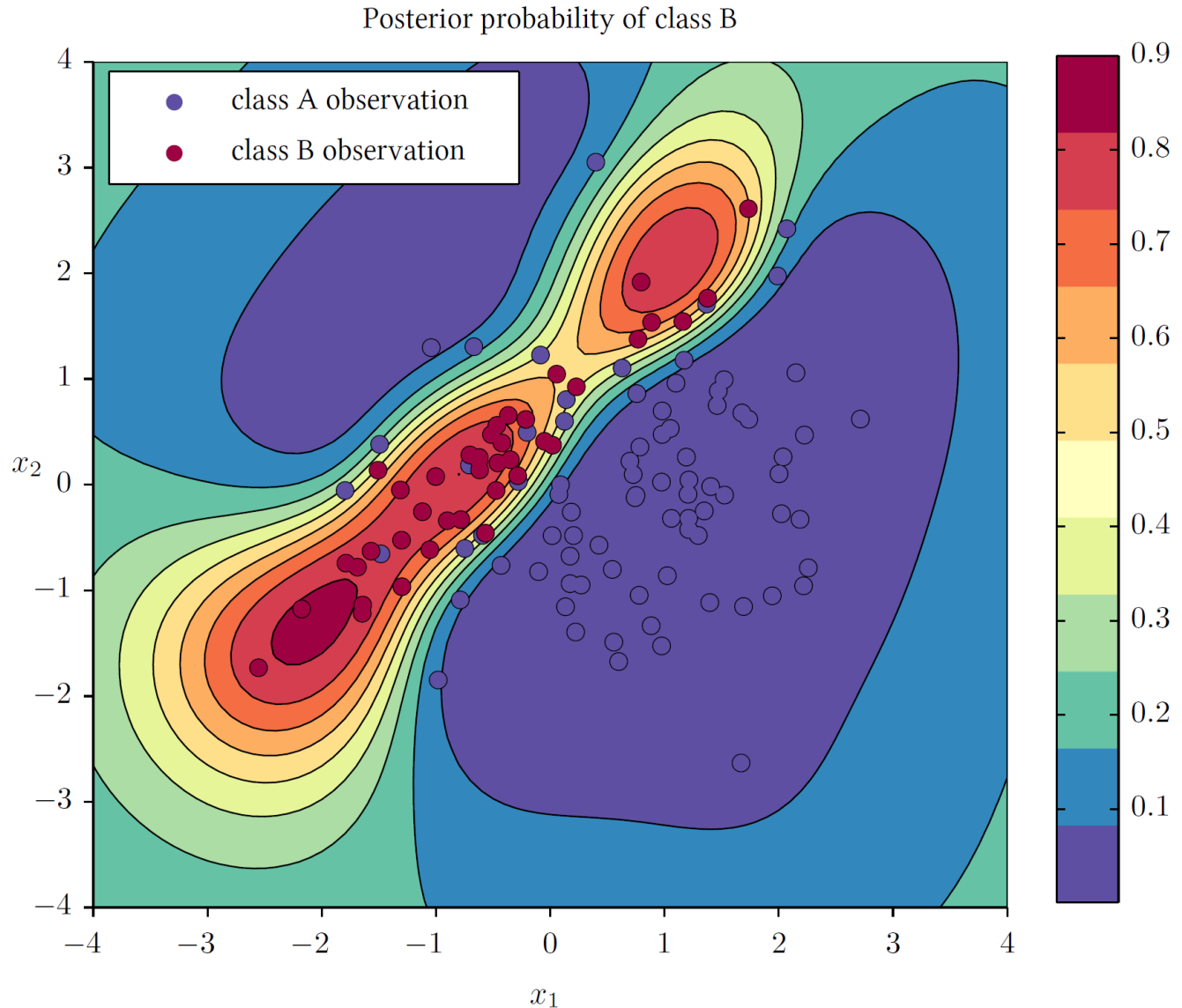
With Bayesian quadrature, we can also estimate the **posterior distributions** for any **hyperparameters**.



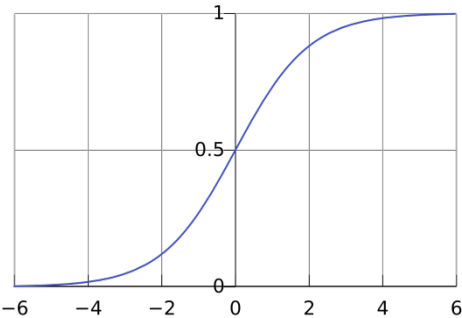
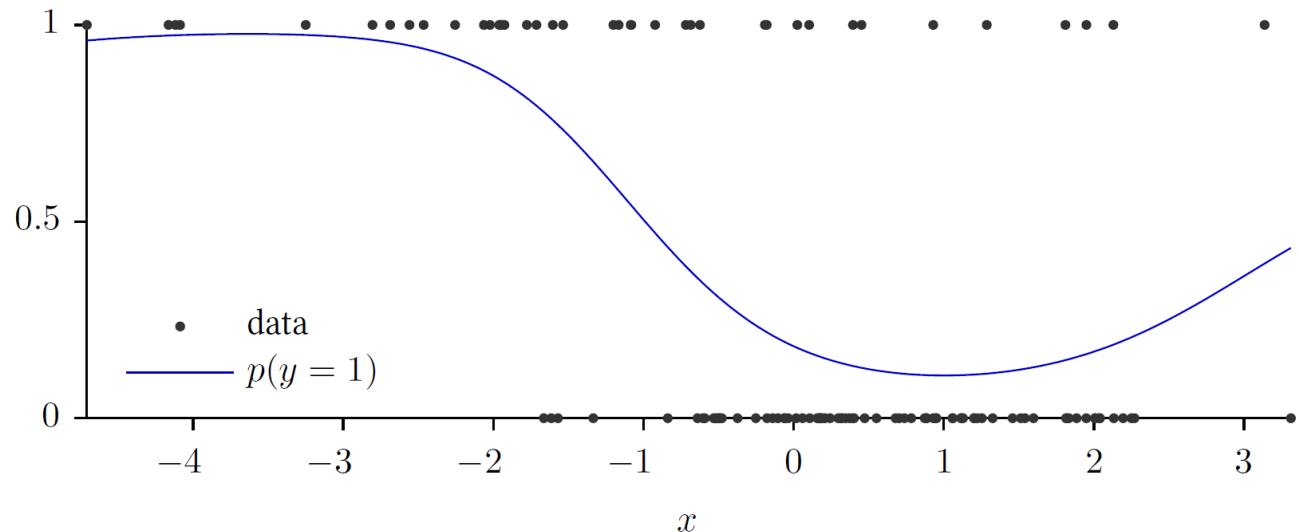
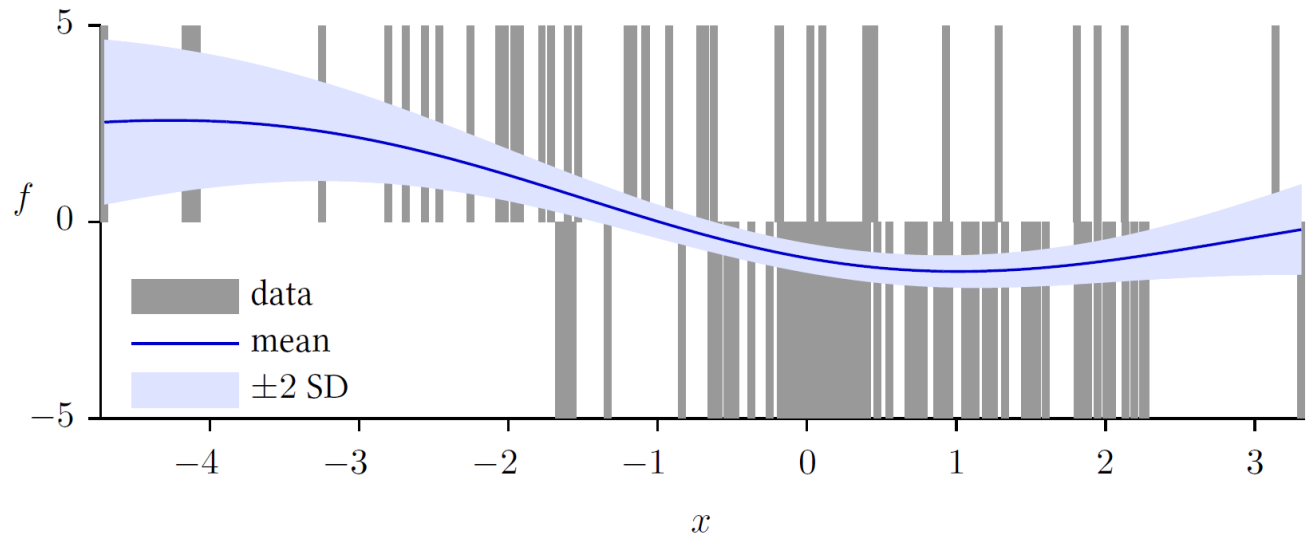
There are many approaches to hyperparameter marginalisation, but only two are recommended.

Likelihood	Marginalisation
Unimodal or high dimensional	Maximum likelihood.
Multimodal or computationally expensive	Bayesian Monte Carlo.

We can put Gaussian processes to work not just for regression. but also for **classification**.



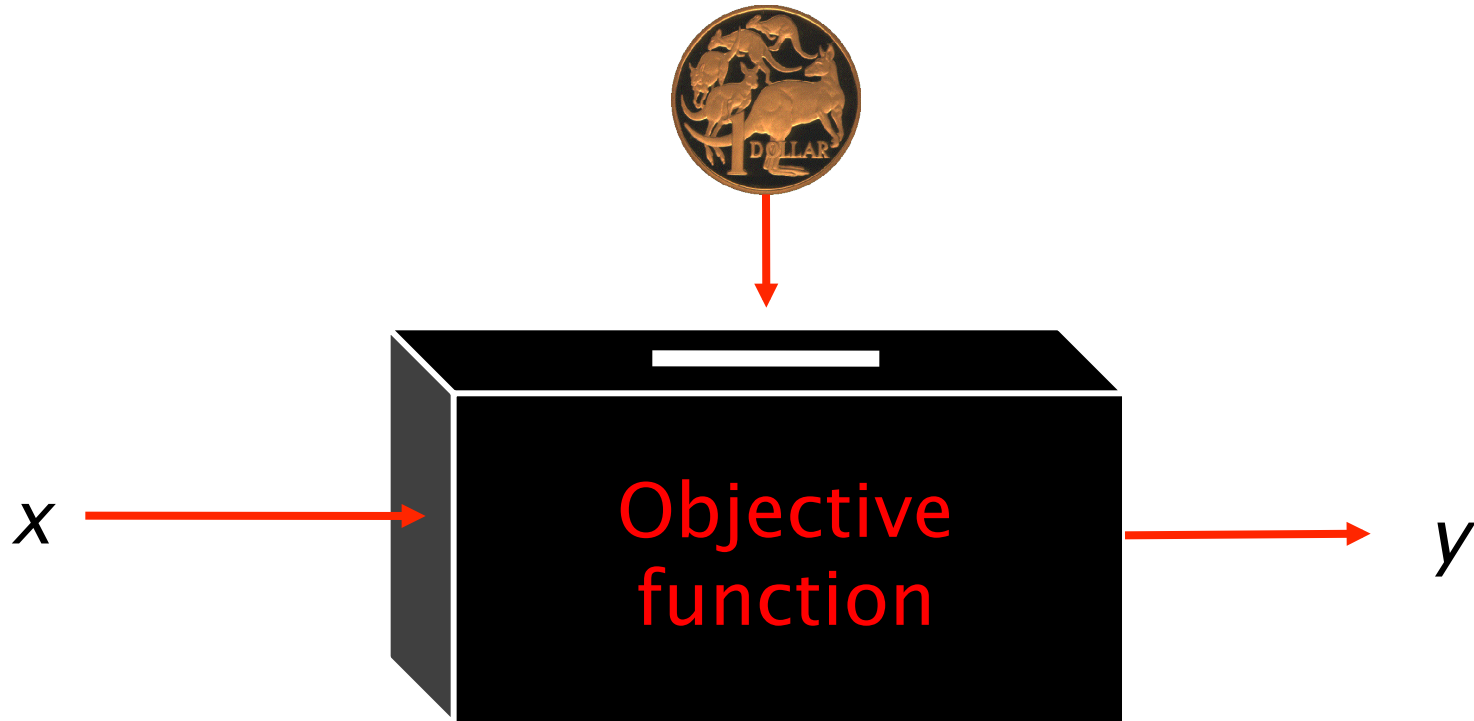
To do so, use a Gaussian process to model a latent variable, **mapped through a sigmoid** to a discrete class label.



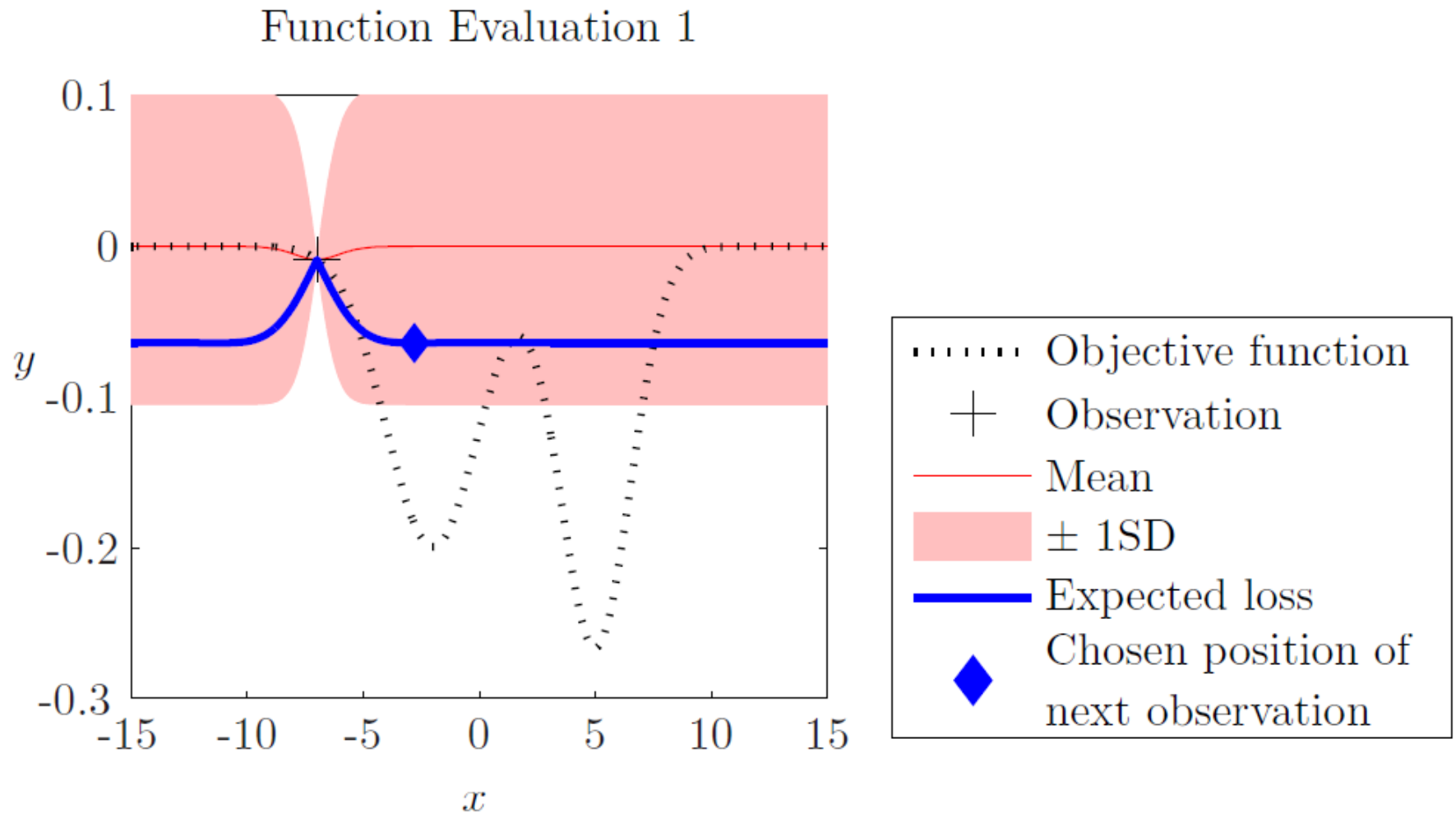
Unfortunately, using this sigmoid makes inference **intractable**. Approximate inference can be achieved using a number of algorithms.

Algorithm	Speed	Accuracy
Laplace approximation	Very fast	Low.
Expectation Propagation	Fast	High.
Markov Chain Monte Carlo	Very slow	Very high.

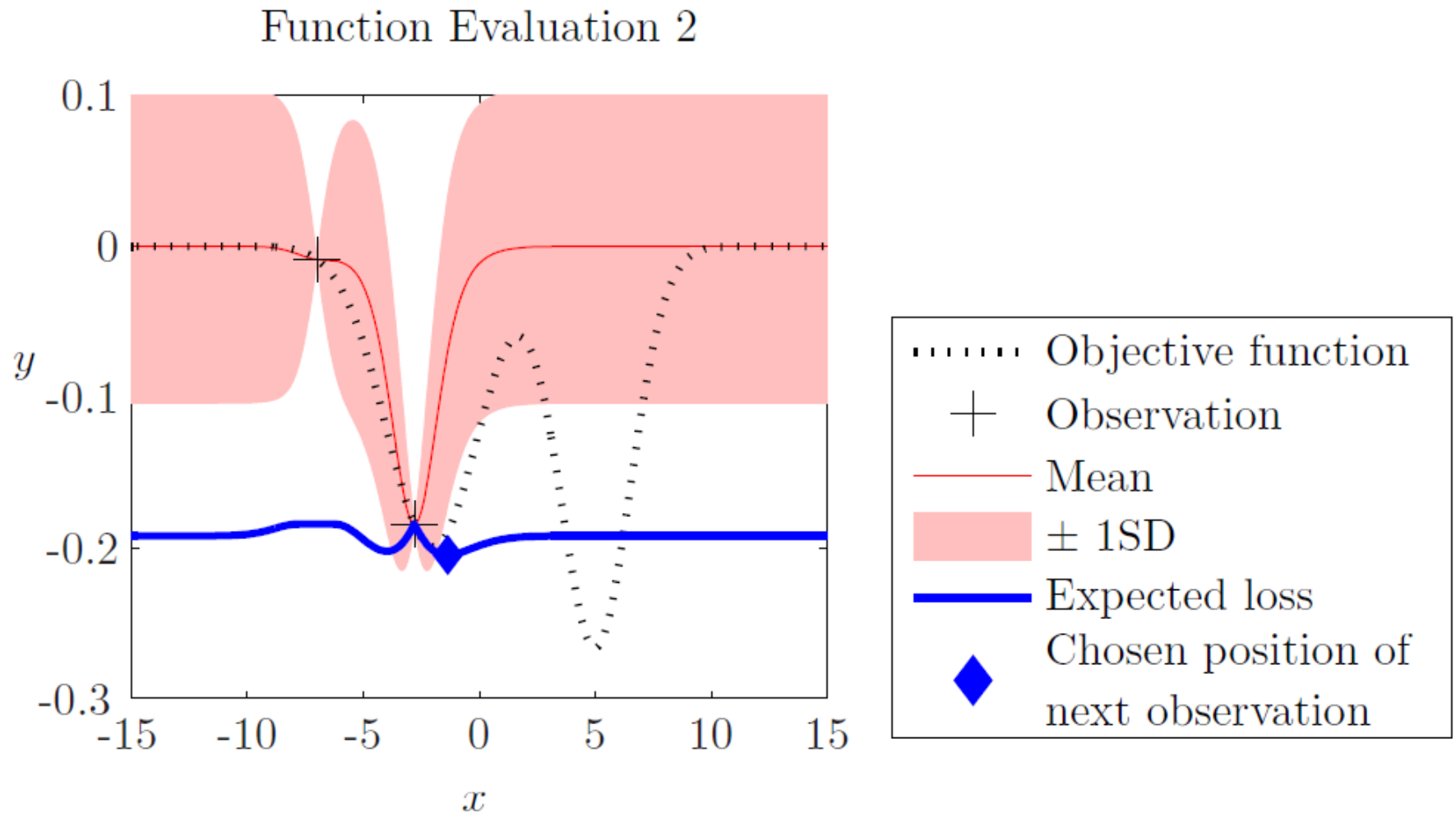
We treat global optimisation as a Bayesian
decision problem.



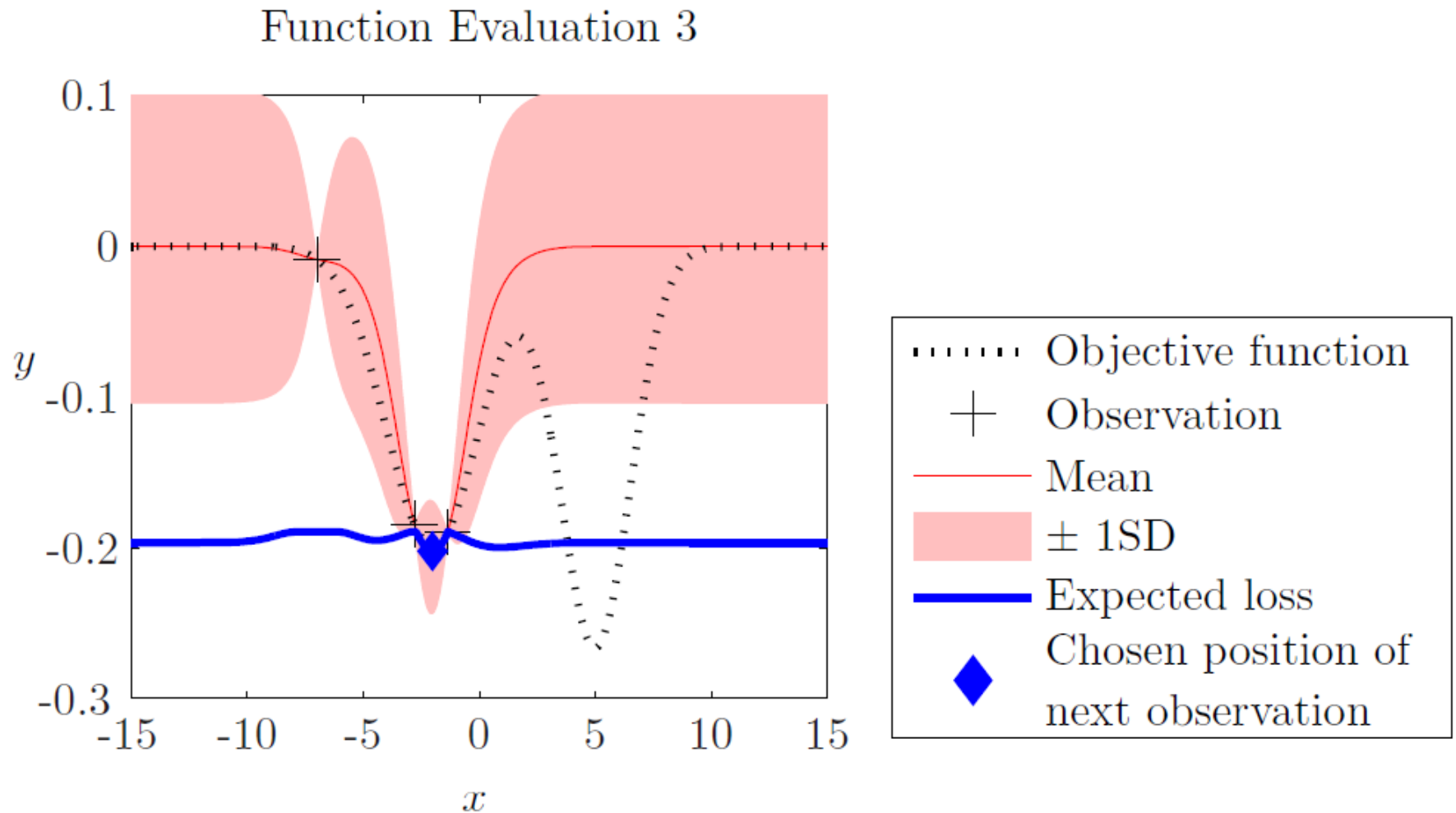
We can also use Gaussian processes for **optimisation**.



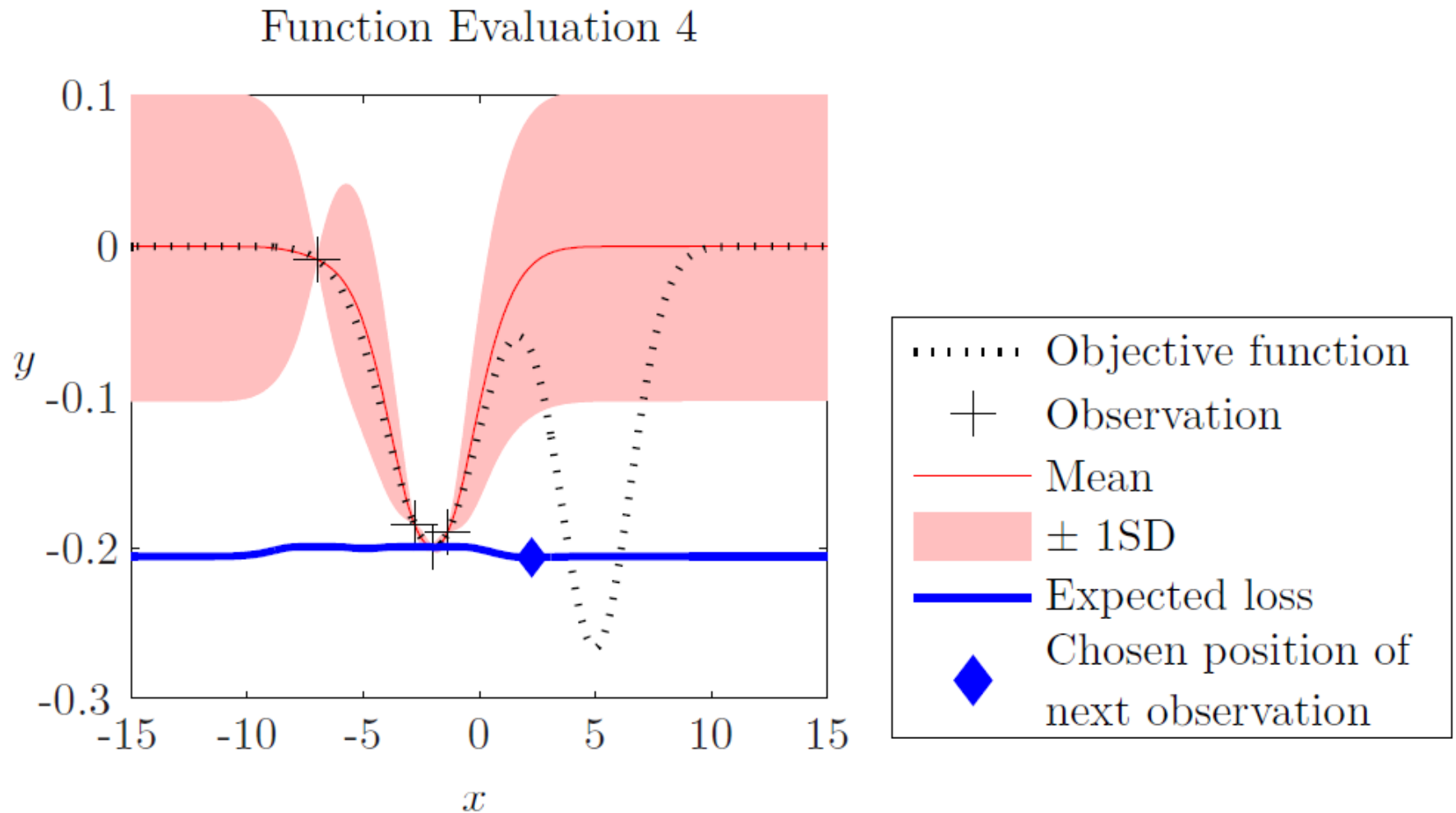
We can also use Gaussian processes for **optimisation**.



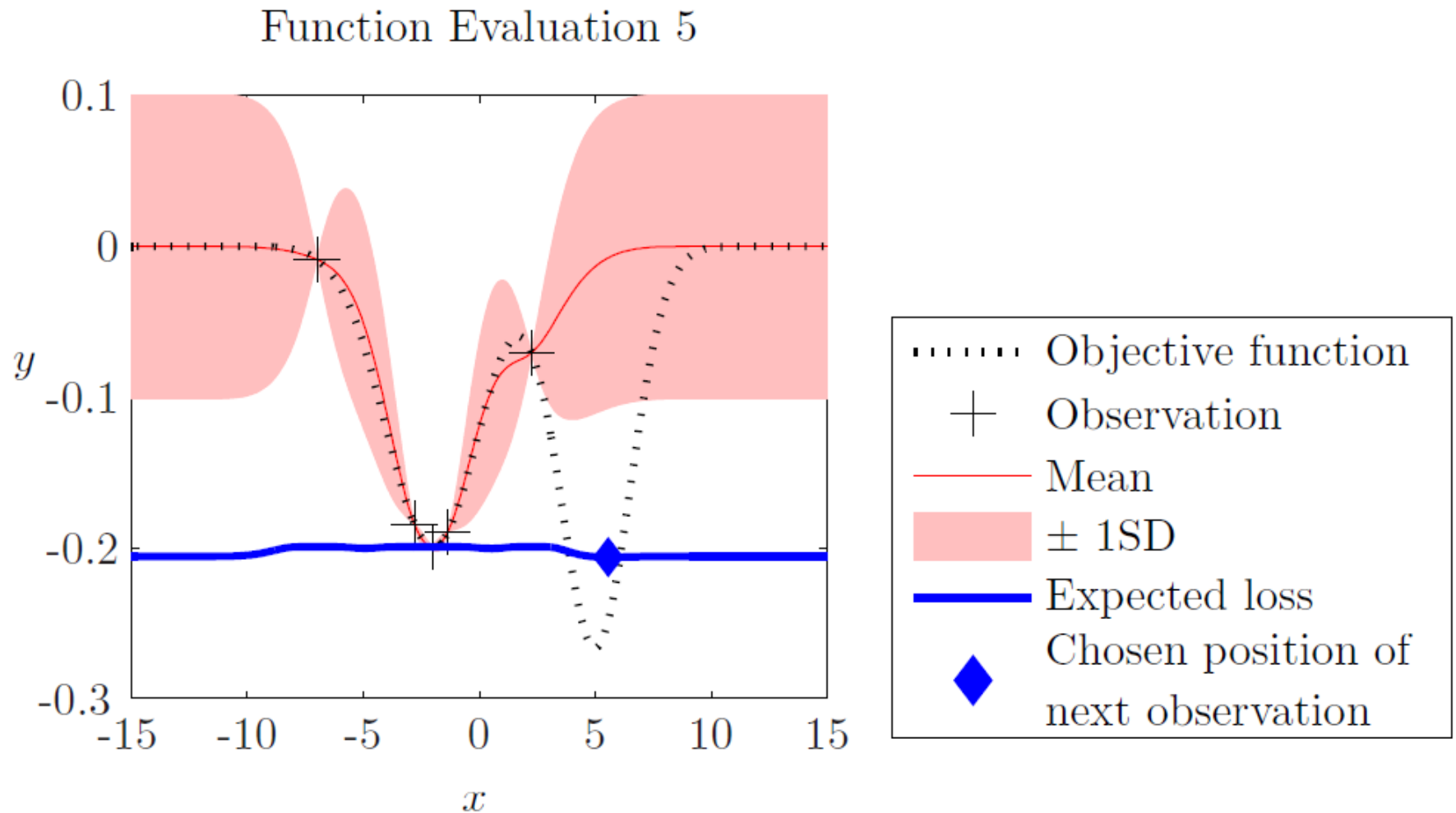
We can also use Gaussian processes for **optimisation**.



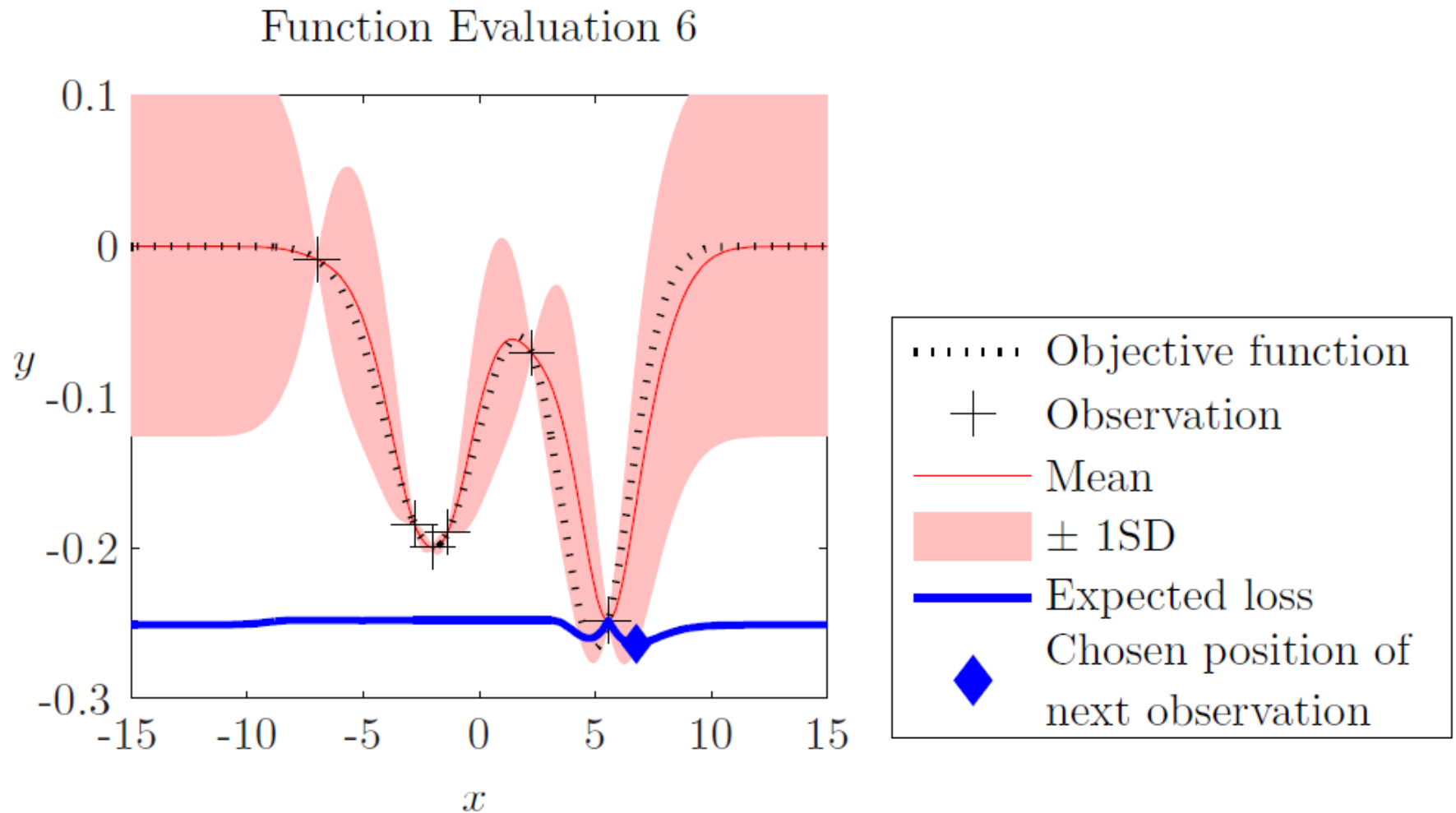
We can also use Gaussian processes for **optimisation**.



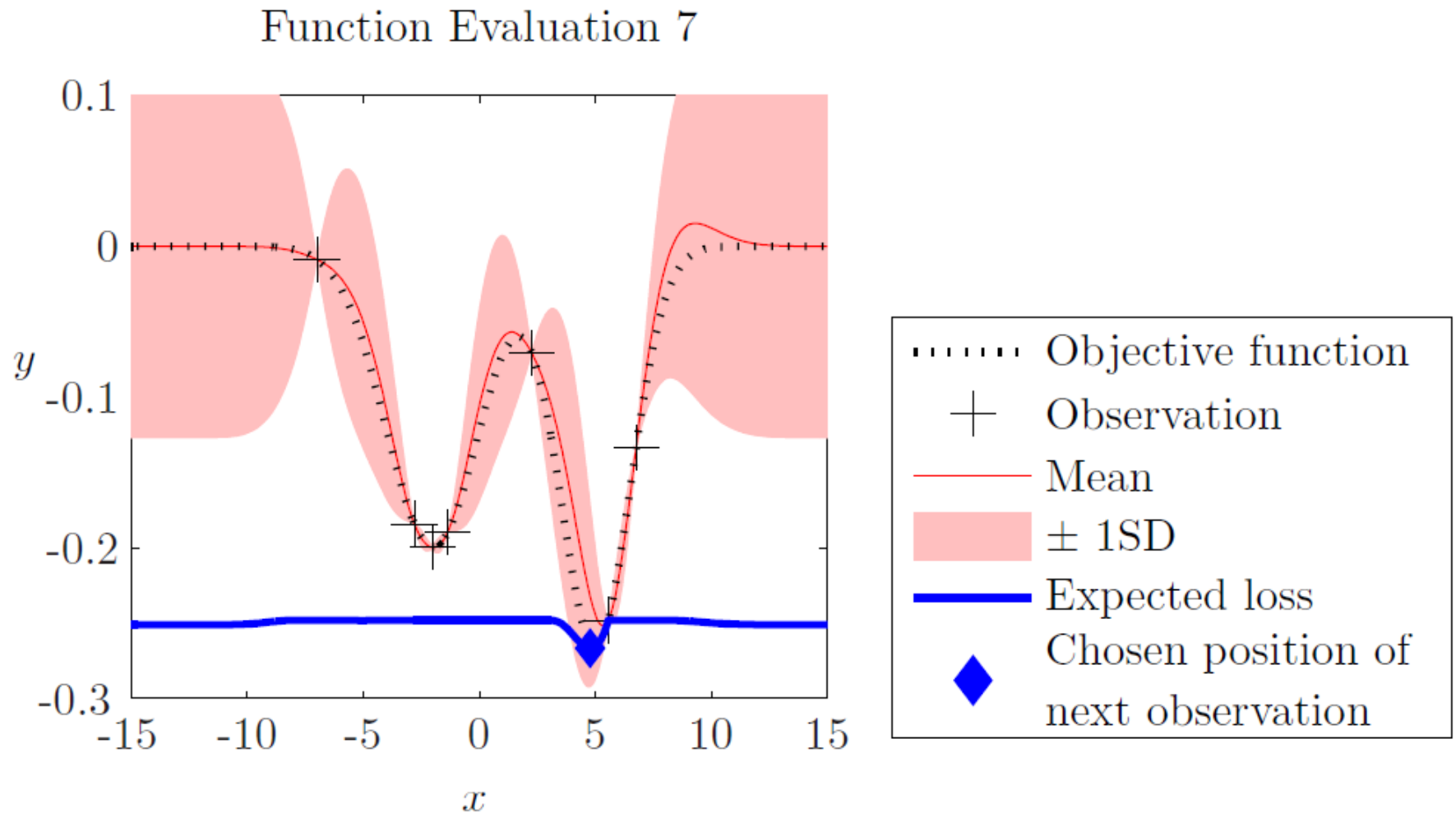
We can also use Gaussian processes for **optimisation**.



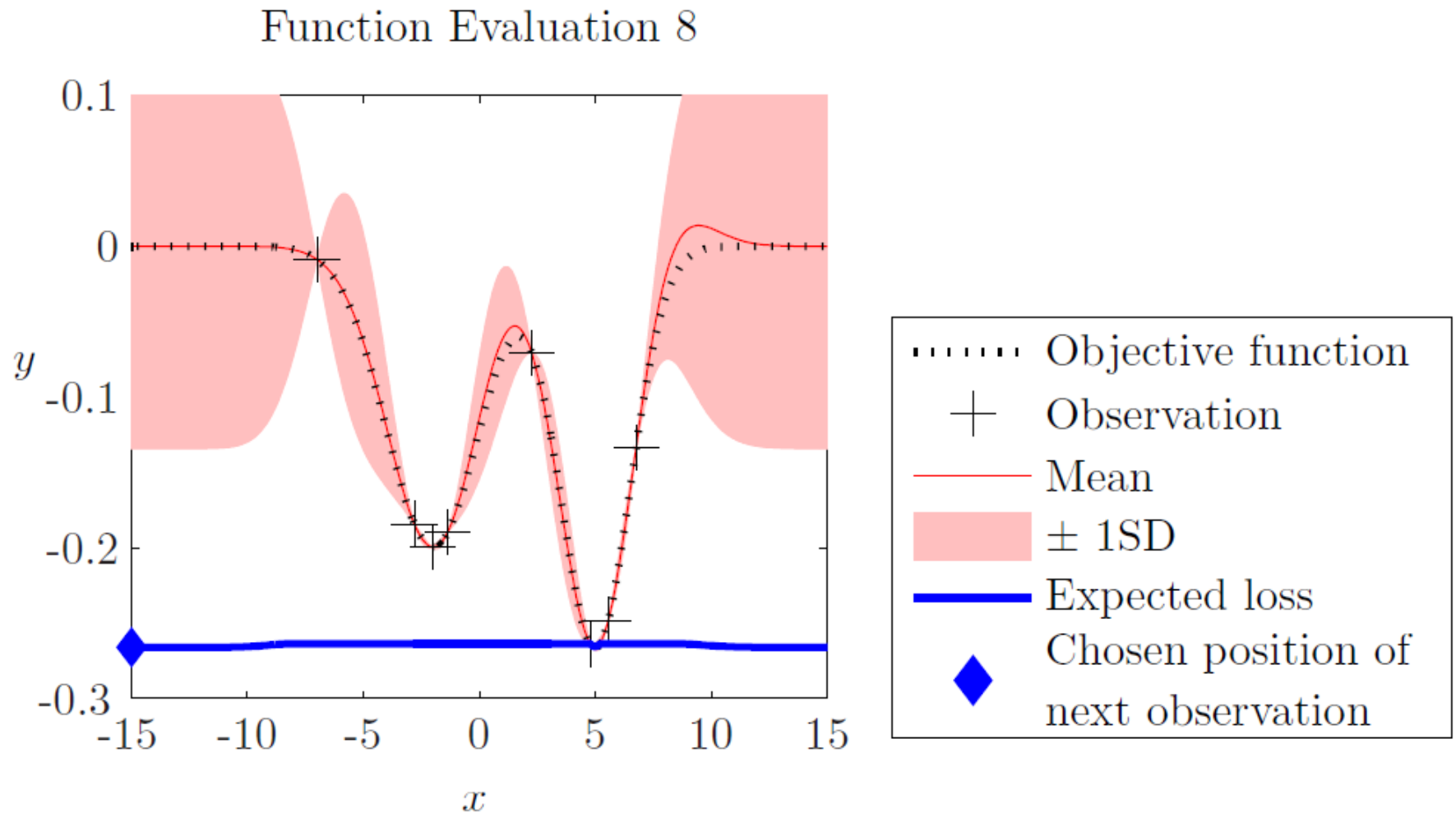
We can also use Gaussian processes for **optimisation**.



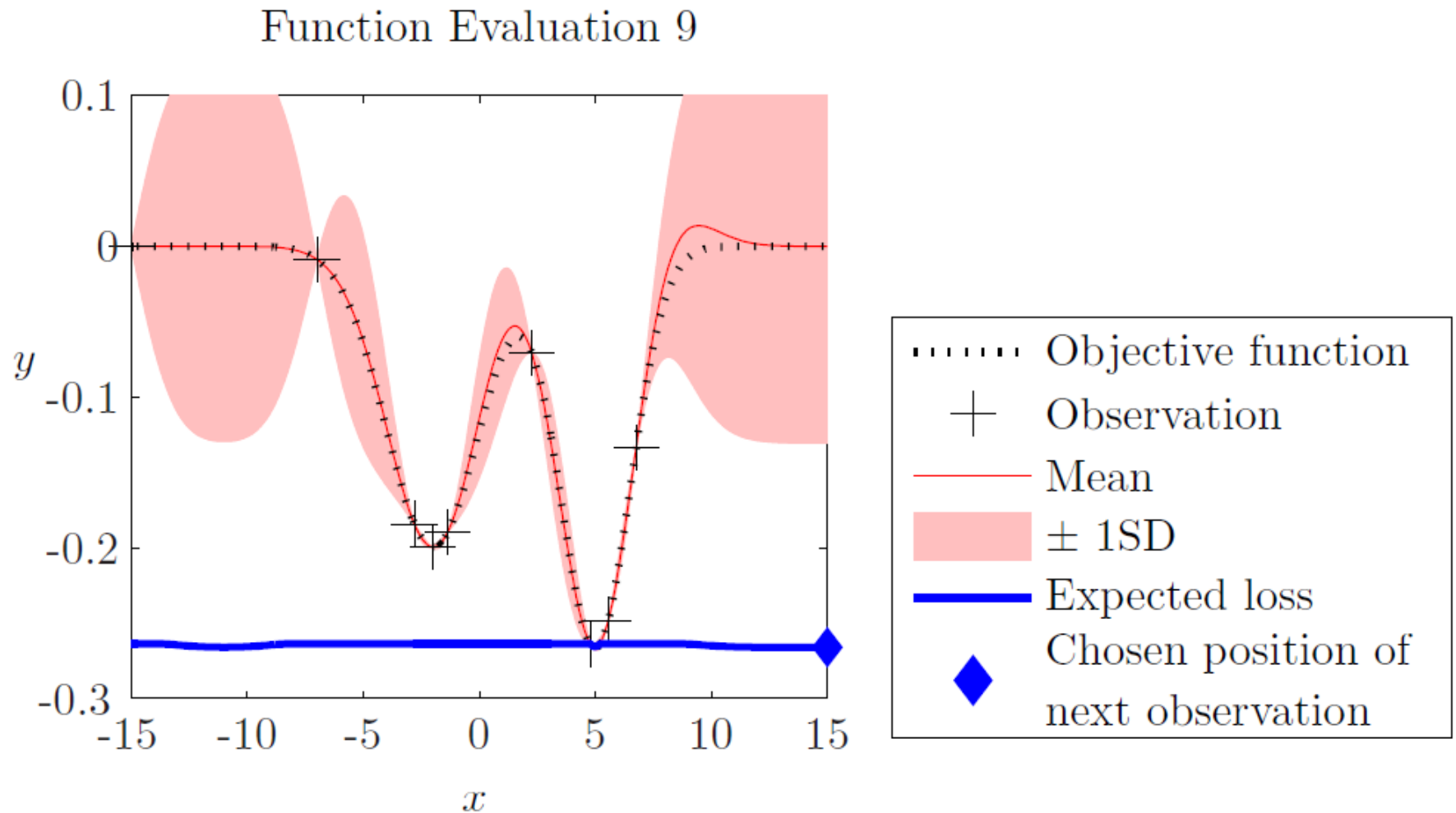
We can also use Gaussian processes for **optimisation**.



We can also use Gaussian processes for **optimisation**.



We can also use Gaussian processes for **optimisation**.



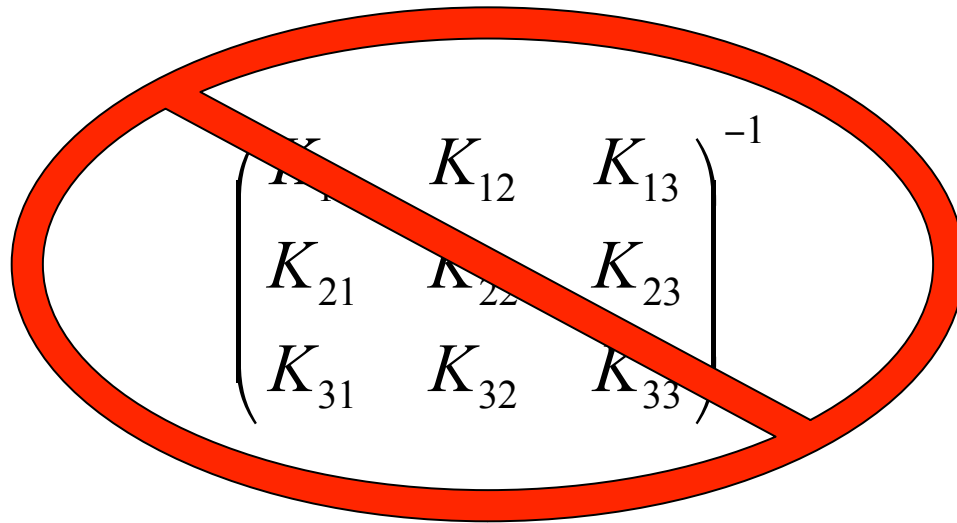
The key **computational bottleneck** associated with Gaussian processes is resolving $\text{inv}(K) v$, or, equivalently, solving $v = K x$ for x .

$$\begin{pmatrix} K_{11} & K_{21} & K_{13} & \cdots \\ K_{21} & K_{22} & K_{23} & \\ K_{31} & K_{32} & K_{33} & \\ \vdots & & & \ddots \end{pmatrix}^{-1} \begin{pmatrix} v_1 \\ v_2 \\ v_3 \\ \vdots \end{pmatrix}$$

Our choice of a method to solve $v = K x$ for x depends on the structure of covariance K .

Covariance matrix	Solving method
Poorly conditioned	Improve conditioning, then see below.
(Just) positive semi-definite	Cholesky factorisation.
Toeplitz	Toeplitz solver.
Kronecker product	Kronecker solver.
Too big and dense	Sparsification.
Updated version of previous matrix	Update, dependent on above.

You should **never** actually **invert** a matrix.



A 3x3 matrix K is shown, enclosed in a red oval with a diagonal slash through it, indicating that matrix inversion is discouraged. The matrix is defined as:

$$K = \begin{pmatrix} K_{11} & K_{12} & K_{13} \\ K_{21} & K_{22} & K_{23} \\ K_{31} & K_{32} & K_{33} \end{pmatrix}^{-1}$$

Inversion is slow, $O(n^3)$ in matrix size n .

Inversion is also unstable; conditioning errors are significant.

Conditioning becomes an issue when we have multiple close observations, giving rows in the covariance matrix that are very similar.

$$\begin{pmatrix} 1 & 0.9999 & 0 & 0 \\ 0.9999 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0.1 \\ 0 & 0 & 0.1 & 1 \end{pmatrix} \left. \vphantom{\begin{pmatrix} 1 & 0.9999 & 0 & 0 \\ 0.9999 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0.1 \\ 0 & 0 & 0.1 & 1 \end{pmatrix}} \right\} \begin{array}{l} \text{Too} \\ \text{similar} \end{array}$$

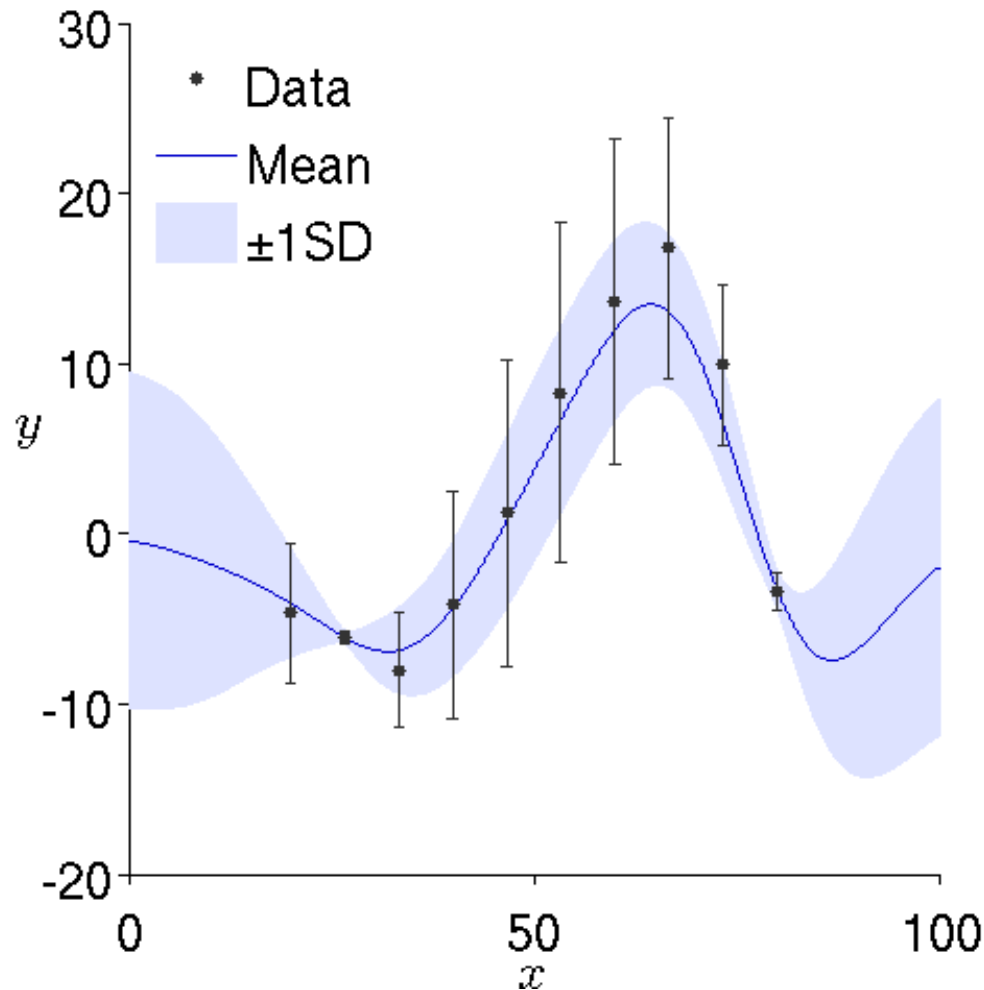
The condition number (cond) of a covariance matrix is the ratio of the largest to the smallest eigenvalue; in Matlab, things break down at about $\text{cond}(K) = 10^{16}$.

The solution to conditioning problems is to add a small positive quantity (*jitter*) to the diagonal of the covariance matrix.

$$\begin{pmatrix} 1.01 & 0.9999 & 0 & 0 \\ 0.9999 & 1.01 & 0 & 0 \\ 0 & 0 & 1.01 & 0.1 \\ 0 & 0 & 0.1 & 1.01 \end{pmatrix} \left. \vphantom{\begin{pmatrix} 1.01 & 0.9999 & 0 & 0 \\ 0.9999 & 1.01 & 0 & 0 \\ 0 & 0 & 1.01 & 0.1 \\ 0 & 0 & 0.1 & 1.01 \end{pmatrix}} \right\} \text{Sufficiently dissimilar}$$



As jitter is effectively imposed noise, adding **jitter** to all diagonal elements (unnecessarily) **dilutes** the **informativeness** of our data.



The **Cholesky** factorisation of a positive semi-definite matrix K is relatively fast (1 / 3 $O(n^3)$ in matrix size n) and more numerically stable.

$$K = R^T R$$

$$R = \text{chol}(K) = \begin{pmatrix} R_{11} & R_{12} & \cdots & R_{1n} \\ 0 & R_{22} & \cdots & R_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & R_{nn} \end{pmatrix}$$

The upper triangular Cholesky factor can then be stored and used to solve $v = Kx$ for x very quickly ($O(n^2)$ in matrix size n) by **back substitution**.

$$v = Kx$$

$$v = R^T x'$$

$$x' = Rx$$

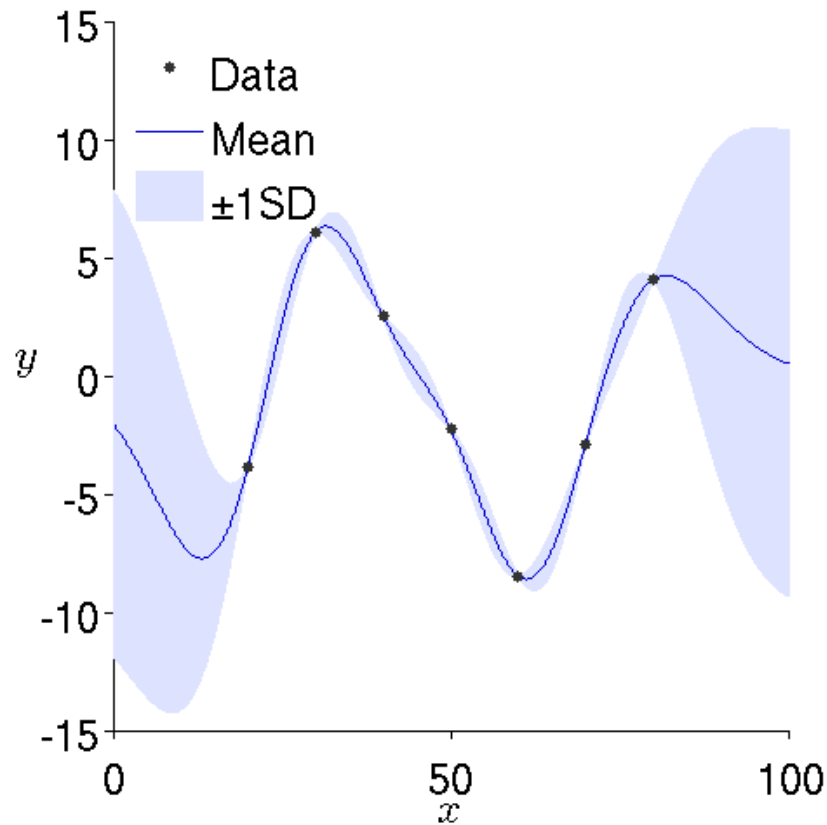
$$\begin{pmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{pmatrix} = \begin{pmatrix} R_{11} & R_{12} & \cdots & R_{1n} \\ 0 & R_{22} & \cdots & R_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & R_{nn} \end{pmatrix} \begin{pmatrix} x'_1 \\ x'_2 \\ \vdots \\ x'_n \end{pmatrix}$$

A symmetric matrix K is **Toeplitz** if it can be written as

$$K = \begin{pmatrix} k_1 & k_2 & k_3 & k_4 & \cdots & k_n \\ k_2 & k_1 & k_2 & k_3 & & \\ k_3 & k_2 & k_1 & k_2 & & \\ k_4 & k_3 & k_2 & k_1 & & \\ \vdots & & & & \ddots & \\ k_n & & & & & k_1 \end{pmatrix}$$

If K is Toeplitz, there exists a very efficient method to solve $v = Kx$ for x ($O(4n^2)$ in matrix size n).

A Gaussian process has a Toeplitz covariance matrix if we have **linearly spaced observations** and a **stationary covariance function**.



$$K = \begin{pmatrix} 100 & 60.7 & 13.5 & 1.11 & 0.03 & 0 & 0 \\ 60.7 & 100 & 60.7 & 13.5 & 1.11 & 0.03 & 0 \\ 13.5 & 60.7 & 100 & 60.7 & 13.5 & 1.11 & 0.03 \\ 1.11 & 13.5 & 60.7 & 100 & 60.7 & 13.5 & 1.11 \\ 0.03 & 1.11 & 13.5 & 60.7 & 100 & 60.7 & 13.5 \\ 0 & 0.03 & 1.11 & 13.5 & 60.7 & 100 & 60.7 \\ 0 & 0 & 0.03 & 1.11 & 13.5 & 60.7 & 100 \end{pmatrix}$$



Some special large matrices can be represented in a compact way using the **Kronecker product**.

$$\mathbf{A} \otimes \mathbf{B} = \begin{bmatrix} a_{11}B & \cdots & a_{1n}B \\ \vdots & \ddots & \vdots \\ a_{m1}B & \cdots & a_{mn}B \end{bmatrix}.$$

$$\mathbf{A} \otimes \mathbf{B} = \begin{bmatrix} a_{11}b_{11} & a_{11}b_{12} & \cdots & a_{11}b_{1q} & \cdots & \cdots & a_{1n}b_{11} & a_{1n}b_{12} & \cdots & a_{1n}b_{1q} \\ a_{11}b_{21} & a_{11}b_{22} & \cdots & a_{11}b_{2q} & \cdots & \cdots & a_{1n}b_{21} & a_{1n}b_{22} & \cdots & a_{1n}b_{2q} \\ \vdots & \vdots & \ddots & \vdots & & & \vdots & \vdots & \ddots & \vdots \\ a_{11}b_{p1} & a_{11}b_{p2} & \cdots & a_{11}b_{pq} & \cdots & \cdots & a_{1n}b_{p1} & a_{1n}b_{p2} & \cdots & a_{1n}b_{pq} \\ \vdots & \vdots & & \vdots & \ddots & & \vdots & \vdots & & \vdots \\ \vdots & \vdots & & \vdots & & \ddots & \vdots & \vdots & & \vdots \\ a_{m1}b_{11} & a_{m1}b_{12} & \cdots & a_{m1}b_{1q} & \cdots & \cdots & a_{mn}b_{11} & a_{mn}b_{12} & \cdots & a_{mn}b_{1q} \\ a_{m1}b_{21} & a_{m1}b_{22} & \cdots & a_{m1}b_{2q} & \cdots & \cdots & a_{mn}b_{21} & a_{mn}b_{22} & \cdots & a_{mn}b_{2q} \\ \vdots & \vdots & \ddots & \vdots & & & \vdots & \vdots & \ddots & \vdots \\ a_{m1}b_{p1} & a_{m1}b_{p2} & \cdots & a_{m1}b_{pq} & \cdots & \cdots & a_{mn}b_{p1} & a_{mn}b_{p2} & \cdots & a_{mn}b_{pq} \end{bmatrix}.$$

e.g. $\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \otimes \begin{bmatrix} 0 & 5 \\ 6 & 7 \end{bmatrix} = \begin{bmatrix} 1 \cdot 0 & 1 \cdot 5 & 2 \cdot 0 & 2 \cdot 5 \\ 1 \cdot 6 & 1 \cdot 7 & 2 \cdot 6 & 2 \cdot 7 \\ 3 \cdot 0 & 3 \cdot 5 & 4 \cdot 0 & 4 \cdot 5 \\ 3 \cdot 6 & 3 \cdot 7 & 4 \cdot 6 & 4 \cdot 7 \end{bmatrix} = \begin{bmatrix} 0 & 5 & 0 & 10 \\ 6 & 7 & 12 & 14 \\ 0 & 15 & 0 & 20 \\ 18 & 21 & 24 & 28 \end{bmatrix}$

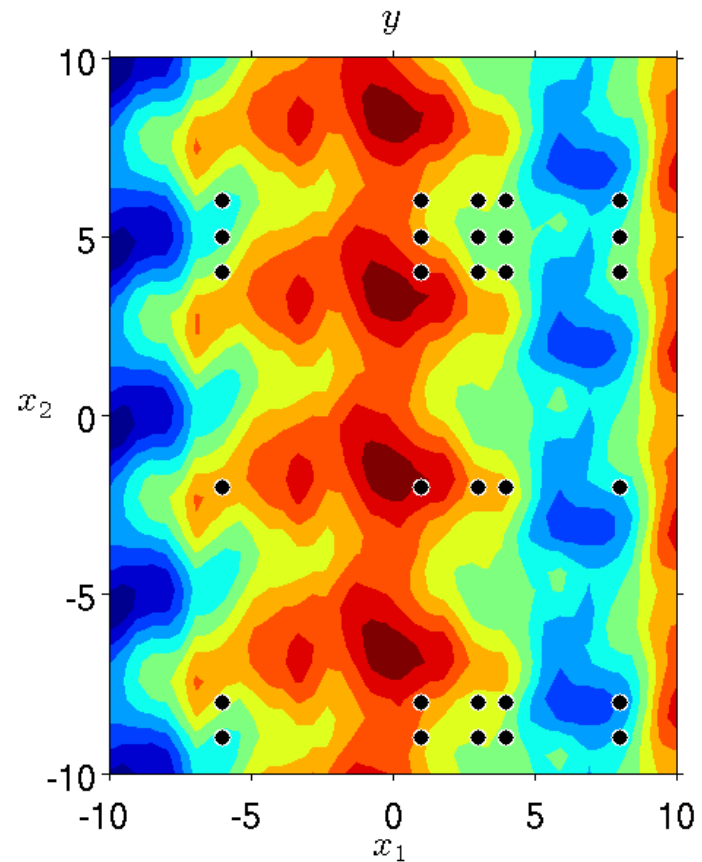
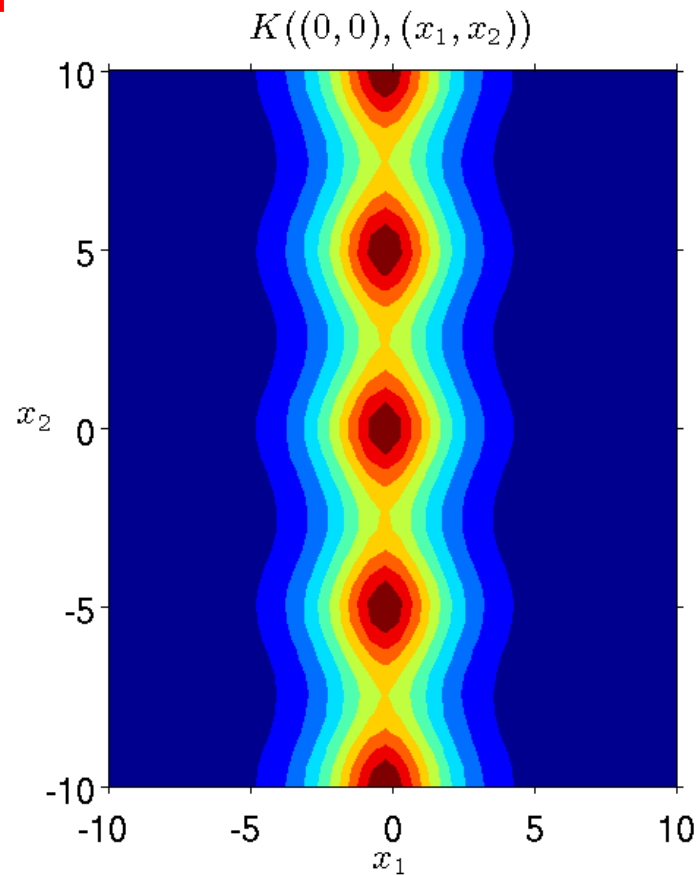


If K is a **Kronecker** product, there exists a very **efficient** method to solve $v = K x$ for x (particularly when v is itself a Kronecker product):

$$\begin{aligned}
 & \text{size } n_a \times n_b \\
 x &= (K_a \otimes K_b)^{-1} (v_a \otimes v_b) \\
 &= \underbrace{(K_a^{-1} v_a)}_{\text{size } n_a} \otimes \underbrace{(K_b^{-1} v_b)}_{\text{size } n_b}
 \end{aligned}$$

Recall that solving operations are typically $O(n^3)$!

A Gaussian process will have a Kronecker product for a covariance matrix if we use a **product covariance function** and a **grid of samples**.

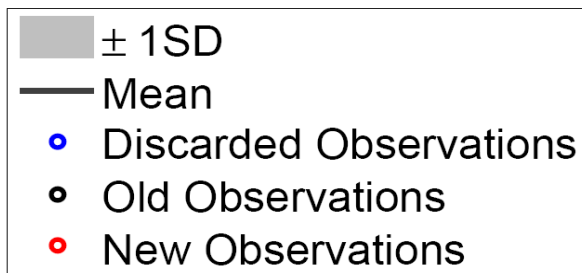
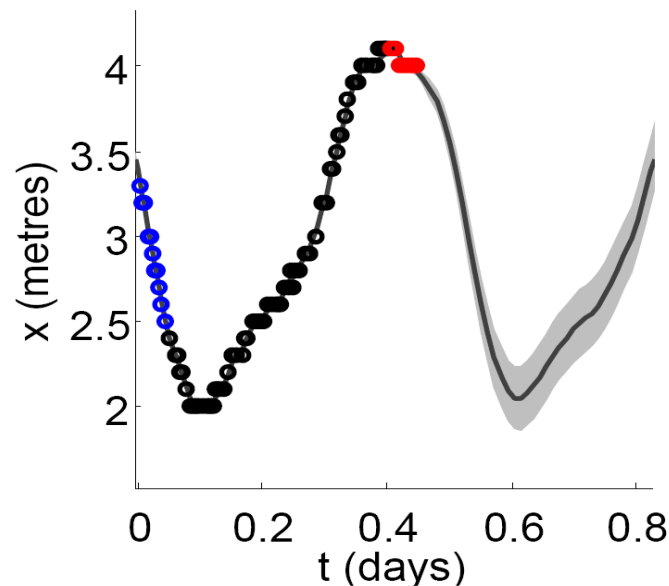


If a very large covariance matrix is not decomposable as a Kronecker product (or otherwise), we may wish to attempt **sparsification**.

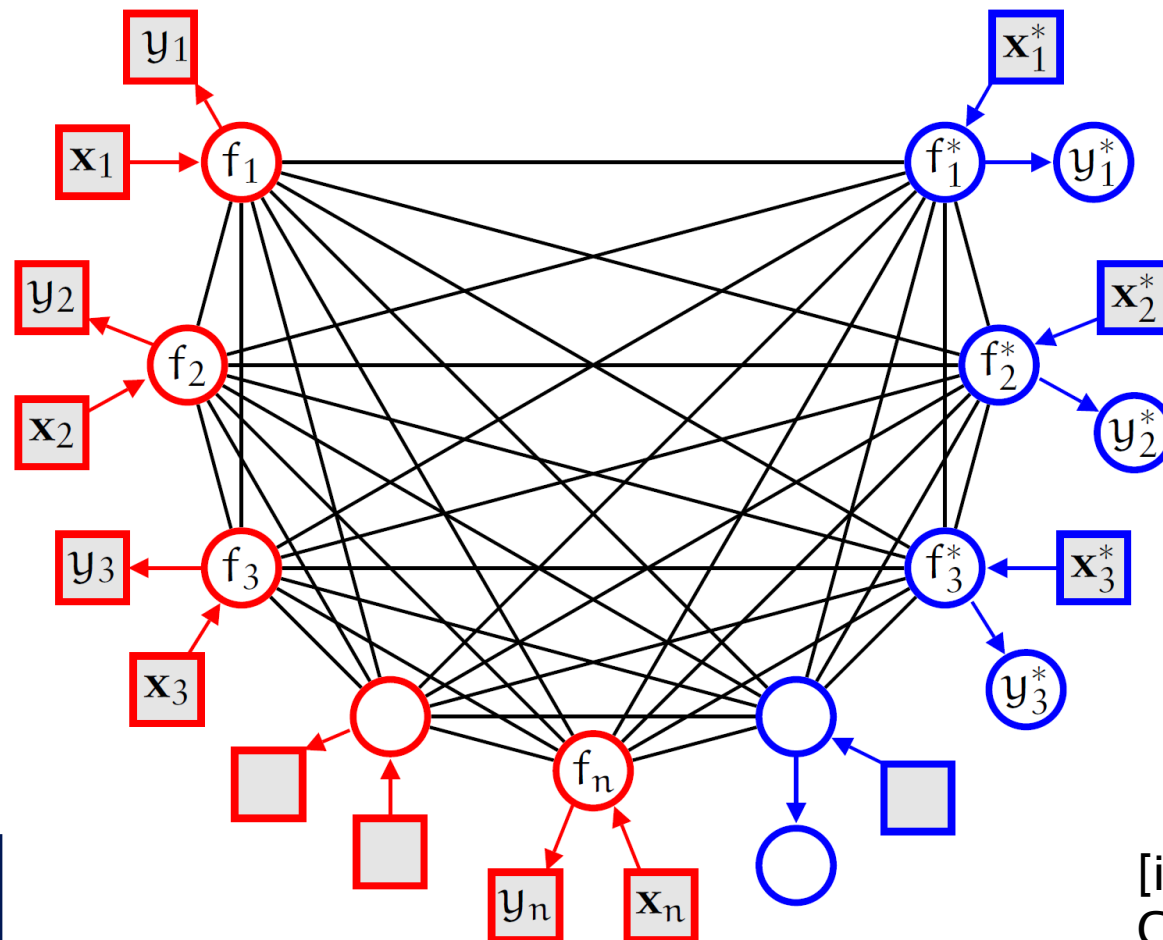
$$K = \begin{pmatrix} K_{11} & K_{12} & 0 & 0 & \dots & 0 \\ K_{21} & K_{22} & K_{23} & 0 & & \\ 0 & K_{32} & K_{33} & K_{34} & & \\ 0 & 0 & K_{43} & K_{44} & & \\ \vdots & & & & \ddots & \\ 0 & & & & & K_{nn} \end{pmatrix}$$

There are many ways to sparsify our data; the simplest involve selecting a subset. **Windowing** represents a reasonable way to do this.

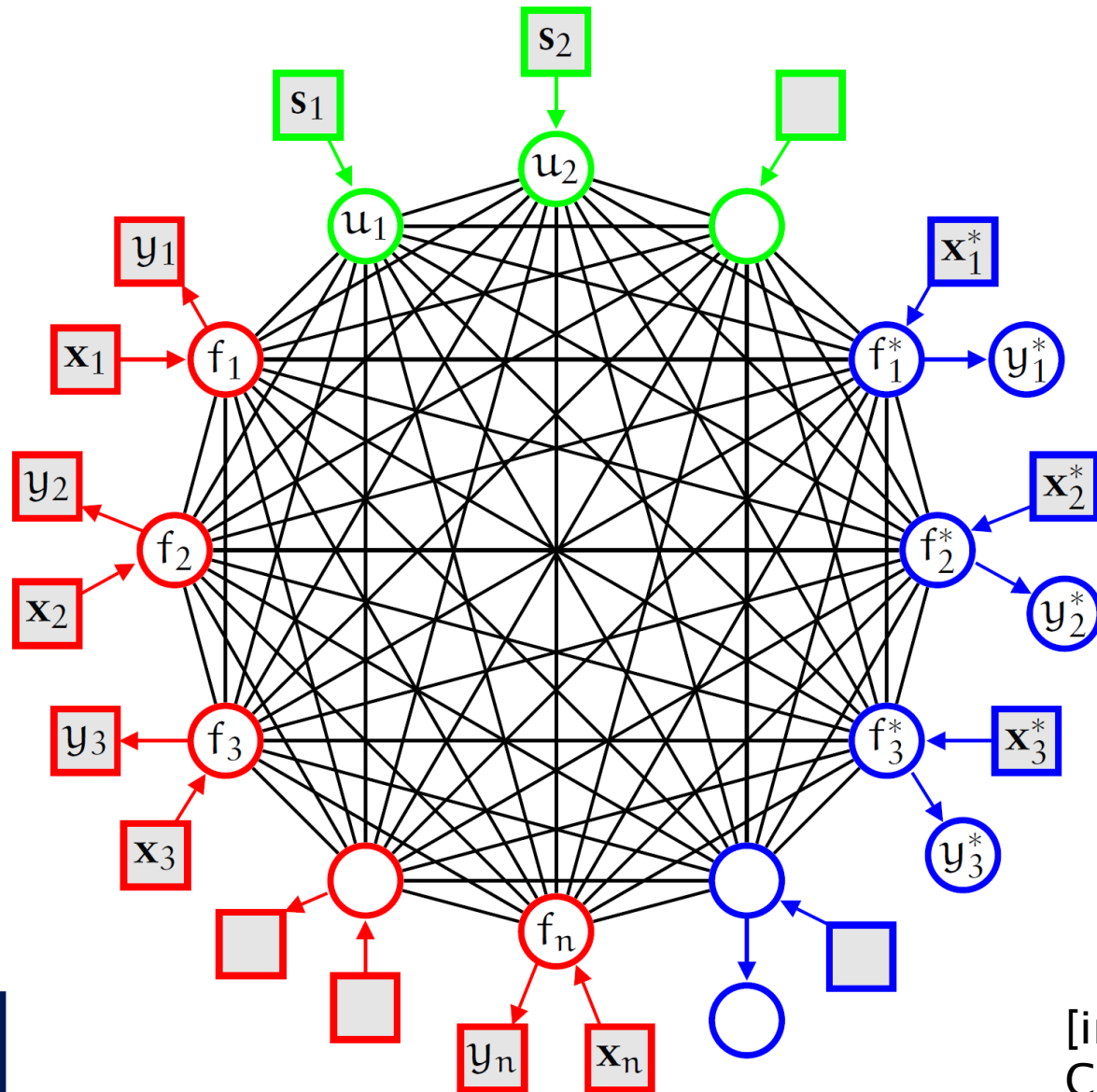
Tide Heights, Independent Sensor 1



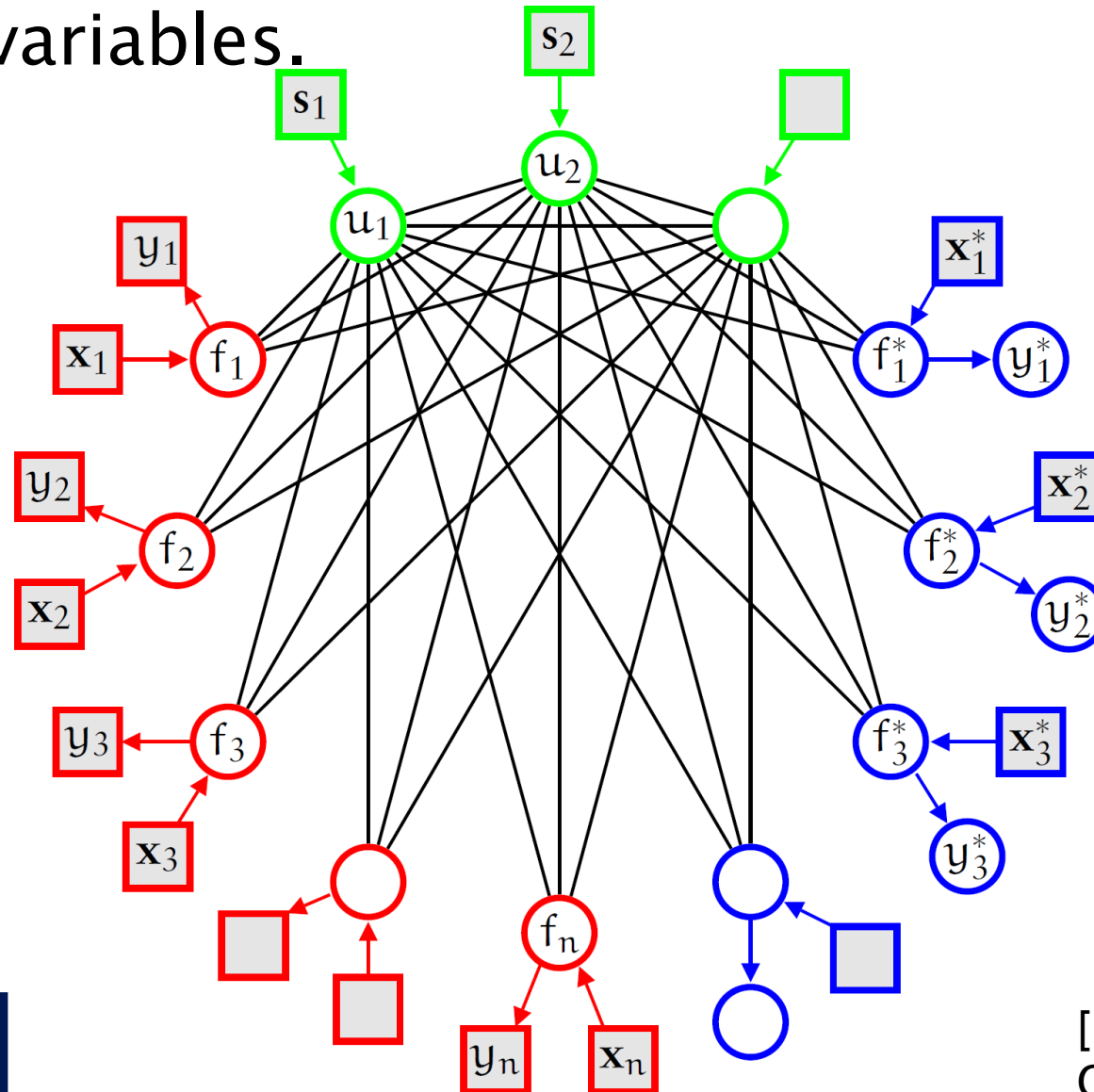
A Gaussian process assumes **all variables f are correlated.**



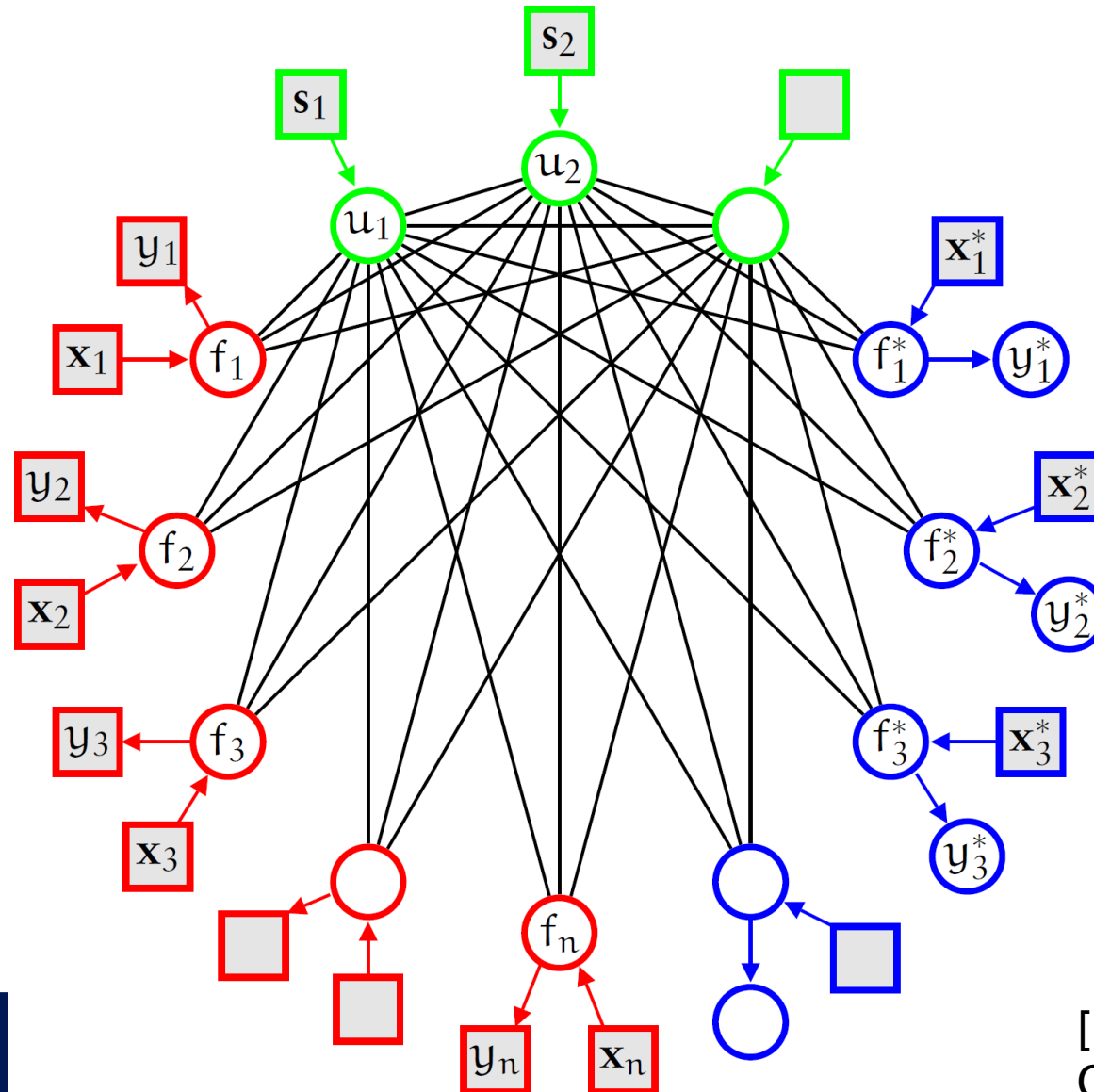
Imagine introducing additional, unobserved
inducing variables u .



We can **sparsify** data by using inducing variables to mediate the interactions between test and training variables.



There are **many such schemes for sparsification**, that differ in the choice of inducing inputs.



Finally, if we already have the Cholesky factor

$$R_{11} = \text{chol}(K_{11}),$$

we can efficiently determine the **updated factor**

$$\begin{pmatrix} R_{11} & R_{12} \\ 0 & R_{22} \end{pmatrix} = \text{chol} \left(\begin{pmatrix} K_{11} & K_{12} \\ K_{12} & K_{22} \end{pmatrix} \right),$$

and similar for other types of Cholesky updates and downdates, and for solutions based upon them. A Toeplitz update is probably also possible.



We want to evaluate a large number of hyperparameter samples to explore hyperparameter space. Fortunately, each sample can be evaluated in **parallel** (possibly on a graphics card).



I hope you have learned how to fit **Gaussian processes** to data.

