Gaussian Processes

Theory, Background, & Recent Advances

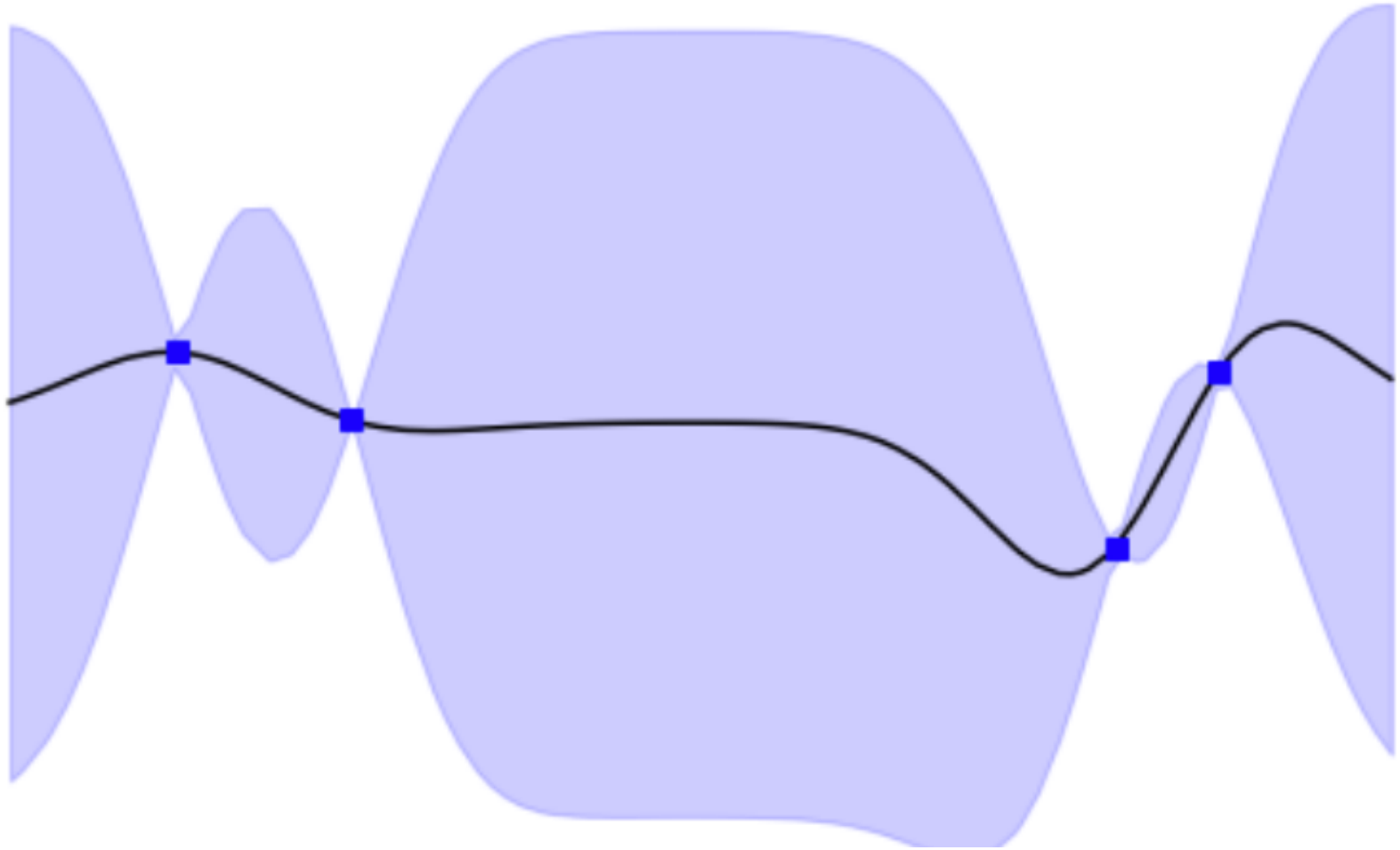
Joel Paulson
The H.C. “Slip” Slider Assistant Professor,
Department of Chemical and Biomolecular Engineering,
The Ohio State University

Sargent Centre Summer School on Bayesian Optimization, 2024

For copies of slides & code, see
https://github.com/joelpaulson/Sargent_Centre_BO_Summer_School_2024

Why care about Gaussian Processes?

Probabilistic models key for active learning (e.g., Bayesian optimization)



Why Gaussian processes and not something else?

- Many other options available for training probabilistic surrogate models
  - Bayesian Linear Regression (or Neural Networks)
  - Random Forests
  - Tree-Structured Parzen Estimators

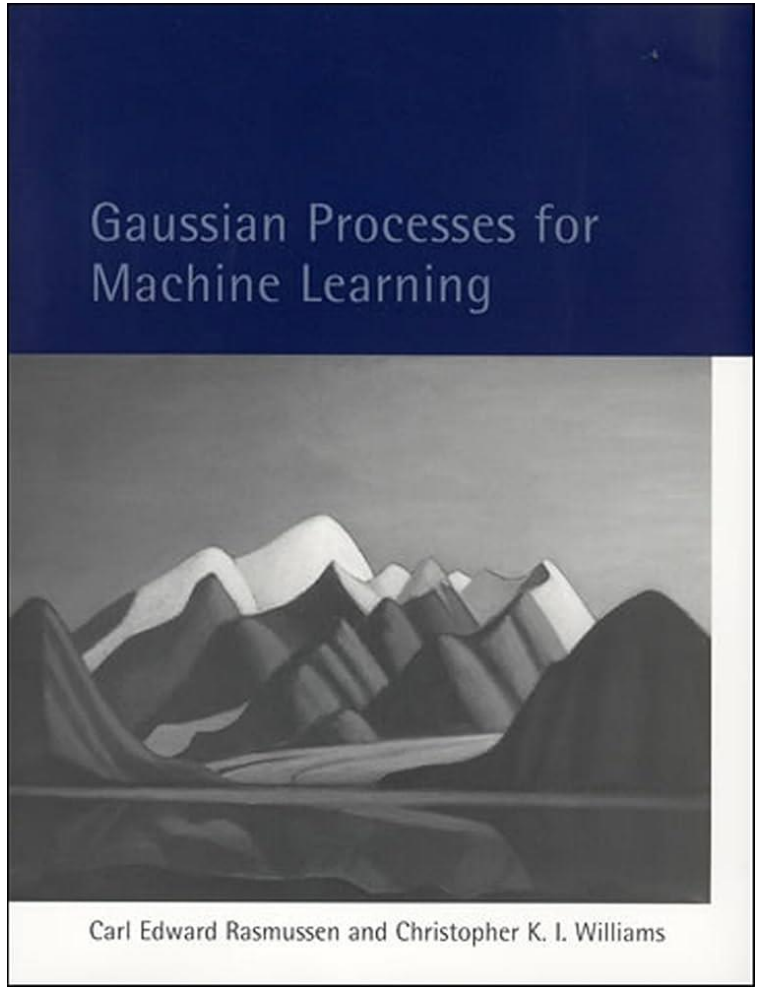
- The model must be probabilistic in order to systematically represent uncertainty (roughly think of this as providing confidence bounds)

- Bayesian optimization (mostly) focuses on Gaussian processes (GPs)
  - By end of talk, I hope to "convince you" that this is not an accident; GPs are **flexible** (universal approximators) but **simple** (not too many free parameters)
  - These models have *strong inductive biases* that concentrate prior support around simple solutions, providing good generalization even on small datasets

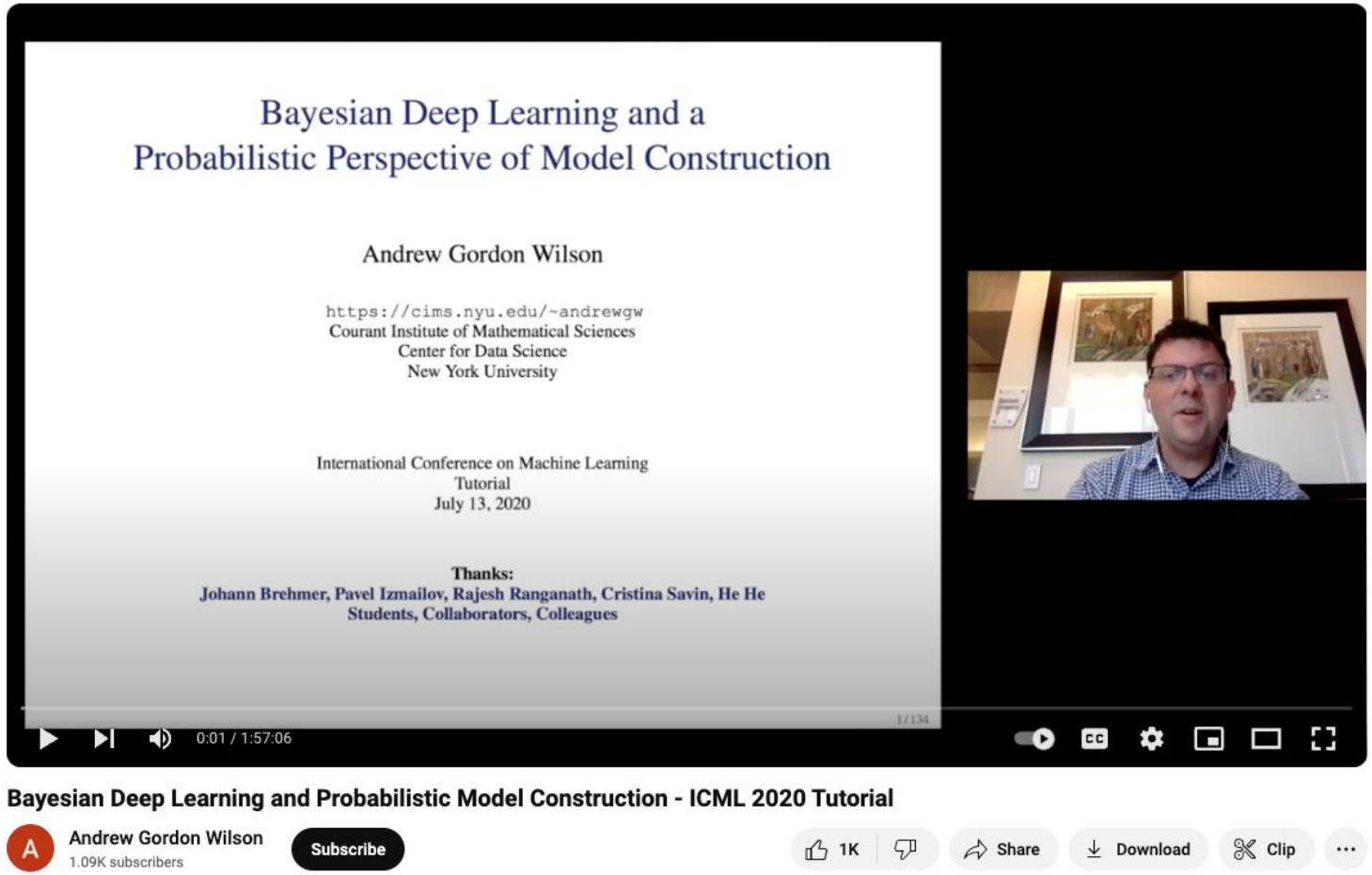
Outline

- Introduction to Bayesian modeling
  - Bayes rule, importance of model averaging, epistemic uncertainty
- The function-space view
  - Gaussian processes, mean and kernel functions, inference
- Deep dive into the kernel
  - Importance of kernel, stationary vs. non-stationary kernels, learning hyperparameters
- Beyond traditional Gaussian processes
  - Heteroskedastic noise, sparse approximations, SAAS, deep kernel learning
- Efficient software packages
  - Quick overview of options, GPyTorch for flexibility and scalability

Before diving in…I highly recommend



https://gaussianprocess.org/gpml/



https://www.youtube.com/watch?v=E1qhGw8QxqY

A quick caveat

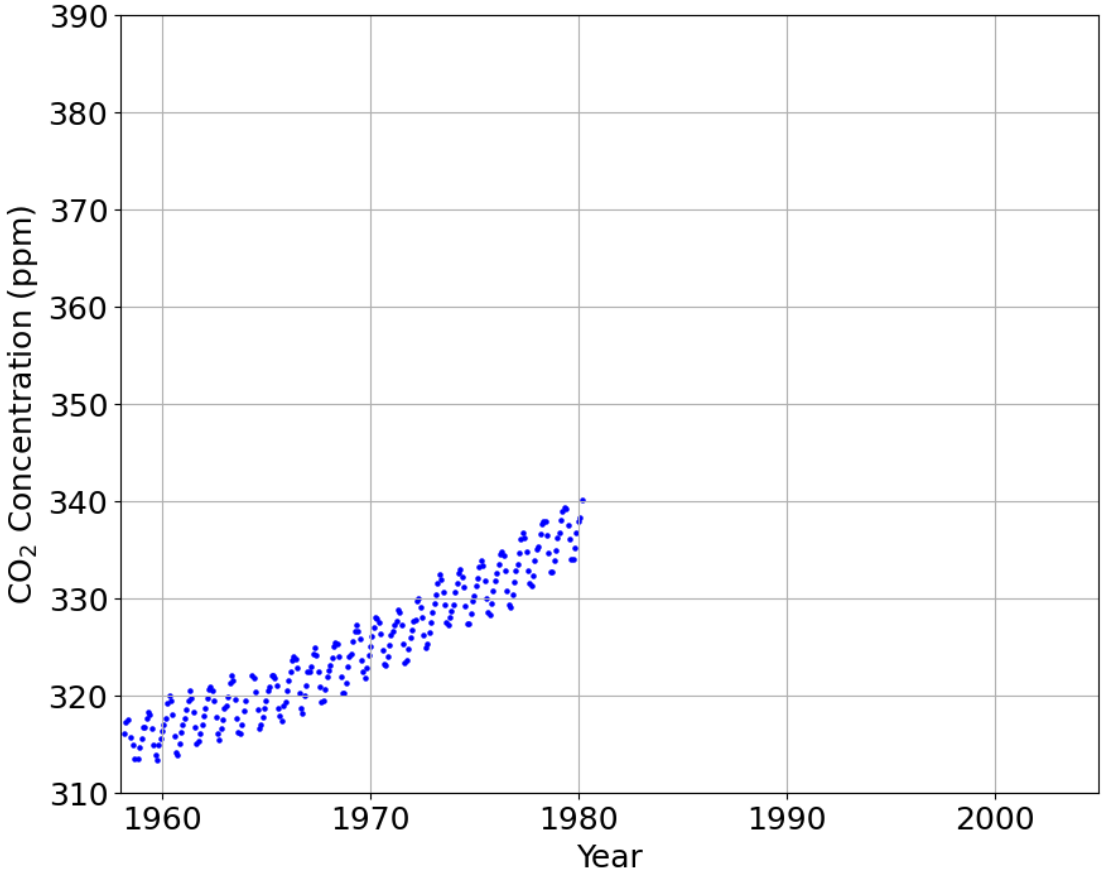
- There has been a lot of fantastic work on Gaussian processes over the past few decades, which I have drawn on to create these slides
  - Several points that I will make have been made by others
  - It is not meant to be a review of all work or all possible interpretations
  - I have done my best to provide references when applicable

- There may be some interesting work / ideas that I have not covered or may have missed referencing
  - If you notice anything you feel should be updated, please send me an email (paulson.82@osu.edu) and I will do my best to fix it in future versions

Outline

- Introduction to Bayesian modeling
  - Bayes rule, importance of model averaging, epistemic uncertainty
- The function-space view
  - Gaussian processes, mean and kernel functions, inference
- Deep dive into the kernel
  - Importance of kernel, stationary vs. non-stationary kernels, learning hyperparameters
- Beyond traditional Gaussian processes
  - Heteroskedastic noise, sparse approximations, SAAS, deep kernel learning
- Efficient software packages
  - Quick overview of options, GPyTorch for flexibility and scalability

Statistical Regression from Scratch



- Basic regression problem
  - Training set of n observations $y = (y_1, \ldots, y_n)^\mathsf{T}$
  - Observations at inputs $X = (x_1, \ldots, x_n)^\mathsf{T}$
  - Want to predict value $y(x_*)$ at a new (test) input x_*

- Given CO_2 concentration values $\boldsymbol{y}$ measured at times X, what will be the concentration when $x_* = 2004$?

How would you solve this type of problem intuitively?

Statistical Regression from Scratch

1. Guess parametric form of a function that could fit observed data
   - $f(x, \boldsymbol{w}) = \boldsymbol{w}^\top x$ [Linear function of $\boldsymbol{w}$ and x]
   - $f(x, \boldsymbol{w}) = \boldsymbol{w}^\top \boldsymbol{\phi}(x)$ [Linear function of $\boldsymbol{w}$ & $\boldsymbol{\phi}(x)$ = nonlinear basis functions]
   - $f(x, \boldsymbol{w}) = g(\boldsymbol{w}^\top \boldsymbol{\phi}(x))$ [Nonlinear in $\boldsymbol{w}$ and x] (e.g., neural network)

2. Choose an error measure $E(\boldsymbol{w})$ & minimize it with respect to $\boldsymbol{w}$
   - Common choice is sum of squared errors

$$E(\boldsymbol{w}) = \sum_{i=1}^{n} \big(y_i - f(x_i, \boldsymbol{w})\big)^2$$

Statistical Regression from Scratch

[Intuitive strategy can be derived from principled probabilistic approach]

- Let's explicitly account for noise in our model
  - $y(x) = f(x, \boldsymbol{w}) + \epsilon(x)$ where $\epsilon(x)$ is a noise function

- Common to assume $\epsilon(x) \sim \mathcal{N}(0, \sigma^2)$ for i.i.d. Gaussian noise, leads to:
  - $p(y(x)|x, \boldsymbol{w}, \sigma^2) = \mathcal{N}(y(x); f(x, \boldsymbol{w}), \sigma^2)$ → observation model
  - $p(\boldsymbol{y}|x, \boldsymbol{w}, \sigma^2) = \prod_{i=1}^{n} \mathcal{N}(y(x_i); f(x_i, \boldsymbol{w}), \sigma^2)$ → likelihood

- Maximize the likelihood of the data $p(\boldsymbol{y}|x, \boldsymbol{w}, \sigma^2)$ with respect to $\boldsymbol{w}, \sigma^2$
  - For Gaussian noise, leads to same solution as the squared error function
    $\log p(\boldsymbol{y}|x, \boldsymbol{w}, \sigma^2) \propto -\frac{1}{2\sigma^2} \sum_{i=1}^{n} \big(y_i - f(x_i, \boldsymbol{w})\big)^2$

Statistical Regression from Scratch

- Probabilistic approach helps interpret error measure $E(\cdot)$ and gives us some sense of noise level σ^2; however, these approaches are prone to *over-fitting* for flexible models $f(x, \boldsymbol{w})$
  - Low error on the training data, but high error on the testing data

- Regularization to reduce over-fitting → penalized log likelihood such as

$$E(\boldsymbol{w}) = \underbrace{\frac{1}{2\sigma^2}\sum_{i=1}^{n}\big(y_i - f(x_i, \boldsymbol{w})\big)^2}_{\text{Model fit}} + \underbrace{\lambda \boldsymbol{w}^\top \boldsymbol{w}}_{\text{Penalty on complexity}}$$

- Questions:
  - How to define and penalize complexity?
  - How to set penalty parameter λ?

Statistical Regression from Scratch

- You may have seen the following interpretation: minimizing the $E(\boldsymbol{w})$ on the previous slide is the same as maximizing the posterior $p(\boldsymbol{w}|\boldsymbol{y}, X)$ when the weights have a Gaussian "prior" $p(\boldsymbol{w})$

- This is technically true because $\log p(\boldsymbol{w}) \propto -\boldsymbol{w}^\top \boldsymbol{w}$ for Gaussian distribution

$$\log p(\boldsymbol{w}|\boldsymbol{y}, X) = \log p(\boldsymbol{y}|\boldsymbol{w}, X) + \log p(\boldsymbol{w}) + c$$

- <u>But important to recognize this is not really Bayesian!</u>
  - Still results in a point estimate and so does not fully capture uncertainty

What is Bayesian Inference?

- Bayes' Rule

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}, \qquad P(A|B) \propto P(B|A)P(A)$$

$$\text{posterior} = \frac{\text{likelihood} \times \text{prior}}{\text{marginal likelihood}}, \qquad p(\boldsymbol{w}|\boldsymbol{y}, X) = \frac{p(\boldsymbol{y}|\boldsymbol{w}, X)p(\boldsymbol{w})}{p(\boldsymbol{y}|X)}$$

- Sum Rule (Marginalization)

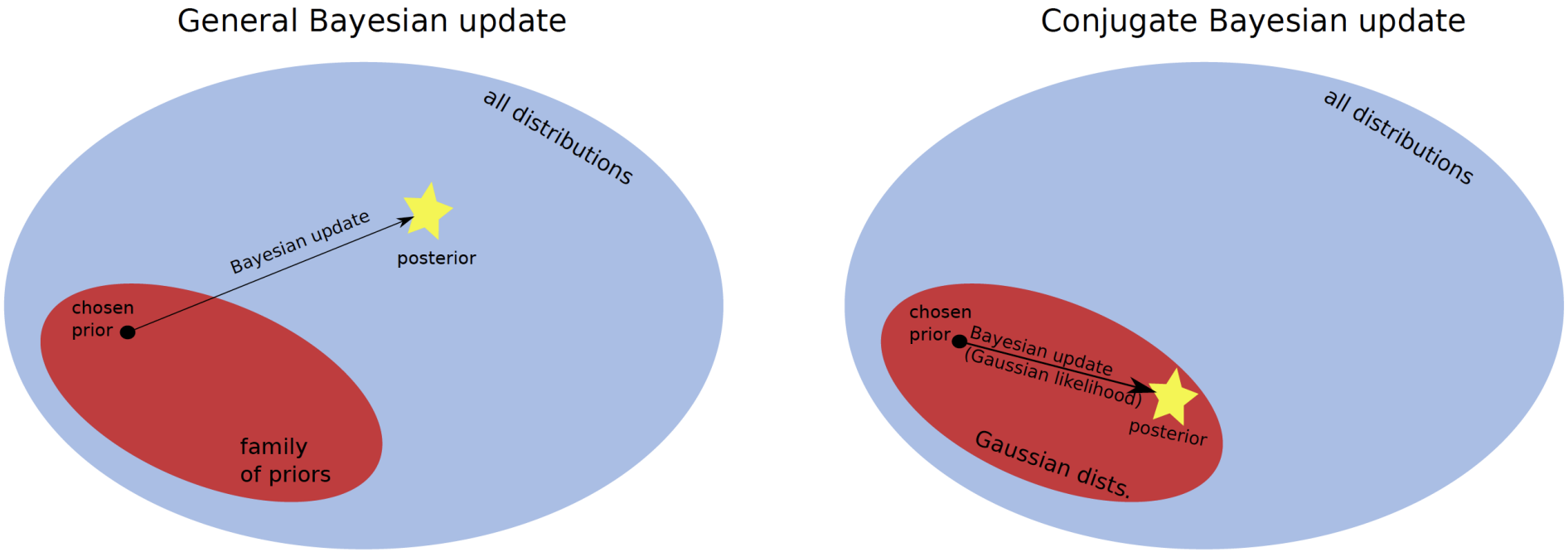
$$p(x) = \sum_y p(x, y) \quad \text{or} \quad \int_y p(x, y)dy$$

- Product Rule (Conditional Probability)

$$p(x, y) = p(x|y)p(y) = p(y|x)p(x)$$

Quick Sidenote: Conjugacy

If posterior distribution $p(\theta|\text{Data})$ is in same family as prior distribution $p(\theta)$, then they are called *conjugate distributions* (prior is conjugate prior for likelihood $p(\text{Data}|\theta)$)



$$p(\theta|\text{Data}) = \frac{p(\text{Data}|\theta)p(\theta)}{p(\text{Data})}$$

Quick Sidenote: Conjugacy

Likelihood	Conjugate Prior Density	Posterior Density
Binomial	Beta	Beta
Negative binomial	Beta	Beta
Poisson	Gamma	Gamma
Normal, with unknown mean	Normal	Normal
Normal, with unknown variance	Inverse gamma	Inverse gamma
Normal, with unknown mean and variance	Normal-inverse gamma	Normal-inverse gamma

Bayesian Posterior Predictive Distribution

- Combine sum rule $p(x) = \int_y p(x, y)dy$ and product rule $p(x, y) = p(y|x)p(x)$ to get:

$$p(y|x_*, \boldsymbol{y}, X) = \int_{\boldsymbol{w}} p(y|x_*, \boldsymbol{w})p(\boldsymbol{w}|\boldsymbol{y}, X)d\boldsymbol{w}$$

  - Think of each $\boldsymbol{w}$ as different model, equation above is a *Bayesian model average*, average of infinitely many models weighted by posterior probabilities
  - Represents epistemic uncertainty over which function $f(x, \boldsymbol{w})$ fits the data
  - Automatically calibrated complexity even with highly flexible models
    - Assuming the prior and model class are properly specified
  - Classical training can be viewed as approximate posterior $\delta(\boldsymbol{w} = \boldsymbol{w}_{MAP})$
    - MAP refers to maximum a posteriori estimate and is the mode of the posterior

Example: Flipping a Potentially Biased Coin

- Suppose we flip a coin with probability λ of landing on tails

1. What is the likelihood of a set of data $\mathcal{D} = \{y_1, \ldots, y_n\}$?
   - Likelihood of data: $p(y_1, \ldots, y_n|\lambda) = \prod_{i=1}^{n} \lambda^{y_i}(1-\lambda)^{1-y_i}$
   - Likelihood of m tails: $p(\mathcal{D}|m, \lambda) = \binom{n}{m}\lambda^m(1-\lambda)^{n-m}$

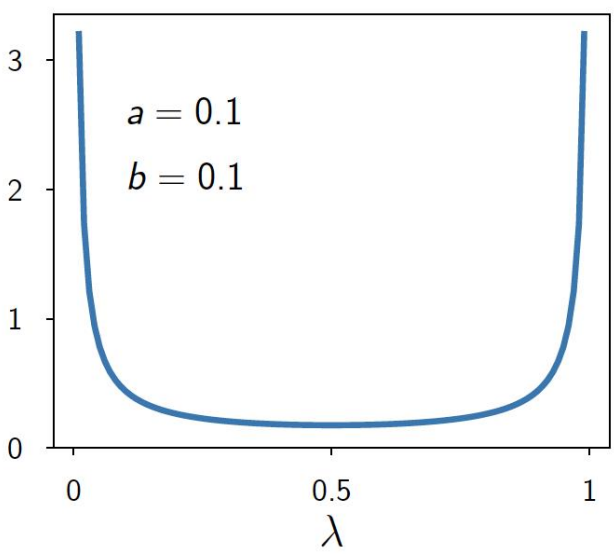
2. What is the maximum likelihood solution for unknown parameter λ?
   - Maximum likelihood: $\lambda_{\text{ML}} = \text{argmax}_\lambda p(\mathcal{D}|m, \lambda) = \frac{m}{n}$

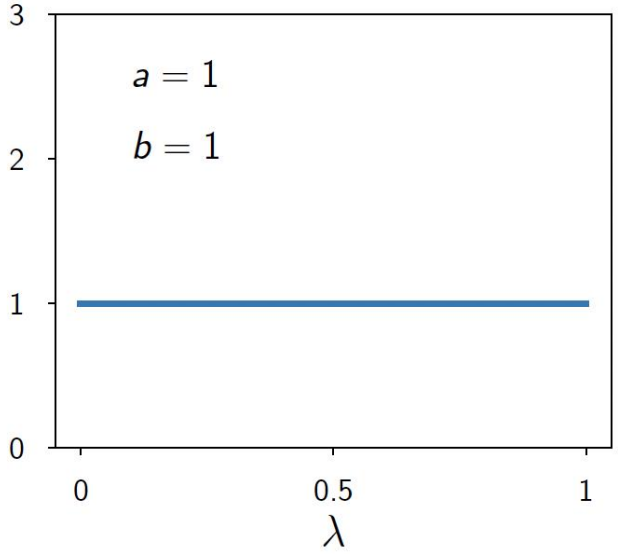
3. Suppose first flip is tails. What is probability that the next flip is tails under the maximum likelihood solution?
   - $\lambda_{\text{ML}} = 1$ → 100% prediction of tails, should we believe this solution?

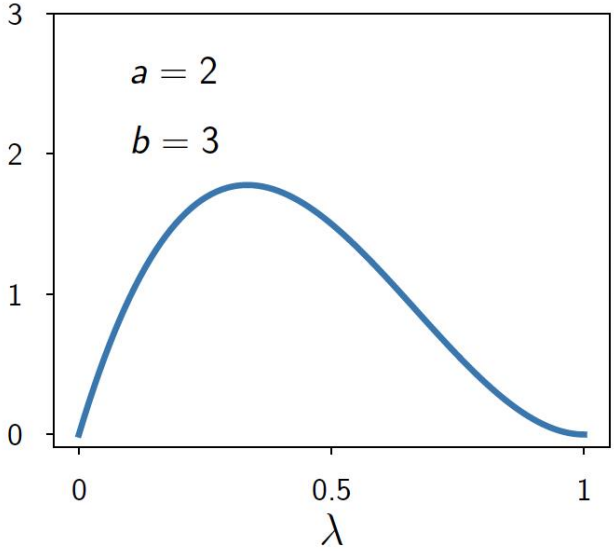
Example: Flipping a Potentially Biased Coin

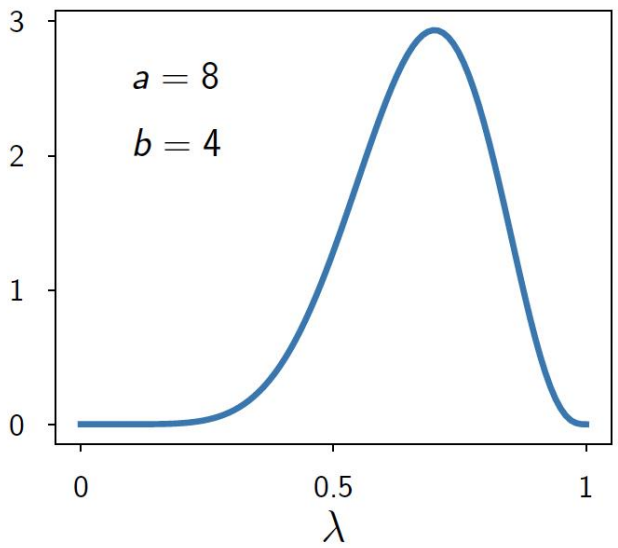
- What if we take a Bayesian perspective?
  - Specify a prior on λ, select beta distribution for conjugacy $p(\lambda) \propto \lambda^a(1-\lambda)^b$

Beta Distribution: Depends on Parameters a and b







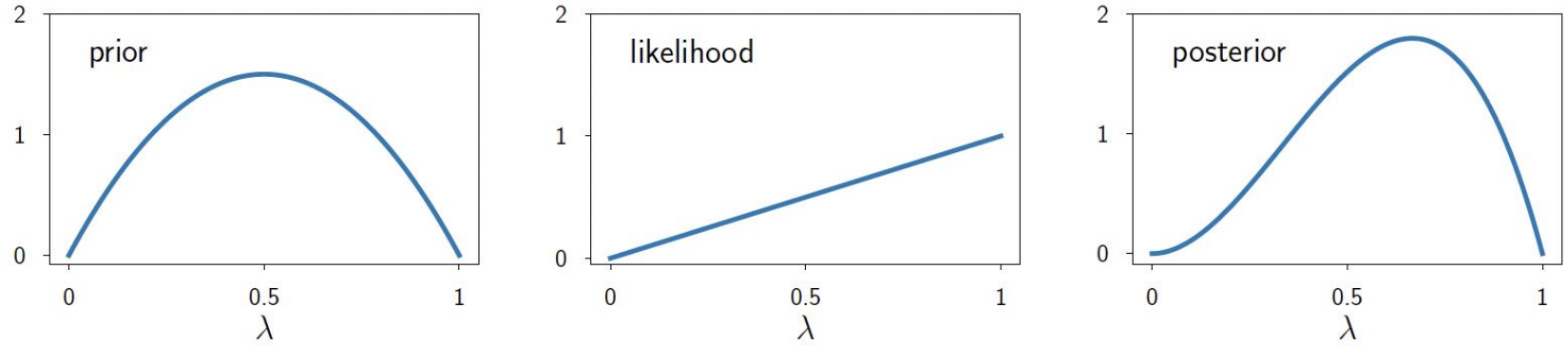


$$\mathbb{E}[\lambda] = \frac{a}{a+b}$$

$$\mathrm{Var}[\lambda] = \frac{ab}{(a+b)^2(a+b-1)}$$

Example: Flipping a Potentially Biased Coin

- What if we take a Bayesian perspective?
  - Specify a prior on λ, select beta distribution for conjugacy $p(\lambda) \propto \lambda^a(1-\lambda)^b$



- Applying Bayes' theorem, we can derive:

$$p(\lambda|\mathcal{D}) = \text{Beta}(\lambda; m + a, n - m + b)$$

Example: Flipping a Potentially Biased Coin

- Mean of the Bayesian posterior for the unknown parameter λ:

$$\mathbb{E}[\lambda|\mathcal{D}] = \frac{m + a}{n + a + b}$$

- We can view a and b as pseudo-observations!

- Questions:
  1. What is the probability that next flip is tails assuming first flip is tails?
  2. What happens if we make a and b large?
  3. What happens in the limit of infinite data $n \to \infty$?

Example: Flipping a Potentially Biased Coin

- Important observation!
  - **The maximum a posteriori (MAP) estimate under a uniform prior does NOT give the same result as Bayesian marginalization**

  - MAP estimate:

$$\lambda_{\text{MAP}} = \text{argmax}_\lambda \log p(\lambda|\mathcal{D}) = \text{argmax}_\lambda \log p(\mathcal{D}|\lambda) + \log p(\lambda)$$

[equal to λ_{ML} when $p(\lambda)$ is uniform since $\log p(\lambda)$ is constant]

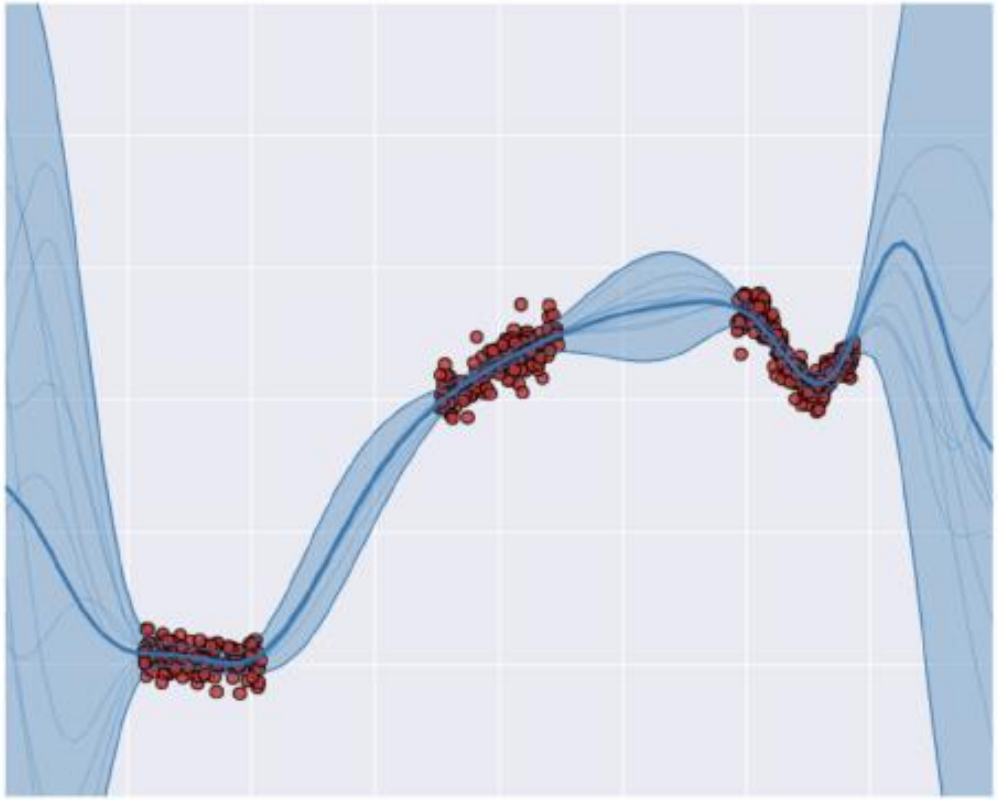
  - Bayesian marginalization:

$$p(\text{next flip tails}|\mathcal{D}) = \int p(\text{next flip tails}|\lambda, \mathcal{D}) p(\lambda|\mathcal{D}) = \int \lambda \, p(\lambda|\mathcal{D}) = \mathbb{E}[\lambda|\mathcal{D}]$$

Outline

- Introduction to Bayesian modeling
  - Bayes rule, importance of model averaging, epistemic uncertainty
- The function-space view
  - Gaussian processes, mean and kernel functions, inference
- Deep dive into the kernel
  - Importance of kernel, stationary vs. non-stationary kernels, learning hyperparameters
- Beyond traditional Gaussian processes
  - Heteroskedastic noise, sparse approximations, SAAS, deep kernel learning
- Efficient software packages
  - Quick overview of options, GPyTorch for flexibility and scalability

What is the function-space view?



The parameters $\boldsymbol{w}$ of a model are separated from the statistical properties of a model…

…yet we spend a lot of our effort trying to learn these parameters $\boldsymbol{w}$

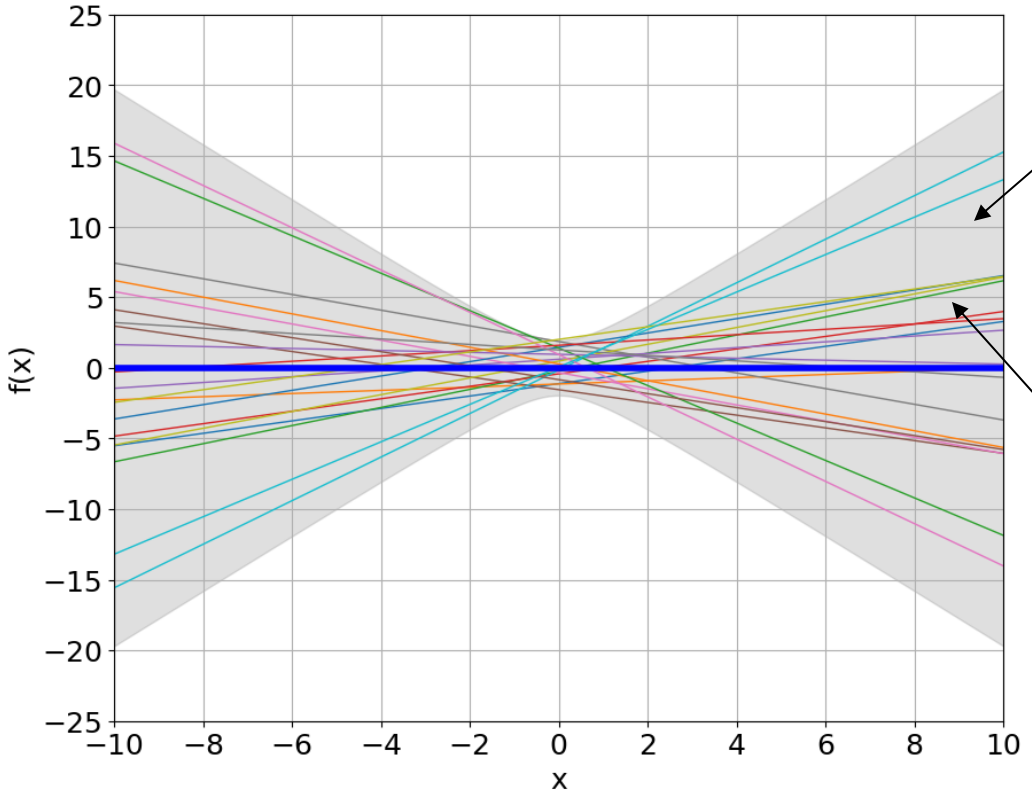
We really care about how those parameters combine with a functional form $f(x, \mathbf{w})$

Can we just perform inference directly in the space of functions? (This would be ideal)

Image from Andrew Wilson's 2020 ICML Tutorial on Bayesian Deep Learning

What is the function-space view?

- Key observation: A distribution over parameters $p(\boldsymbol{w})$ induces a distribution over functions $p(f(x))$
  - For example, a simple linear model $f(x, \boldsymbol{w}) = w_0 + w_1 x$ with $w_0, w_1 \sim \mathcal{N}(0,1)$ can be used to generate functions of the type shown in the figure below



95% confidence region

Light lines are different samples drawn from the model above

What is the function-space view?

- We can express the distribution over functions directly:

$$f(x, \boldsymbol{w}) = \boldsymbol{w}^\top \boldsymbol{\phi}(x)$$
$$p(\boldsymbol{w}) \sim \mathcal{N}(0, \Sigma_p)$$

where $\boldsymbol{\phi}(x)$ are basis functions

– For example, $\boldsymbol{\phi}(x) = (1, x, x^2, x^3)$ for $x \in \mathbb{R}$ → $f(x, \boldsymbol{w}) = w_0 + w_1 x + w_2 x^2 + w_3 x^3$

- What are the moments of the induced distribution over functions?

$$\mathbb{E}[f(x, \boldsymbol{w})] = ???$$

$$\mathrm{Cov}\big[f(x_i), f(x_j)\big] = \mathbb{E}\big[f(x_i) f(x_j)\big] - \mathbb{E}[f(x_i)]\mathbb{E}\big[f(x_j)\big] = ???$$

What is the function-space view?

- Moments of the induced distribution over functions:

$$\mathbb{E}[f(x, \boldsymbol{w})] = \mathbb{E}[\boldsymbol{w}^\top \boldsymbol{\phi}(x)] = \mathbb{E}[\boldsymbol{w}^\top]\boldsymbol{\phi}(x) = 0$$

This is a mean function, $\mu(x)$

$$\begin{aligned}\mathrm{Cov}\big[f(x_i), f(x_j)\big] &= \mathbb{E}\big[f(x_i)f(x_j)\big] - \mathbb{E}[f(x_i)]\mathbb{E}\big[f(x_j)\big] \\ &= \mathbb{E}\big[\boldsymbol{\phi}(x_i)^\top \boldsymbol{w}\boldsymbol{w}^\top \boldsymbol{\phi}(x_j)\big] - 0 \\ &= \boldsymbol{\phi}(x_i)^\top \mathbb{E}[\boldsymbol{w}\boldsymbol{w}^\top]\boldsymbol{\phi}(x_j) \\ &= \boldsymbol{\phi}(x_i)^\top \Sigma_w \boldsymbol{\phi}(x_j)\end{aligned}$$

This is a covariance function (or kernel), $k(x, x')$

What is the function-space view?

- Moments of the induced distribution over functions:

$$\mathbb{E}[f(x, \boldsymbol{w})] = \mathbb{E}[\boldsymbol{w}^\top \boldsymbol{\phi}(x)] = \mathbb{E}[\boldsymbol{w}^\top]\boldsymbol{\phi}(x) = 0$$

This is a mean function, $\mu(x)$

$$\begin{aligned}\mathrm{Cov}\big[f(x_i), f(x_j)\big] &= \mathbb{E}\big[f(x_i)f(x_j)\big] - \mathbb{E}[f(x_i)]\mathbb{E}\big[f(x_j)\big] \\ &= \mathbb{E}\big[\boldsymbol{\phi}(x_i)^\top \boldsymbol{w}\boldsymbol{w}^\top \boldsymbol{\phi}(x_j)\big] - 0 \\ &= \boldsymbol{\phi}(x_i)^\top \mathbb{E}[\boldsymbol{w}\boldsymbol{w}^\top]\boldsymbol{\phi}(x_j) \\ &= \boldsymbol{\phi}(x_i)^\top \Sigma_w \boldsymbol{\phi}(x_j)\end{aligned}$$

This is a covariance function (or kernel), $k(x, x')$

- **Using μ and k, we can do inference directly over $f(x)$ instead of w!**

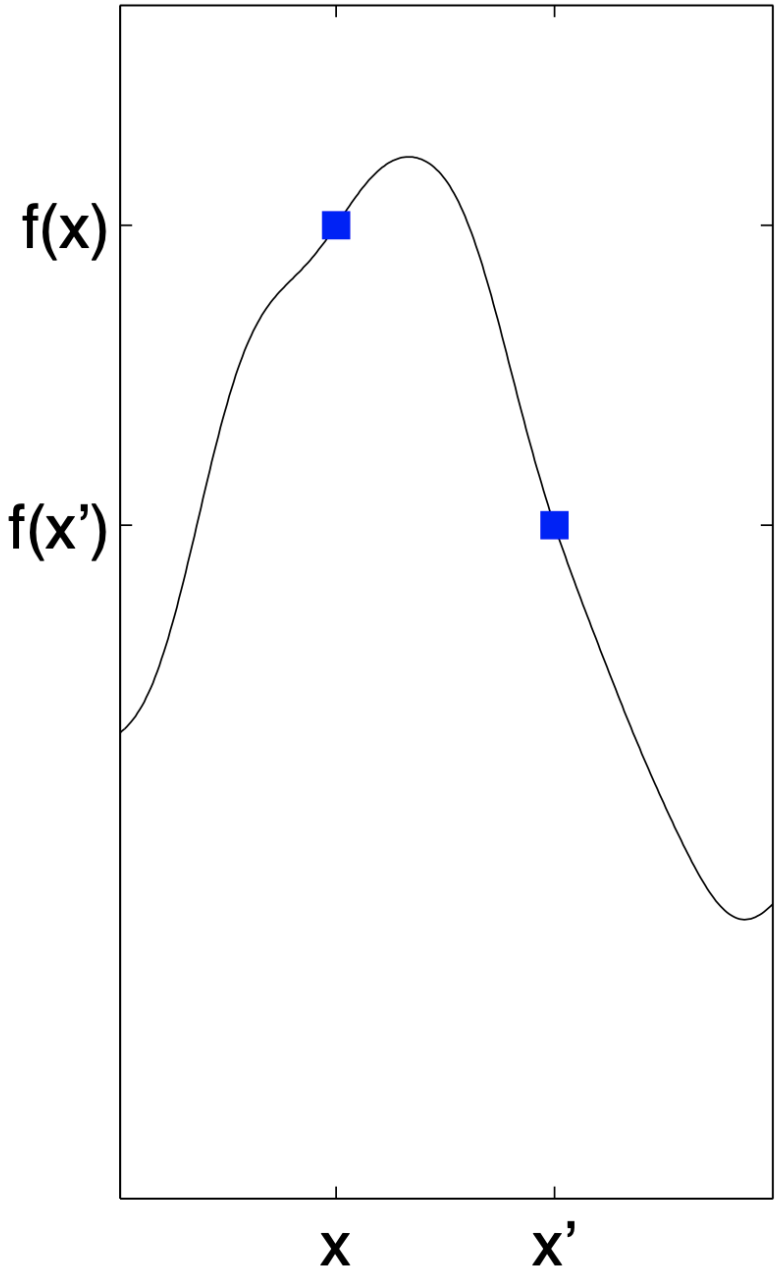
Gaussian Processes

- A Gaussian process (GP) is a collection of random variables, any finite number of which have a joint Gaussian distribution, i.e., $f(x) \sim \mathcal{GP}(\mu_0, k_0)$

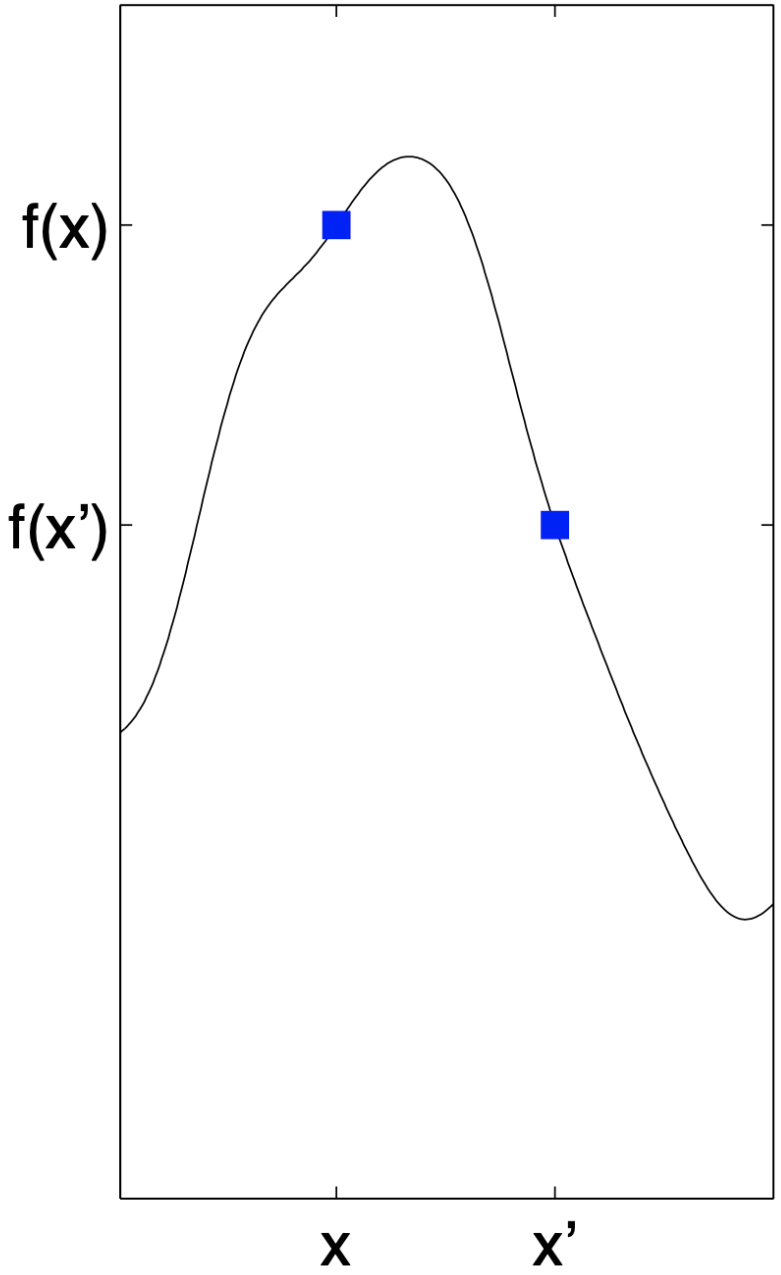
$$\begin{bmatrix} f(x_1) \\ \vdots \\ f(x_k) \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \mu_0(x_1) \\ \vdots \\ \mu_0(x_k), \end{bmatrix}, \begin{bmatrix} k_0(x_1, x_1) & \cdots & k_0(x_1, x_k) \\ \vdots & \ddots & \vdots \\ k_0(x_k, x_1) & \cdots & k_0(x_k, x_k) \end{bmatrix} \right)$$

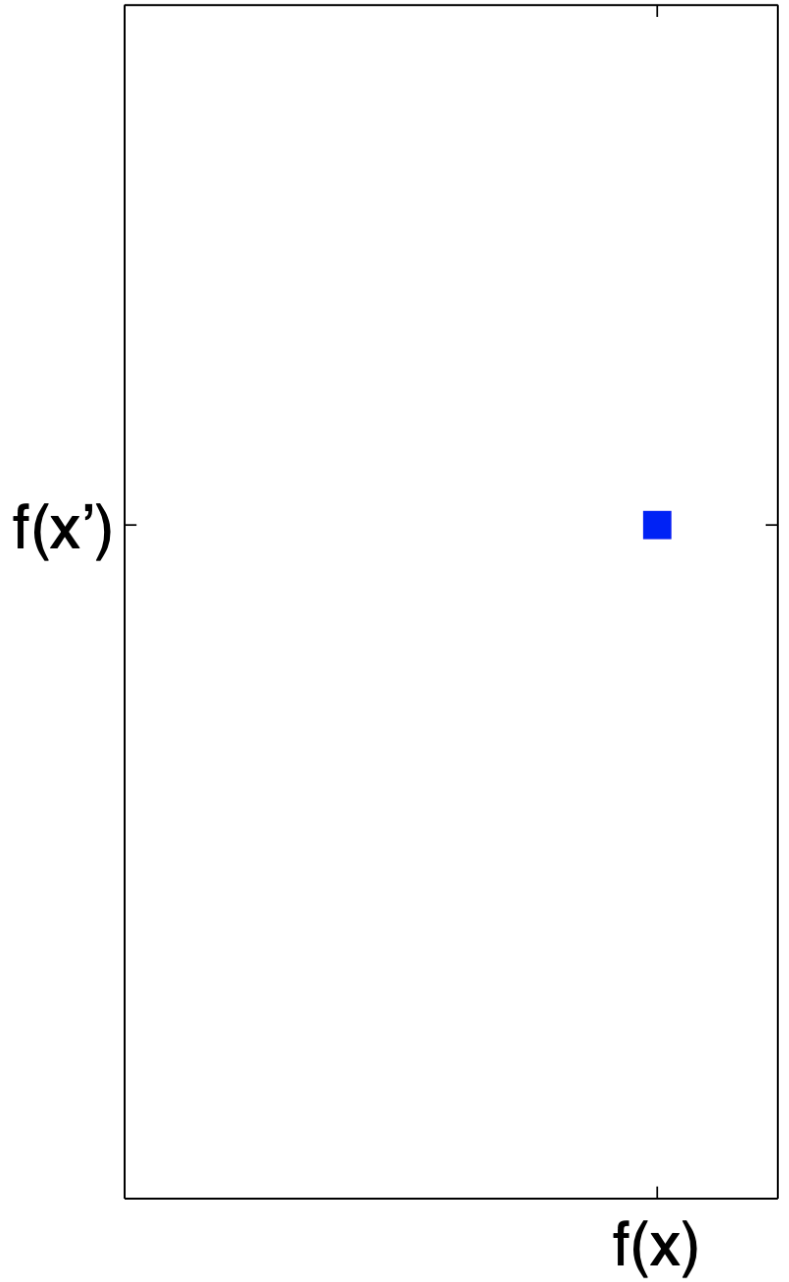
  for any finite collection of points $x_1, \ldots, x_k$.

→ Let's try to visually investigate what this means assuming $k_0(x, x')$ decreases with $\|x - x'\|$, implying less correlation when inputs far apart

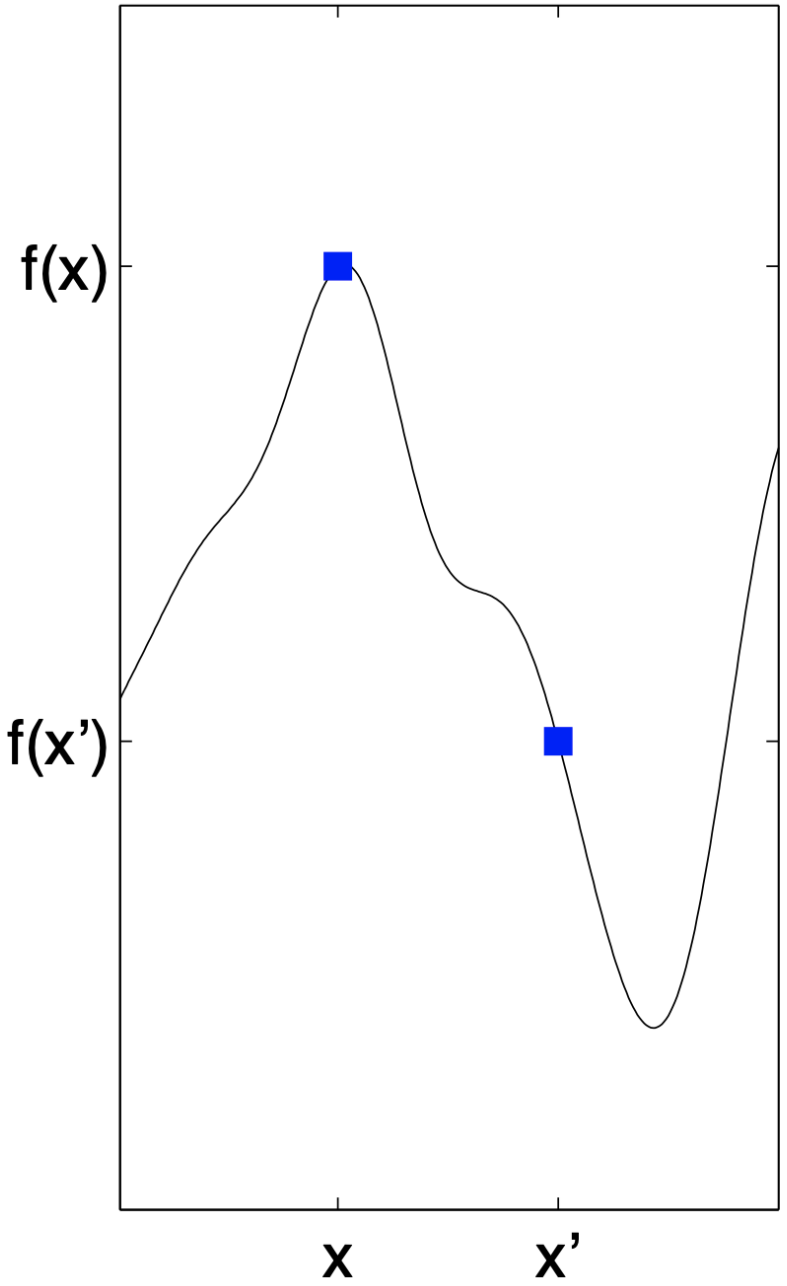


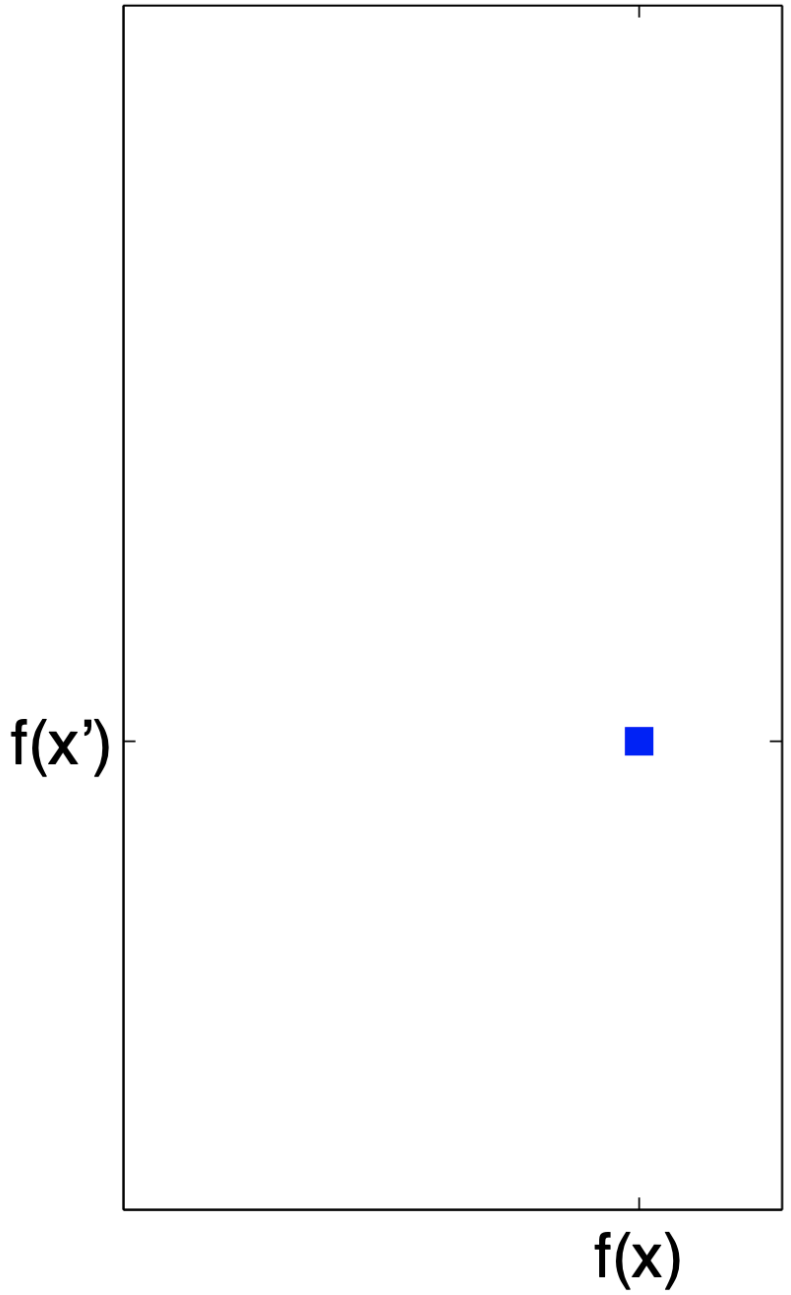
- Fix two points x and x′
- Consider the vector of outputs [f(x), f(x′)]



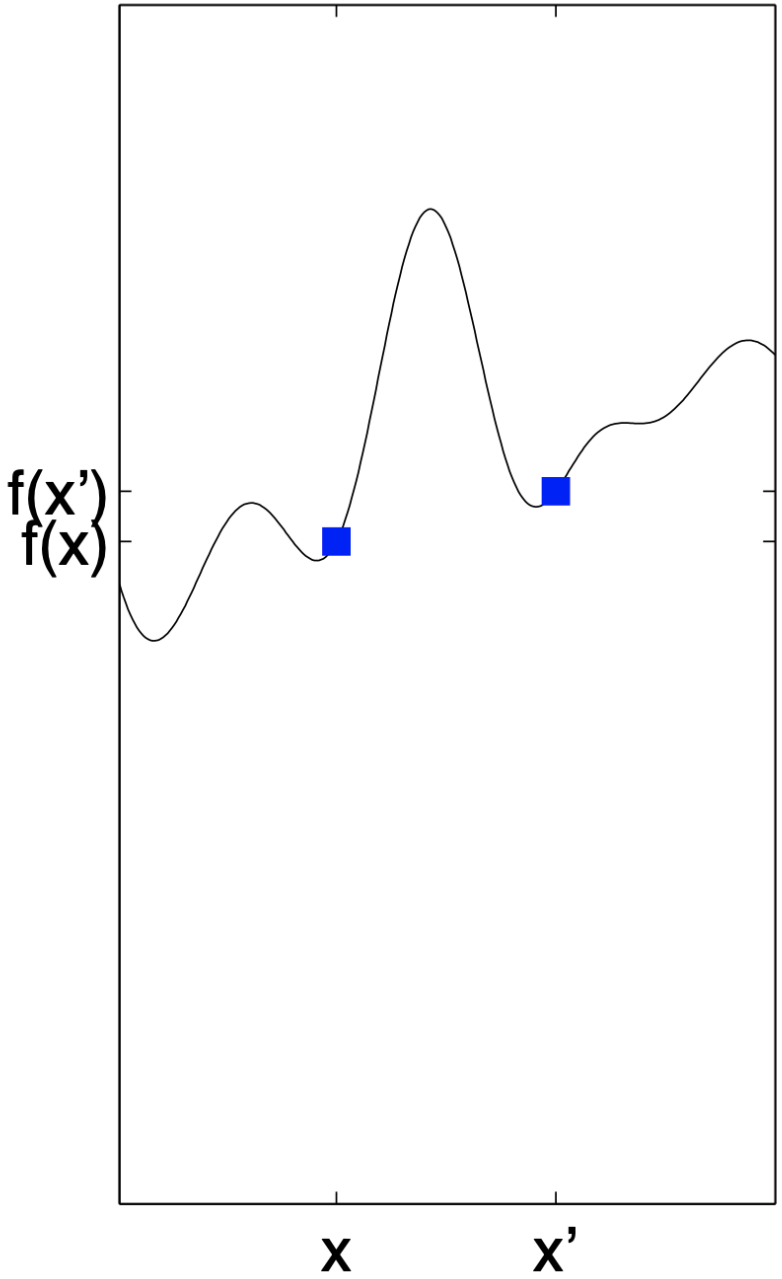


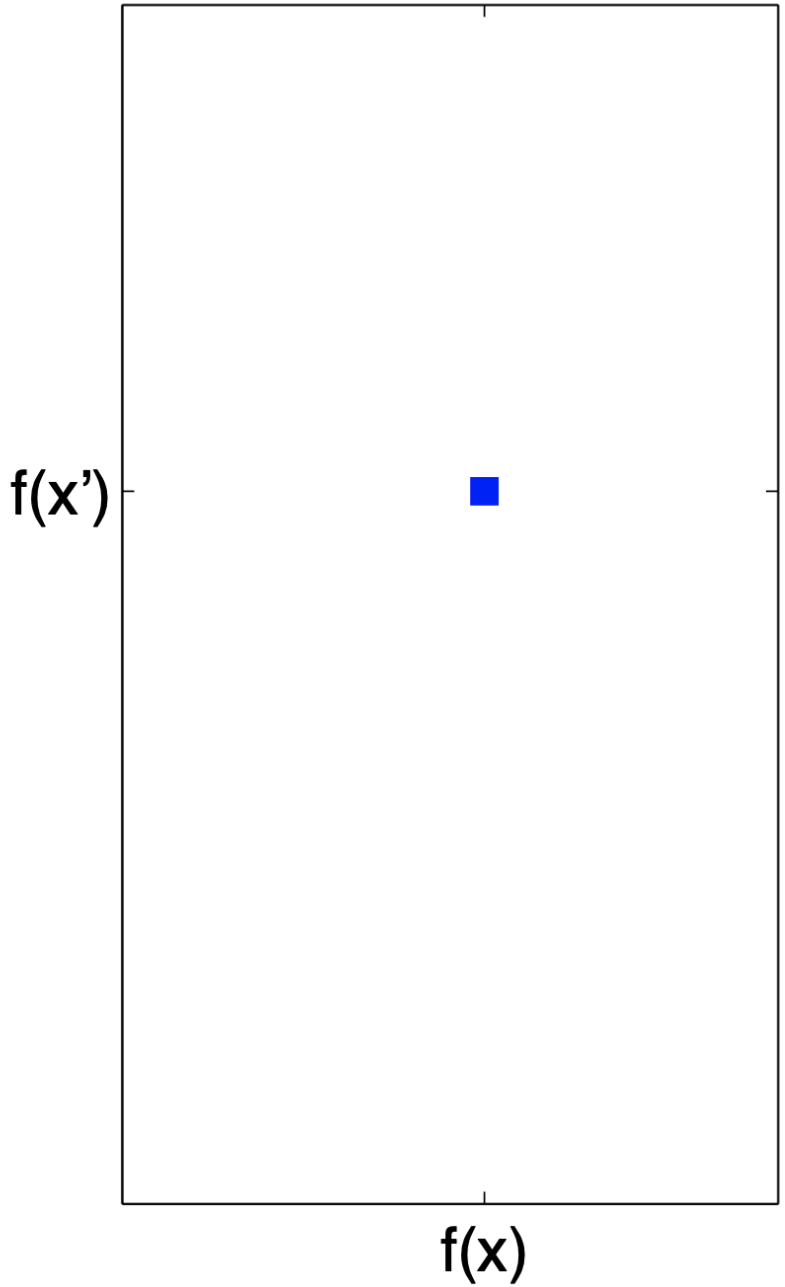
P. Frazier, INFORMS Tutorial, 2018



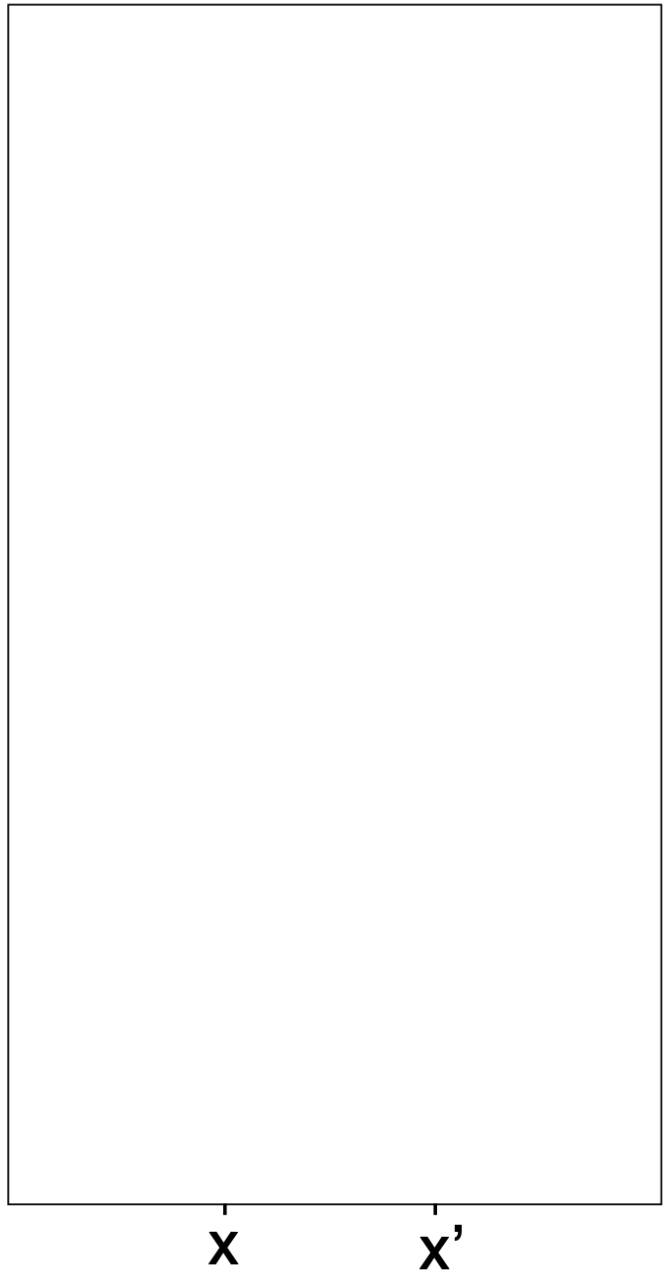


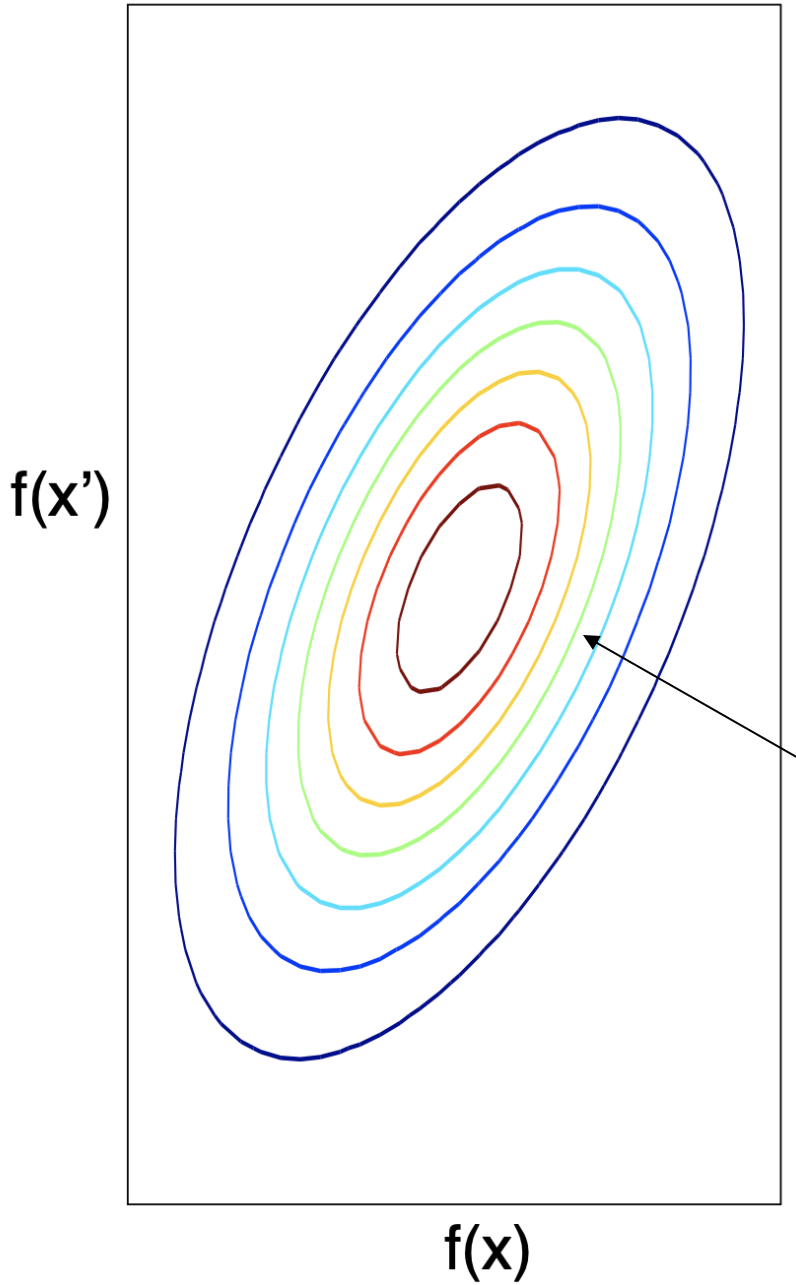
P. Frazier, INFORMS Tutorial, 2018



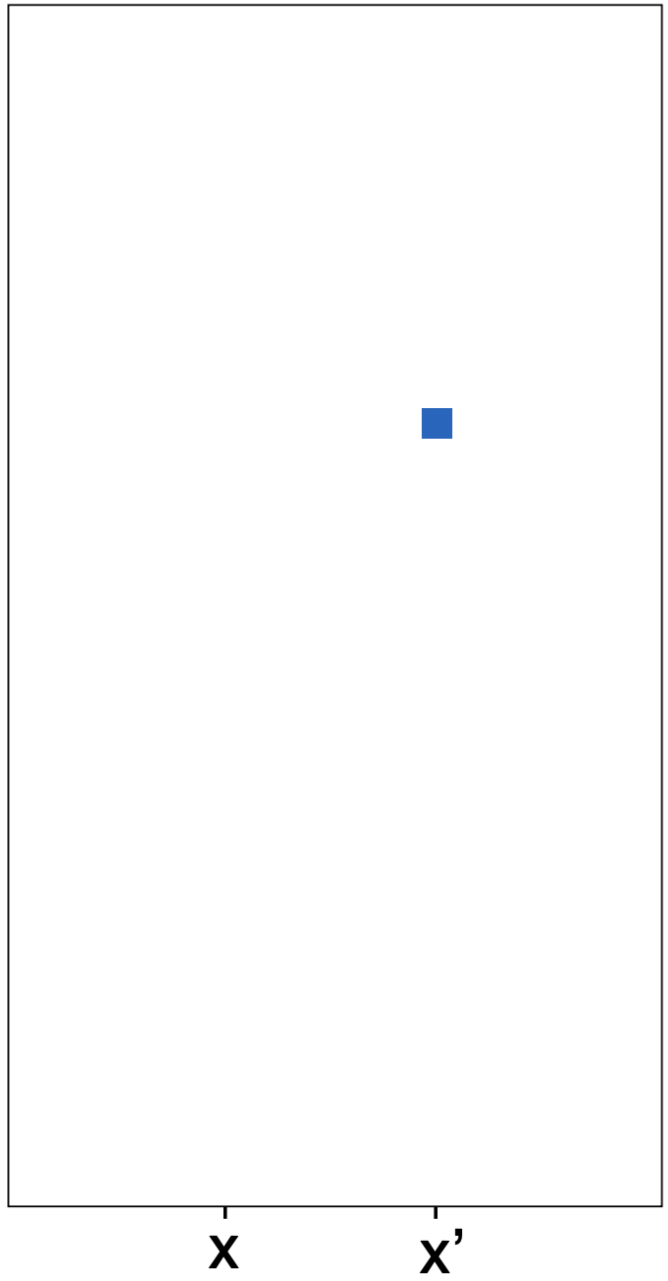


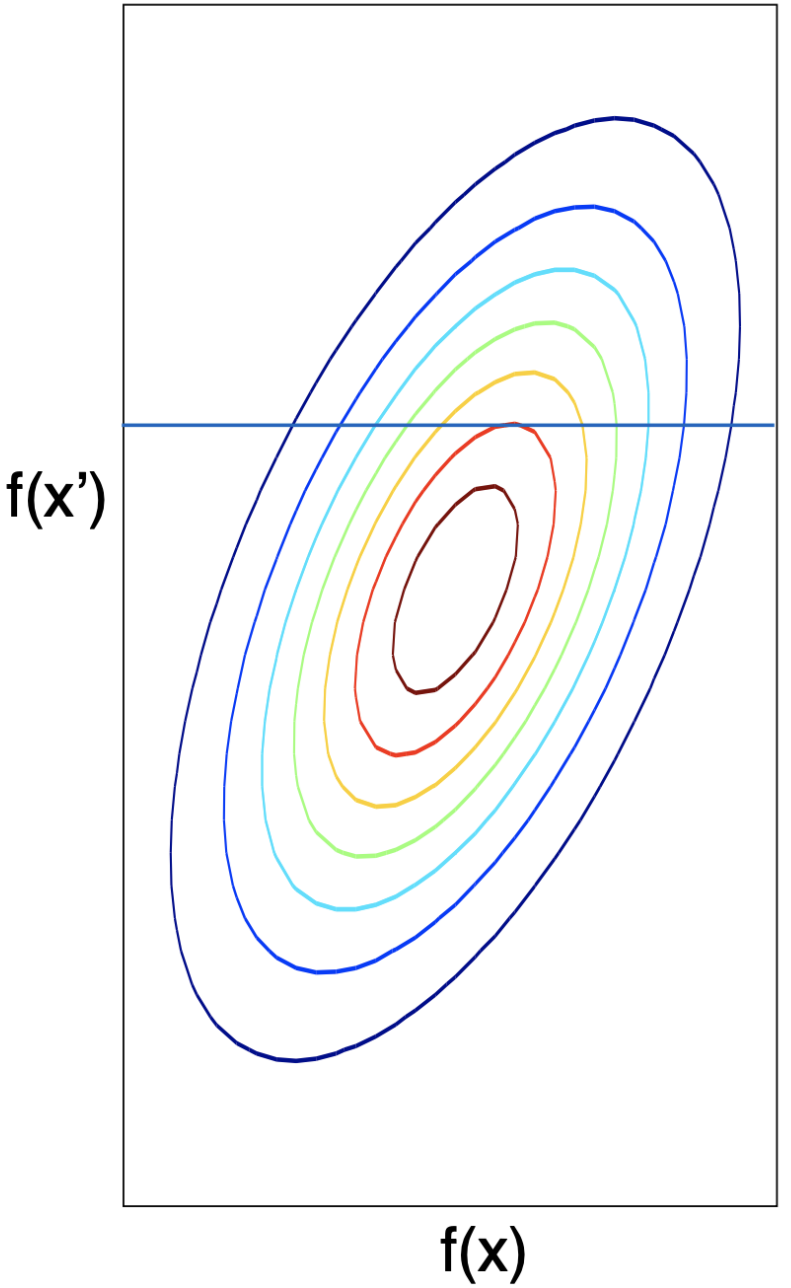
P. Frazier, INFORMS Tutorial, 2018



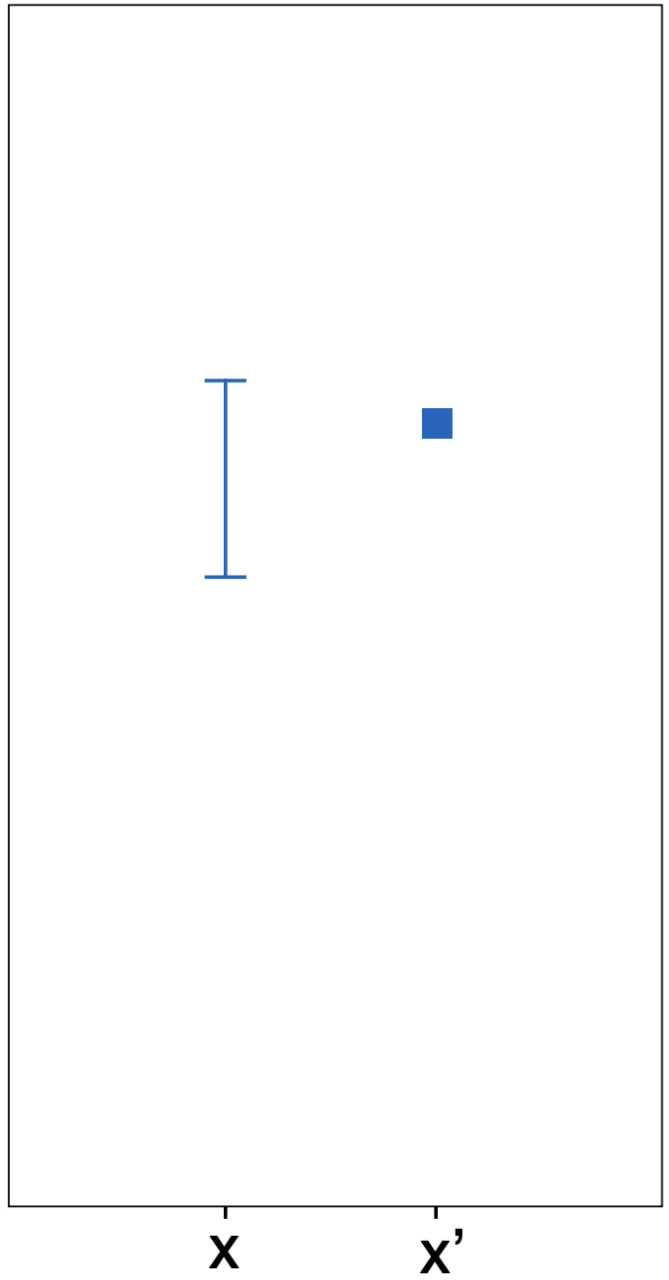


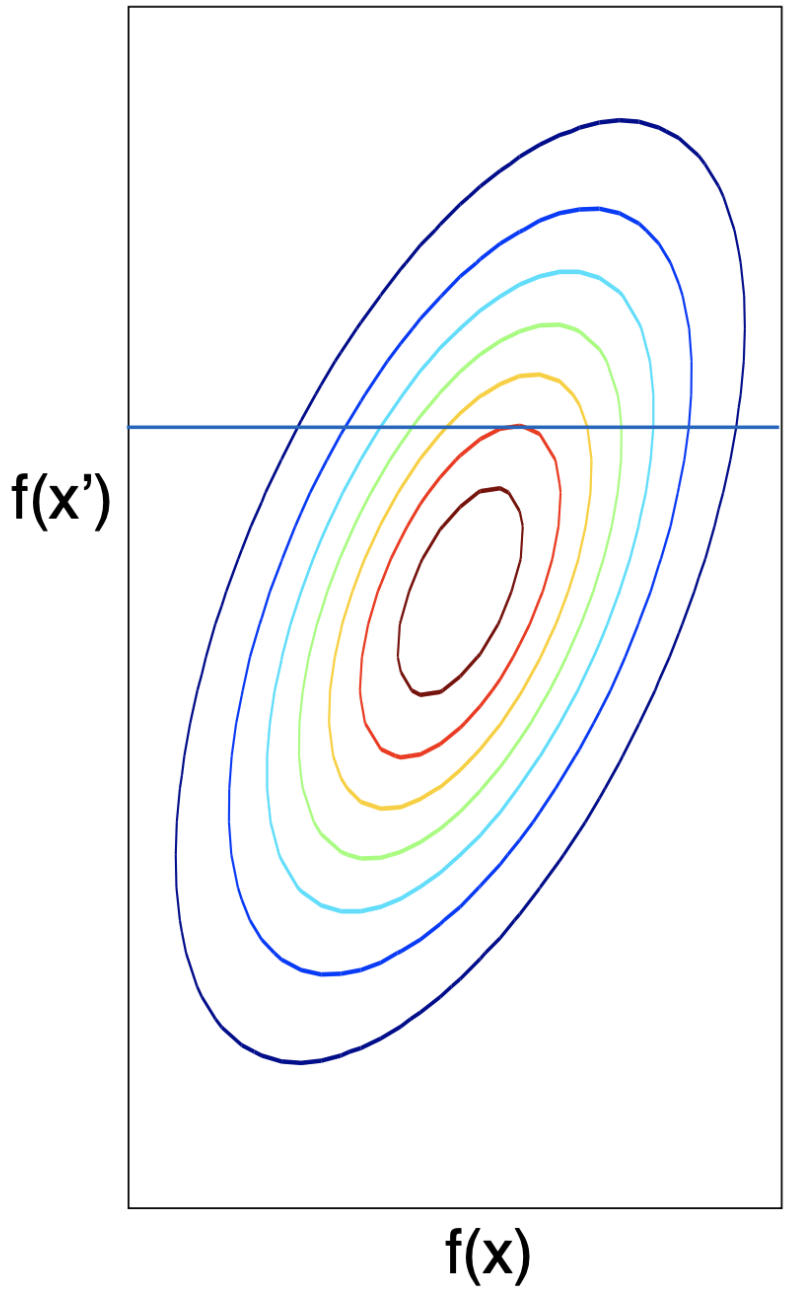
GP assumes a multivariate Gaussian prior for outcomes of any fixed inputs



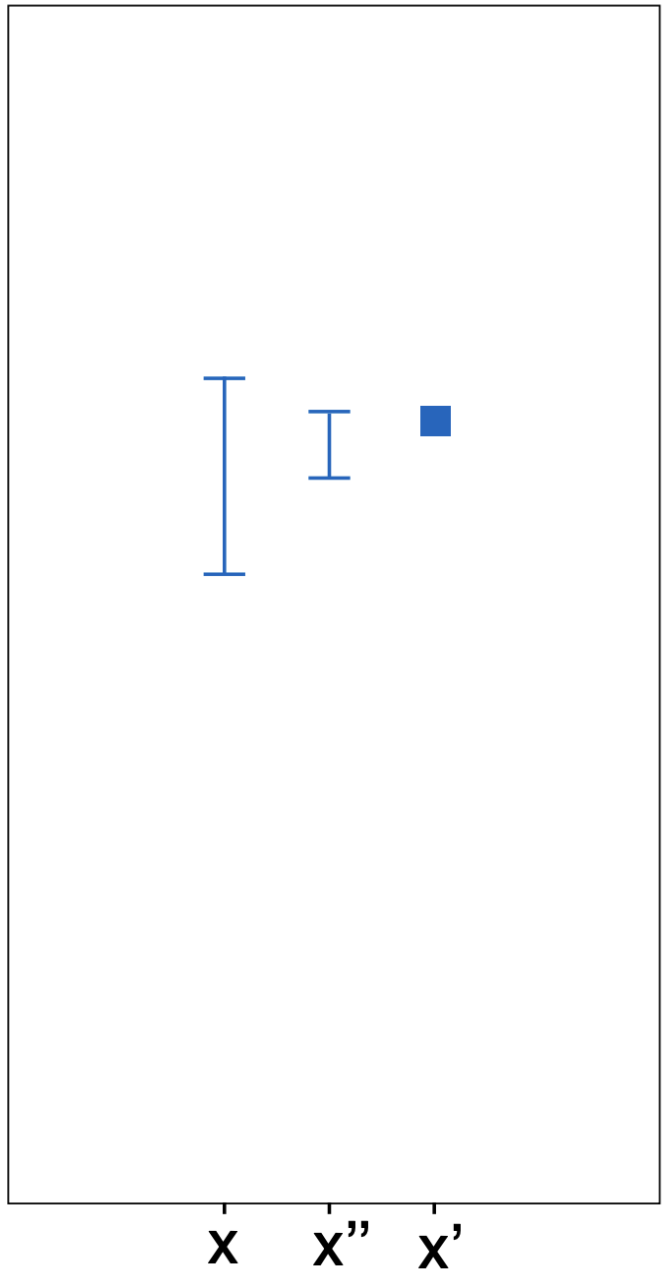


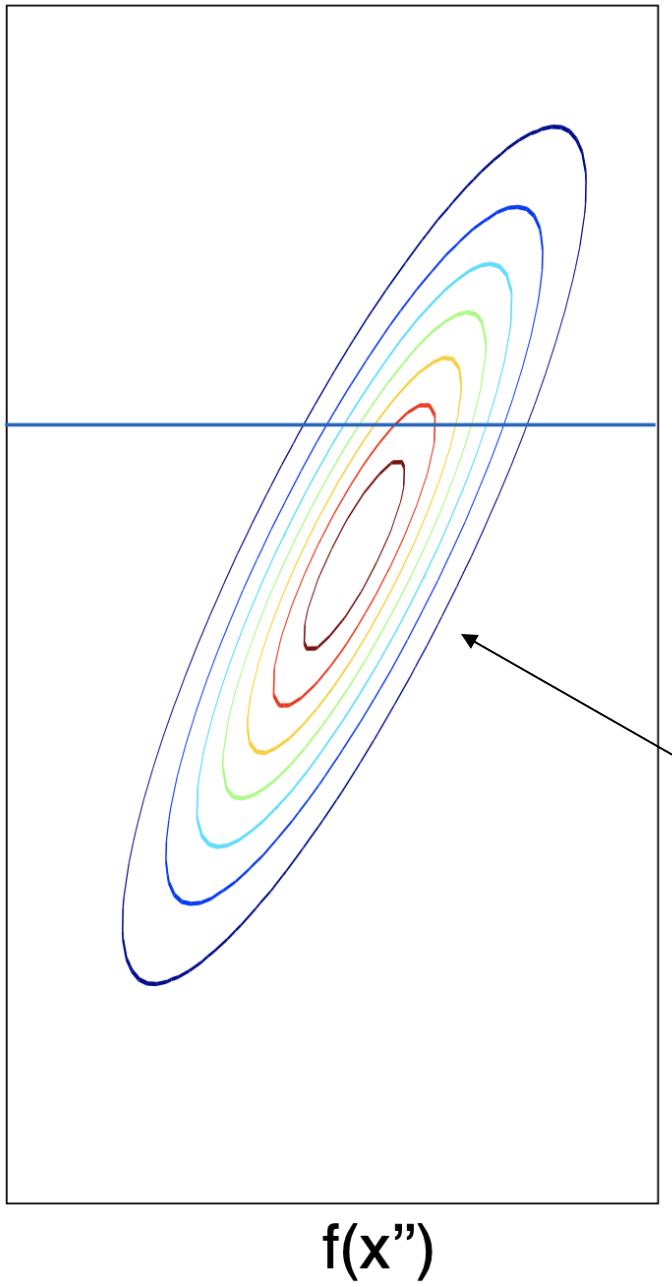
P. Frazier, INFORMS Tutorial, 2018





P. Frazier, INFORMS Tutorial, 2018





Notice how the covariance “shrinks” as x and x' get closer together

P. Frazier, INFORMS Tutorial, 2018

Before moving to inference, let’s discuss 3 properties of Gaussian random variables

1. The Gaussian Law (Linear Transformation)

- Gaussian distributions are closed under linear transformations:

$$X \sim \mathcal{N}(\mu_X, \Sigma_X) \quad \Rightarrow \quad Y = AX + b \sim \mathcal{N}(A\mu_X + b, A\Sigma_X A^\top)$$

- An immediate consequence is any n-dimensional Gaussian random vector can be produced from $\mathcal{N}(0, I_n)$ (independent Gaussian RVs)

- This is often known as the "whitening transformation" and can be expressed in terms of the Cholesky decomposition $\Sigma_X = LL^\top$:

$$X = \mu_X + L\mathcal{N}(0, I) \sim \mathcal{N}(\mu_X, \Sigma_X)$$

2. Gaussian Marginalization

- Assume that we have two random vectors that are jointly Gaussian, so that they have the following distribution

$$(\mathbf{x}, \mathbf{y}) \sim \mathcal{N}\left(\begin{bmatrix} \mathbf{a} \\ \mathbf{b} \end{bmatrix}, \begin{bmatrix} A & C \\ C^\top & B \end{bmatrix}\right)$$

- What is the distribution of the random variable $\mathbf{x}$?

$$\mathbf{x} \sim \mathcal{N}(\mathbf{a}, A)$$

Prove by treating marginalization as projection operation: $\mathbf{x} = \begin{bmatrix} I & 0 \end{bmatrix} \begin{bmatrix} \mathbf{x} \\ \mathbf{y} \end{bmatrix}$

3. Gaussian Conditional

- It turns out any conditional of a Gaussian distribution is also Gaussian

$$(\mathbf{x}, \mathbf{y}) \sim \mathcal{N}\left(\begin{bmatrix} \mathbf{a} \\ \mathbf{b} \end{bmatrix}, \begin{bmatrix} A & C \\ C^\top & B \end{bmatrix}\right)$$

$$\mathbf{x}|\mathbf{y} \sim \mathcal{N}(\mathbf{a} + CB^{-1}(\mathbf{y} - \mathbf{b}), A - CB^{-1}C^\top)$$

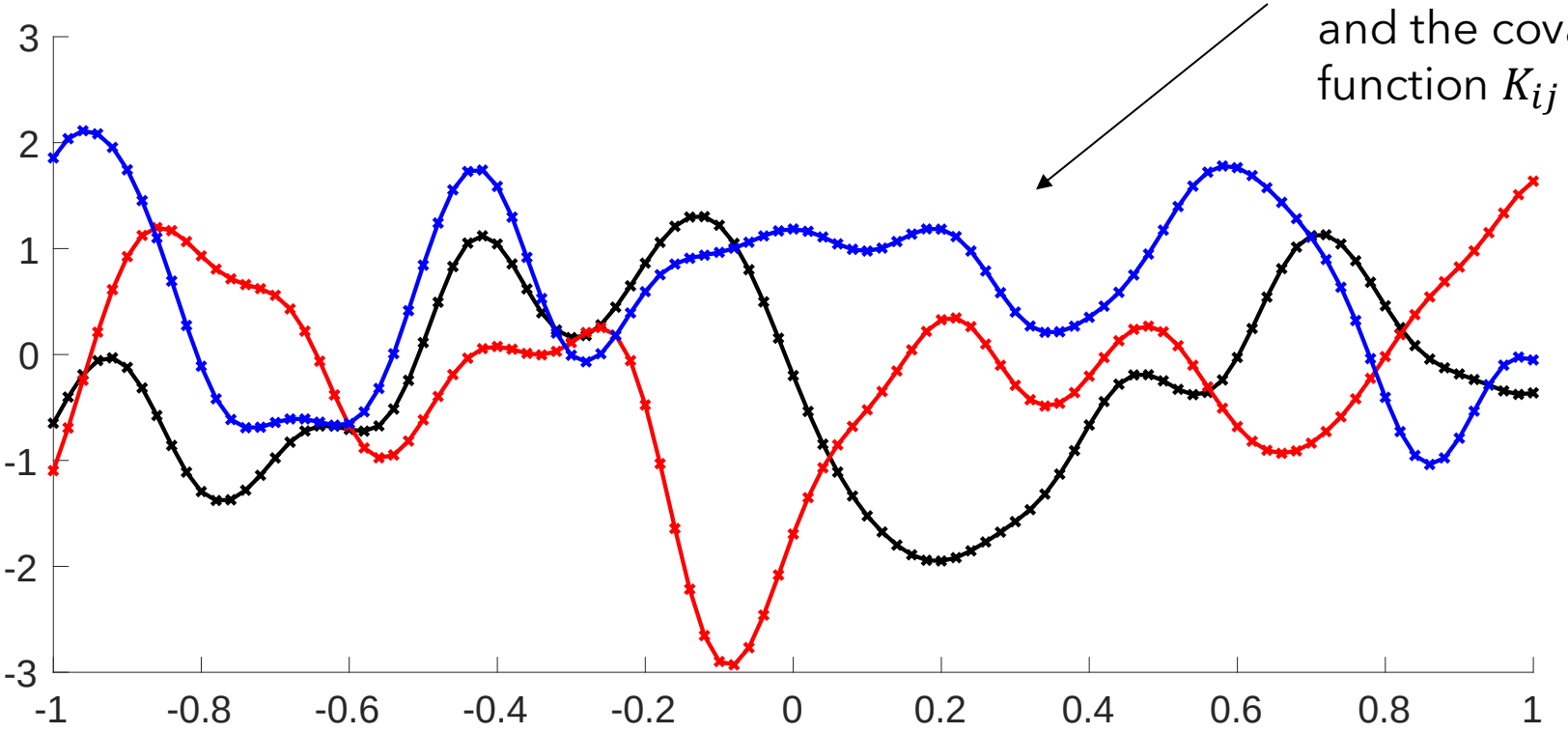
Showing this result is not as straightforward, but is a standard result that can be looked up / proved in several textbooks

Now back to GPs

Sampling From a Gaussian Process

- The complete GP is infinite-dimensional; however, we can draw finite parts of a GP sample using the marginalization property of GPs

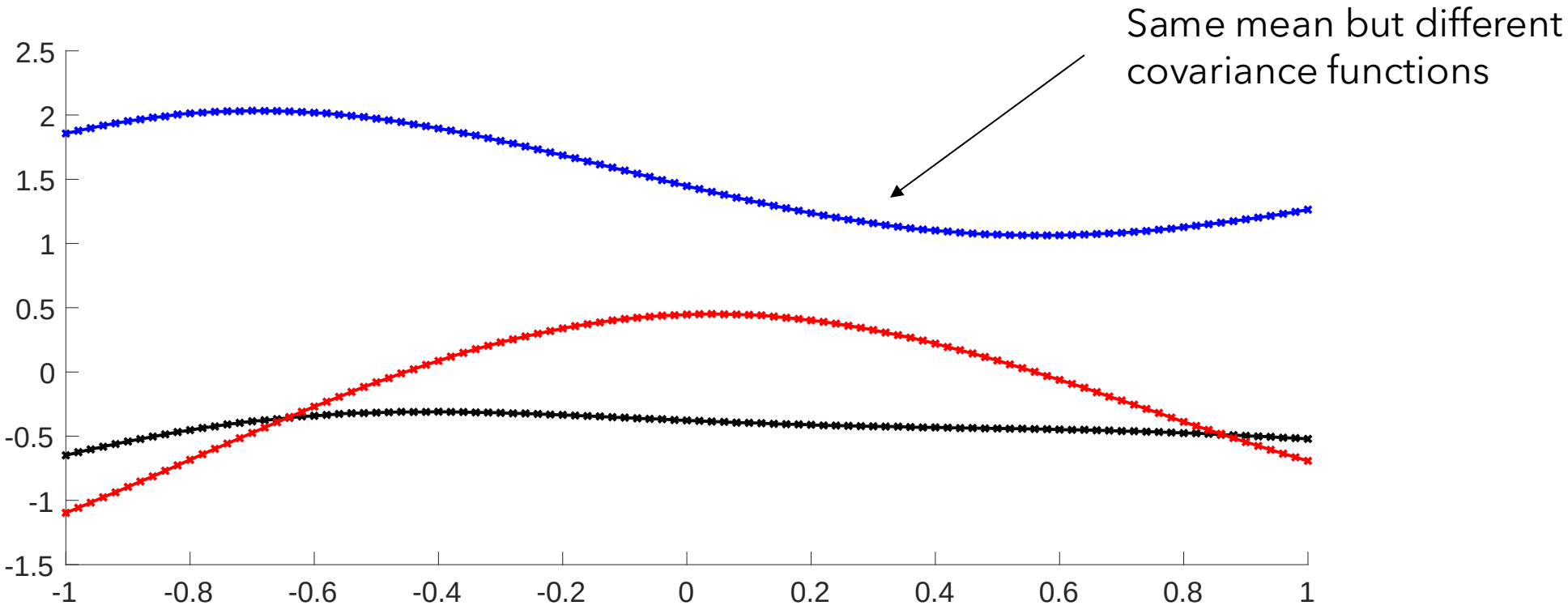
Need to select mean
(in this case zeros(100,1))
and the covariance
function $K_{ij} = k(x_i, x_j)$



- Here are 3 realizations on a 100-dimensional grid, drawn from a GP

Sampling From a Gaussian Process

- The complete GP is infinite-dimensional; however, we can draw finite parts of a GP sample using the marginalization property of GPs



- Here are 3 realizations on a 100-dimensional grid, drawn from a GP

Inference using a Gaussian Process (Nonparametric Regression)

- Now, we assume our function has a GP prior: $f(x) \sim \mathcal{GP}(\mu_0, k_0)$

- Assuming we have past data $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^n$ generated from this prior, can treat the regression problem as a Bayesian inference problem:

$$p(f(x)|\mathcal{D}) \propto p(\mathcal{D}|f(x))p(f(x))$$

$$[\text{GP posterior} \propto \text{Likelihood} \times \text{GP prior}]$$

- **Can we derive GP posterior if we assume Gaussian likelihood?**

We Can Compute GP Posterior Analytically!

- The posterior on f(x′) (at a test point)* given n past observations $y_1,\ldots,y_n$ is normal with mean $\mu_n(x')$ and variance $\sigma_n^2(x')$:

$$\mu_n(x') = k_0(x', x_{1:n}) \left(k_0(x_{1:n}, x_{1:n}) + \sigma^2 I_n\right)^{-1} y_{1:n}$$

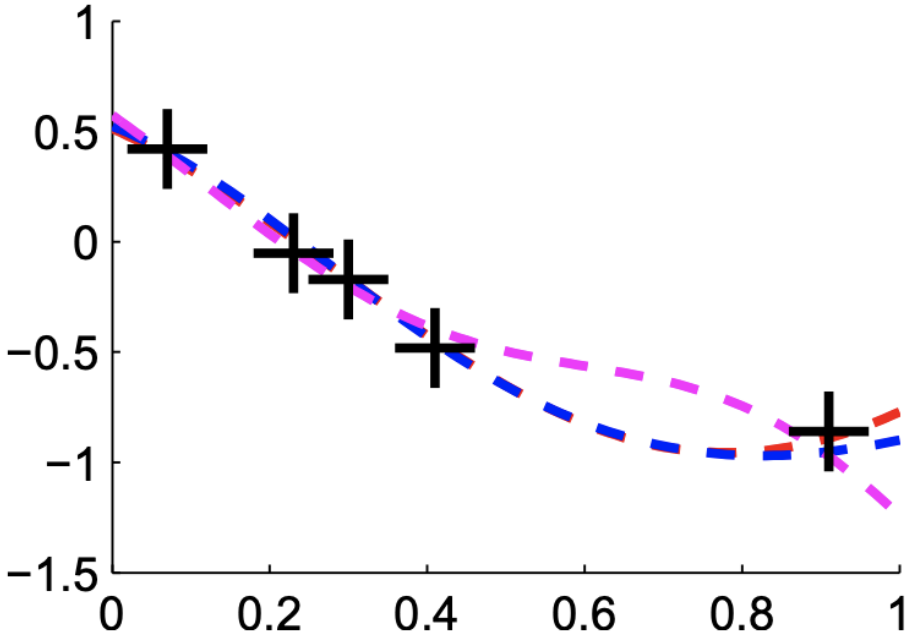
$$\sigma_n^2(x') = k_0(x', x') - k(x', x_{1:n}) \left(k(x_{1:n}, x_{1:n}) + \sigma^2 I_n\right)^{-1} k(x_{1:n}, x')$$

- Formula assumes $\mu_0(x) = 0$ but can be easily modified for general μ_0

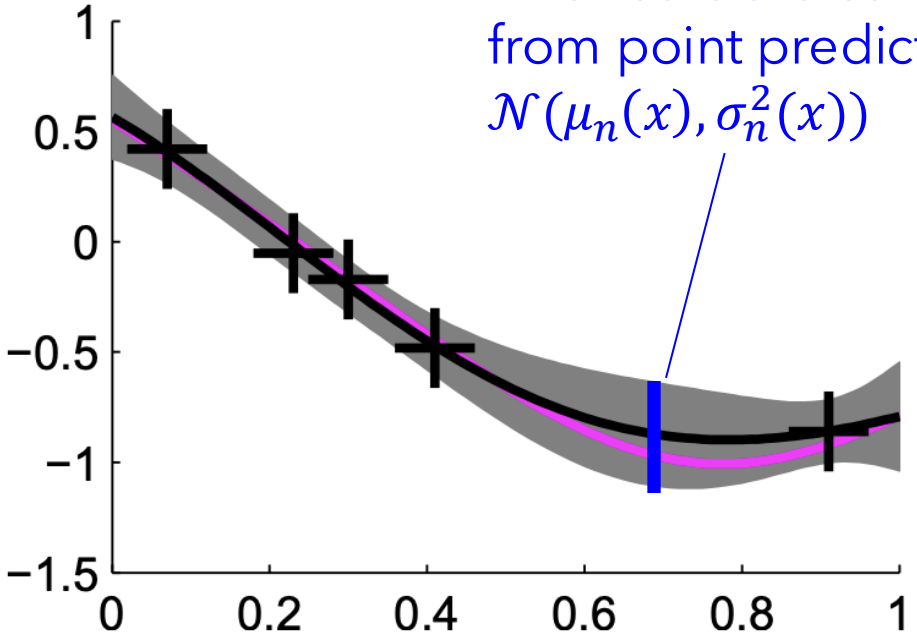
*Can be derived from Gaussian conditional formula

GP Posterior

Two (incomplete) ways of visualizing what we have learned:



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

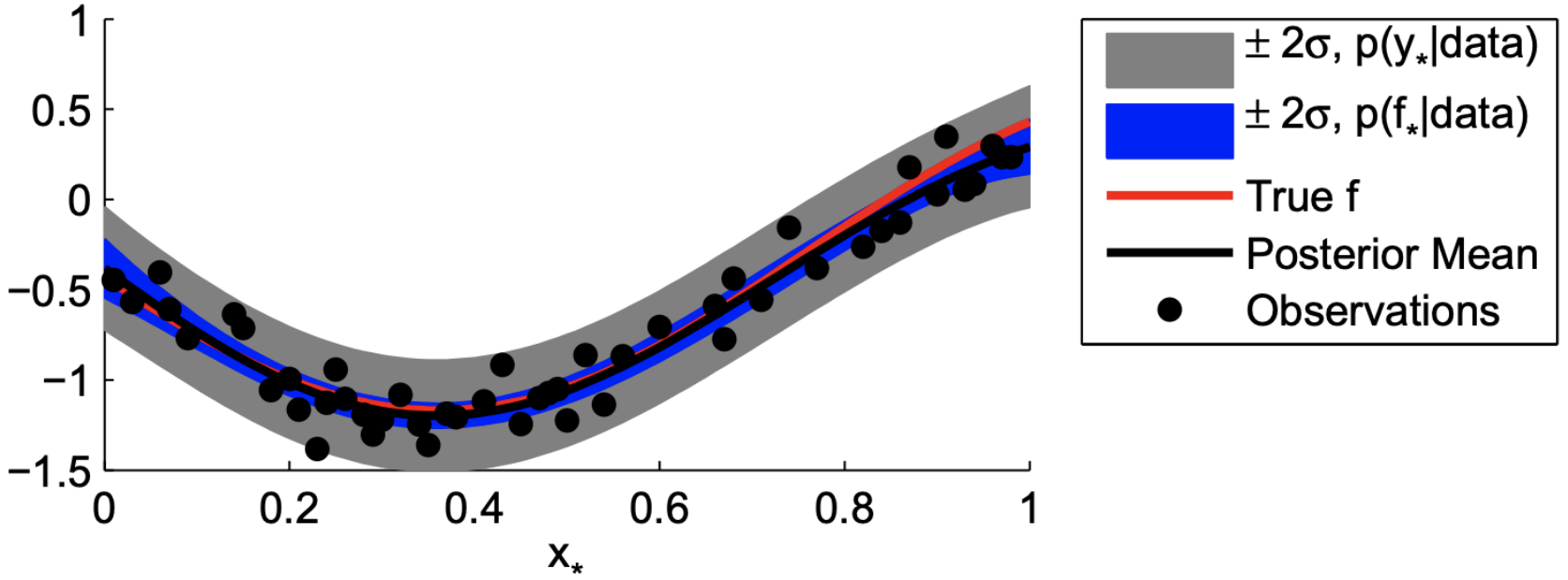


Error bars are constructed from point predictions $\mathcal{N}(\mu_n(x), \sigma_n^2(x))$

Mean and error bars

Discovery or Prediction?

- What should the error bars show?



- $f_*|y_{1:n} \sim \mathcal{N}(\mu_n(x), \sigma_n^2(x))$ says what we know about noiseless function
- $y_*|y_{1:n} \sim \mathcal{N}(\mu_n(x), \sigma_n^2(x) + \sigma^2)$ predicts what we will see next at point x_*

Let's recap what we learned in this section

- We can represent a function as a big (infinite) vector $\mathbf{f}$
- We assume this unknown vector was drawn from a big (infinite) correlated Gaussian distribution, which we call a Gaussian process (GP)
  - This is why GPs are often called "non-parametric" since they cannot be represented by a finite set of weight parameters
- Observing elements of function (optionally corrupted by i.i.d. Gaussian noise) creates a posterior distribution that is also a GP (conjugacy)
- From marginalization principle, we can easily ignore all positions x_i that are neither observed nor queried

Outline

- Introduction to Bayesian modeling
  - Bayes rule, importance of model averaging, epistemic uncertainty
- The function-space view
  - Gaussian processes, mean and kernel functions, inference
- Deep dive into the kernel
  - Importance of kernel, stationary vs. non-stationary kernels, learning hyperparameters
- Beyond traditional Gaussian processes
  - Heteroskedastic noise, sparse approximations, SAAS, deep kernel learning
- Efficient software packages
  - Quick overview of options, GPyTorch for flexibility and scalability

The kernel

- The main part that has been missing in our discussion so far is where does the kernel (covariance function) $k(x, x')$ come from?

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

- It turns out that the choice of kernel is extremely flexible and in fact be can selected in a way to represent most of the common model classes (including linear models, neural networks, and beyond)

The Squared Exponential (SE) Kernel

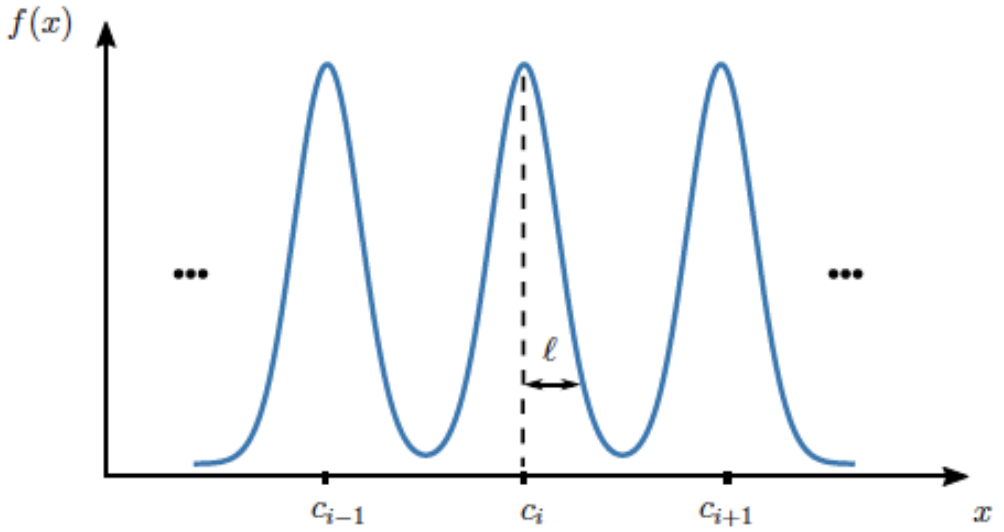
- So far, saw the linear kernel $k(x, x') = \sigma_0^2 + \sigma_1^2 \langle x, x' \rangle$…this leads to a GP that is equivalent to a Bayesian linear regression (weight-space view)

- One of the most popular kernels is the squared exponential (SE) (also known as radial basis function (RBF)) kernel

$$k_{\mathrm{SE}}(x, x') = a^2 \exp\left(-\frac{1}{2} \sum_{d=1}^{D} \frac{(x_d - x'_d)^2}{l_d^2} \right)$$

- Not always "best" choice since it assumes f is infinitely differentiable

Why is the SE kernel so popular?



$$f(x) = \sum_{i=1}^{J} w_i \phi_i(x)\,, \quad w_i \sim \mathcal{N}\left(0, \frac{\sigma^2}{J}\right)\,, \quad \phi_i(x) = \exp\left(-\frac{(x - c_i)^2}{2\ell^2}\right)$$

$$\therefore k(x, x') = \frac{\sigma^2}{J} \sum_{i=1}^{J} \phi_i(x)\phi_i(x')$$

Take limit as $J \to \infty$ assuming $\Delta c = \frac{2\log(J)}{J}$

→ You will get $k(x, x') \propto k_{SE}(x, x')$

Why is the SE kernel so popular?

- Remarkably, we can work with an infinite set of basis functions using a finite amount of computation using the *kernel trick* – replacing inner products of basis functions with kernel operations

- GPs using SE kernels are **flexible** (universal approximators) but **simple**

- By concentrating prior support around relatively simple solutions, these models can provide good generalization even on small datasets while also keeping substantial flexibility

- Are all GPs universal approximators?

Why is the SE kernel so popular?

- Remarkably, we can work with an infinite set of basis functions using a finite amount of computation using the *kernel trick* – replacing inner products of basis functions with kernel operations

- GPs using SE kernels are **flexible** (universal approximators) but **simple**

- By concentrating prior support around relatively simple solutions, these models can provide good generalization even on small datasets while also keeping substantial flexibility

- Are all GPs universal approximators?
  - No! Only kernels with a reproducing property (e.g., polynomial kernels correspond to finite polynomial models that cannot model all smooth functions)

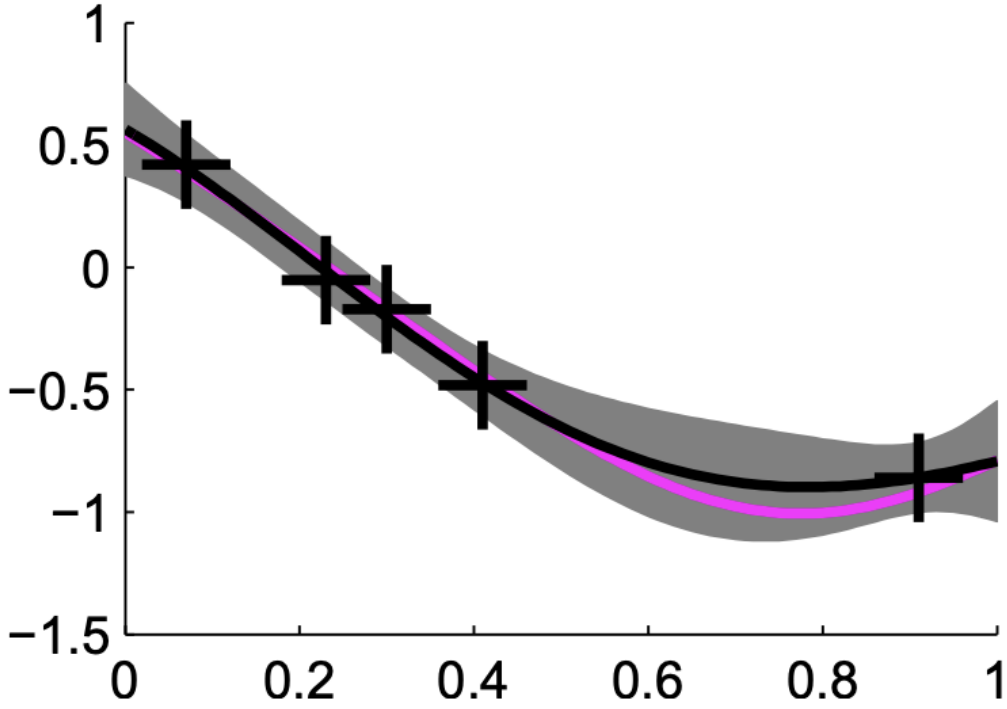
Back to the SE Kernel

$$k_{\mathrm{SE}}(x, x') = a^2 \exp\left(-\frac{1}{2}\sum_{d=1}^{D}\frac{(x_d - x'_d)^2}{l_d^2}\right)$$

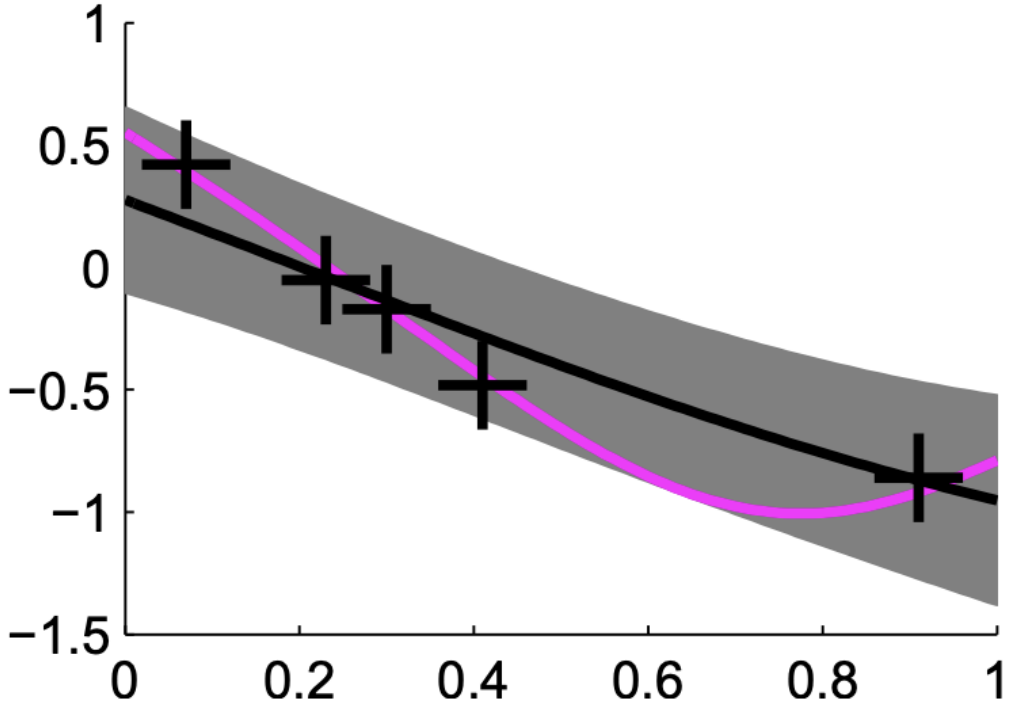
- Notice the kernel has hyperparameters
  - a^2 which controls the marginal function variance (scale)
  - l_d in each dimension controls the correlation length (distance between peaks)

- Many kernels will have similar hyperparameters, what is their impact?

Different Hyperparameters = Different Explanations of Data



$l = 0.5, a = 0.05$



$l = 1.5, a = 0.15$

Marginal Likelihood for Model Selection (and Hyperparameter Learning)

- The *marginal likelihood* (also known as the *Bayesian evidence*) is the probability we would generate dataset $\mathcal{D}$ with model $\mathcal{M}$ if we randomly sample from a prior over its parameters

$$p(\mathcal{D}|\mathcal{M}) = \int p(\mathcal{D}|, \mathbf{f}, \mathcal{M}) p(\mathbf{f}|\mathcal{M}) \, d\mathbf{f}$$

- $\mathcal{M}$ could represent different kernels or hyperparameters of specific kernel
  - Or even both, which we will touch on a bit later (called a fully Bayesian method)

- For a GP with Gaussian likelihood, can find an exact expression for integral

GP Regression using the Marginal Likelihood

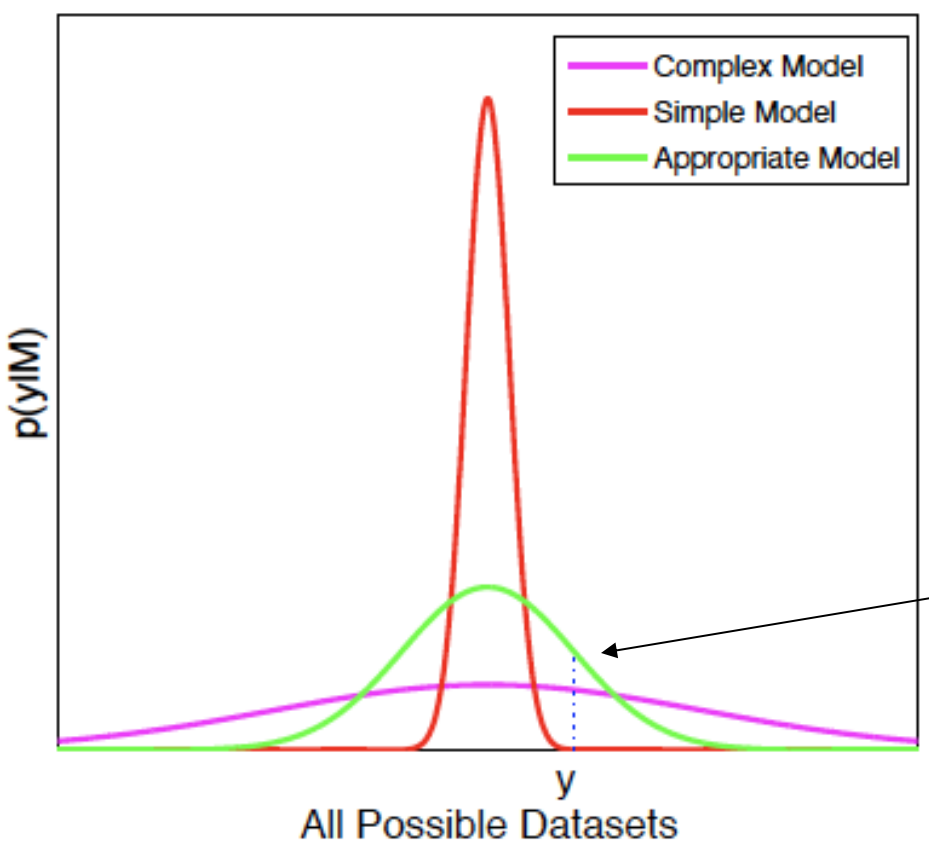
- The probability of the observed data $\boldsymbol{y}$ is just a Gaussian

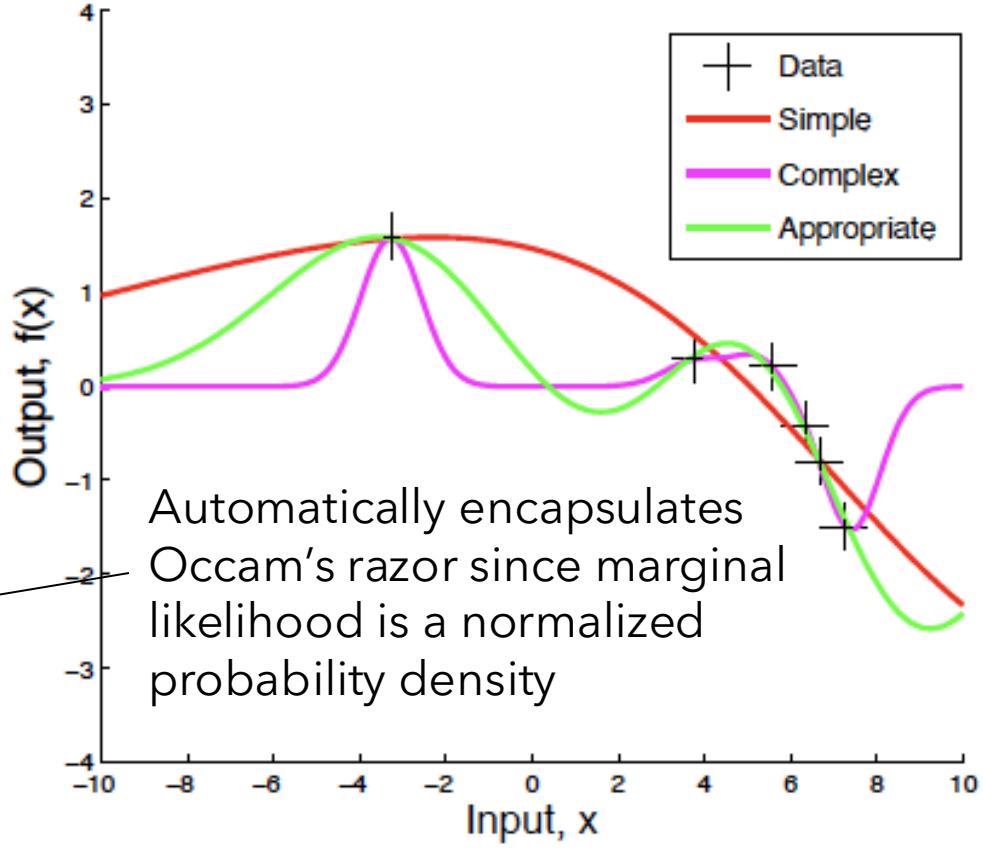
- Therefore, we can select the hyperparameters by maximizing the likelihood (same as minimizing the negative log-likelihood function)

$$-\log p(\mathbf{y} \mid \theta) = \underbrace{\frac{1}{2}(\mathbf{y}-\boldsymbol{\mu})^\top \Sigma_{\mathbf{y}}^{-1}(\mathbf{y}-\boldsymbol{\mu})}_{\text{Data fit term}} + \underbrace{\frac{1}{2}\log(|\Sigma_{\mathbf{y}}|)}_{\text{Complexity penalty}} + \frac{n_y}{2}\log(2\pi)$$

- We can compute derivatives of this function with respect to θ such that we can optimize the hyperparameters using standard algorithms

Marginal Likelihood for Model Selection (and Hyperparameter Learning)



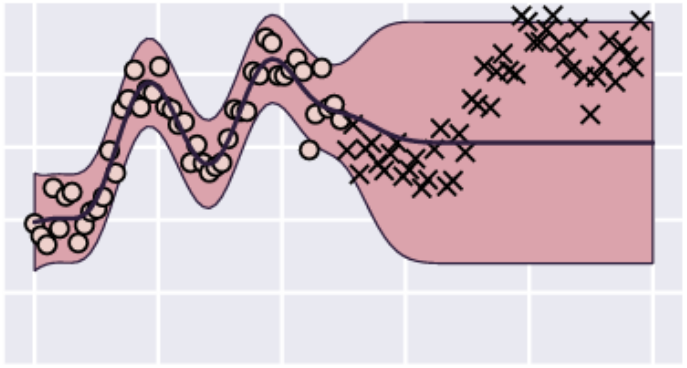


Gaussian Processes for Machine Learning. Rasmussen and Williams, 2006

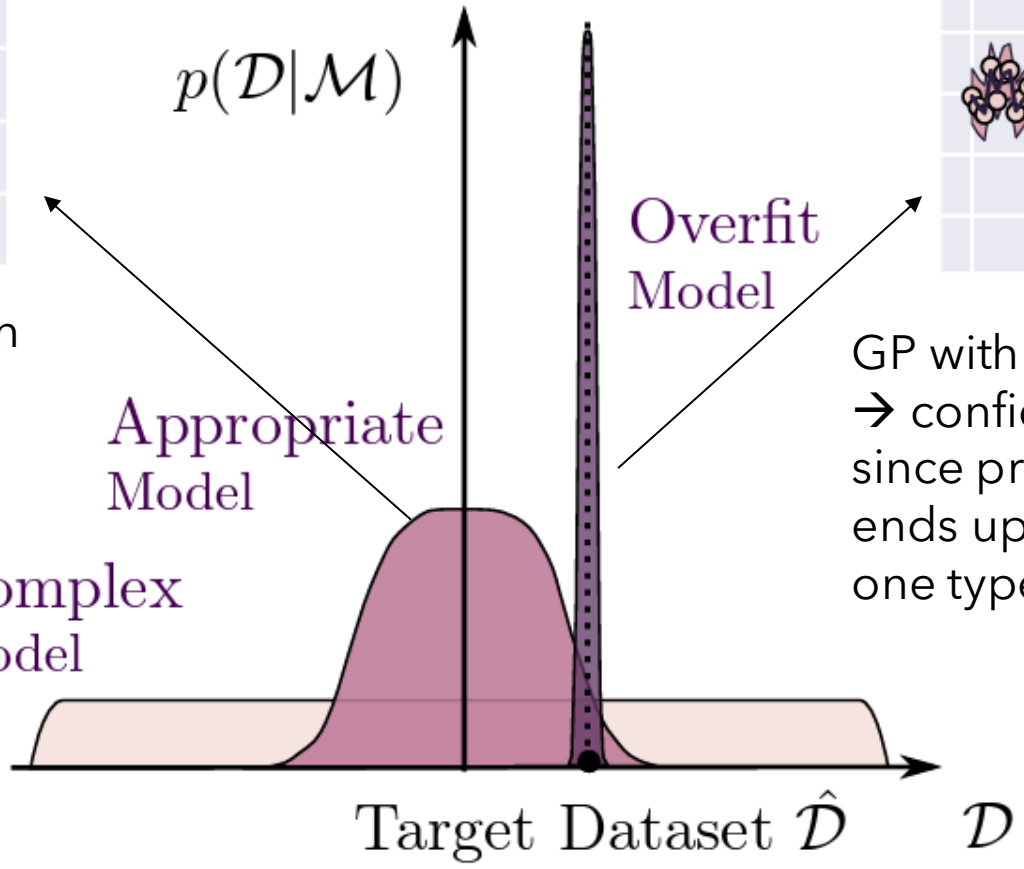
Bayesian Methods for Adaptive Models. MacKay, 1992.

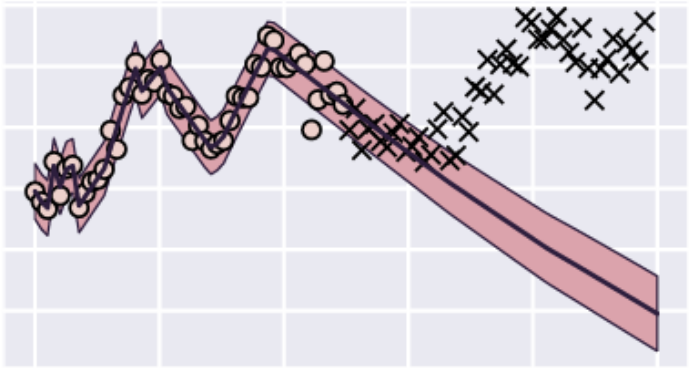
Covariance Kernels for Fast Automatic Pattern Discovery and Extrapolation with Gaussian Processes. Wilson, 2014.

CAUTION: Marginal Likelihood can Overfit



Standard GP with constant mean



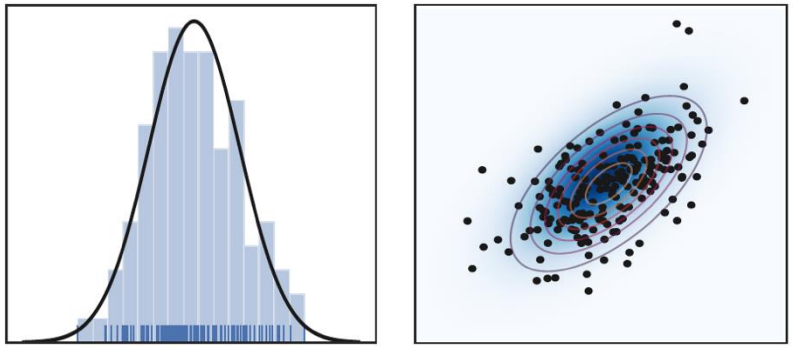


GP with neural network mean function → confidently overfits the training data since prior is extremely flexible and ends up being only able to produce one type of solution

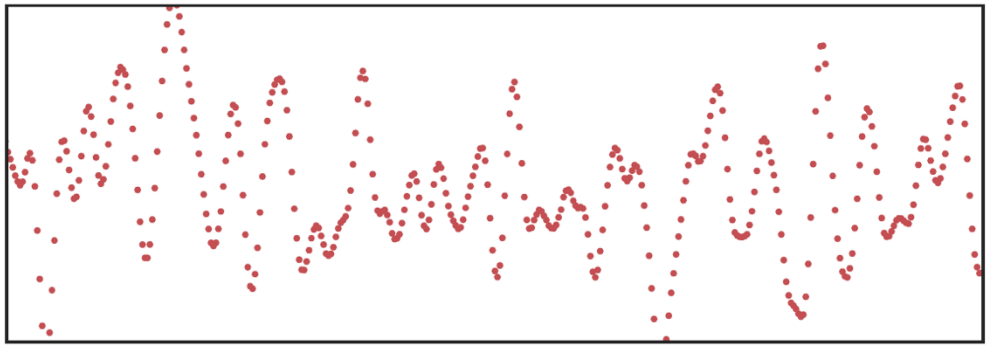
Lofti et al., Bayesian Model Selection, the Marginal Likelihood, and Generalization, ICML, 2022

Why Do We Have to Parametrize Kernel?

Finite Gaussians: # samples >> parameter dimension



Gaussian process: # samples << 1 << param dimension



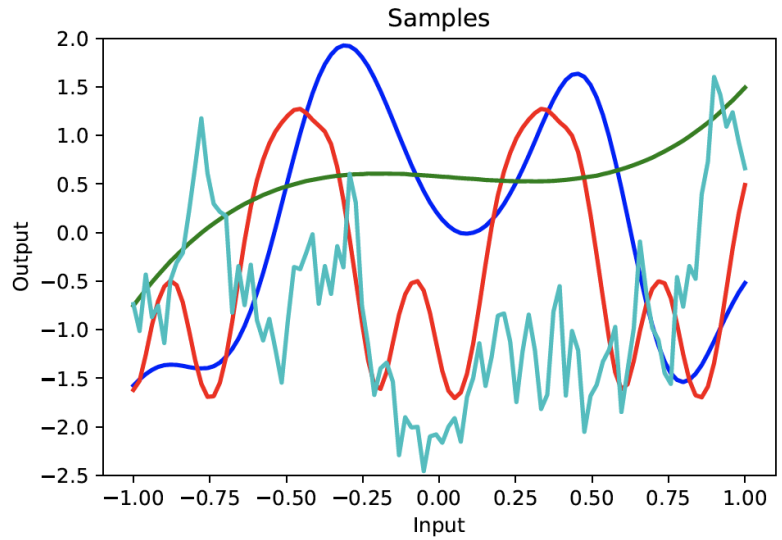
- Learning finite Gaussians in d dimensions requires many observations to fit $d + \frac{d(d+1)}{2}$ parameters (usually **overdetermined** problem)

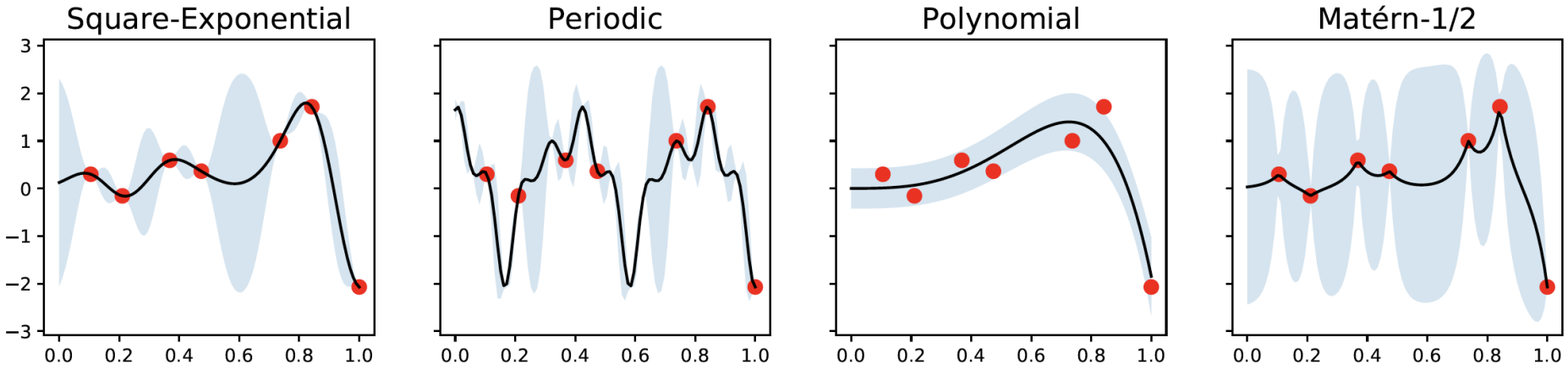
- With GPs, we do not even have one full observation of the model, but only parts of it. Therefore, learning a GP is inherently an **underdetermined** problem

To learn a GP, we must *tie* the values of mean and covariance to reduce their degrees of freedom → cannot freely assign covariance to each pair (x, y) and instead choose a function $k_\theta(x, y)$ and compare different values of θ

Important: Different Kernels, Different Predictions

- Square-Exponential: $k_{\mathbf{SE}}(x, x') = \sigma^2 \exp\left(-\frac{(x-x')^2}{2l^2}\right)$
- Periodic: $k_{\mathbf{Per}}(x, x') = \sigma^2 \exp\left(-\frac{2\sin(\pi|x-x'|/p)}{l^2}\right)$
- Polynomial: $k_{\mathbf{Pol}}(x, x') = (x^\top x' + c)^d$
- Matérn-1/2: $k_{\mathbf{M}}(x, x') = \sigma^2 \exp\left(-\frac{|x-x'|}{2l^2}\right)$





Lincoln et al. The Art of the Gaussian Process

Stationary Kernels

- A stationary kernel is a function of the difference of its inputs

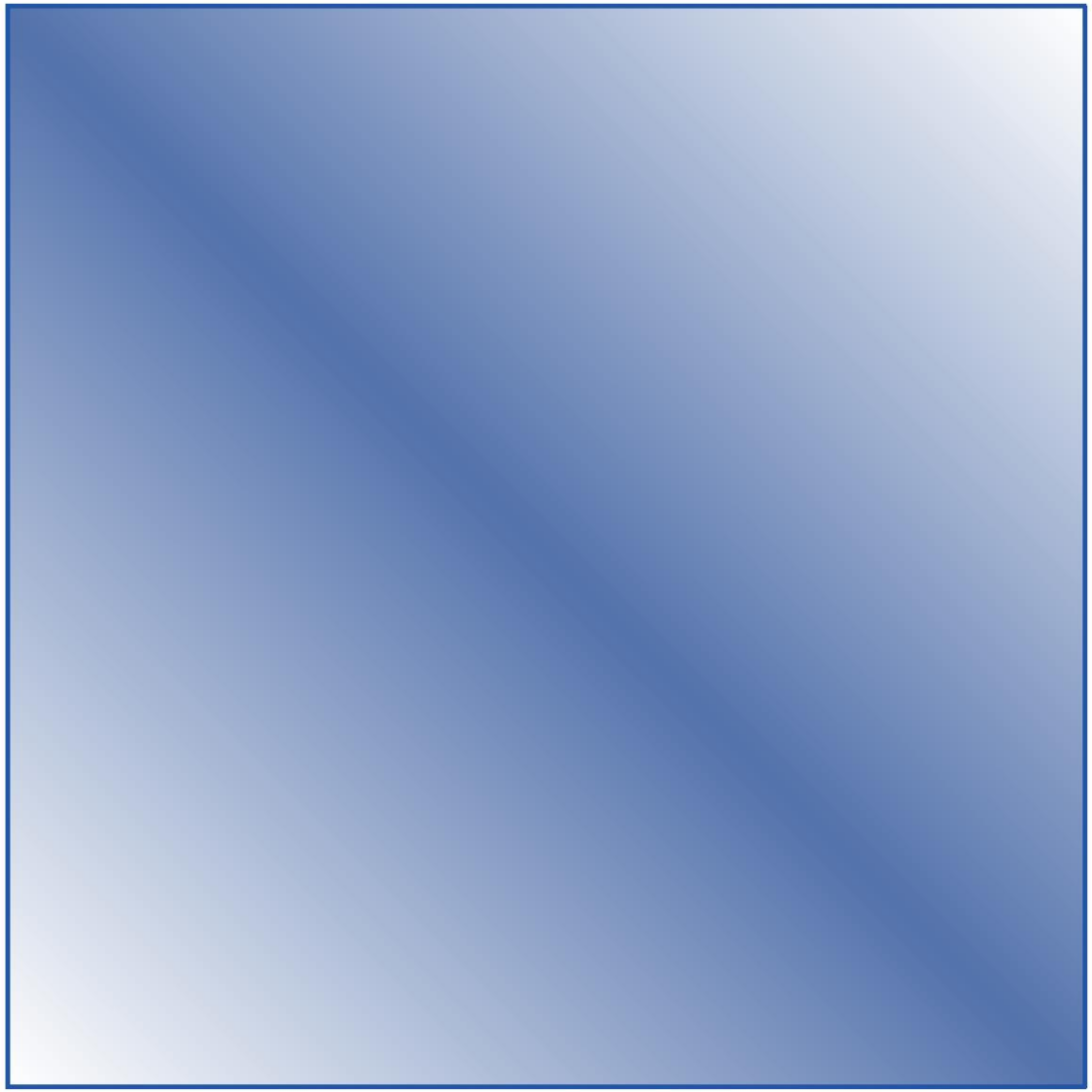
$$k_\theta(x, x') = k_\theta(x - x')$$

- Observation 1: GPs with stationary kernels are invariant under translations

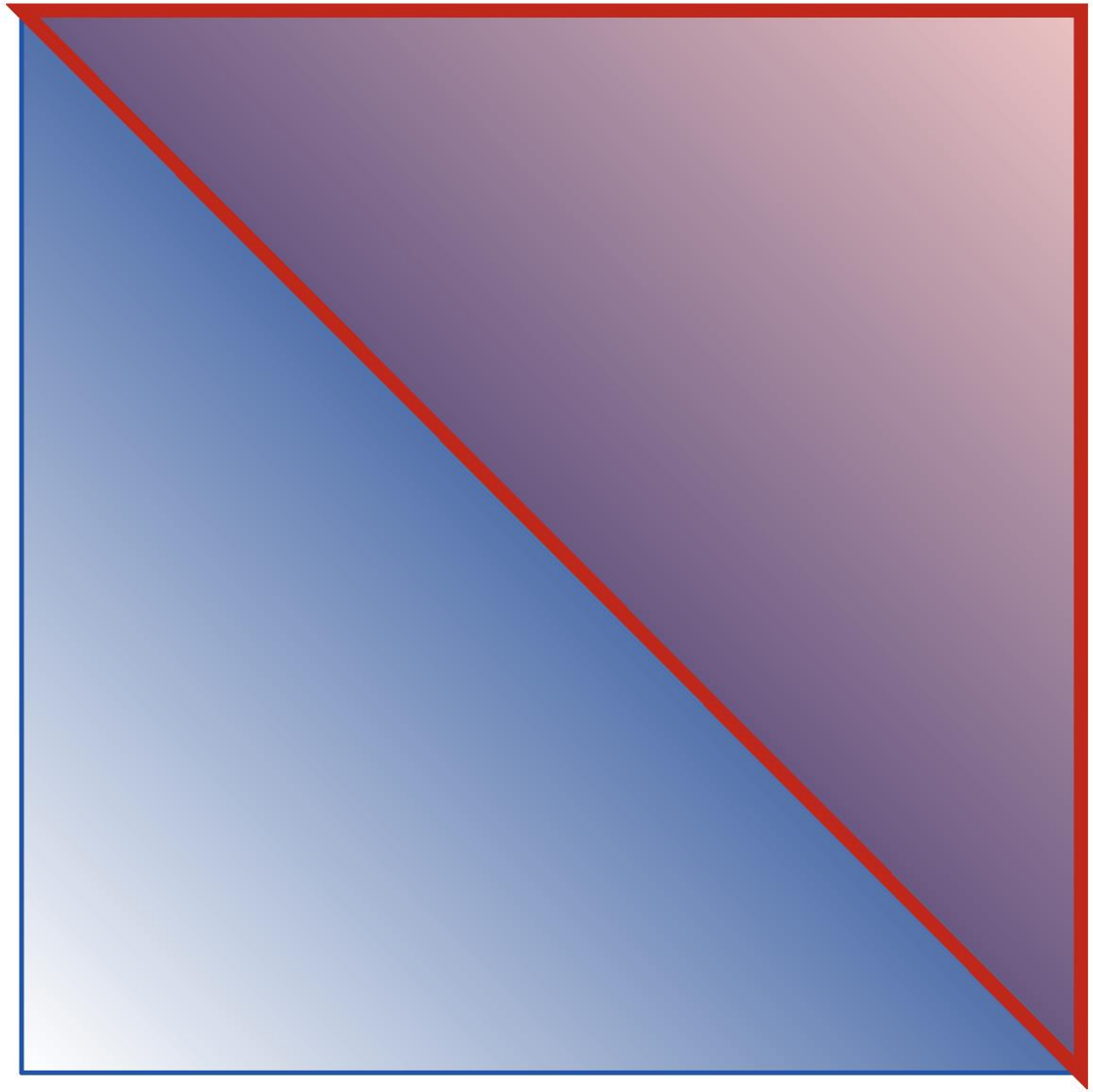
$$k_\theta(x, x') = k_\theta(x - \delta, x' - \delta), \qquad \forall \delta \in \mathbb{R}$$

- Observation 2: The assumption of stationarity dramatically reduces the design of a kernel → helpful modeling assumption (but be careful!)

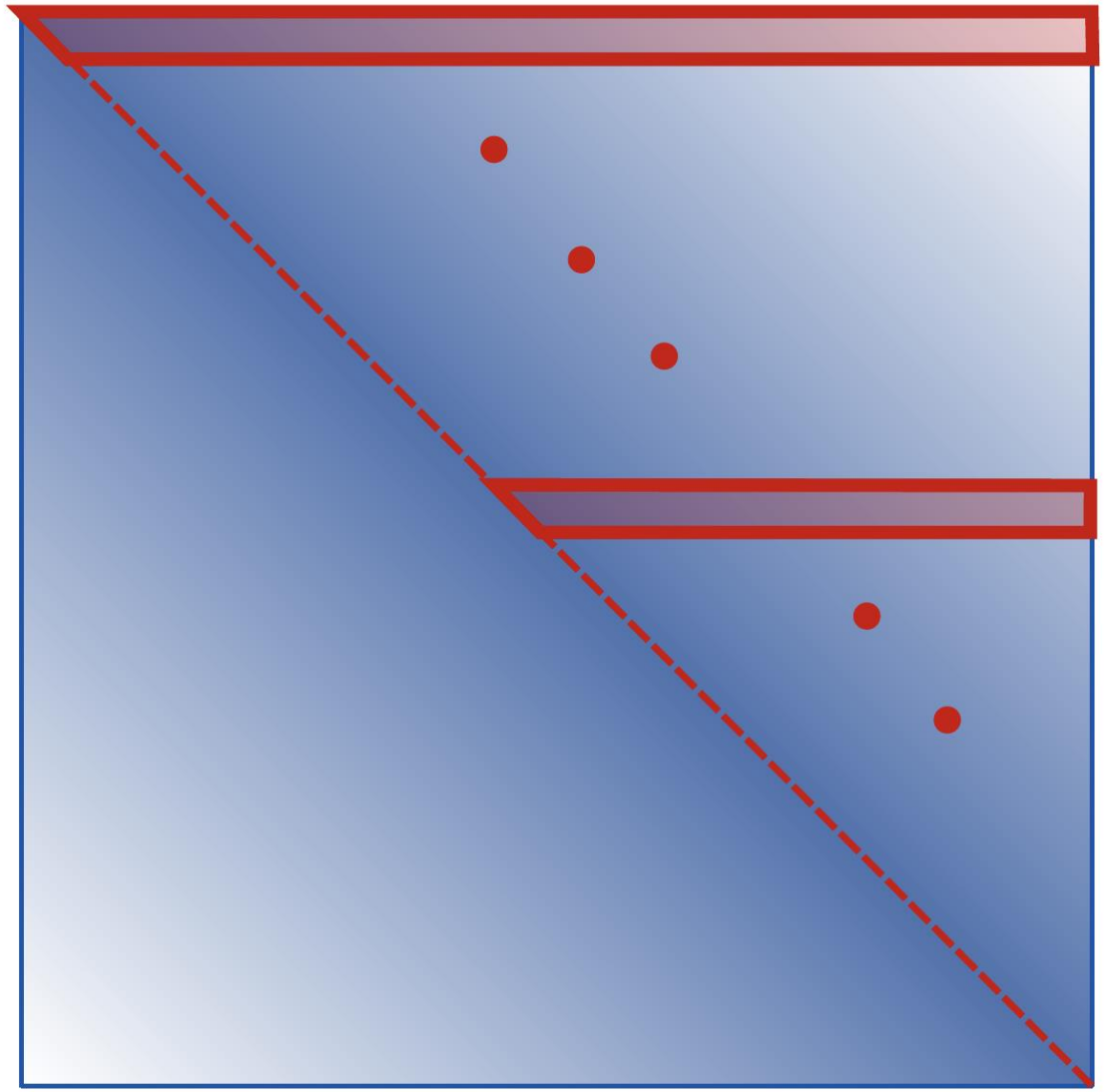
Stationary Kernels



Stationary Kernels



Stationary Kernels



Knowing one "slice" of the kernel allows me to populate the rest of the kernel, so dimensionality has been drastically reduced

Common Practice when Learning Hyperparameters in GPs

- Set initial hyperparameters using domain knowledge wherever possible
  - For example, set mean function using a physics-based model

- Otherwise:
  - Standardize input data (min-max scaling) and set lengthscales ~ 1
  - Standardize output data (mean-std scaling) and set function variance ~ 1

- Often useful: Set initial noise level high, even if you think your data has low noise to make the optimization surface easier to move in

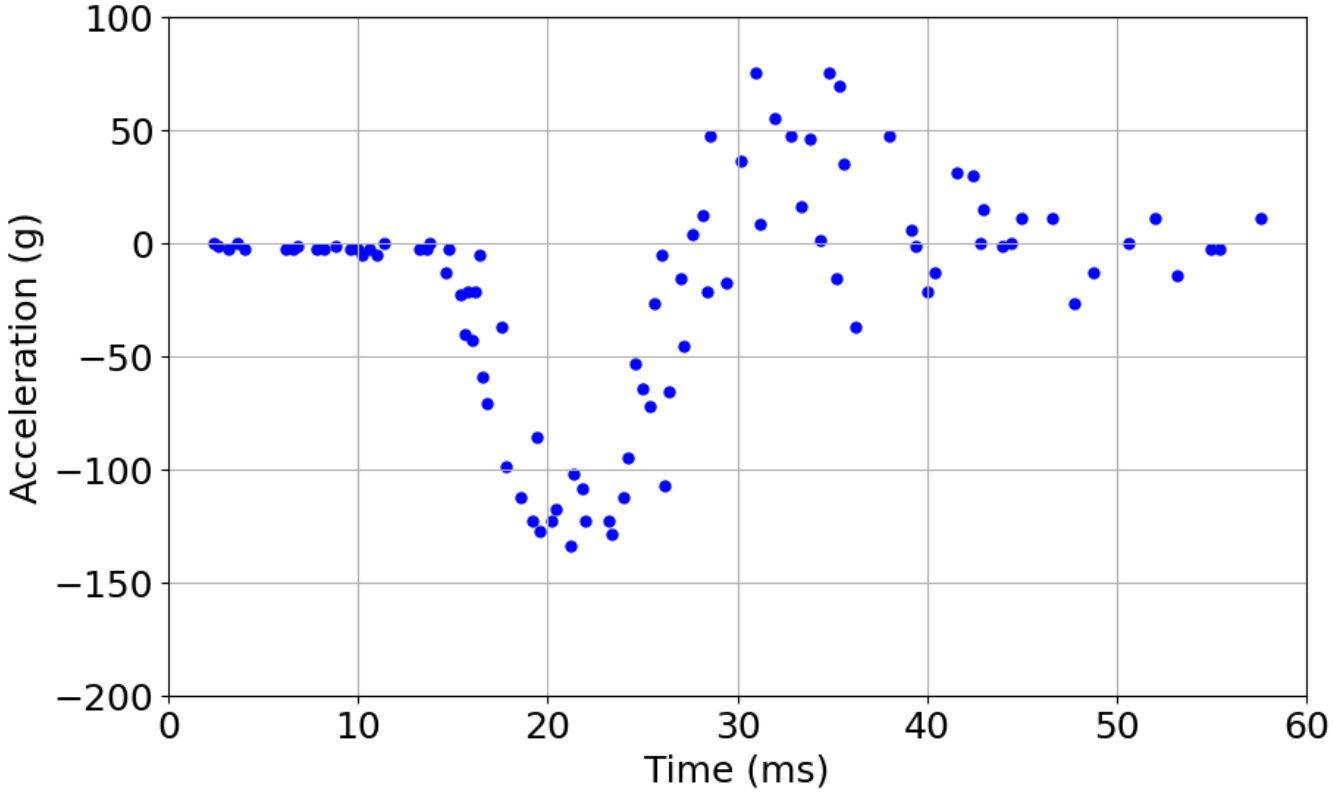
- Random restarts / other tricks to avoid local optima are advised when minimizing negative log-likelihood using gradient-based methods

Outline

- Introduction to Bayesian modeling
  - Bayes rule, importance of model averaging, epistemic uncertainty
- The function-space view
  - Gaussian processes, mean and kernel functions, inference
- Deep dive into the kernel
  - Importance of kernel, stationary vs. non-stationary kernels, learning hyperparameters
- Beyond traditional Gaussian processes
  - Heteroskedastic noise, sparse approximations, SAAS, deep kernel learning
- Efficient software packages
  - Quick overview of options, GPyTorch for flexibility and scalability

Heteroskedastic Noise

- **Challenge:** Traditional GP regression assumes a Gaussian likelihood with constant variance $y_i = f(x_i) + \epsilon_i$, $\epsilon_i \sim \mathcal{N}(0, \sigma_y^2)$ where σ_y^2 is independent of x_i, but that is not always a good assumption



Heteroskedastic Noise

- **Challenge:** Traditional GP regression assumes a Gaussian likelihood with constant variance $y_i = f(x_i) + \epsilon_i$, $\epsilon_i \sim \mathcal{N}(0, \sigma_y^2)$ where σ_y^2 is independent of x_i, but that is not always a good assumption

- **Goal:** Account for input dependence on the noise distribution

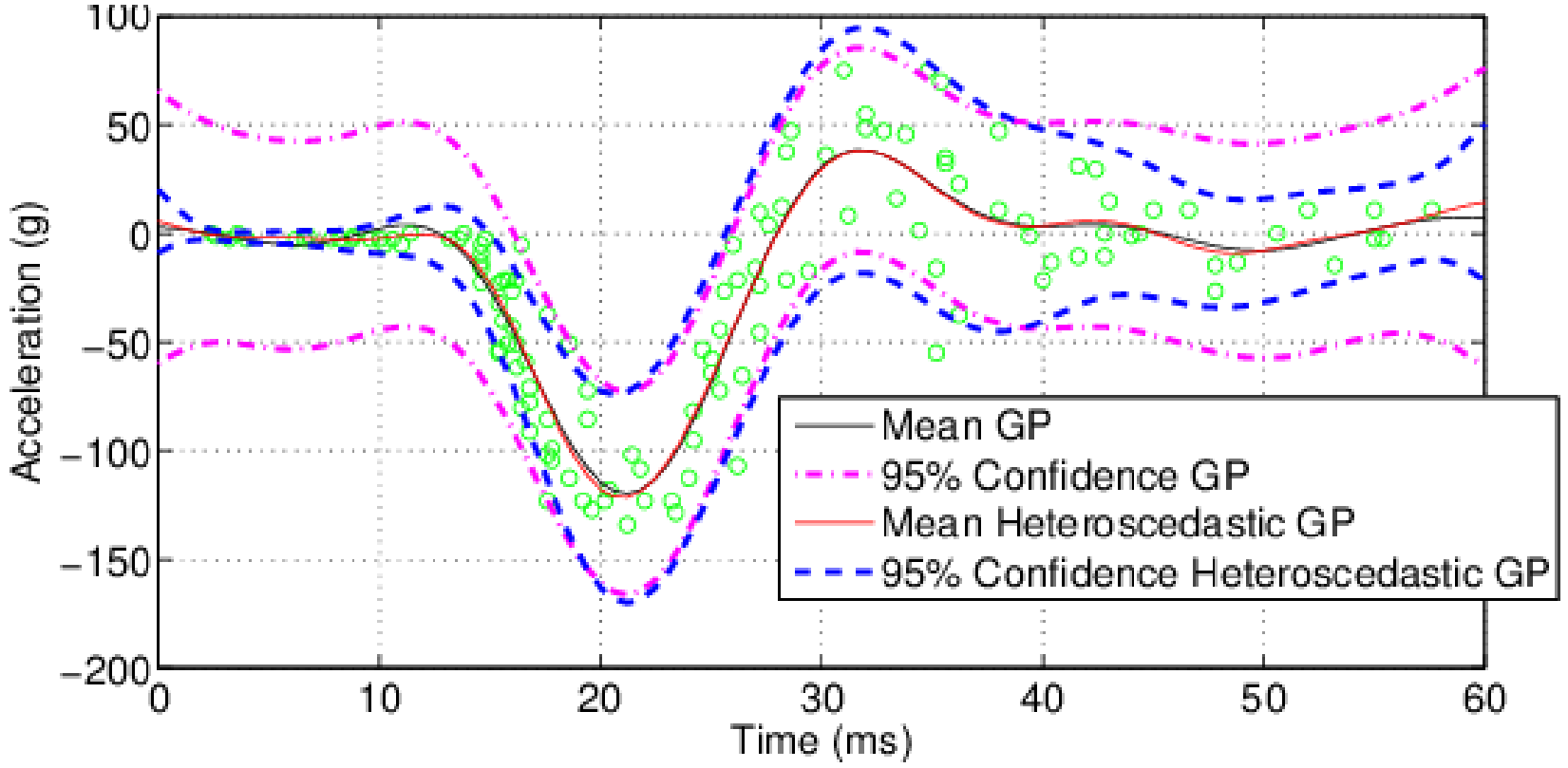
- **Idea:** Assume the noise $\epsilon \sim \mathcal{N}(0, e^{z(x)})$ where $z(x)$ (logarithm of noise) has an independent GP prior with a possibly different kernel
  - The predictive distribution is no longer Gaussian, as we need to marginalize:

$$p(y_*|x_*, \mathcal{D}) = \int\int p(y_*|x_*, \boldsymbol{z}, z_*, \mathcal{D})\, p(\boldsymbol{z}, z_*|x_*, \mathcal{D}) d\boldsymbol{z} dz_*$$

  - Need some type of approximation to integral (e.g., MCMC or Laplace approximation)

Heteroskedastic Noise

[Result using "most likely" method in *]



*Kersting et al., Most Likely Heteroscedastic Gaussian Process Regression, ICML, 2017

**Goldberg et al., Regression with input-dependent noise: A Gaussian process treatment, NeurIPS, 1997

Sparse Approximations

- **Challenge:** Despite flexibility of GP models, the marginal likelihood and predictive distribution require evaluation of $|\boldsymbol{K}_f + \sigma_y^2 \boldsymbol{I}|$ and $(\boldsymbol{K}_f + \sigma_y^2 \boldsymbol{I})^{-1}$

$$\overbrace{\log p(\boldsymbol{y}|\boldsymbol{X})}^{\text{marg. likelihood}} = -\frac{1}{2}\log|\boldsymbol{K}_f + \sigma_y^2\boldsymbol{I}| - \frac{1}{2}\boldsymbol{y}^\top(\boldsymbol{K}_f + \sigma_y^2\boldsymbol{I})^{-1}\boldsymbol{y} - \frac{N}{2}\log(2\pi),$$

$$\underbrace{p(y_*|\boldsymbol{x}_*, \boldsymbol{y}, \boldsymbol{X})}_{\text{predictive dist.}} = \mathcal{N}\left(y_*|\boldsymbol{k}_{f*}^\top(\boldsymbol{K}_f + \sigma_y^2\boldsymbol{I})^{-1}\boldsymbol{y}, k_{**} - \boldsymbol{k}_{f*}^\top(\boldsymbol{K}_f + \sigma_y^2\boldsymbol{I})^{-1}\boldsymbol{k}_{f*} + \sigma_y^2\right).$$

→ Naive computation scales with $\mathcal{O}(N^3)$.
→ Storage scales with $\mathcal{O}(N^2)$.

Lincoln et al. The Art of the Gaussian Process

Sparse Approximations

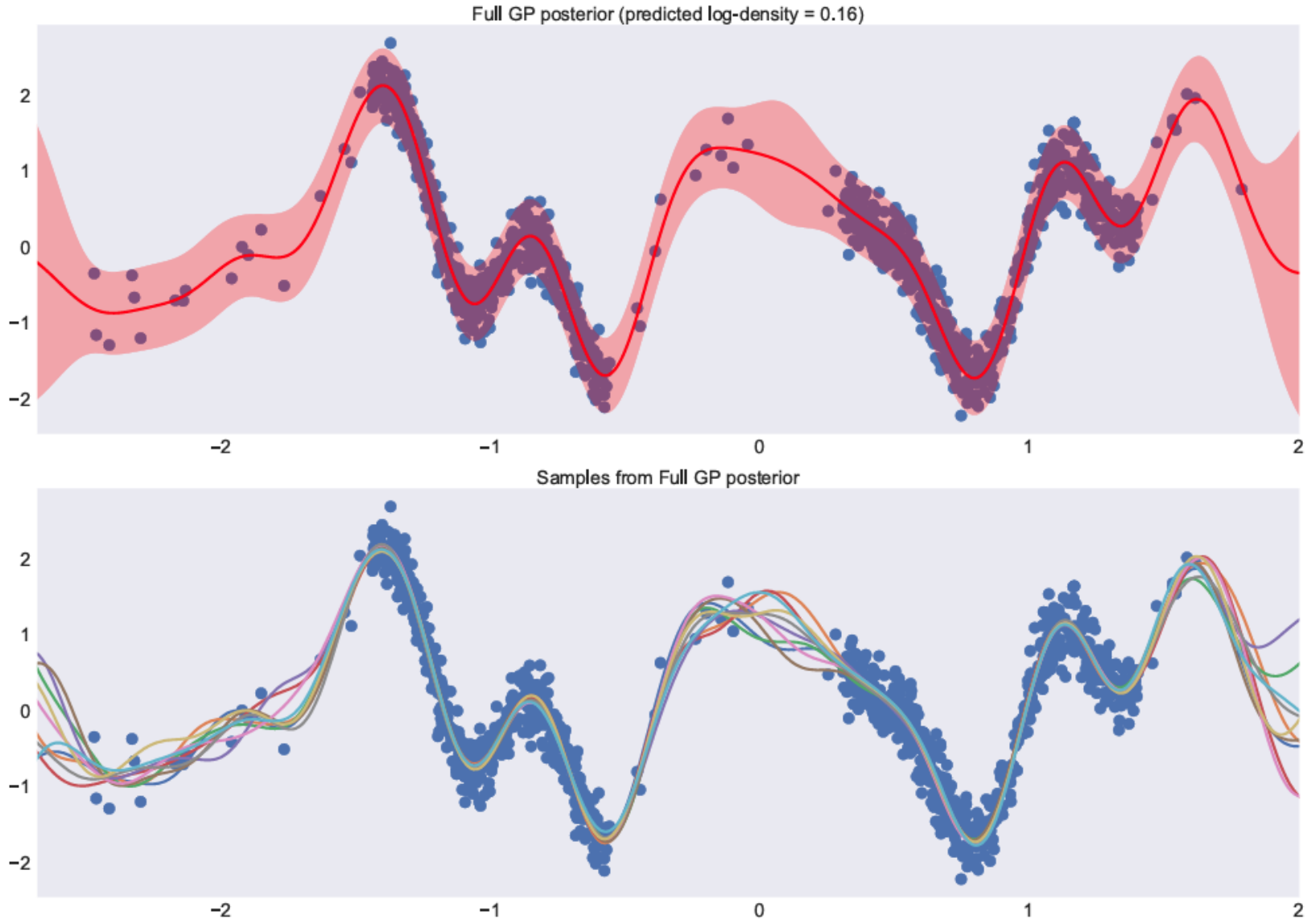
- **Challenge:** Despite flexibility of GP models, the marginal likelihood and predictive distribution require evaluation of $|\boldsymbol{K}_f + \sigma_y^2 \boldsymbol{I}|$ and $(\boldsymbol{K}_f + \sigma_y^2 \boldsymbol{I})^{-1}$
  - Naïve computation of matrix inverse $\mathcal{O}(N^3)$
  - Storage scales as $\mathcal{O}(N^2)$

- **Goal:** Mitigate the influence of number of data points N on cost

- **Idea:** Optimally select a smaller set of points to summarize the data
  - Introduce inducing variables $\boldsymbol{u} \in \mathbb{R}^M$ at pseduo-inputs $\{\boldsymbol{z}_j\}_{j=1}^M \in \mathbb{R}^d$ to approximate the posterior → optimize using variational bound
  - Computation and storage scale as $\mathcal{O}(NM^2)$ and $\mathcal{O}(NM)$

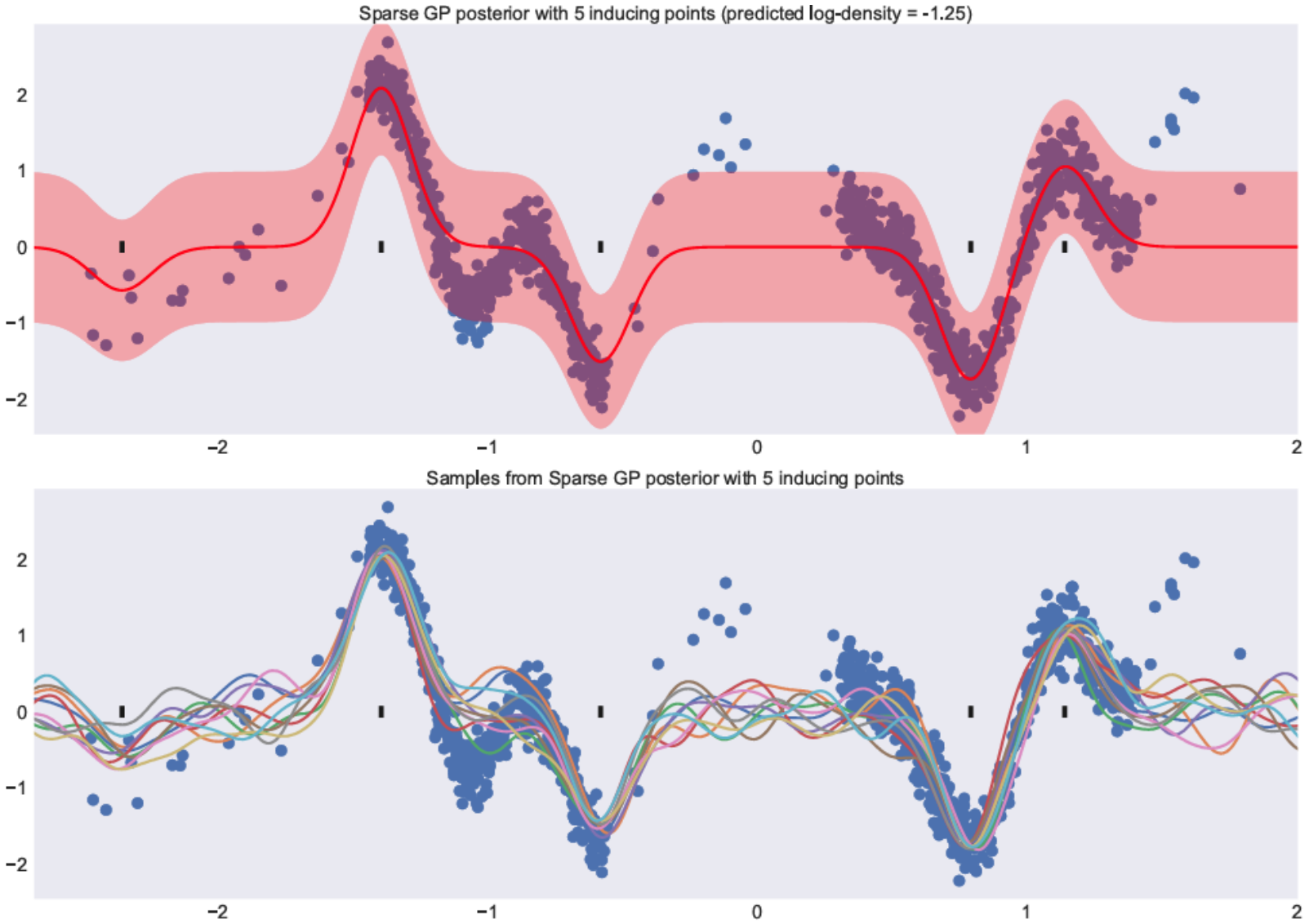
Lincoln et al. The Art of the Gaussian Process

Sparse Approximations (1083 observations, full GP)



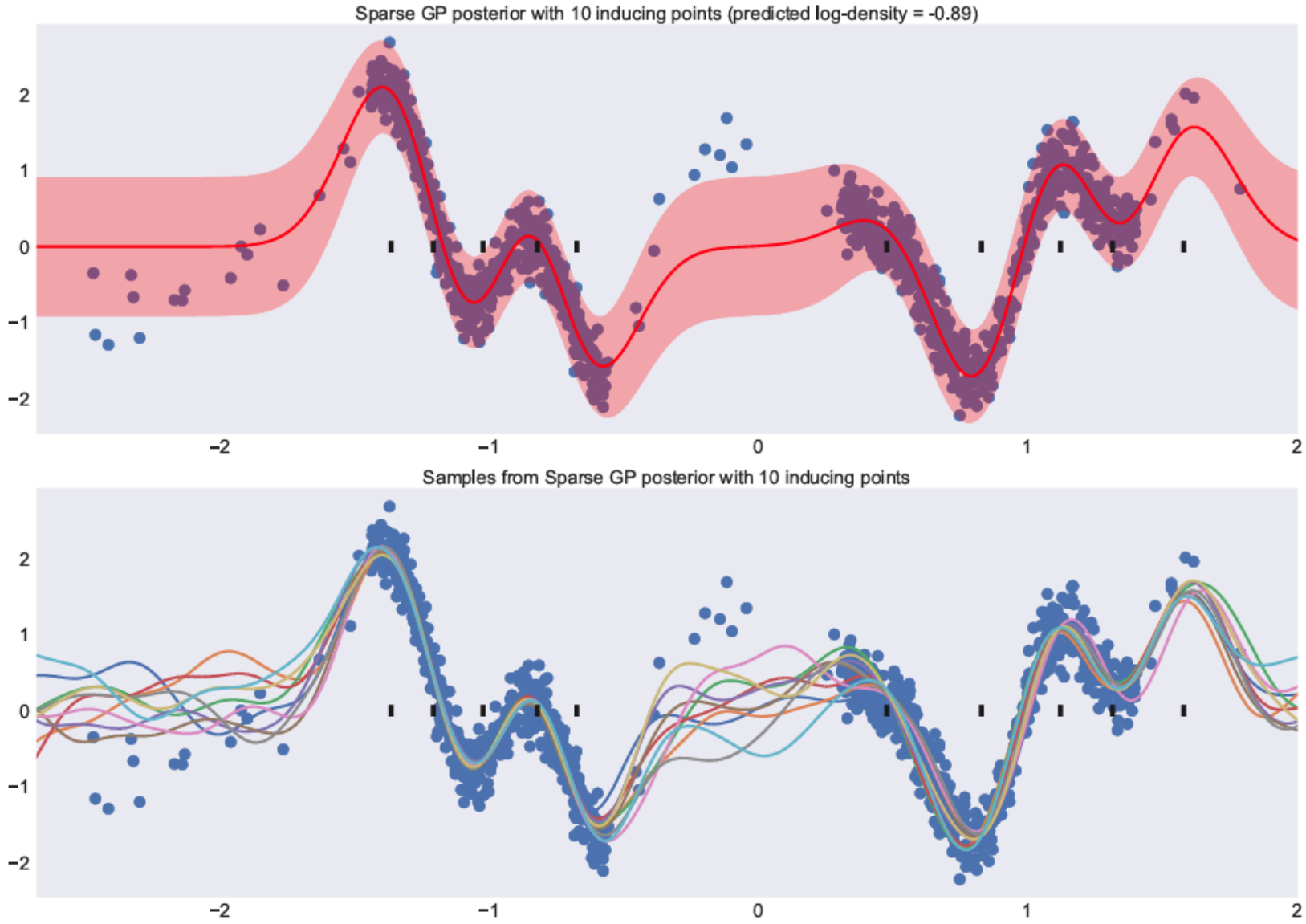
Lincoln et al. The Art of the Gaussian Process

Sparse Approximations (1083 observations, 5 inducing points)



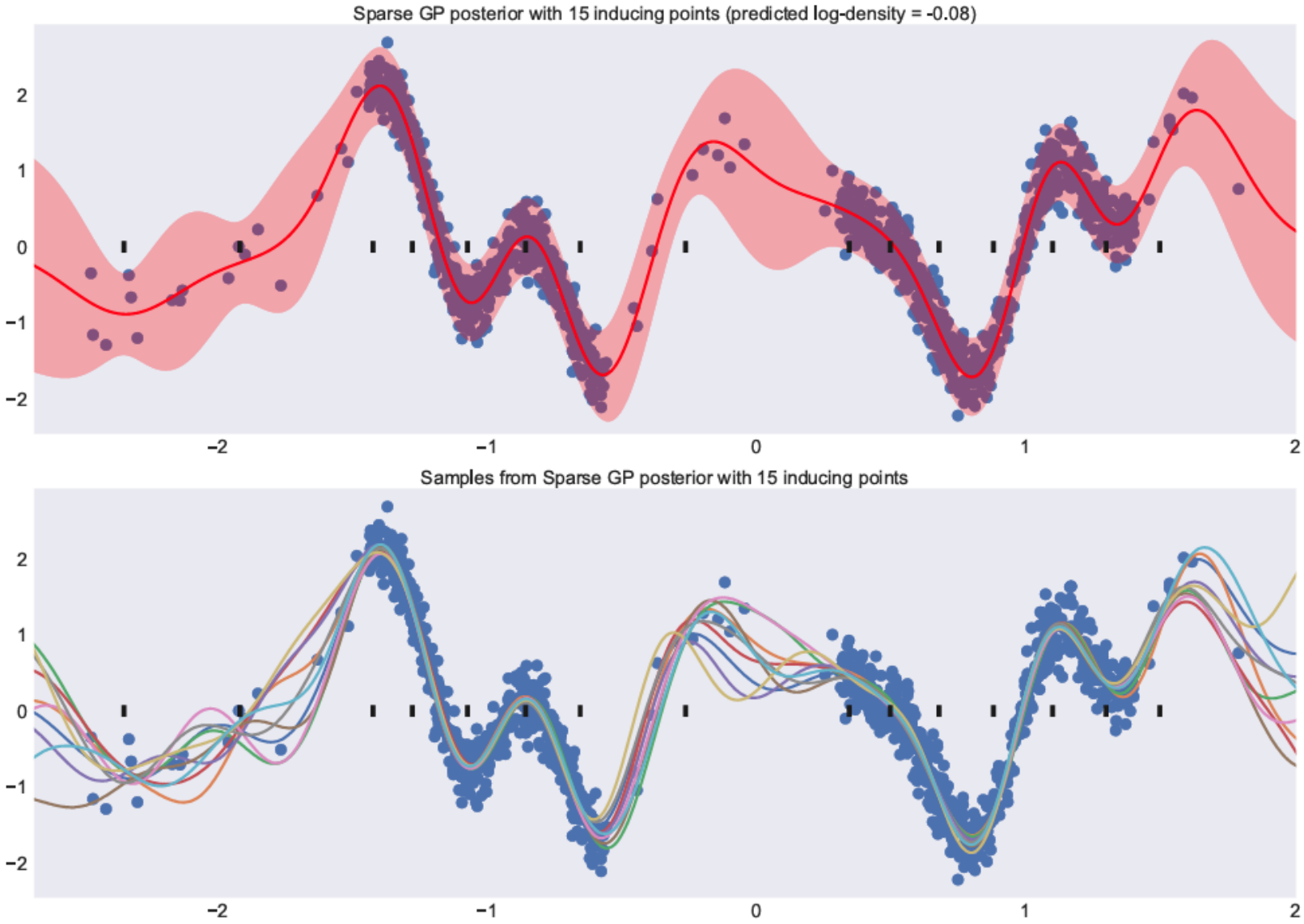
Lincoln et al. The Art of the Gaussian Process

Sparse Approximations (1083 observations, 10 inducing points)



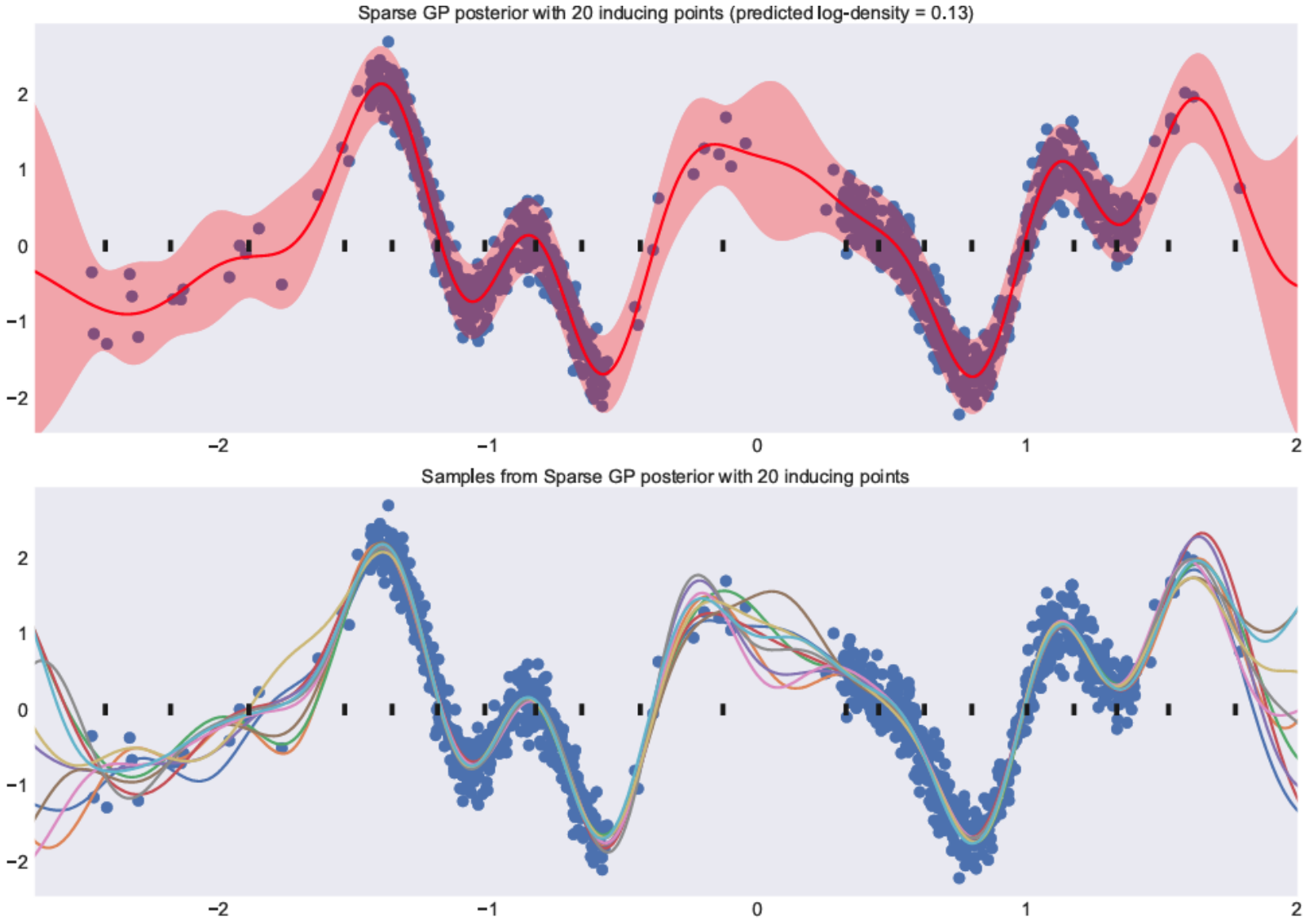
Lincoln et al. The Art of the Gaussian Process

Sparse Approximations (1083 observations, 15 inducing points)



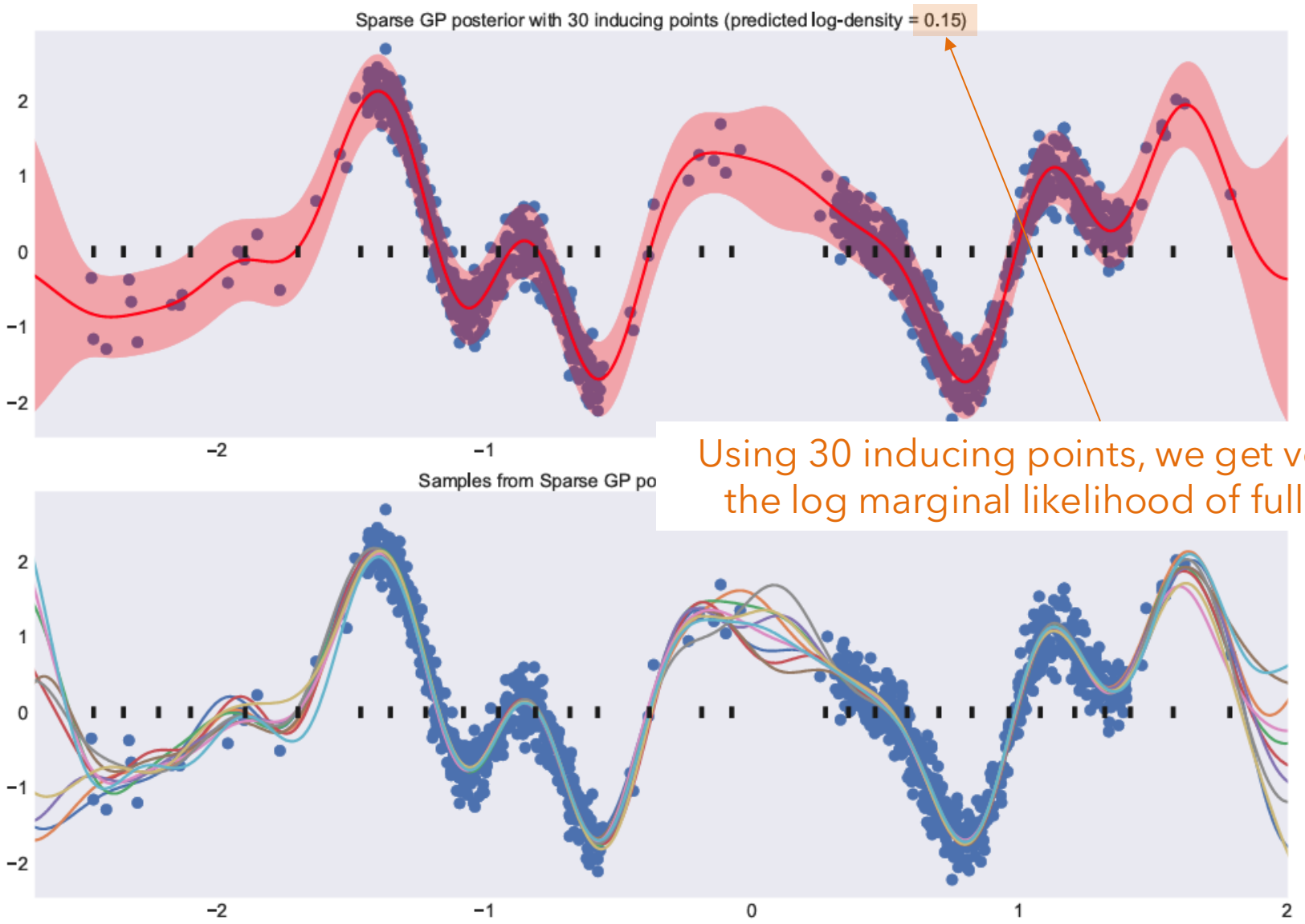
Lincoln et al. The Art of the Gaussian Process

Sparse Approximations (1083 observations, 20 inducing points)



Lincoln et al. The Art of the Gaussian Process

Sparse Approximations (1083 observations, 30 inducing points)



Using 30 inducing points, we get very close to the log marginal likelihood of full GP (0.16)

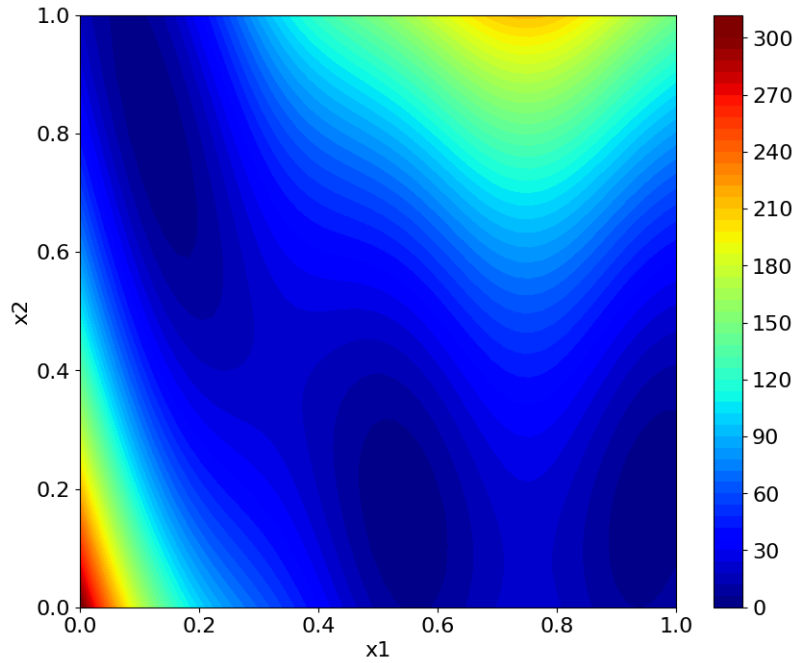
Lincoln et al. The Art of the Gaussian Process

Fully Bayesian GPs for High-Dimensional Problems

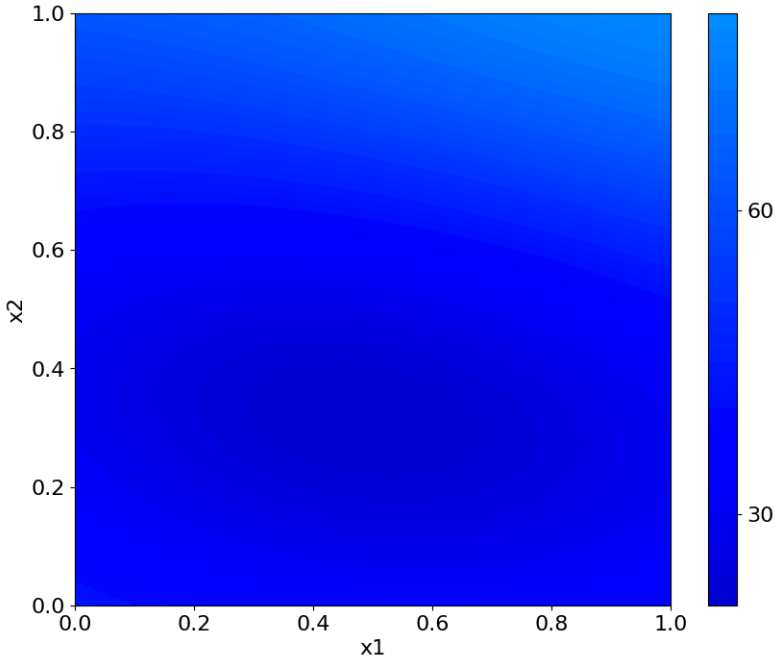
- **Challenge:** When number of input dimensions $\boldsymbol{x} \in \mathbb{R}^D$ is large, most test points are "far" from training points & kernel has many hyperparameters (easy to overfit)
  - Commonly referred to as "curse of dimensionality"

(30 random training data points)

Projection of $D = 30$ dimensional function onto 2 sensitive dimensions



GP prediction trained using traditional maximum marginal likelihood method



Fully Bayesian GPs for High-Dimensional Problems

- **Challenge:** When number of input dimensions $\boldsymbol{x} \in \mathbb{R}^D$ is large, most test points are "far" from training points & kernel has many hyperparameters (easy to overfit)
  - Commonly referred to as "curse of dimensionality"

- **Goal:** Achieve improved predictive performance in many dimensions by assuming a hierarchy of relevance in the dimensions $\{x_1 \ldots, x_D\}$
  - For example, $\{x_5, x_{12}\}$ important, $\{x_2, x_6, x_{24}\}$ semi-important, and rest are marginally important, we just do not know which dimensions are in each bin

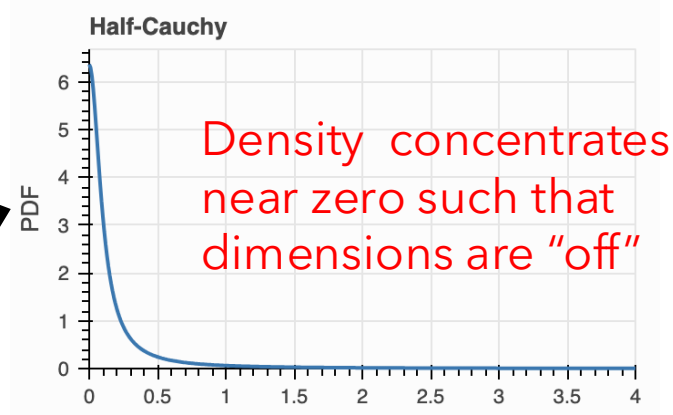
- **Idea:** Introduce a prior on the kernel hyperparameters to reflect this conjectured sparse model structure
  - Easiest to assume dimensions are irrelevant by default and, as we get more data, further dimensions can be "unlocked" to support a richer class of functions

Fully Bayesian GPs for High-Dimensional Problems

[Sparse Axis-Aligned Subspace Prior]*

- The (inverse squared) lengthscales $\rho_i = l_i^{-2}$ control sparsity

- SAAS prior:
  - Kernel variance: $\sigma_f^2 \sim \text{LogNormal}(0, 10^2)$
  - Global shrinkage: $\tau \sim \text{HalfCauchy}(\alpha)$
  - Length-scales: $\rho_i \sim \text{HalfCauchy}(\tau), \;\; i = 1, \ldots, D$
  - Function values: $\mathbf{f} \sim \text{Normal}(\mathbf{0}, K_{\mathbf{XX}}^{\psi})$ where $\psi = \{\rho_1, \ldots, \rho_D, \sigma_f^2\}$
  - Observations: $\mathbf{y} \sim \text{Normal}(\mathbf{f}, \sigma^2 I_n)$

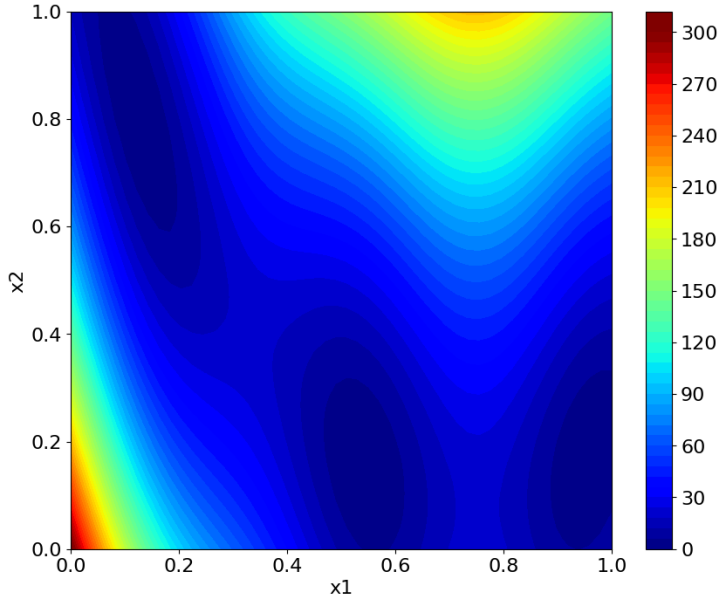


- Use Hamiltonian Monte Carlo (specifically NUTS) to generate samples from the posterior $p(\tau, \psi \mid \mathbf{y}, \mathbf{X}) \propto p(\mathbf{y} \mid \mathbf{X}, \psi) p(\psi \mid \tau) p(\tau)$

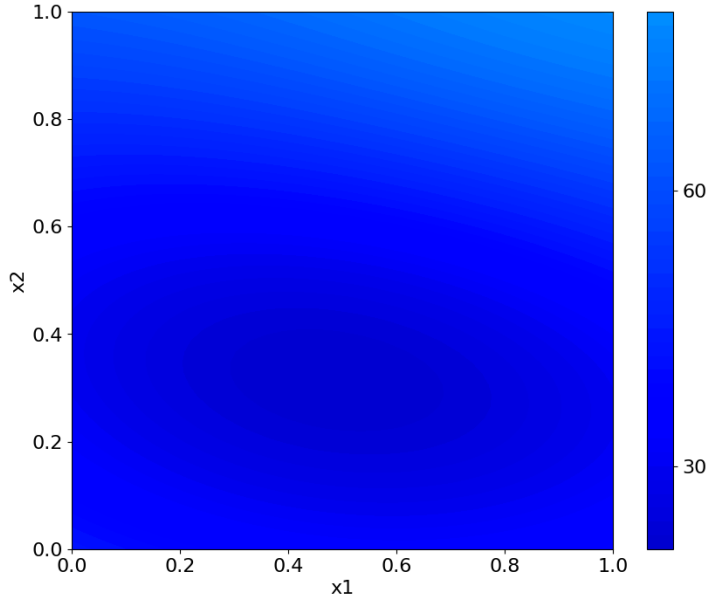
*[Eriksson and Jankowiak, *UAI*, 2021]

Comparison between traditional GP and SAAS-GP

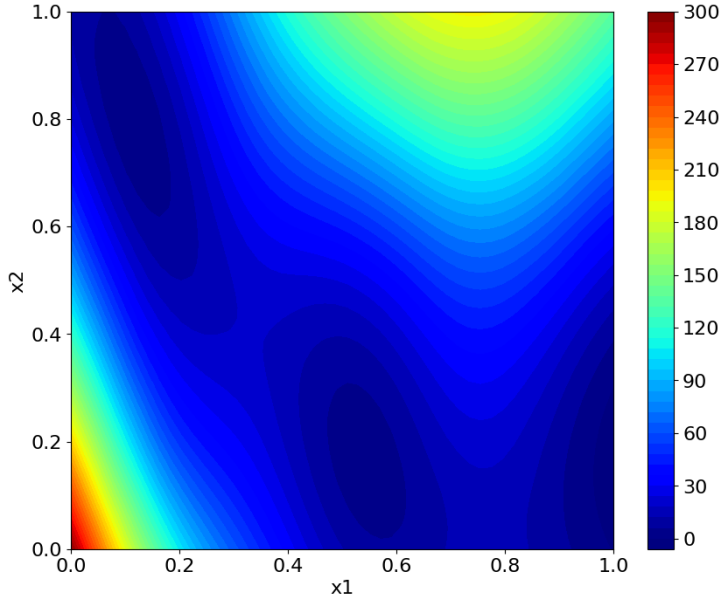
Exact Function



Traditional GP

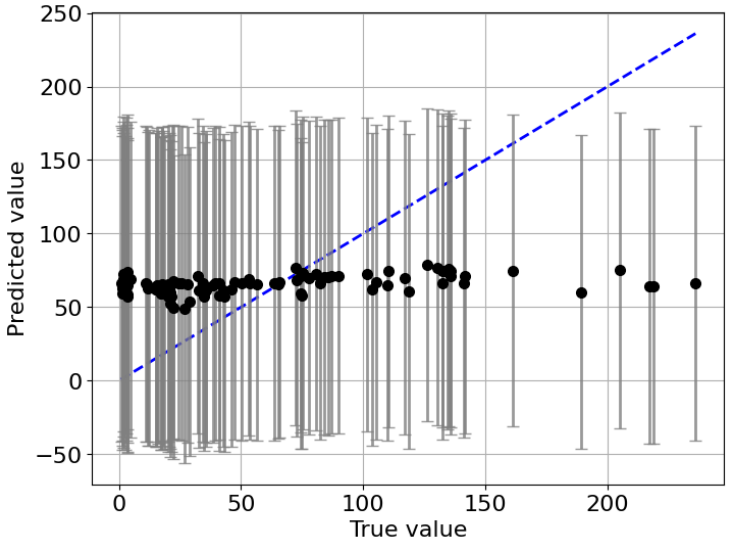


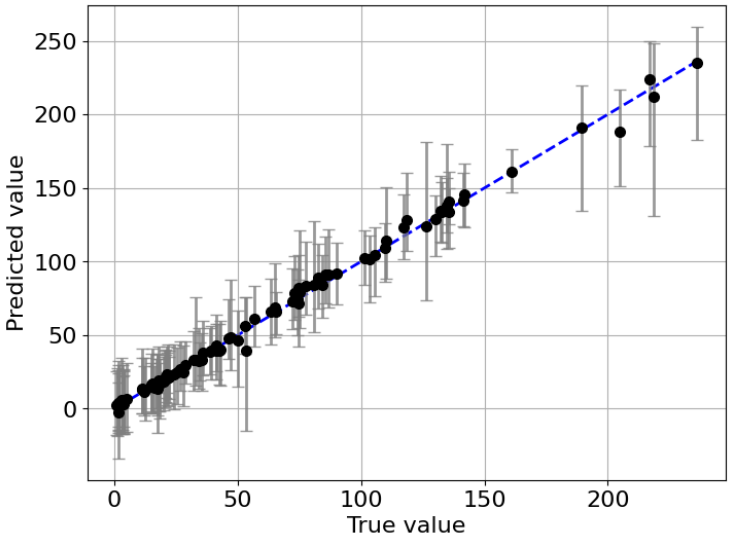
SAAS-GP



- Projection of $D = 30$ function onto 2 sensitive dimensions

- Parity plot on 100 test points





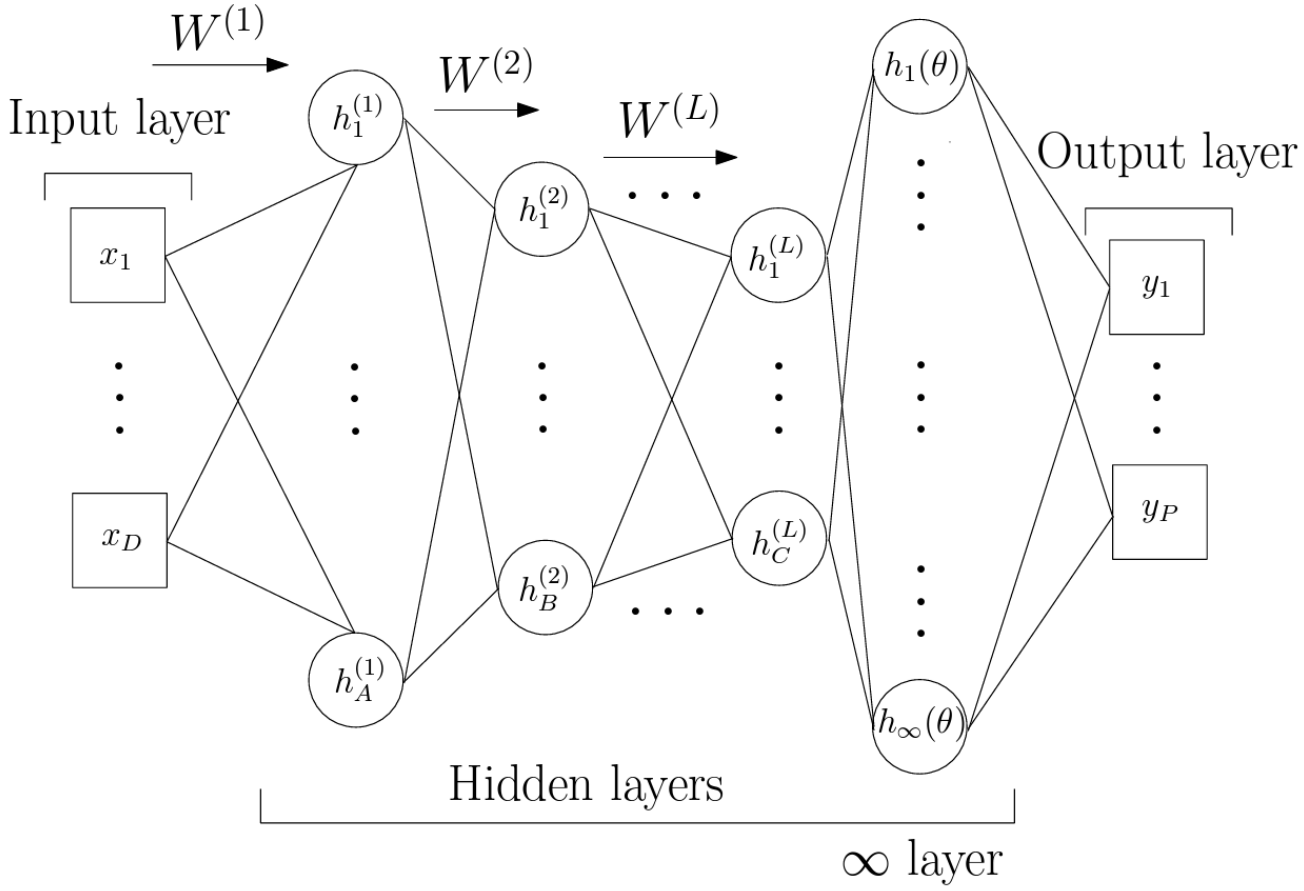
Deep Kernel Learning (DKL)

- **Challenge:** A central critique of GP regression is that it does not learn representations of the data (only adjusts degree of smoothing), which limits their applicability to high-dimensional and structured inputs (e.g., images)

- **Goal:** Combine uncertainty representation advantages of GPs with representation learning power of large parametric models like neural networks

- **Idea:** Transform inputs $\boldsymbol{x}$ using a parametrized nonlinear mapping $g(\boldsymbol{x}, \boldsymbol{w})$ and feed these intermediate values as inputs to a standard kernel
  - Generates an effective kernel $k_{\mathrm{DKL}}(\boldsymbol{x}, \boldsymbol{x}'|\theta, \boldsymbol{w}) = k(g(\boldsymbol{x}, \boldsymbol{w}), g(\boldsymbol{x}', \boldsymbol{w})|\theta)$
  - Deep kernel hyperparameters $\{\theta, \boldsymbol{w}\}$ trained *jointly* including weights of network

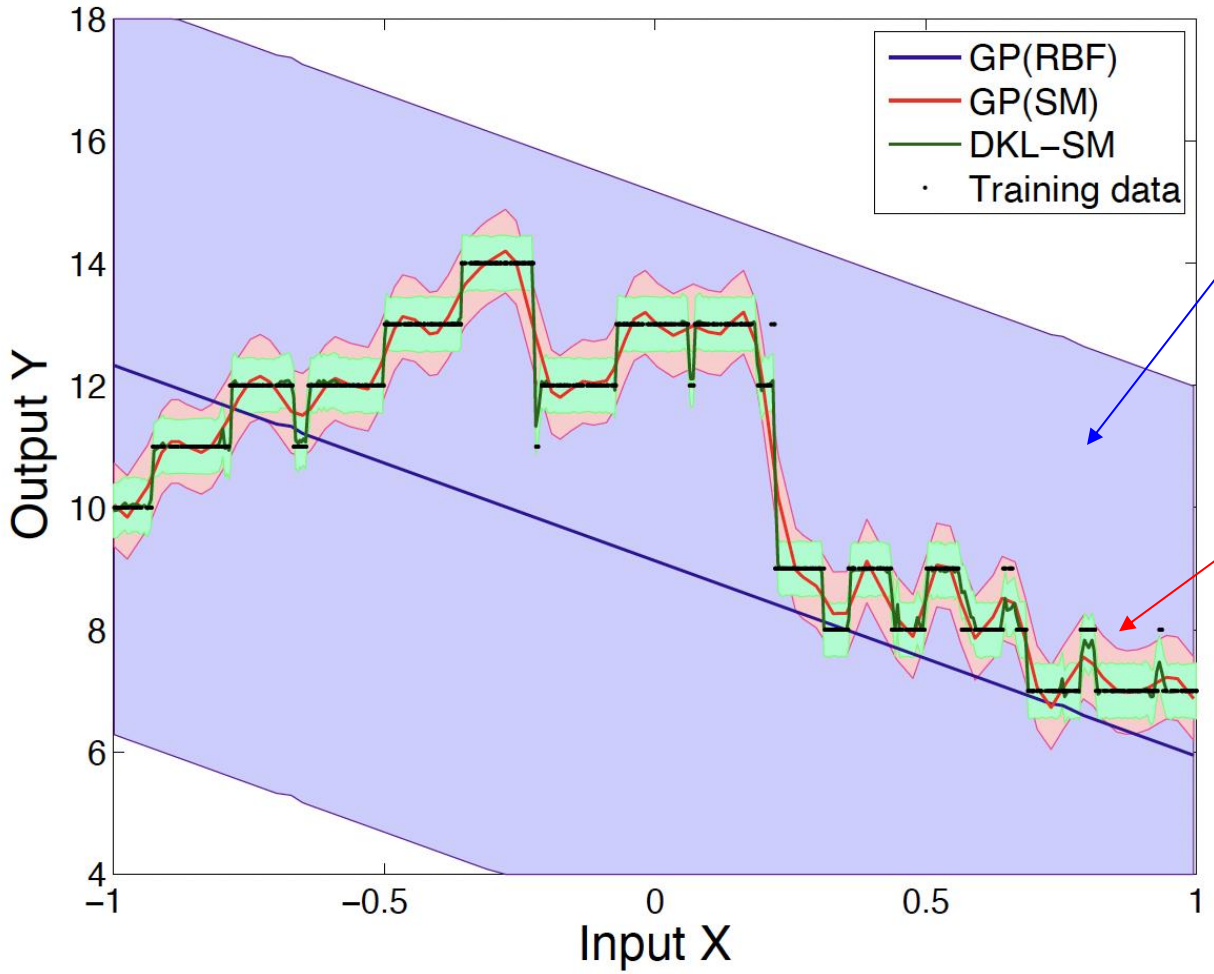
Deep Kernel Learning (DKL)



Base kernel hyperparameters θ and deep network hyperparameters $\boldsymbol{w}$ are jointly trained through the marginal likelihood objective

Deep Kernel Learning. Wilson, Hu, Salakhutdinov, and Xing, AISTATS, 2016

DKL can handle non-stationarity in data!



SE (or RBF) kernel struggles to fit the highly discontinuous data

Spectral mixture (SM) kernel does a better job than SE but still struggles with the sharp changes and so has overestimated uncertainty that varies too smoothly

DKL accurately encodes the discontinuities of the function and provides good uncertainty quantification throughout the domain

Deep Kernel Learning. Wilson, Hu, Salakhutdinov, and Xing, AISTATS, 2016

However…DKL has tendency to overfit, so must be careful!

The Promises and Pitfalls of Deep Kernel Learning

Sebastian W. Ober[1] **Carl E. Rasmussen**[1,2] **Mark van der Wilk**[3]

[1]Department of Engineering, University of Cambridge, Cambridge, United Kingdom
[2]Secondmind.ai, Cambridge, United Kingdom
[3]Department of Computing, Imperial College London, London, United Kingdom

- Observed the marginal likelihood overfits by over-correlating datapoints, as it tries to correlate all data, not just points that should be correlated
- Stochastic mini-batching can mitigate overfitting and helps DKL in practice
- Fully Bayesian treatment of hyperparameters helps DKL achieve “best of both worlds”

Outline

- Introduction to Bayesian modeling
  - Bayes rule, importance of model averaging, epistemic uncertainty
- The function-space view
  - Gaussian processes, mean and kernel functions, inference
- Deep dive into the kernel
  - Importance of kernel, stationary vs. non-stationary kernels, learning hyperparameters
- Beyond traditional Gaussian processes
  - Heteroskedastic noise, sparse approximations, SAAS, deep kernel learning
- Efficient software packages
  - Quick overview of options, GPyTorch for flexibility and scalability

Several Software Packages for GPs in Python

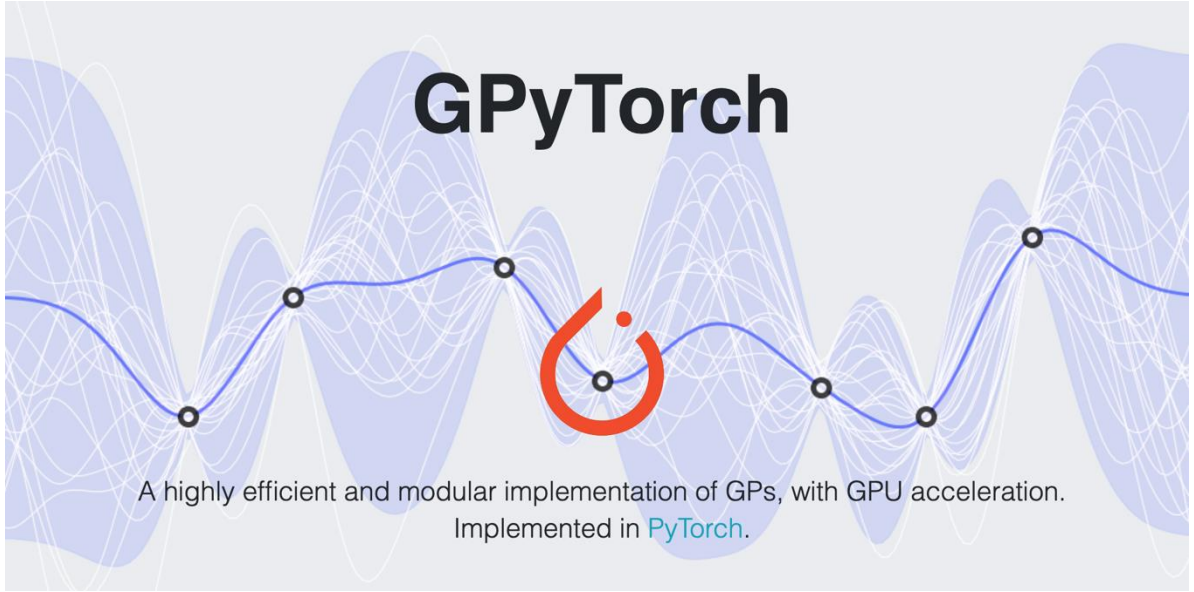
[not comprehensive]

- GPy
  - Good for educational purposes, quick GP prototyping; less suited for large datasets
- Scikit-learn
  - Good for basic GP regression and integration with other ML models; much less flexible
- George
  - Efficient for large datasets using special factorization of kernel matrix; less comprehensive
- GPflow
  - Scalable GP models via implementation in TensorFlow; can be complex to setup
- GPyTorch
  - Great for large datasets & deep learning integration (built on PyTorch); bit of a learning curve
- Many Bayesian optimization packages with "custom" GP regression tools
  - BoTorch, pyGPGO, Spearmint → typically have default prior settings on hyperparameters

GPyTorch: Flexible & Scalable Gaussian Processes

- I personally recommend using GPyTorch, especially if you have prior experience with the PyTorch ecosystem (tensors, batching, GPU support)
  - Relatively easy to "extend" (e.g., define new kernels) – we will see in code review

- Possible to run **exact** GPs on millions of points in minutes, but also provides support for approximate approaches and many advanced features we discussed
  - Important idea is the use of conjugate gradient methods that can be run on GPU to greatly accelerate the inversion of the kernel matrix



GPyTorch: Flexible & Scalable Gaussian Processes

Setup training data

```
# Training data is 100 points in [0,1] inclusive regularly spaced
train_x = torch.linspace(0, 1, 100)
# True function is sin(2*pi*x) with Gaussian noise
train_y = torch.sin(train_x * (2 * math.pi)) + torch.randn(train_x.size()) * math.sqrt(0.0
```

Define model class with desired mean and kernel functions (object oriented)

```
# We will use the simplest form of GP model, exact inference
class ExactGPModel(gpytorch.models.ExactGP):
    def __init__(self, train_x, train_y, likelihood):
        super(ExactGPModel, self).__init__(train_x, train_y, likelihood)
        self.mean_module = gpytorch.means.ConstantMean()
        self.covar_module = gpytorch.kernels.ScaleKernel(gpytorch.kernels.RBFKernel())

    def forward(self, x):
        mean_x = self.mean_module(x)
        covar_x = self.covar_module(x)
        return gpytorch.distributions.MultivariateNormal(mean_x, covar_x)

# initialize likelihood and model
likelihood = gpytorch.likelihoods.GaussianLikelihood()
model = ExactGPModel(train_x, train_y, likelihood)
```

GPyTorch: Flexible & Scalable Gaussian Processes

Call an optimizer to train the model (for some number of steps)

```
# Find optimal model hyperparameters
model.train()
likelihood.train()

# Use the adam optimizer
optimizer = torch.optim.Adam(model.parameters(), lr=0.1)  # Includes GaussianLikelihood pa

# "Loss" for GPs - the marginal log likelihood
mll = gpytorch.mlls.ExactMarginalLogLikelihood(likelihood, model)

for i in range(training_iter):
    # Zero gradients from previous iteration
    optimizer.zero_grad()
    # Output from model
    output = model(train_x)
    # Calc loss and backprop gradients
    loss = -mll(output, train_y)
    loss.backward()
    print('Iter %d/%d - Loss: %.3f   lengthscale: %.3f   noise: %.3f' % (
        i + 1, training_iter, loss.item(),
        model.covar_module.base_kernel.lengthscale.item(),
        model.likelihood.noise.item()
    ))
    optimizer.step()
```

All model parameters are optimized, which includes mean, kernel, and likelihood parameters (unless they are explicitly set to not trainable)

GPyTorch: Flexible & Scalable Gaussian Processes

Turn model into eval mode and evaluate model to get predictions

```
# Get into evaluation (predictive posterior) mode
model.eval()
likelihood.eval()

# Test points are regularly spaced along [0,1]
# Make predictions by feeding model through likelihood
with torch.no_grad(), gpytorch.settings.fast_pred_var():
    test_x = torch.linspace(0, 1, 51)
    observed_pred = likelihood(model(test_x))
```

Plot mean with confidence bounds calculated from predictions