Deconstructing Gaussian Processes
Bonus Part: Hamiltonian Monte Carlo

Vidhi Lalchand

Department of Physics
University of Cambridge

3rd December 2019

UNIVERSITY OF CAMBRIDGE
Cavendish Laboratory

Outline

1. Tackling the nomenclature
2. Gaussian Processes
3. Where Deep Learning falls short
4. Advances in Gaussian Process
5. Inference: Hamiltonian Monte Carlo

Non-probabilistic vs. Probabilistic Machine Learning

- Step 1: Data $D = \{x_i, y_i\}_{i=1}^{N}$ + Parametric model, eg: $y_i = w_1 x_i + b_1$, $\theta = \{w_1, b_1\}$.
- Step 2: Train the model $\rightarrow$ **'learn'** the parameters $\theta = \left\{\hat{w}_1, \hat{b}_1\right\}$.
- Step 3: Make predictions for unseen x_* by plugging in, $y_* = \hat{w}_1 x_* + \hat{b}_1$.

Training step: $\underset{\theta}{\textit{argmin}}\ \mathcal{L}(\theta)$

Parameters θ are fixed, unknown quantities

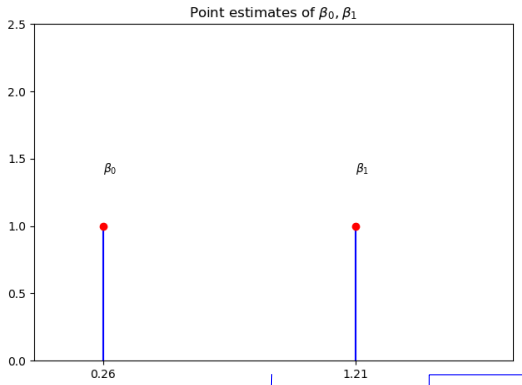
Non-probabilistic vs. Probabilistic Machine Learning

- Step 1: Data $(X, y) = \{x_i, y_i\}_{i=1}^{N}$ + Parametric model, eg: $y_i = w_1 x_i + b_1$, $\theta = \{w_1, b_1\}$.
- Step 2: Specify data likelihood $p(y|\theta)$ and prior $p(\theta)$
- Step 3: Inference step $\rightarrow$ **'learn'** the posterior distribution $p(\theta|X, y)$ over parameters $\theta = \left\{\hat{w_1}, \hat{b_1}\right\}$
- Step 4: Prediction step: $p(y_*|x_*, y) = \int p(y_*|x_*, y, \theta)p(\theta|y)d\theta$

Training step: $p(\theta|y) = \dfrac{p(y|\theta)p(\theta)}{p(y)}$ where $p(y) = \int p(y|\theta)p(\theta)d\theta$

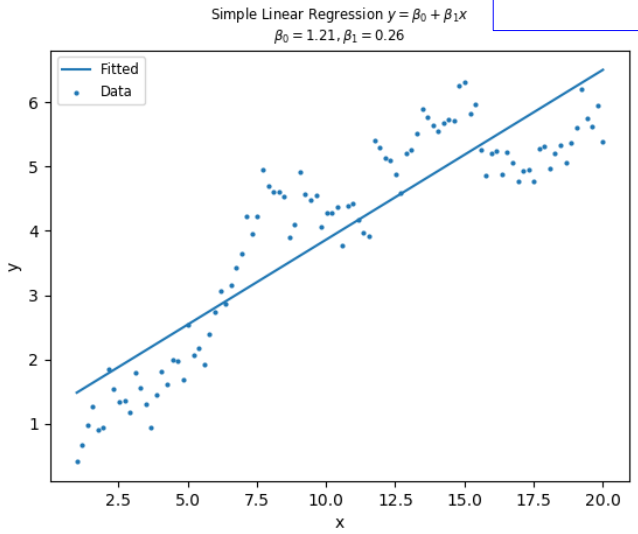
Predictions: Point estimates vs. Distributions

Conventional

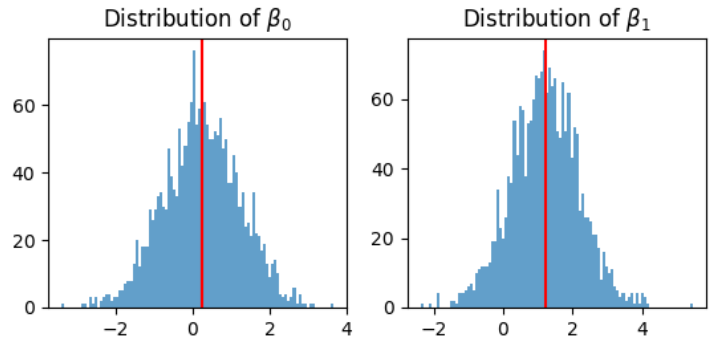


$\theta = (\beta_0, \beta_1)$

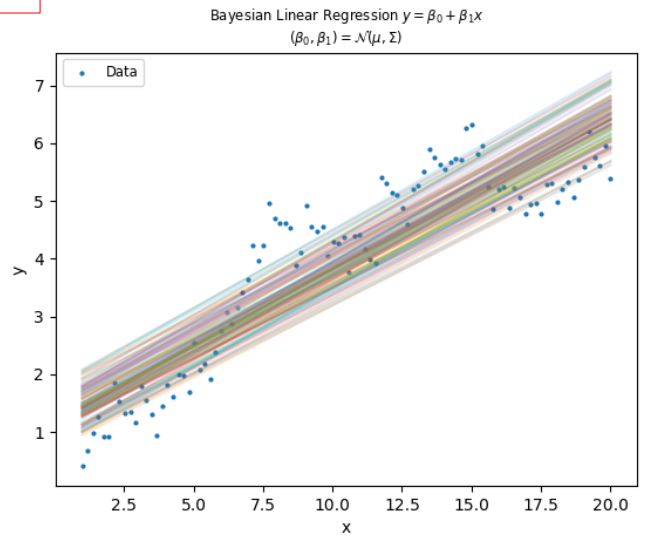
Fixed, unknown quantities



Bayesian

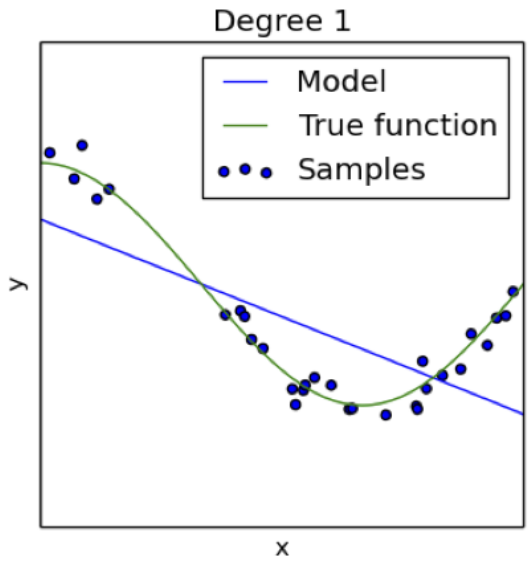


Random variables

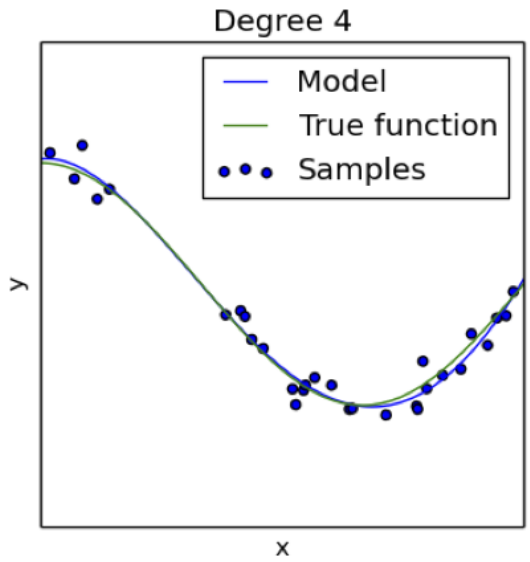


Parametric Regression

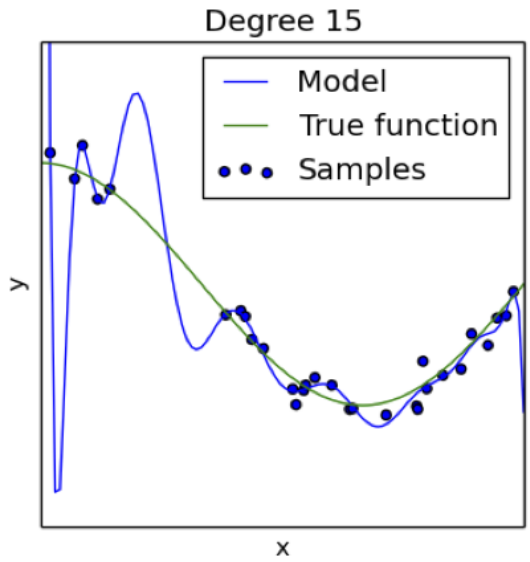
Model complexity has to be explicitly calibrated.



$$y = \beta_0 + \beta_1 x$$



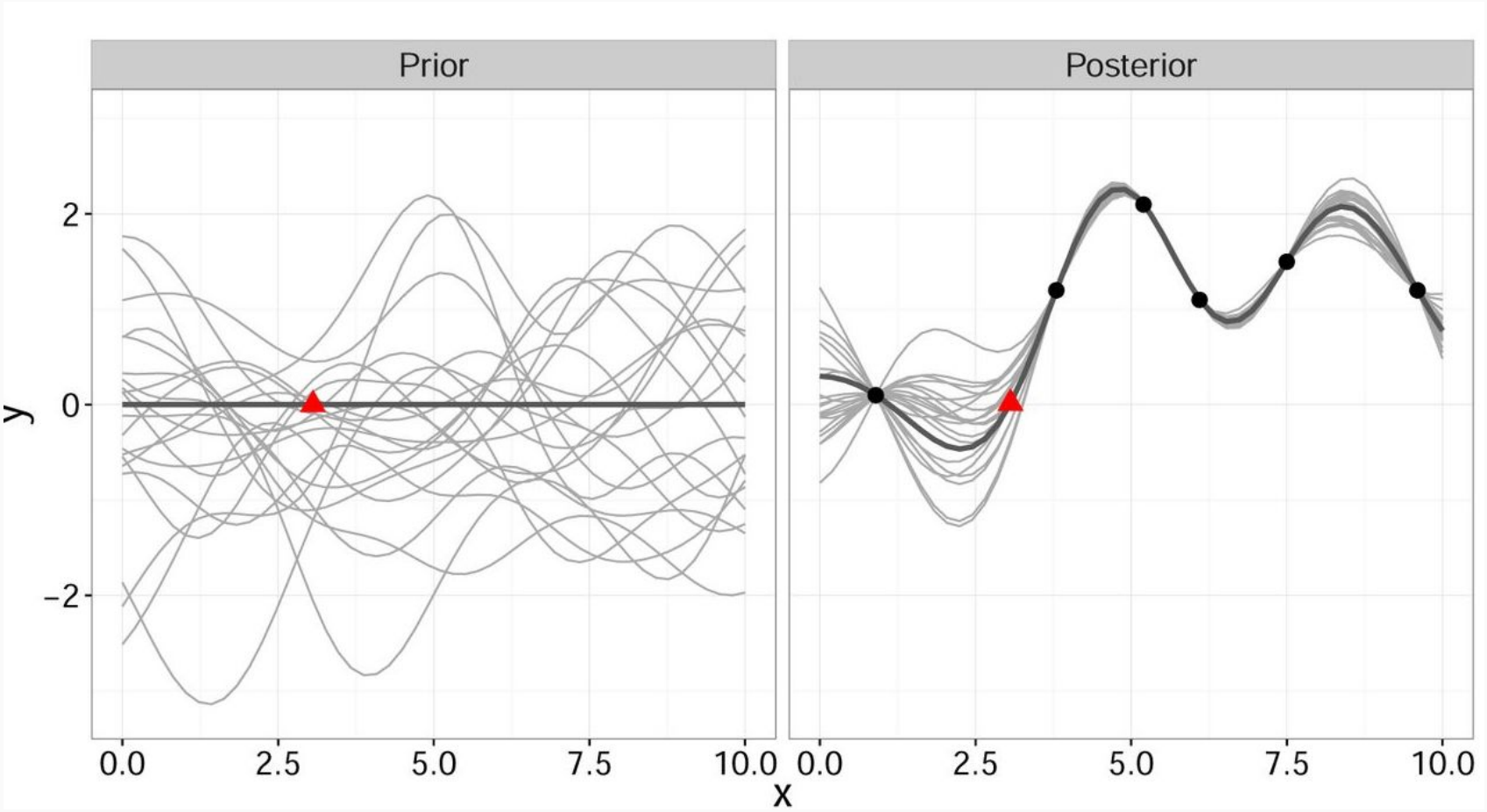
$$y = \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3 + \beta_4 x^4$$



$$y = \beta_0 + \beta_1 x + \beta_2 x^2 + \cdots + \beta_{15} x^{15}$$

Non-parametric Regression

Automatic calibration of model complexity.



The non-parametric approach

- We don't select the functional form of the model, ~~$y = \beta_0 + \beta_1 x$~~.

The non-parametric approach

- We don't select the functional form of the model, ~~$y = \beta_0 + \beta_1 x$~~.
- It is *"letting the data speak for itself"* $\rightarrow$ the model becomes more complex as the size and the complexity of the data grow.

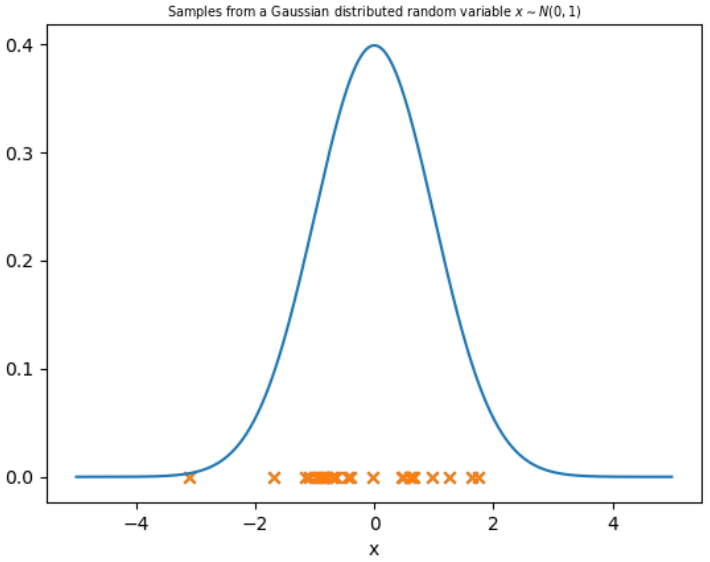
The non-parametric approach

- We don't select the functional form of the model, ~~$y = \beta_0 + \beta_1 x$~~.
- It is *"letting the data speak for itself"* $\rightarrow$ the model becomes more complex as the size and the complexity of the data grow.
- The model structure (a.k.a functional form) and the parameters are both part of the *"learning"* in a non-parametric model.

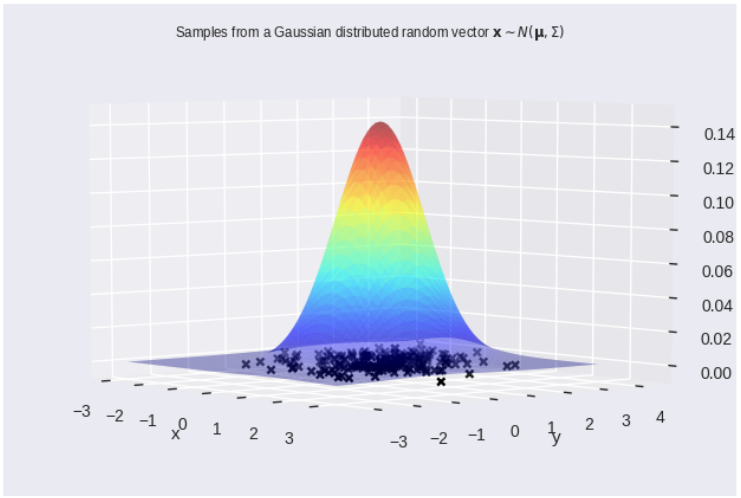
Gaussian Processes

What is a GP?

The most intuitive way of understanding GPs is understanding the correspondence between Gaussian distributions and Gaussian Processes.



(a) samples from a univariate gaussian distribution



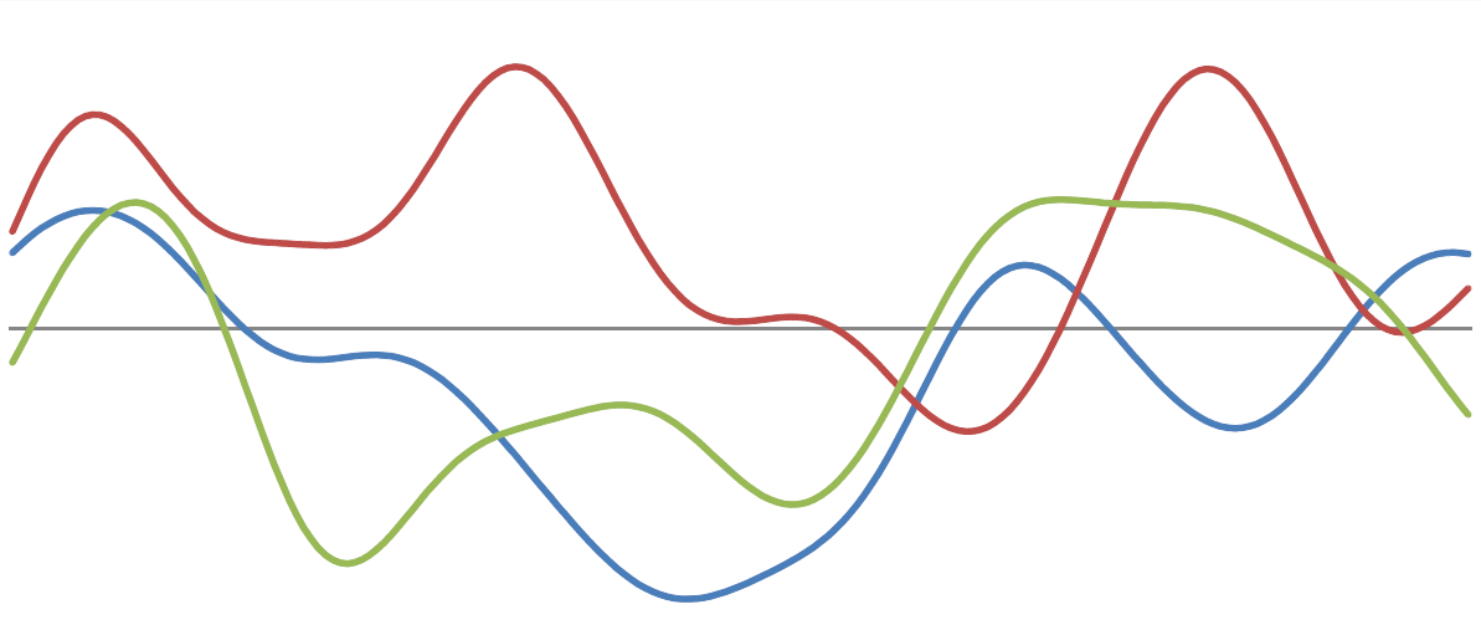
(b) samples from a bi-variate gaussian distribution

What is a GP?

GPs are just Gaussian probability distributions of random functions $f(x)$, hence sampling from a Gaussian process gives functions $f(x)$.

$$f(x) \sim \mathcal{GP}(m(x), k(x, x')) \tag{1}$$

where $m(x)$ represents the mean function and $k(x, x')$ represents a covariance function.



Primer: Functions as infinite vectors

1. When we think of a 'function' in a mathematical sense we immediately try to think of a parametric form. For example, $5x - 2, x^2, 3x^3 - x, e^x$.

2. But in GP world there is a fundamental shift in thinking about functions. **We completely abandon the parametric form viewpoint**.

3. Think of functions as a vector of infinite length $f(x) = [f(x_1), f(x_2) \ldots\ldots]$

4. GPs represent functions $f(x)$ obliquely (but rigorously) by selecting the covariance function $k(x, x')$.

Primer: Marginals and Conditionals

Gaussians have some really nice properties:

$$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \mu_1 \\ \mu_2 \end{bmatrix}, \begin{bmatrix} \Sigma_{11} & \Sigma_{12} \\ \Sigma_{12}^T & \Sigma_{22} \end{bmatrix} \right)$$

Marginalisation is trivial. Just ignore dimensions you don't want.

$$x_1 \sim \mathcal{N}(\mu_1, \Sigma_{11}) \tag{2}$$

Conditioning is straightforward:

$$x_1 | x_2 \sim N(\mu_3, \Sigma_3)$$

where,

$$\mu_3 = \mu_1 + \Sigma_{12}\Sigma_{22}^{-1}(A_2 - \mu_2)$$
$$\Sigma_3 = \Sigma_{11} - \Sigma_{12}\Sigma_{22}^{-1}\Sigma_{21}$$

($x_2|x_1$ can be evaluated using symmetry)

Rigorous definition

A GP is a collection of random variables, any finite number of which have a joint Gaussian distribution. Infinite dimensional analogue:

$$f(x) \sim \mathcal{GP}(m(x), k(x, x'))$$

where m represents the mean function and $k(x, x')$ represents a covariance function.

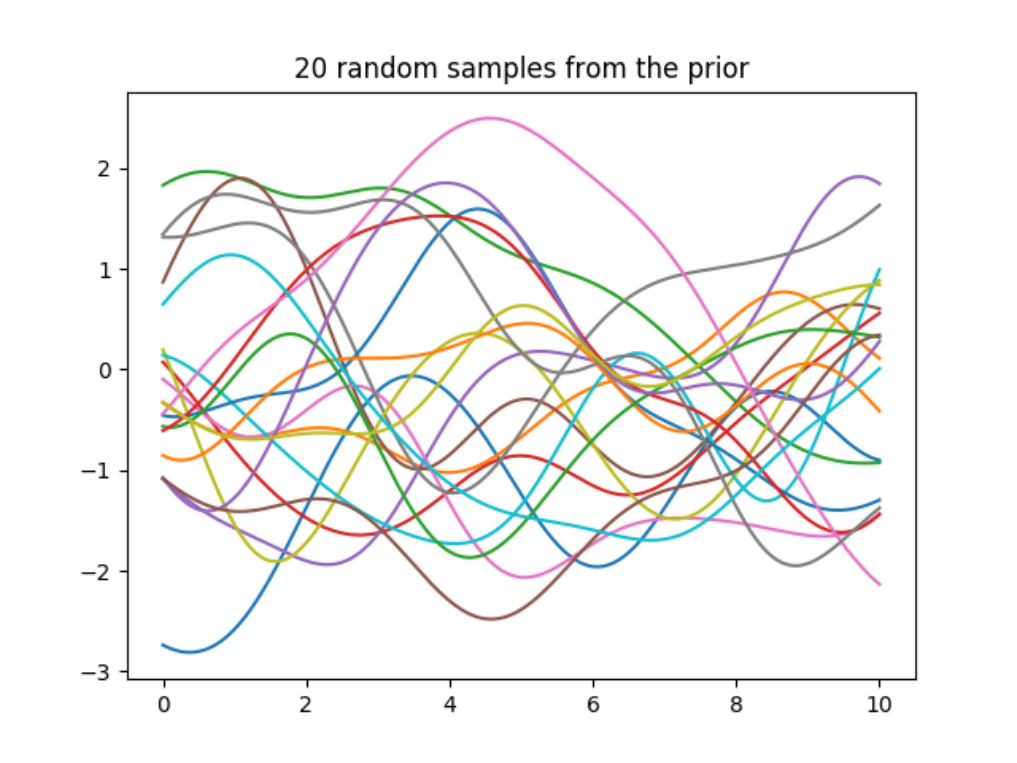
In order to compute with GPs we only use finite subsets of this infinite dimensional space.

$$\begin{bmatrix} f_1 \\ \vdots \\ f_{499} \\ f_{500} \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \mu_1 \\ \vdots \\ \mu_{499} \\ \mu_{500} \end{bmatrix}, \begin{bmatrix} k(x_1, x_1) & \dots & \dots k(x_1, k_{500}) \\ \vdots & \ddots & \vdots \\ \vdots & \ddots & \vdots \\ k(x_{500}, x_1) & \dots & \dots k(x_{500}, k_{500}) \end{bmatrix} \right)$$

In going from finite to infinite we are marginalising out everything not in our index set $x = [x_1, x_2, \dots, \dots, x_{500}]$.

Rigorous definition

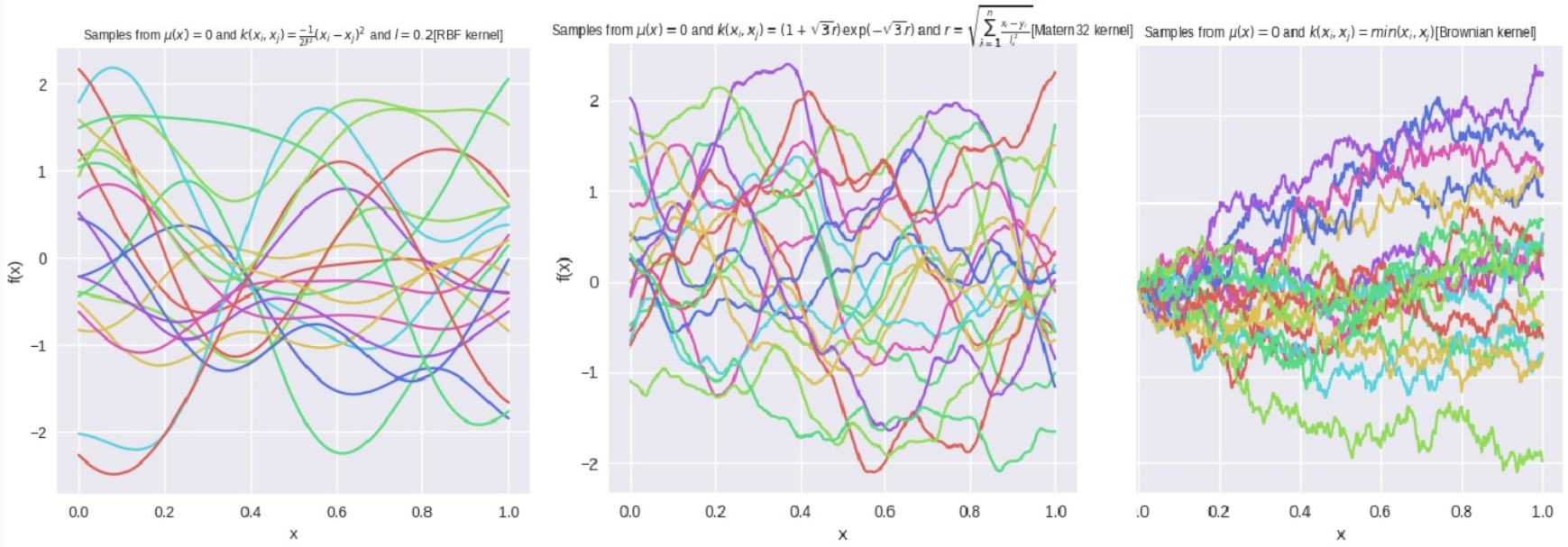
GPs induce a prior over random functions $f(x)$:



Samples from a GP with the squared exponential kernel

$k(x_1, x_2) = \sigma_f^2 \exp\{-\frac{(x_1 - x_2)^2}{2\gamma^2}\}$, with hyperparameter vector $\{\sigma_f^2, \gamma\}$

Structure with Kernels



- A kernel is a map $k: X \times X \rightarrow R$ - which encodes similarity between two inputs.
- Kernels control the shape, periodicity, smoothness and other structural properties of the functions being modelled.
- Valid kernels yield positive semi-definite covariance matrices (Mercer's theorem).
- Kernels can be defined over all types of data structures: text, strings, trees, matrices and kernels themselves!

$$k(x, x') = g(x)k_1(x, x')g(x')$$
$$k(x, x') = q(k_1(x, x'))$$
$$k(x, x') = \exp(k_1(x, x'))$$
$$k(x, x') = k_1(x, x') + k_2(x, x')$$
$$k(x, x') = k_1(x, x')k_2(x, x')$$
$$k(x, x') = k_3(\boldsymbol{\phi}(x), \boldsymbol{\phi}(x'))$$
$$k(x, x') = x^\top A x'$$

Gaussian Process Regression

1. Observe some noisy data $\boldsymbol{y} = \{y_i\}_{i=1}^N$ at X input locations.
2. Standard regression set-up $\boldsymbol{y} = f(x) + \epsilon, \quad \epsilon \sim \mathcal{N}(0, \sigma_n^2)$.
3. Data Likelihood: $\quad y|f \sim \mathcal{N}(f(x), \sigma^2)$
4. Specify GP Prior: $\quad f \sim \mathcal{GP}(0, k_\theta)$

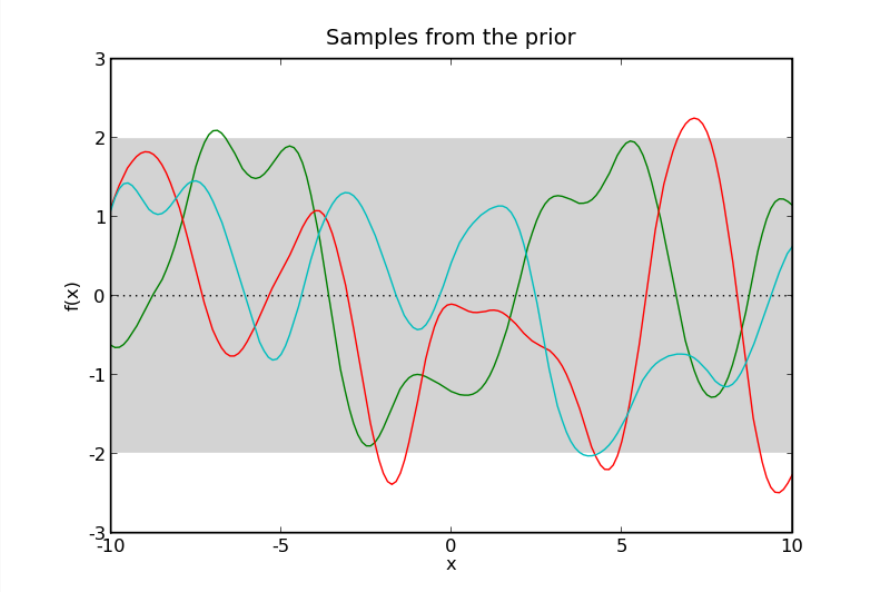
We want to infer latent function values f_* at arbitrary input locations X_*, or in other words we want $p(f_*|X_*, y, \theta)$.

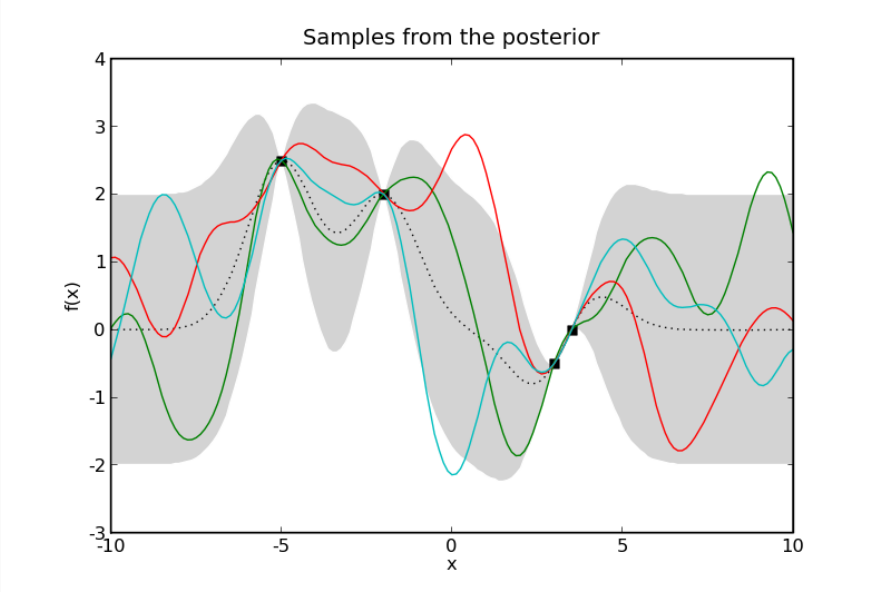
Joint distribution:

$$p(f, f_*) = \mathcal{N}\left(0, \begin{bmatrix} K(X,X) & K(X,X_*) \\ K(X_*,X) & K(X_*,X_*) \end{bmatrix}\right) = \mathcal{N}\left(0, \begin{bmatrix} K & K_*^T \\ K_* & K_{**} \end{bmatrix}\right)$$

$$p(y, f_*) = \mathcal{N}\left(0, \begin{bmatrix} K + \sigma^2\mathbb{I} & K_*^T \\ K_* & K_{**} \end{bmatrix}\right)$$

Learning and Inference





Conditional:

$$p(f_*|y) = \mathcal{N}(\mu_*, \ \Sigma_*)$$
$$\mu_* = K_*(K_\theta + \sigma_n^2)^{-1} f$$
$$\Sigma_* = K_{**} - K_*(K_\theta + \sigma_n^2)^{-1} K_*^T$$

Learning in a GP

Learning occurs through adapting the hyperparameters of the kernel function to the data. The objective that is used for this adaptation is the marginal likelihood.

$$\theta_\star = \underset{\theta}{\text{argmax}}\, \mathcal{L}(\theta)$$

$$p(y) = \int p(y|f)p(f)df = \mathcal{N}(0, K + \sigma^2 \mathbb{I})$$

$$\text{where,} \mathcal{L}(\theta) = \log p(y) = -\overbrace{\frac{1}{2} y^T (K_\theta + \sigma_n^2)^{-1} y}^{\text{model fit}} - \overbrace{\frac{1}{2} log|K_\theta + \sigma_n^2 \mathbb{I}|}^{\text{complexity penalty}} - \frac{-n}{2} log 2\pi$$

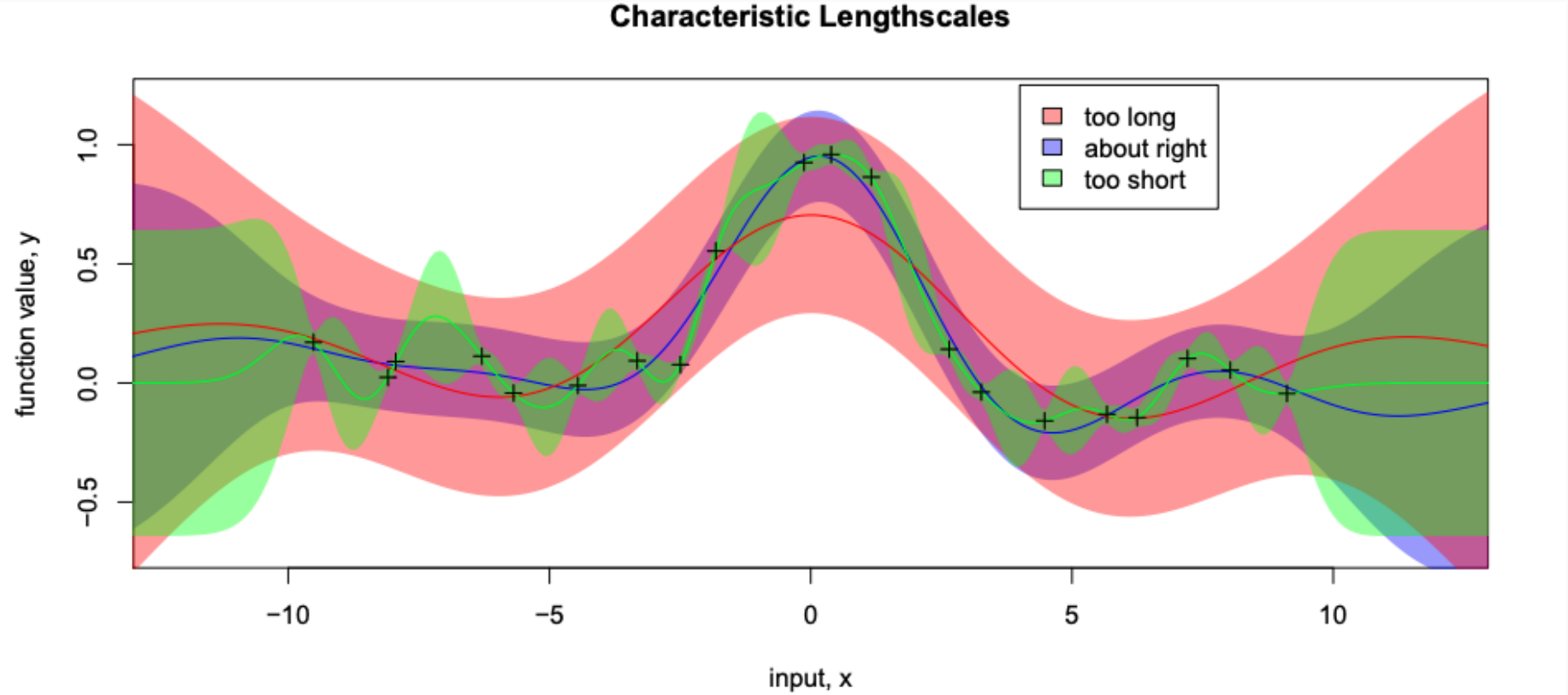
Inference: Plug in θ_* in the conditional distribution.

$$p(f_*|y) = \mathcal{N}(\mu_*, \ \Sigma_*)$$

$$\mu_* = K_*(K_\theta + \sigma_n^2)^{-1} f$$

$$\Sigma_* = K_{**} - K_*(K_\theta + \sigma_n^2)^{-1} K_*^T$$

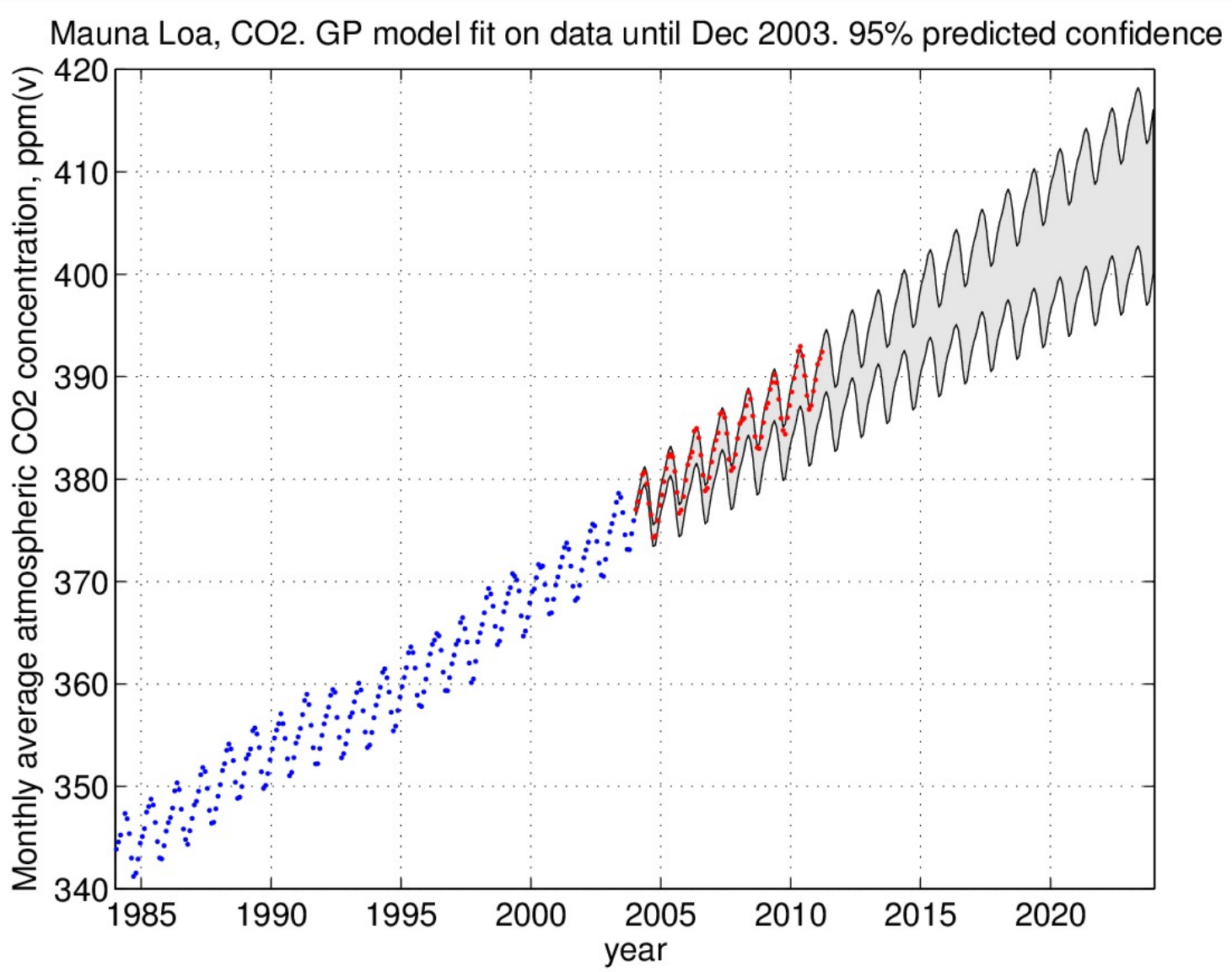
Learning in a GP



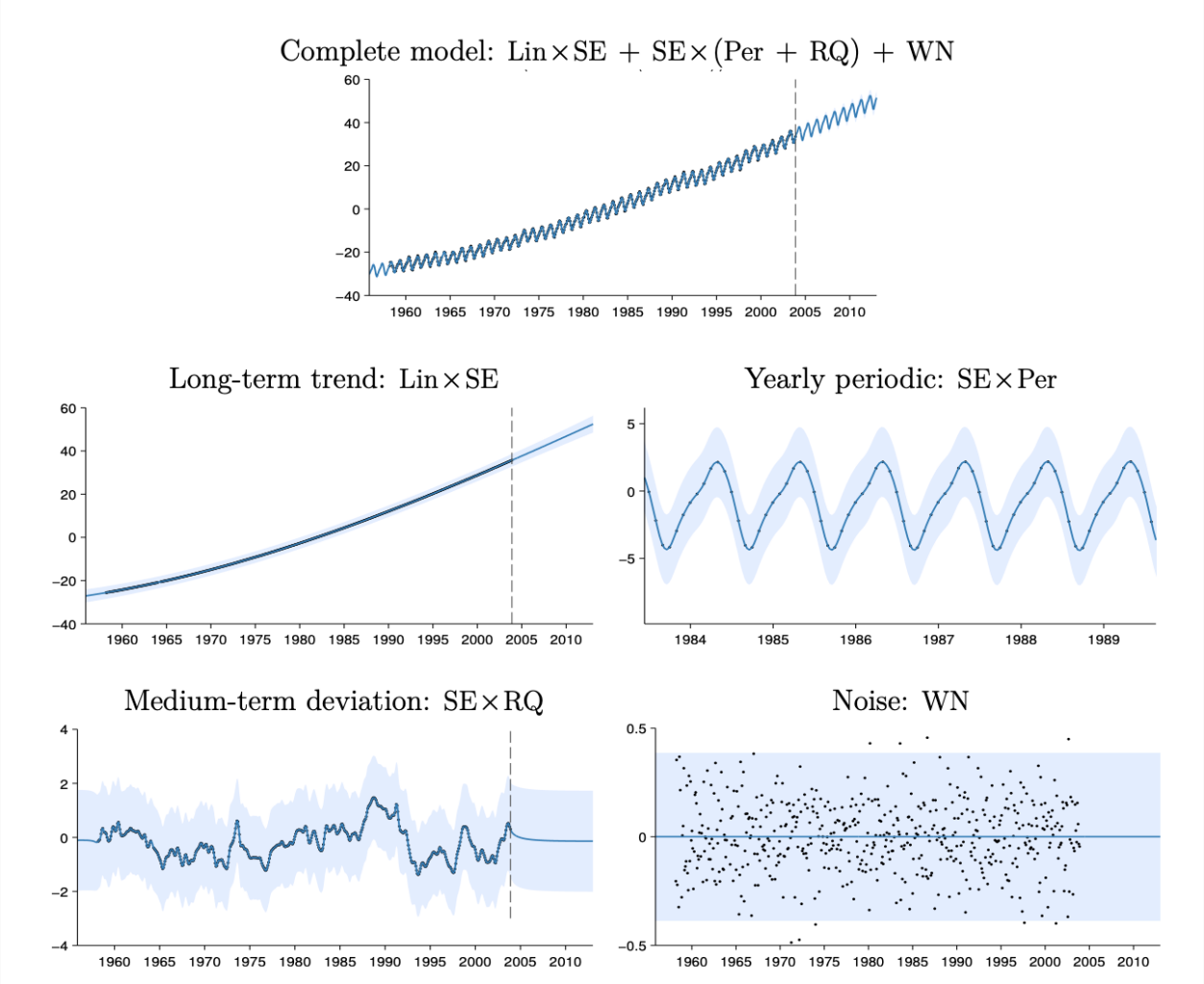
The over-fitted and under-fitted models are not favoured by the marginal likelihood.

$$\log p(y) = \overbrace{-\frac{1}{2} y^T (K_\theta + \sigma_n^2)^{-1} y}^{\text{model fit}} \overbrace{-\frac{1}{2} log|K_\theta + \sigma_n^2 \mathbb{I}|}^{\text{complexity penalty}} - \frac{-n}{2} log 2\pi$$

Encoding rich structure



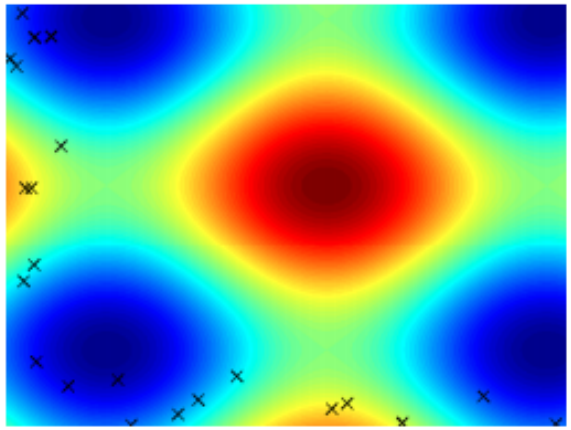
Composite Kernel



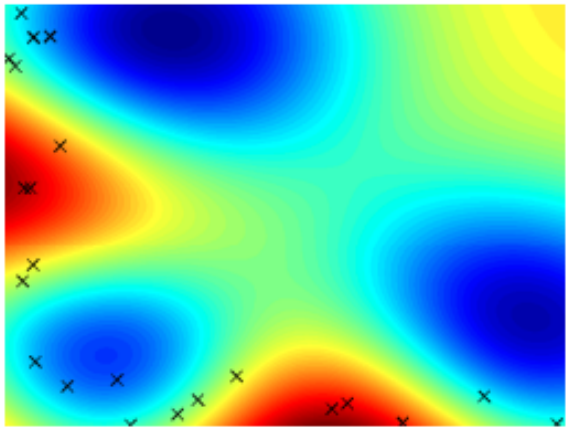
[1]

[1] David Duvenaud. "Automatic model construction with Gaussian processes". PhD thesis. University of Cambridge, 2014.

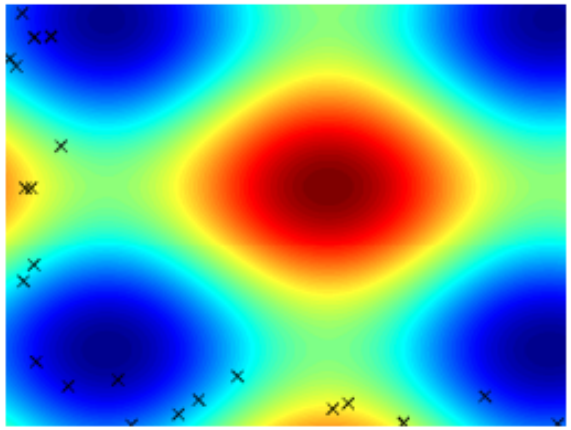
2d Regression



True Function
& data locations



Squared-exp GP
posterior mean



Additive GP
posterior mean

[2]

The composite kernels picks up on unseen structure about the original function.

[2] David K Duvenaud, Hannes Nickisch, and Carl E Rasmussen. "Additive gaussian processes". In: *Advances in neural information processing systems*. 2011, pages 226–234.

Probabilistic and Non-parametric

1. It takes learning one step further by automatically calibrating model complexity. So, the model is **learnt** rather than chosen or fixed.

Probabilistic and Non-parametric

1. It takes learning one step further by automatically calibrating model complexity. So, the model is **learnt** rather than chosen or fixed.
2. Prediction Uncertainty - a fitted model needs to know when it does not know.
3. Other Models: Dirichlet Processes, Infinite Mixture Models

Summary

- GPs are a powerful probabilistic and non-parametric paradigm for modelling non-linear functions in low and high dimensions.
- They are expensive to train as the cost of inverting a matrix of size N is $\mathcal{O}(N^3)$.
- Memory requirements for storing a matrix of size N is $\mathcal{O}(N^2)$.
- Choosing or designing the right kernel is a bit of a black art.

1 slide on Neural Nets

Using deep architectures for modelling functions, capturing non-linear relationships or structure inherent in high dimensional data.

Where Deep learning falls short

- Cannot probe deep learning using statistical theory or probability calculus (unless you use Bayesian Neural Nets.)

Where Deep learning falls short

- Cannot probe deep learning using statistical theory or probability calculus (unless you use Bayesian Neural Nets.)
- Very compute intensive and data hungry.

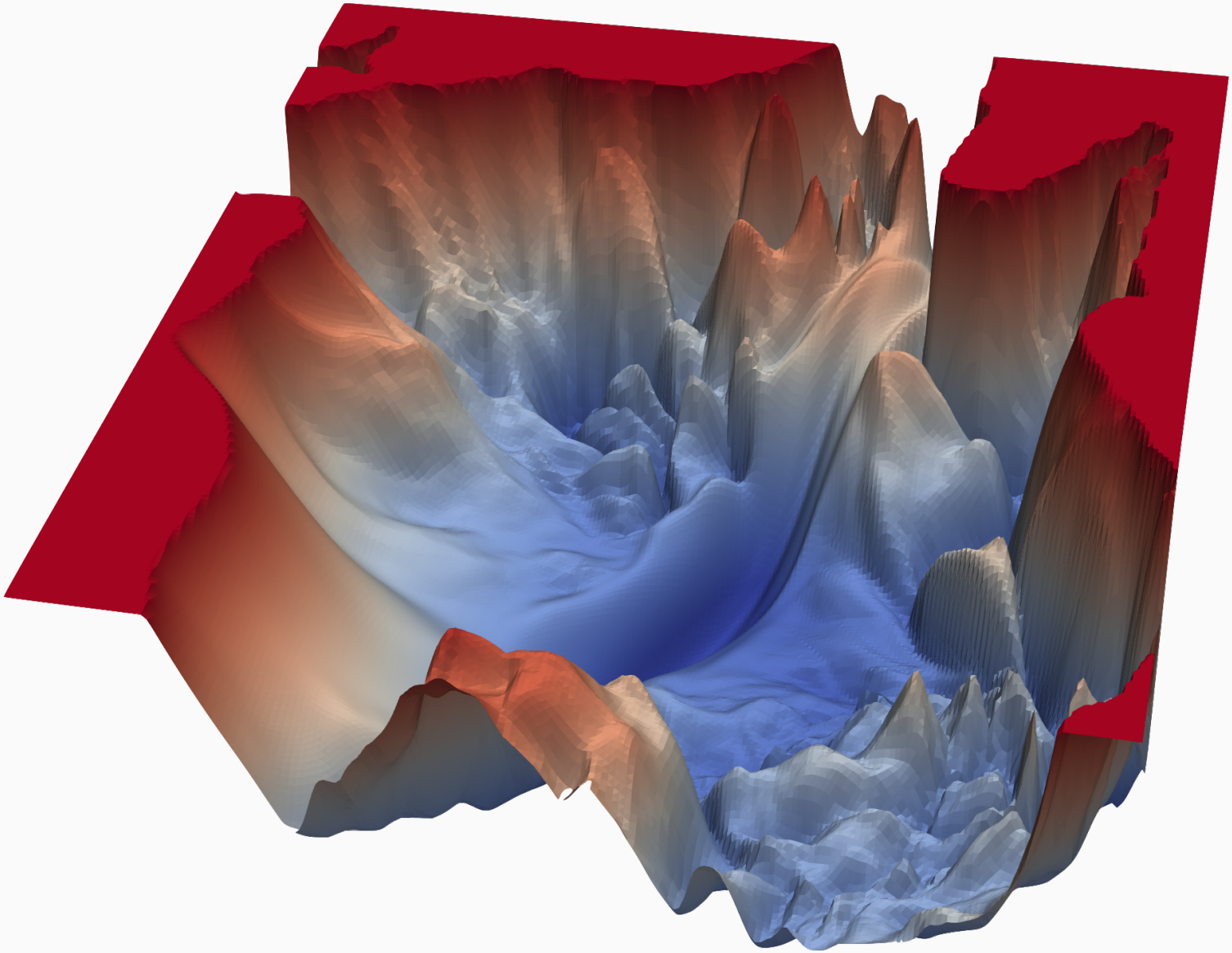
Where Deep learning falls short

- Cannot probe deep learning using statistical theory or probability calculus (unless you use Bayesian Neural Nets.)
- Very compute intensive and data hungry.
- Non-convexity of loss surface, catastrophic local minima / saddle point problem.

Where Deep learning falls short

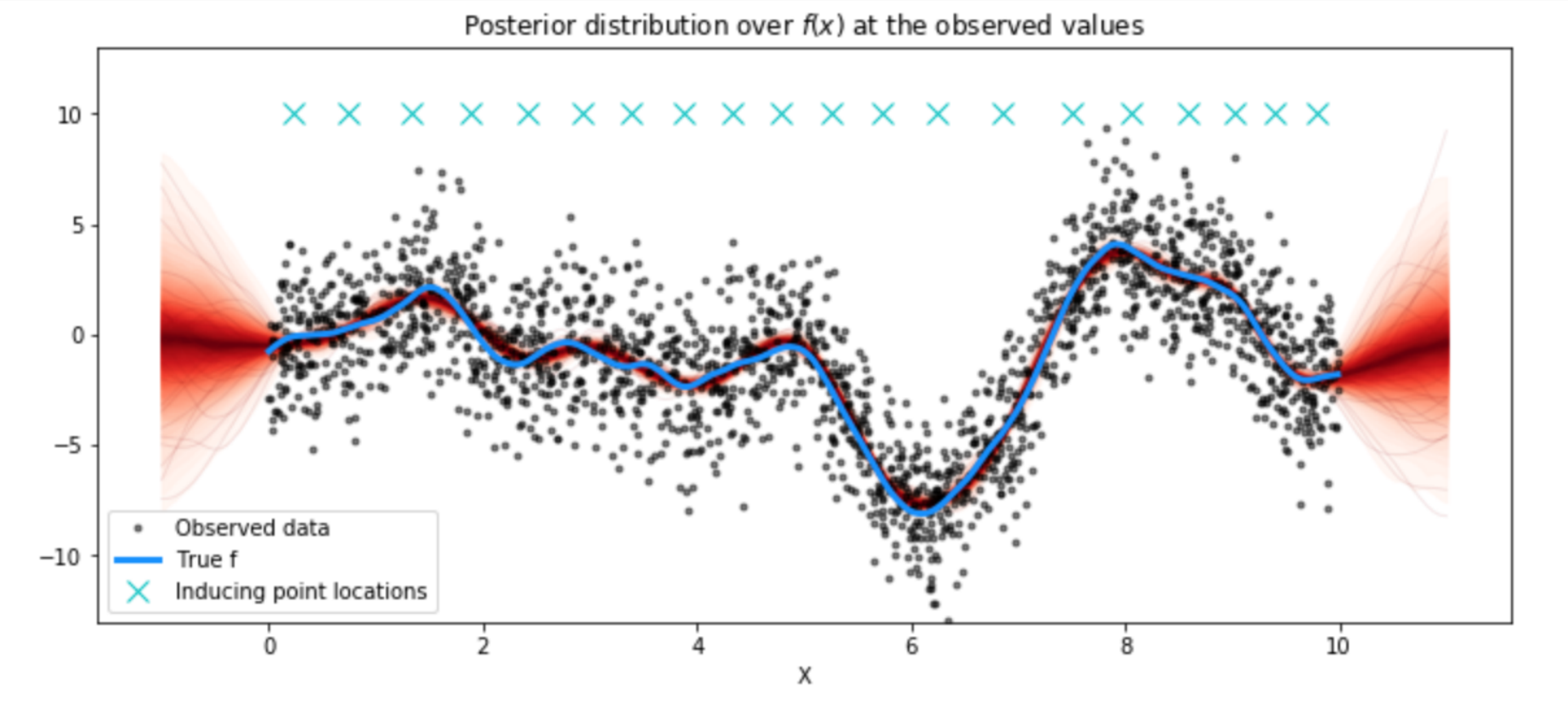
- Cannot probe deep learning using statistical theory or probability calculus (unless you use Bayesian Neural Nets.)
- Very compute intensive and data hungry.
- Non-convexity of loss surface, catastrophic local minima / saddle point problem.
- Neural architecture, learning rate, activation functions all have to be selected before training and most of the times these choices are not theoretically guided.

Loss Surface



Advanced Gaussian Processes: Sparsity

Sparse Gaussian Processes



[3] John Salvatier, Thomas V Wiecki, and Christopher Fonnesbeck. "Probabilistic programming in Python using PyMC3". In: *PeerJ Computer Science* 2 (2016), e55.

Advanced Gaussian Processes: Deep Kernels

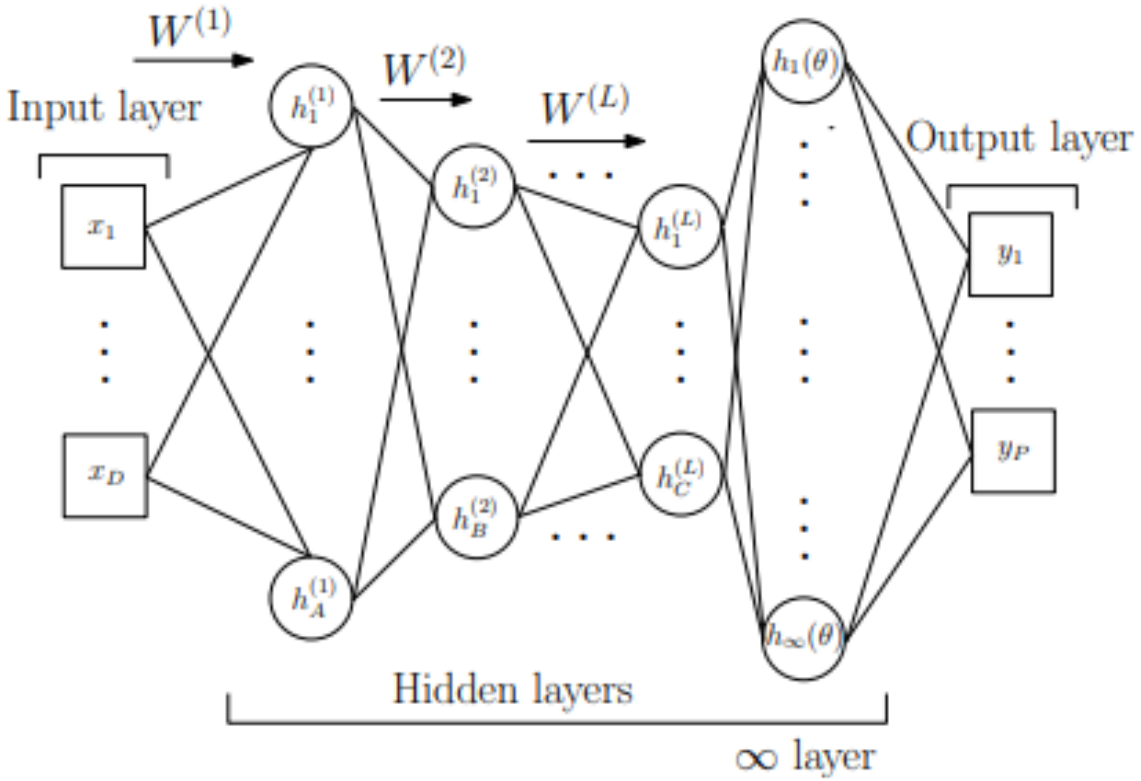


Figure 1: Deep Kernel Learning: A Gaussian process with a deep kernel maps D dimensional inputs $\mathbf{x}$ through L parametric hidden layers followed by a hidden layer with an infinite number of basis functions, with base kernel hyperparameters $\boldsymbol{\theta}$. Overall, a Gaussian process with a deep kernel produces a probabilistic mapping with an infinite number of adaptive basis functions parametrized by $\boldsymbol{\gamma} = \{\mathbf{w}, \boldsymbol{\theta}\}$. All parameters $\boldsymbol{\gamma}$ are learned through the marginal likelihood of the Gaussian process.

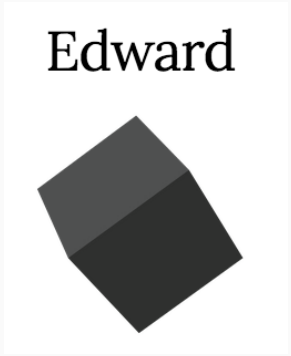
[4]

[4] Andrew Gordon Wilson, Zhiting Hu, Ruslan Salakhutdinov, and Eric P Xing. "Deep kernel learning". In: *Artificial Intelligence and Statistics*. 2016, pages 370–378.

Resources: Probabilistic programming

STAN (C++ backend with custom syntax)
pymc3
GPFlow (tf)
Edward
GPytorch

All offer HMC and Variational Inference



Hamiltonian Monte Carlo

HMC is a fundamental tool for MCMC inference in advanced GP models. It hinges on using gradient information to suppress random walk behaviour.

Here we look at the standard form of the algorithm.

On Probabilistic Machine Learning

- Posterior Inference (computing $p(\theta|y)$) is intractable in most models which are used in production.
- $p(\theta|y) = \frac{p(y|\theta)p(\theta)}{p(y)}$ where $p(y) = \int p(y|\theta)p(\theta)d\theta$
- Inference step: Sample $\theta_i \sim p(\theta|y)$ using HMC.
- Prediction step: $p(y_*|x_*, y) = \frac{1}{M}\sum_{i=1}^{M} p(y_*|x_*, y, \theta_i)$

The physics picture

The Hamiltonian $H(p, q)$ defines the phase space of the physical system in terms of a position vector q and a momentum vector p. Hamiltonian mechanics describe the time evolution of a physical system in terms of Hamilton's equations

$$\frac{dq}{dt} = \frac{\partial \mathcal{H}}{\partial p} \tag{3}$$

$$\frac{dp}{dt} = \frac{-\partial H}{\partial q} \tag{4}$$

Consider a closed system of a friction-less particle moving along a surface of varying height. The partial derivatives w.r.t time define a mapping of the state from any time t to the state at any time $t + s$.

Hamiltonian Dynamics $\rightarrow$MCMC

The Hamiltonian admits a decomposition in terms of potential and kinetic energy,

$$H(p.q) = V(q) + K(p)$$

The task is to sample from a density $p(\boldsymbol{x})$ (typically, a posterior density), in order to facilitate the use of the dynamics, we augment with momentum variable $\boldsymbol{p} \sim (0, M)$ and define a joint density,

$$p(\boldsymbol{x}, \boldsymbol{p}) = p(\boldsymbol{x})p(\boldsymbol{p}) \propto e^{-H(\boldsymbol{x},\boldsymbol{p})}$$

where the negative log-joint has the form,

$$H(\boldsymbol{x}, \boldsymbol{p}) = -\log p(\boldsymbol{x}) + \frac{1}{2}\boldsymbol{p}^T M^{-1}\boldsymbol{p}$$

Hamiltonian Dynamics →MCMC

Define,

$$V(q) = -\log p(x),$$
$$K(p) = \frac{1}{2} p^T M^{-1} p$$

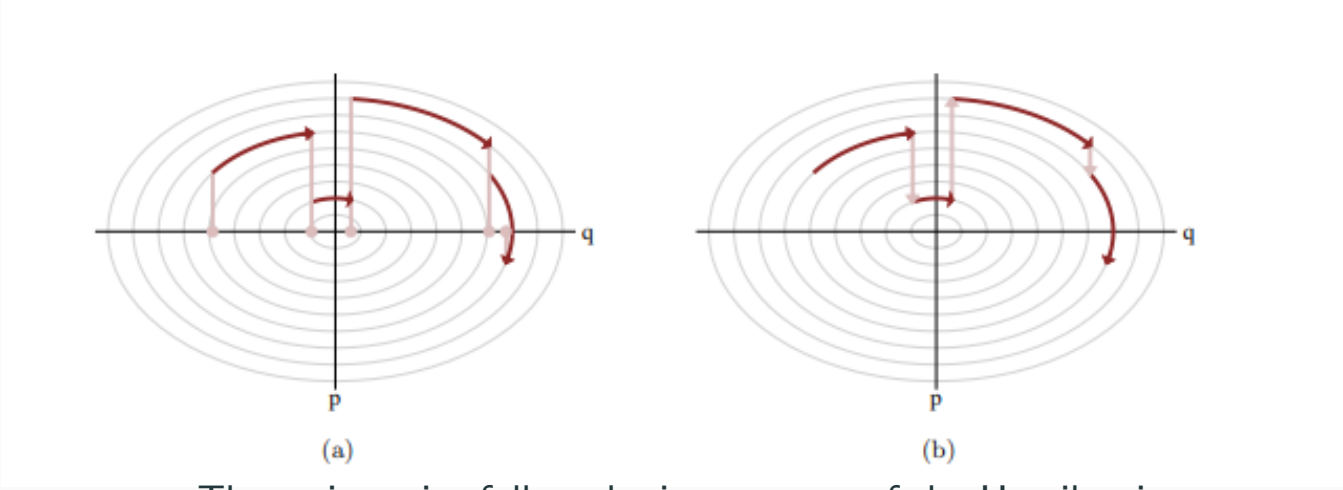
So x, the variable of interest plays the role of the position coordinate q and $K(p)$ corresponds to minus log probability density of a zero mean Gaussian with covariance matrix M.

With these forms Hamilton's equations become,

$$\frac{dq}{dt} = \frac{\partial H}{\partial p} = M^{-1} p$$
$$\frac{dp}{dt} = -\frac{\partial H}{\partial q} = -\frac{\partial V}{\partial q}$$

(Side Note: $V(q) = -log[\pi(q)L(q|y)]$, for simplicity of notation we just use $p(q)$ to denote the posterior or target)

Canonical HMC



[5]

The trajectories follow the isocontours of the Hamiltonian

HMC

Given q_0, ϵ, L, M
for t = 1, . . . , N **do**
  Sample momentum $p_t \sim \mathcal{N}(0, M)$
  Leapfrog $(q_t, p_t) \rightarrow (q_{t+\epsilon L}, p_{t+\epsilon L})$ *depends on L, ϵ and current state*
  Draw $u \sim Unif[0, 1]$
  if $u < min[1, \exp(H(q_t, p_t) - H(q_{t+\epsilon L}, p_{t+\epsilon L})]$
  $(q_{t+1}, p_{t+1}) = (q_{t+\epsilon L}, p_{t+\epsilon L})$
  else
  $(q_{t+1}, p_{t+1}) = (q_t, p_t)$
end for

[5] **betancourt2017conceptual**.

HMC: Leapfrog Integrator

Hamilton's equation are not explicitly solvable, instead they must be approximated by discretizing time using a small finite step size ϵ.

Leapfrog: $(q_t, p_t) \to (q_{t+\epsilon L}, p_{t+\epsilon L})$

1: **for** j $= 1, \ldots,$ L **do**

2: $\quad p_{t+\epsilon/2} = p_t - (\epsilon/2)\frac{\partial V}{\partial q}|_{q_t}$

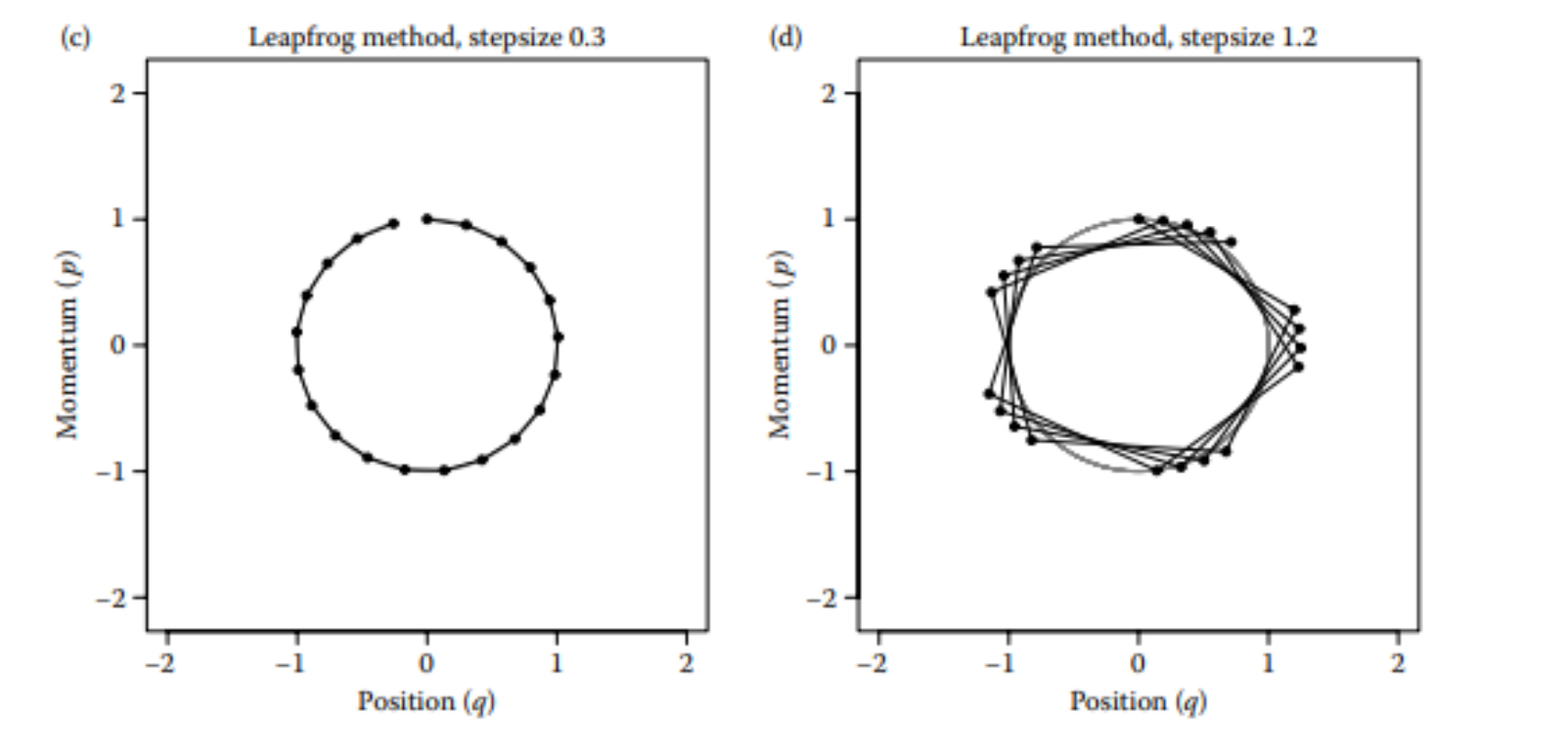
3: $\quad q_{t+\epsilon} = q_t + \epsilon M^{-1}(p_{t+\epsilon/2})$

4: $\quad p_{t+\epsilon} = p_{t+\epsilon/2} - (\epsilon/2)\frac{\partial V}{\partial q}|_{q_{t+\epsilon}}$

5: **end for**

Leapfrog method traces out an approximate Hamiltonian trajectory in phase space.

HMC: Leapfrog Integrator



[6]

The effect of step-size on the discretisation error, path length $L = 20$ and $H(q, p) = q^2/2 + p^2/2$

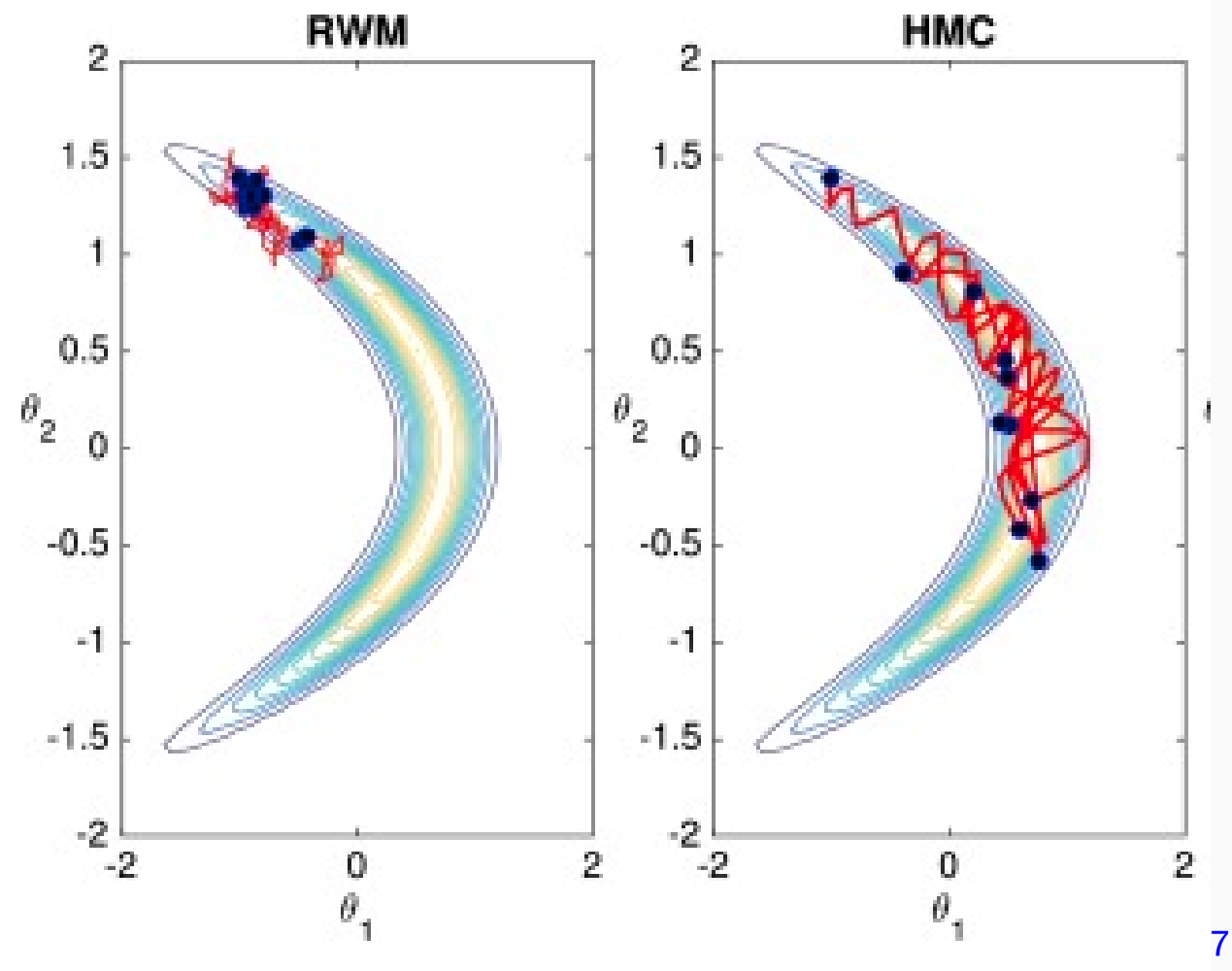
[6] **neal2011mcmc**.

Properties

1. **Reversibility:** The mapping $(q_t, p_t) \to (q_{t+s}, p_{t+s})$ is 1-1.
2. **Conservation of the Hamiltonian:** The dynamics keep the Hamiltonian invariant $\frac{\partial H}{\partial t} = 0$
3. **Symplecticness:** The phase space (q, p) of a Hamiltonian system is a symplectic manifold.

2. is not achieved exactly as we simulate dynamics numerically using the leapfrog method.

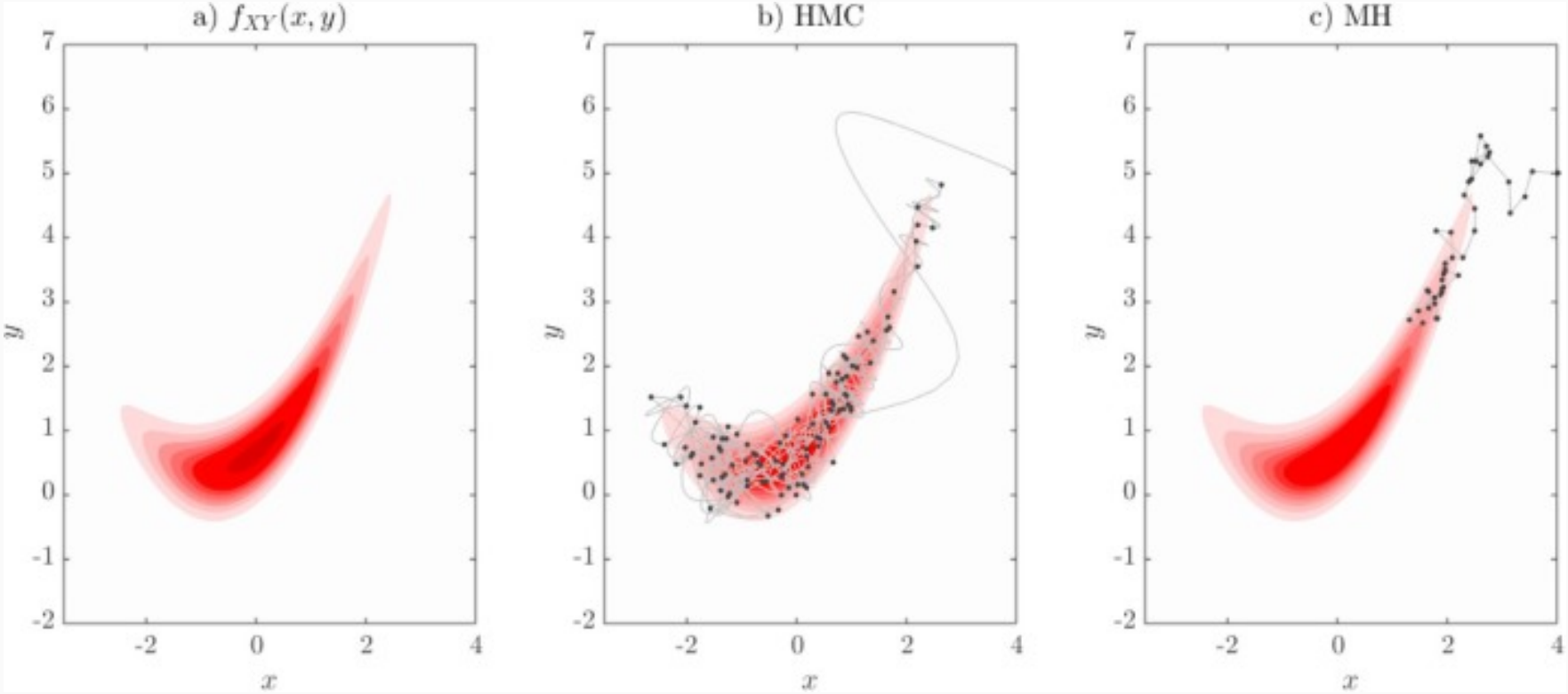
HMC on a warped Gaussian



[7]

[7] **lan2016emulation**.

HMC on a warped Gaussian

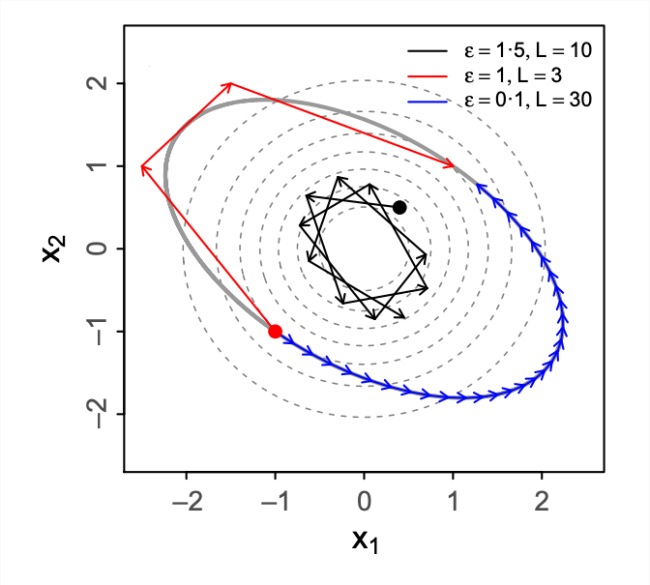


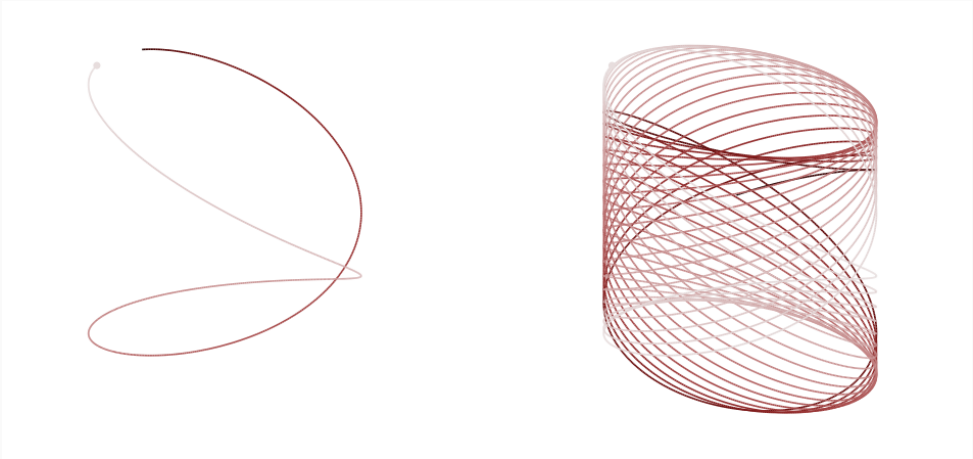
$\epsilon = 0.05, L = 20$, showing accepted proposals from 100 trajectories.

HMC in practice

Performance of HMC depends on choosing suitable values for ϵ and L.

1. ϵ too small $\rightarrow$ Inefficient exploration.
2. ϵ too large $\rightarrow$ Will miss areas of the target density with a small spatial scale, higher rejection rate.
3. L too small $\rightarrow$ Slow mixing, resembling random-walk.
4. L too large $\rightarrow$ Redundant computation due to cyclical nature of trajectories.





Thank you!

vr308@cam.ac.uk

@VRLalchand