Crash course on Gaussian Process

a sip of theory and a mouthful of practice

Yingzhao Lian, Emilio Maddalena

June 17, 2021

Labotoire d'Automatique, EPFL

Table of Contents

Recap of Bayes

Recap of Bayes

Bayes rule: Estimate the underlying paramters θ from the measurement y

$$\underbrace{\mathbb{P}(\theta|y)}_{\text{A posterior}} = \underbrace{\mathbb{P}(\theta)}_{\text{A priori}} \frac{\overbrace{\mathbb{P}(y|\theta)}^{\text{likelihood}}}{\mathbb{P}(y)}$$

- **A priori** is your prior knowledge about the θ.
- **Likelihood** models how the parameter θ changes the measurement y.
- **A posterior** is your new knowledge of θ updated by your measurement y.

Why do we use Bayes

- Incorporate our assumption on θ: linearity, Gaussian, e.t.c
- Avoid overfitting

Recap of Bayes

Bayes rule: Estimate the underlying paramters θ from the measurement y

$$\underbrace{\mathbb{P}(\theta|y)}_{\text{A posterior}} = \underbrace{\mathbb{P}(\theta)}_{\text{A priori}} \frac{\overbrace{\mathbb{P}(y|\theta)}^{\text{likelihood}}}{\mathbb{P}(y)}$$

- **A priori** is your prior knowledge about the θ.
- **Likelihood** models how the parameter θ changes the measurement y.
- **A posterior** is your new knowledge of θ updated by your measurement y.

Take home messages

- A priori is the only difference between maximal likelihood and maximal a posterior
- The effect of a priori will diminish as the amount of measurements increases ("data overwhelms the prior").

Recap of Bayes

When we use Bayes rule to "learn", it finds the most possible parameter that satisfies our assumption. However, the explicit solutions of Bayes rule are normally intractable.

- MCMC approach: simulation based approach to sample from the posterior distribution
- Variational approach: approximate posterior distribution with some parametric distribution, and then minimize the approximation error.

Bayes and Gaussian Process

When we "learn" a Gaussian process, we first assume that the underlying function is a Gaussian process (**A priori**). The "learnt" model(**A posterior**) is calculated from noisy measurements (**Likelihood**).

Gaussian Process

Gaussian Process

Gaussian process is a distribution of function, there are two main approaches to characterize an unknown function

- Model the function **evaluation**: *Function space viewpoint*
- Model the **weight** w.r.t a collection of basis functions: *Weight space viewpoint*

Function Space Viewpoint

Definition

A Guassian process $\mathcal{GP}(\mu(\cdot), k(\cdot,\cdot))$ is a distribution of continuous real-valued function, whose function evaluation $y := \{f(x_i)\}_{i=1}^N$ at $X := \{x_i\}_{i=1}^N$ forms a multivariate Gaussian distribution $\mathcal{N}(\mu_X, K_{XX})$ with

$$\mu_X = \begin{bmatrix} \mu(x_1) \\ \vdots \\ \mu(x_N) \end{bmatrix}, \; K_{XX,i,j} = k(x_i, k_j) \,,$$

where $\mu(\cdot)$ is called the **mean function** and $k(\cdot,\cdot)$ is called the kernel.

Intuitively, mean function models the most possible structure of $f(x)$. For example, we can have $\mu(x) = Ax$ to enforce linearity. The kernel models the interaction of function **evaluation** at different locations.

Remark

The mean function and the kernel function uniquely define a Gaussian process

Function Space Viewpoint

- **A priori**: $f \sim \mathcal{GP}(\mu(\cdot), k(\cdot, \cdot))$.
- **Likelihood**: the measurement is contaminated by Gaussian measurement noise $\mathcal{N}(0, \sigma^2)$, we have a dataset $\{x_i, y_i\}_{i=1}^N$.

How do we get the posterior?

Function Space Viewpoint

- **A priori**: $f \sim \mathcal{GP}(\mu(\cdot), k(\cdot, \cdot))$.
- **Likelihood**: the measurement is contaminated by Gaussian measurement noise $\mathcal{N}(0, \sigma^2)$, we have a dataset $\{x_i, y_i\}_{i=1}^N$.

How do we get the posterior?

Remember that we model the **evaluation**!

Function Space Viewpoint

- **A priori**: $f \sim \mathcal{GP}(\mu(\cdot), k(\cdot,\cdot))$.
- **Likelihood**: the measurement is contaminated by Gaussian measurement noise $\mathcal{N}(0, \sigma^2)$, we have a dataset $\{x_i, y_i\}_{i=1}^N$.

How do we get the posterior?

Remember that we model the **evaluation**!

Take a random location x^*, by definition we know that

$$\begin{bmatrix} f(x_1) \\ \vdots \\ f(x_N) \\ f(x^*) \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \mu(x_1) \\ \vdots \\ \mu(x_N) \\ \mu(x^*) \end{bmatrix}, \begin{bmatrix} k(x_1,x_1) & k(x_1,x_2) & \dots & k(x_1,x^*) \\ k(x_2,x_1) & k(x_2,x_2) & \dots & k(x_2,x^*) \\ \vdots & & \ddots & \vdots \\ k(x_N,x_1) & k(x_N,x_2) & \dots & k(x_N,x^*) \\ k(x^*,x_1) & k(x^*,x_2) & \dots & k(x^*,x^*) \end{bmatrix} + \sigma^2 I \right)$$

$$\implies f(x^*) \sim \mathcal{N}(\overbrace{\mu(x^*) + k_{*X}[K_{XX} + \sigma^2 I]^{-1}(y - \mu_X)}^{\text{posterior mean function}},$$

$$\underbrace{k(x^*,x^*) - K_{*X}[K_{XX} + \sigma^2 I]^{-1} K_{X*}}_{\text{posterior kernel function}})$$

Weight Space Viewpoint

Consider that a function is spanned by some basis function $\{\phi_m({}_c dot)\}_{m=1}^M$

$$f(x) = \sum_{m=1}^{M} \alpha_m \phi_m(x) \ ,$$

where $\alpha \sim \mathcal{N}(\mu_\alpha, K_\alpha)$, then f follows a Gaussian process characterized by

$$\mu(x) = \sum_{m=1}^{M} \mu_{\alpha,m} \phi_m(x)$$

$$k(x, y) = \phi(x)^T K_\alpha \phi(y)$$

$k(x, y)$ is an "weighted" inner-product, hence we can use the **kernel trick**

Remark

- The basis function uniquely defines a kernel (*Moore-Aronszajn theorem*), however the basis function for a kernel is not unique.
- When the K_α is diagonal, we call these set of basis functions eigenfunctions. (*Mercer's theorem*)

Weight Space Viewpoint

We have observed that kernel incorporate the information of the basis functions, here are some take home messages

- Kernel determines the space of the function that we can model. For example, if $k(x, y) = x^\top y$, then we can only model linear functions.
- Usually, a mean function is used to model information beyond the kernel.
- Most interesting kernel has infinite many but countable eigenfunctions, such as **RBF** kernel $k(x, y) := e^{-\frac{\|x-y\|^2}{2L^2}}$

Construction of kernel

summation and product of two kernels is still a kernel function.

What do we "train" when we use any GP toolboxes?

The hyper-parameters of the kernel is trained, for example, by maximal **posterior**. Take RBF kernel for example

$$\max_{L,\sigma} \log \mathbb{P}(y|x) = \max_{l,\sigma} \underbrace{-(y-\mu_X)[K_{XX}+\sigma^2 I]^{-1}(y-\mu(x))}_{\text{fitting error } O(N^2 \log(N))} + \underbrace{\log(K_{XX}+\sigma^2 I)}_{\text{model capacity } O(N^3)}$$

Weight Space Viewpoint

Training/learning a GP means adjusting its hyperparameters

Most common objective: log marginal likelihood $\max_\theta \mathcal{L} = \log p(y|\theta)$

$$\mathcal{L} = -\frac{1}{2} y^\top (K_{XX} + \sigma^2 I)^{-1} y - \frac{1}{2} \log \det(K_{XX} + \sigma^2 I) - \frac{N}{2} \log(2\pi)$$

Data term (Mahalanobis distance), complexity term (Occam's razor)

Tunes lengthscales, vertical scales, noise variance, etc

Non-convex, unconstrained optimization problem

A first example

The Gramacy and Lee function

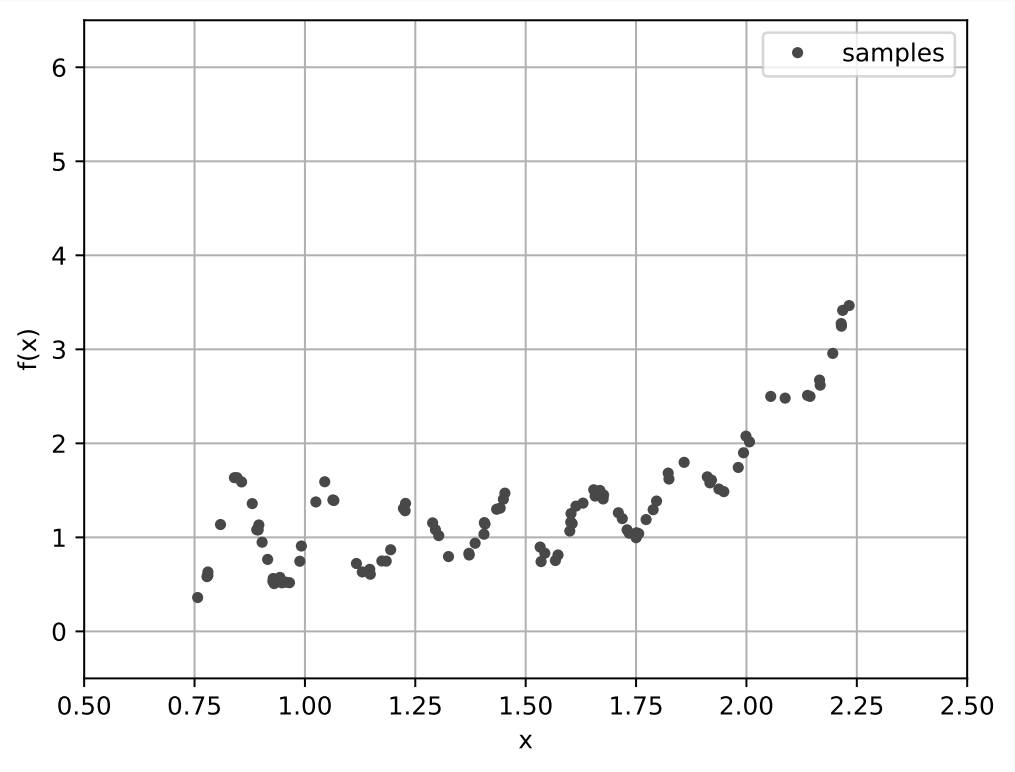


Figure 1: $N = 100$ samples, Gaussian likelihood with $\sigma^2 = 0.05$

A first example

Training your GP

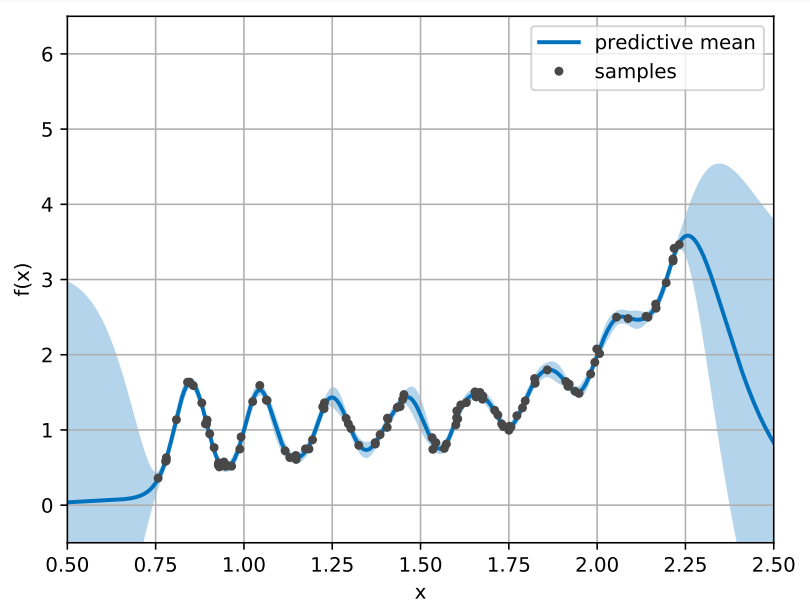
```
N      = 100
sigma  = 0.05
offset = 0.25

# Latent function
f     = GandL_1D()
X     = np.random.uniform(f.xmin+offset, f.xmax-offset, (N, f.xdim))
delta = np.random.normal(0, sigma, (N, 1))
Y     = f(X) + delta
data  = (tf.convert_to_tensor(X, "float64"), tf.convert_to_tensor(Y, "float64"))
```

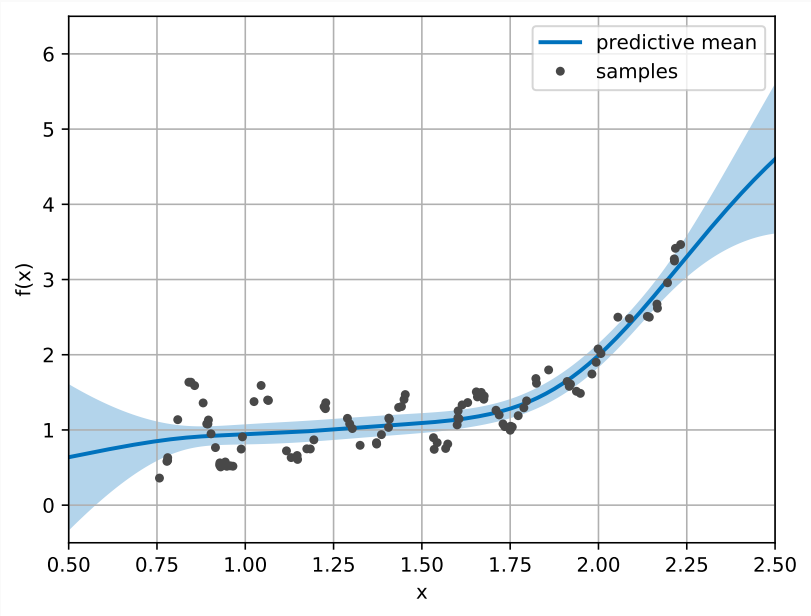
```
# Defining the GP
kernel = gpflow.kernels.Matern52(lengthscales=0.5*np.random.rand(f.xdim))
my_gp  = gpflow.models.GPR(data, kernel=kernel)

# Learning the GP
opt = gpflow.optimizers.Scipy()
opt.minimize(my_gp.training_loss, my_gp.trainable_variables)

# Predictions
xx = np.linspace(f.xmin, f.xmax, 1000).reshape(-1, 1)
mean, var = my_gp.predict_f(xx)
```

A first example

Training your GP

How come sometimes “it works” and sometimes “it doesn’t”?

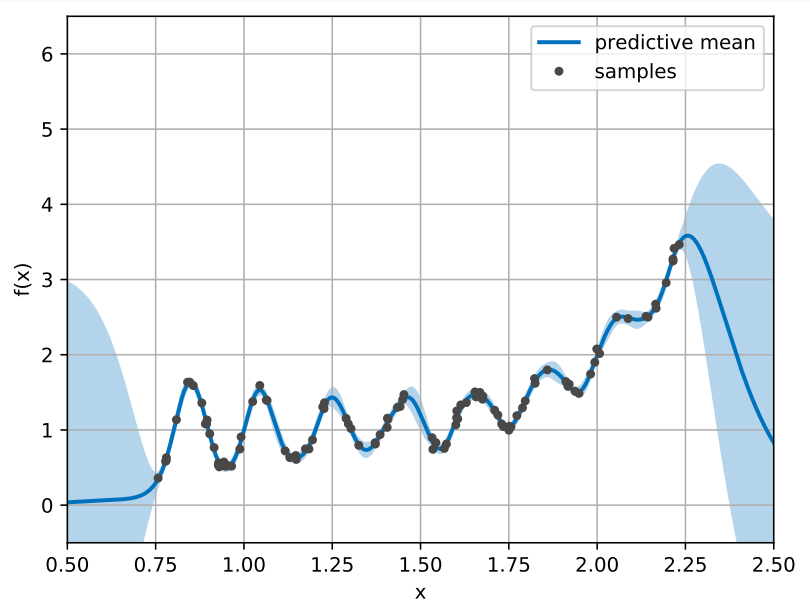


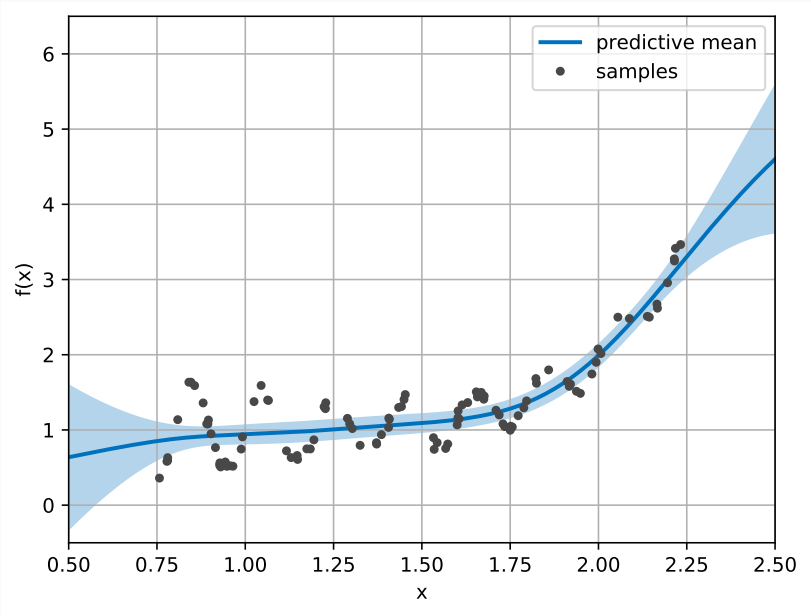


A first example

Training your GP

How come sometimes "it works" and sometimes "it doesn't"?





Training the hyperparameters (through MLE) is a non-convex procedure!

Attained likelihood: 77.16 (left) vs -33.24 (right) $\rightarrow$ left is better ☺

Limitations

Properties of GP

- A GP remains GP undergoing a linear operator.
  - Integration/differentiation of a GP is a GP
  - Koopman operator on a GP is a GP.

Limitations

Properties of GP

- A GP remains GP undergoing a linear operator.
  - Integration/differentiation of a GP is a GP
  - Koopman operator on a GP is a GP.
- Summation of two independent GP is a GP. (Not true for dependent GP)
- A GP remains GP after marginalization (*Similar to the marginalization of a multivariate Gaussian distribution remains a Gaussian distribution*).

Limitations

Properties of GP

- A GP remains GP undergoing a linear operator.
  - Integration/differentiation of a GP is a GP
  - Koopman operator on a GP is a GP.
- Summation of two independent GP is a GP. (Not true for dependent GP)
- A GP remains GP after marginalization (*Similar to the marginalization of a multivariate Gaussian distribution remains a Gaussian distribution*).

What we have done up to this point are **all** application of these properties, which also forms the limitations of GP.

Anything beyond the realm of these properties becomes Non-GP

Limitations

What we have done up to this point are **all** application of these properties, which also forms the limitations of GP.

Anything beyond the realm of these properties becomes Non-GP

- Consider a function $f : \mathbb{R} \to \mathbb{R}$, if f is nonlinear (*e.g.* a GP), then $f \circ \mathcal{GP}(\mu, k))$ is no longer GP.
- If the measurement noise is **non-Gaussian**, then the a posterior is non-GP.
- Vector-varlued kernel is well-defined but the vector-valued GP is **not** well-defined.
- The underlying function $f \sim \mathcal{GP}(\mu(\cdot), k(\cdot, \cdot))$ is deterministic $\Longrightarrow$ can only consider additive homoscedasticity addtive process/measurement noise

Application in Control

- Model the system dynamics $x^+ = f(x, u)$ with a Gaussian process model.
- Discover the latent-space of a complex dynamics (**Gaussian Process Latent Variable Model**)
- Controller tuning by modeling the performance index with GP (**Bayesian Optimization**)
- GP-based reinforcement learning, where a GP is used to model the value function/unknown dynamics.
- Model the solution of ODE/SDE as Gaussian process, and then do filtering or inverse optimization.

Computational Issues

Ill-conditioning

The posterior distribution of a GP

$$f(x^*) \sim \mathcal{N}(\overbrace{\mu(x^*) + k_{*X}[K_{XX} + \sigma^2 I]^{-1}(y - \mu_X)}^{\text{posterior mean function}}, \underbrace{k(x^*, x^*) - K_{*X}[K_{XX} + \sigma^2 I]^{-1} K_{X*}}_{\text{posterior kernel function}})$$

When x_i is similar to x_j, the i-th row of K_{XX}

$$K_{XX,i,:} = [k(x_i, x_1), k(x_i, x_2), \ldots, k(k_i, x_N)]$$

is similar (more or less *colinear*) to the j-th row

$$K_{XX,j,:} = [k(x_j, x_1), k(x_j, x_2), \ldots, k(k_j, x_N)] \ .$$

Ill-conditioning

The posterior distribution of a GP

$$f(x^*) \sim \mathcal{N}(\overbrace{\mu(x^*) + k_{*X}[K_{XX} + \sigma^2 I]^{-1}(y - \mu_X)}^{\text{posterior mean function}}, \underbrace{k(x^*, x^*) - K_{*X}[K_{XX} + \sigma^2 I]^{-1} K_{X*}}_{\text{posterior kernel function}})$$

When x_i is similar to x_j, the i-th row of K_{XX}

$$K_{XX,i,:} = [k(x_i, x_1), k(x_i, x_2), \ldots, k(k_i, x_N)]$$

is similar (more or less *colinear*) to the j-th row

$$K_{XX,j,:} = [k(x_j, x_1), k(x_j, x_2), \ldots, k(k_j, x_N)] \ .$$

K_{XX} is ill-conditioned for dense dataset.

$\Longrightarrow$ More data is not-necessarily better in term of numerical stability.

Ill-conditioning

Solutions:

- Use pseudo inverse instead of matrix inverse.
- Add small diagonal matrix to regulate the matrix K_{XX}, (usually good enough with 10^{-6} perturbation)
- Discard similar data points.
- Use iterative algorithms (*e.g.* conjugate gradient) to solve

$$[K_{XX} + \sigma^2 I]^{-1}(y - \mu_X) \ , \ [K_{XX} + \sigma^2 I]^{-1} K_{X*}$$

Computational Issues

The computational cost to calculate the a posterior GP is $O(N^3)$ with a memory cost of $O(N^2)$. The evaluation of a posterior GP is $O(N)$.

The main idea to make GP scalable is to approximate the operations with K_{XX}. There are three main appoaches

- Approximate the big kernel matrix with a smaller kernel matrix: *Sparse GP*
- Acceleration by efficient matrix-vector operations:*Fast GP*
- Approximate the kernel with finite basis functions:*Spectral Methods*

Sparse GP

The **key idea** of all the sparse GP algorithm is to approximate the big dataset $\{x_i\}_{i=1}^N$ with a sparse but informative small dataset $\{u_i\}_{i=1}^M$, coined **inducing points**.

In general, there are mainly three family of sparse GP

- Subset of Regressors (**SOR**)
- Fully-independent Training conditional (**FITC**).
- Variational Free Energy (**VFE**)

Other methods are small modification of these methods.

Sparse GP

The **key idea** of all the sparse GP algorithm is to approximate the big dataset $\{x_i\}_{i=1}^N$ with a sparse but informative small dataset $\{u_i\}_{i=1}^M$, coined **inducing points**.

In general, there are mainly three family of sparse GP

- Subset of Regressors (**SOR**)
- Fully-independent Training conditional (**FITC**).
- Variational Free Energy (**VFE**)

From SoR to FITC

SOR: $K_{XX} \approx K_{XU} K_{UU}^{-1} K_{UX} =: K_{SOR}$
FITC: $K_{XX} \approx K_{SOR} + \text{diag}(K_{XX} - K_{SOR})$

FITC is a regularized SOR. In particular, it models the correct variance at training data point x_i and approximate the covariance among datapoints by SOR.

Sparse GP

The **key idea** of all the sparse GP algorithm is to approximate the big dataset $\{x_i\}_{i=1}^N$ with a sparse but informative small dataset $\{u_i\}_{i=1}^M$, coined **inducing points**.

In general, there are mainly three family of sparse GP

- Subset of Regressors (**SOR**)
- Fully-independent Training conditional (**FITC**).
- Variational Free Energy (**VFE**)

From FITC to VFE

- **FITC**: Maximal a poesteiror: $\max \mathbb{P}(Y|U)$.
- **VFE**: Minimize KL-divergence: $\min \mathcal{KL}(\mathbb{P}(Y|X), \mathbb{P}(Y|U))$

take home messages:

- VFE approximate the posterior GP $\Longrightarrow$ better variance fit
- FITC approximate the posterior inference $\Longrightarrow$ better mean fit (risk of overfit)

Sparse GP

The **key idea** of all the sparse GP algorithm is to approximate the big dataset $\{x_i\}_{i=1}^N$ with a sparse but informative small dataset $\{u_i\}_{i=1}^M$, coined **inducing points**.

In general, there are mainly three family of sparse GP

- Subset of Regressors (**SOR**)
- Fully-independent Training conditional (**FITC**).
- Variational Free Energy (**VFE**)

From VFE to "Big Data"

Training cost per iteration: $O(N^3) \overset{a}{\Longrightarrow} O(NM^2) \overset{b}{\Longrightarrow} O(mM^2)$

a: application of sparse GP

b: application of stochastic variational inference, which is a stochastic gradient descent for variational inference and m is the batch size.

A second example

The Ackley function

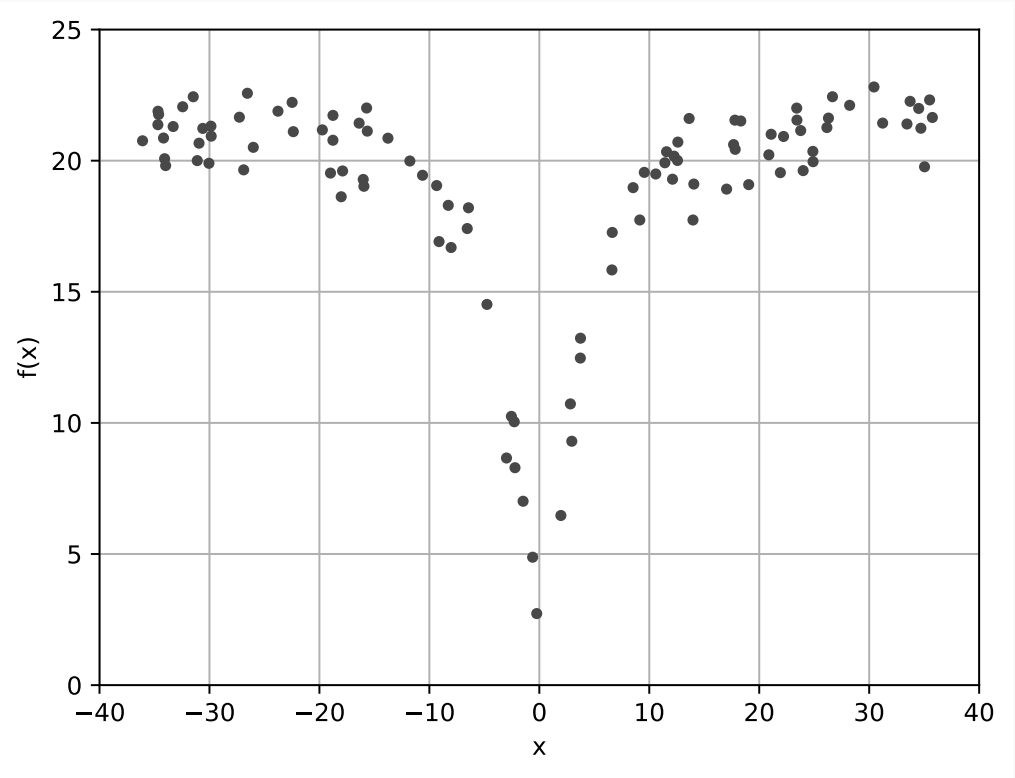
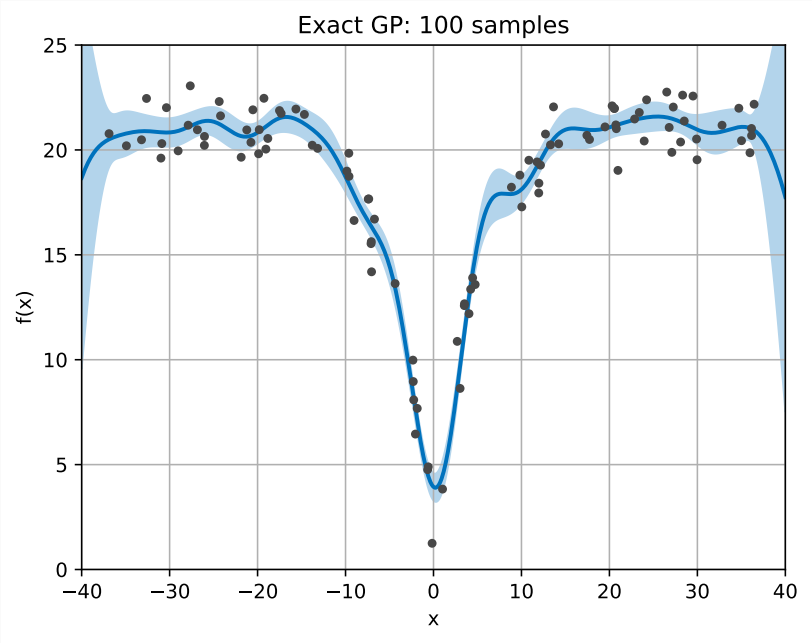
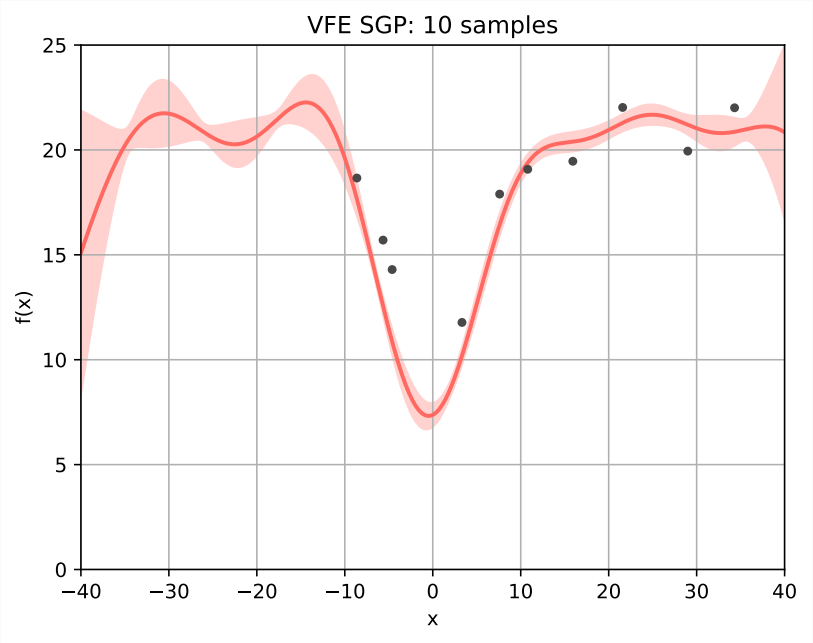


Figure 2: $N = 100$ samples, Gaussian likelihood with $\sigma^2 = 0.5$

A second example

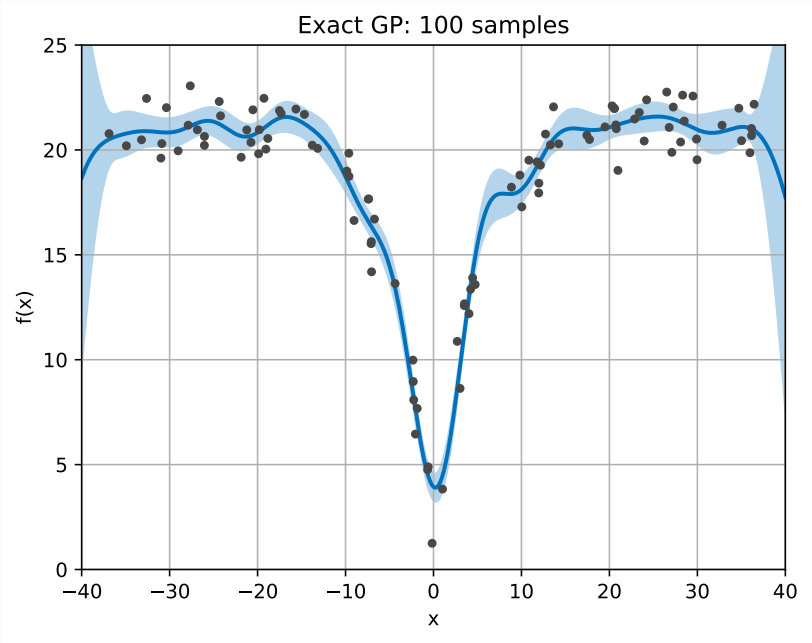
Exact VS sparse

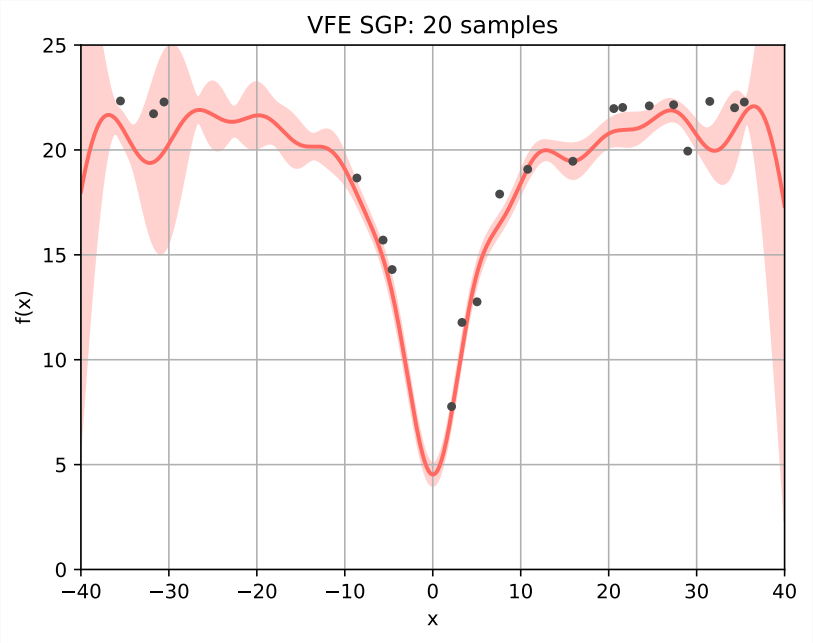




A second example

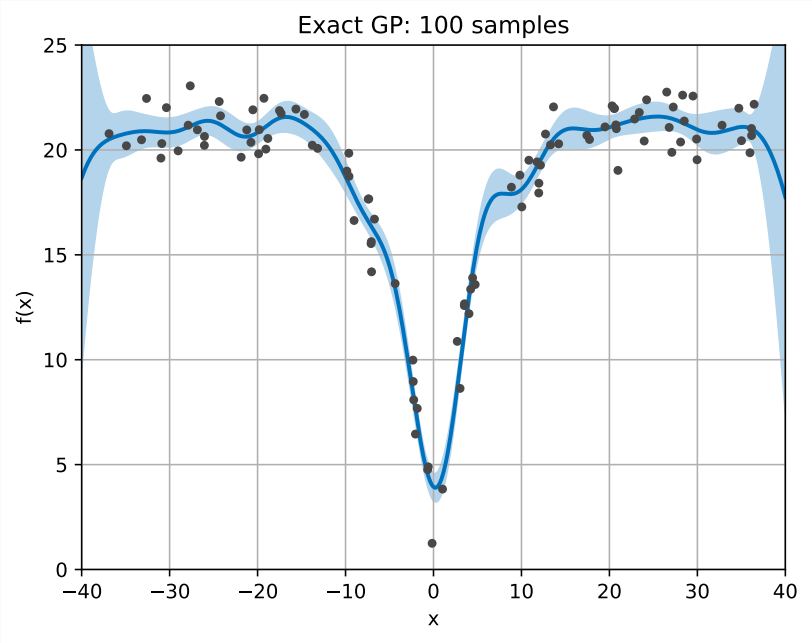
Exact VS sparse

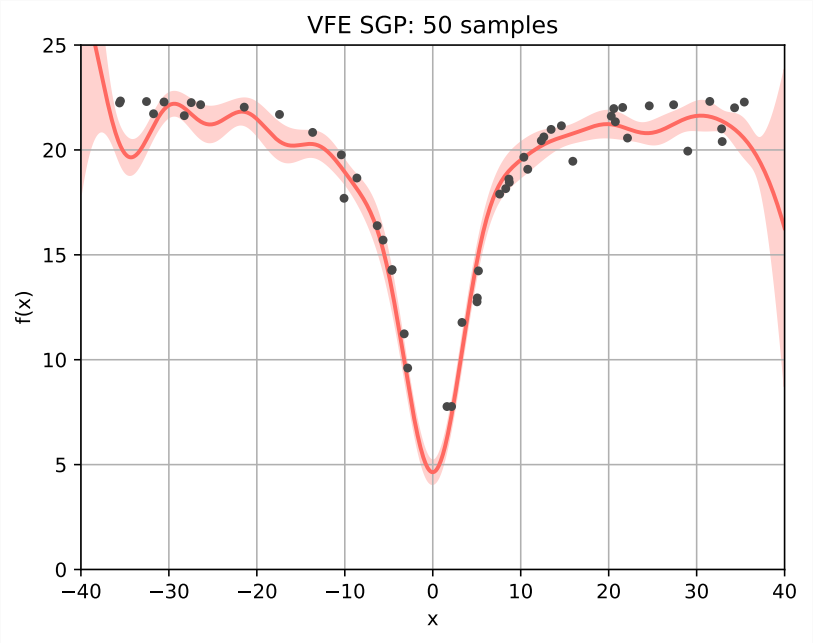




A second example

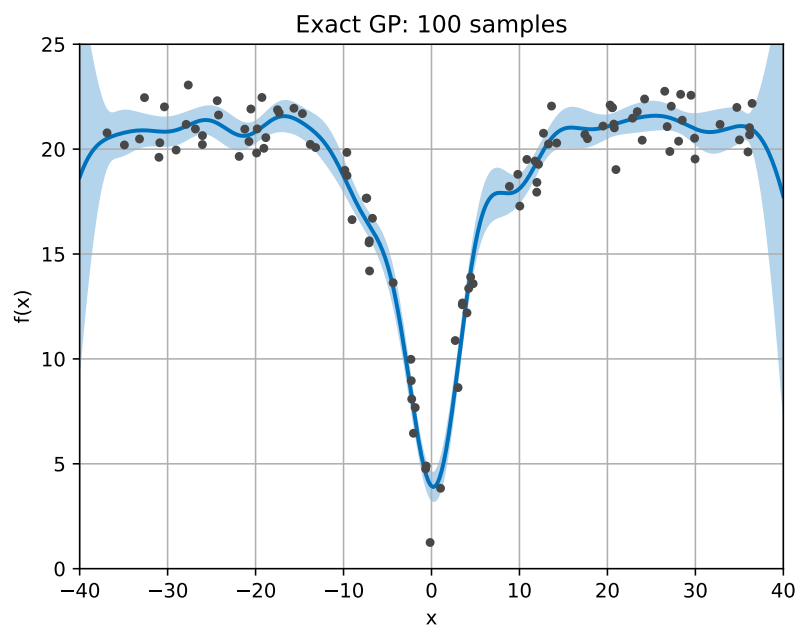
Exact VS sparse

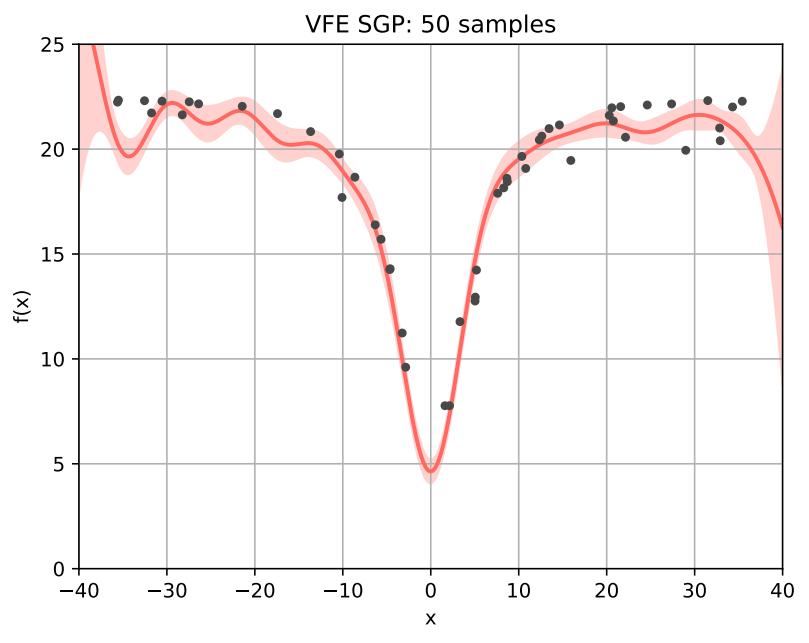




A second example

Exact VS sparse





Keep in mind, SPGs are still non-parametric ☺

How many inducing points? What are the SGPs training objectives?

Are they harder to train? Can we compare likelihoods?

Takeaway messages:

It does speed up computations a lot!

time: $\mathcal{O}(N^3) \to \mathcal{O}(NM^2)$, **memory**: $\mathcal{O}(N^2) \to \mathcal{O}(NM)$

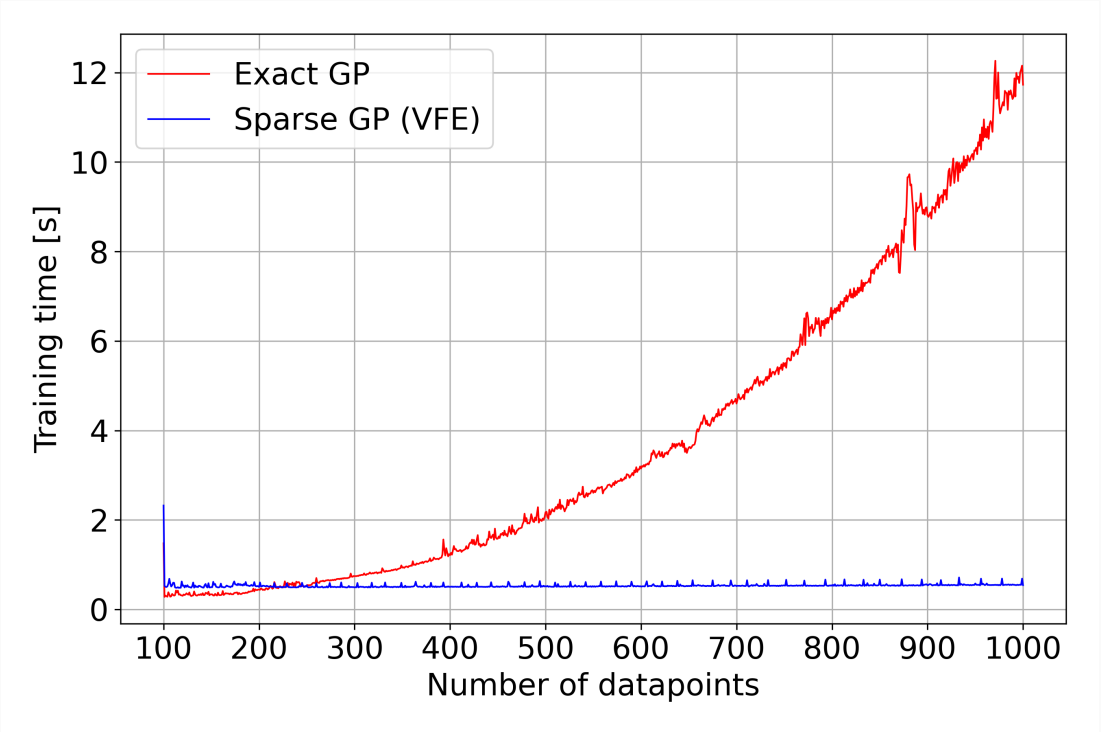


Figure 3: Matern32 kernel, rnd initialization, L-BFGS (5'000 iterations)

Sparse GP

Takeaway messages:

It does speed up computations a lot!

time: $\mathcal{O}(N^3) \to \mathcal{O}(NM^2)$, **memory**: $\mathcal{O}(N^2) \to \mathcal{O}(NM)$

Parsing the literature requires some statistical training! ☹☹☹

But packages are easy to use (`GPy`, `GPFlow` and `GPytorch`) ☺☺☺

Fast GP

The main computational bottleneck of GP comes from the matrix inversion K_{XX}^{-1}. There are three main approaches

- Nyström preconditioned conjugate gradient ($O(N\sqrt{N}\log(N))$).
- Matrix-matrix Lancoz predictioned conjugate gradient ($O(N^2)$)
- structural interpolation (Down to $O(M)$).

Fast GP

The main computational bottleneck of GP comes from the matrix inversion K_{XX}^{-1}. There are three main approaches

- Nyström preconditioned conjugate gradient ($O(N\sqrt{N}\log(N))$).
- Matrix-matrix Lancoz predictioned conjugate gradient ($O(N^2)$)
- structural interpolation (Down to $O(M)$).

Nyström based conjugate gradient

In-place Cholesky decomposition of the kernel matrix K_{XX} and then run conjugate gradient on the linear system predictioned by the Cholesky decompostion

- Customized for GPU with efficient memory allocation
- Gradient calculation requires extra memory allocation.

Fast GP

The main computational bottleneck of GP comes from the matrix inversion K_{XX}^{-1}. There are three main approaches

- Nyström preconditioned conjugate gradient ($O(N\sqrt{N}\log(N))$).
- Matrix-matrix Lancoz predictioned conjugate gradient ($O(N^2)$)
- structural interpolation (Down to $O(M)$).

Matrix-matrix based conjugate gradient

Convert the determinant and trace calculation to maxtrix-matrix product by Rademacher stochastic estimator. Solve matrix-matrix multiplication system by conjugate gradient preconditioned by pivoted Cholesky low rank approximation.

- Customized for GPU with efficient memory allocation
- Compatible with any sparse GP algorithms
- Available in `GPytorch`
- Compatible with product kernel acceleration

Fast GP

The main computational bottleneck of GP comes from the matrix inversion K_{XX}^{-1}. There are three main approaches

- Nyström preconditioned conjugate gradient ($O(N\sqrt{N}\log(N))$).
- Matrix-matrix Lancoz predictioned conjugate gradient ($O(N^2)$)
- structural interpolation (Down to $O(M)$).

Structural Interpolation

Choose inducing point on equidistant grids and approximate the original kernel matrix by: $K_{XX} \approx WK_{UU}W$ with W an (known) interpolation matrix. A kernel matrix defined on a grid can be decomposed by kronecker product arithmetic and each submatrix is evaluted efficiently by Toeplitz arthmetic (*e.g.* matrix inversion is equivalent to an FFT).

- Down to $O(M)$ complexity
- Only applicable to low dimensional, compact dataset.

Spectral Methods

Given a stationary kernel ($k(x_1, y_1) = k(x_2, y_2)$ if $x_1 - y_1 = x_2 - y_2$), it is uniquely characterized by its spectrum (*Bochner theoremz*).

The **key idea** of spectral method is to use finite Fourier series to approximate the infinite feature maps of the chosen kernel.

$\Longrightarrow$ equivalent to Generalized Bayesian linear regression and the computational cost is $O(m^3)$ with m the number of frequency components.

Remark

- The frequency components are optimized in a way similar to **SOR** or **FITC**. The "**VFE**"-style spectral method is defined on compact set.
- The Fourier series approximation converge uniformly, hence has great practical performance.

Fourier Series or eigen functions?

As Fourier series is actually eigenfunction w.r.t the Lebeges measure, it is also possible to approximate kernel with other eigenfunctions. However, up to what I know, most eigenfunctions are non-trivial. For example, the eigenfunction of the RBF kernel w.r.t the Gaussian distribution is a weighted Gamma function.

Some of our kernel-related works

Deterministic error bounds

The dataset $\{(x_i, y_i)\}_{i=1}^d$

Observational model $y_i = f(x_i) + \delta_i$

Uniformly bounded noise $|\delta_i| \leq \bar{\delta}$

Then, computing

$$C(x) = \sup_{f \in \mathcal{H}} \{f(x) \,:\, |f(x_i) - y_i| \leq \bar{\delta},\, \forall i,\, \|f\|_{\mathcal{H}} \leq \Gamma\}$$

Is equivalent to solving the QCQP

$$\begin{aligned} C(x) \;=\; \max_{c \in \mathbb{R}^d, c_x \in \mathbb{R}} \quad & c_x \\ \text{subj. to} \quad & \begin{bmatrix} c \\ c_x \end{bmatrix}^\top \begin{bmatrix} K_{XX} & K_{Xx} \\ K_{xX} & k(x,x) \end{bmatrix}^{-1} \begin{bmatrix} c \\ c_x \end{bmatrix} \leq \Gamma^2 \\ & \|c - \mathrm{y}\|_\infty \leq \bar{\delta} \end{aligned}$$

Deterministic error bounds

How to come up with $\Gamma \geq \|f\|_{\mathcal{H}}$?

The norm can be estimated from data $\{(x_i, f(x_i))\}_{i=1}^{d}$ since:

- $\hat{\Gamma} := f_X^\top K_{XX}^{-1} f_X$ increases with d
- $\hat{\Gamma} \leq \|f\|_{\mathcal{H}}^2$, for any d

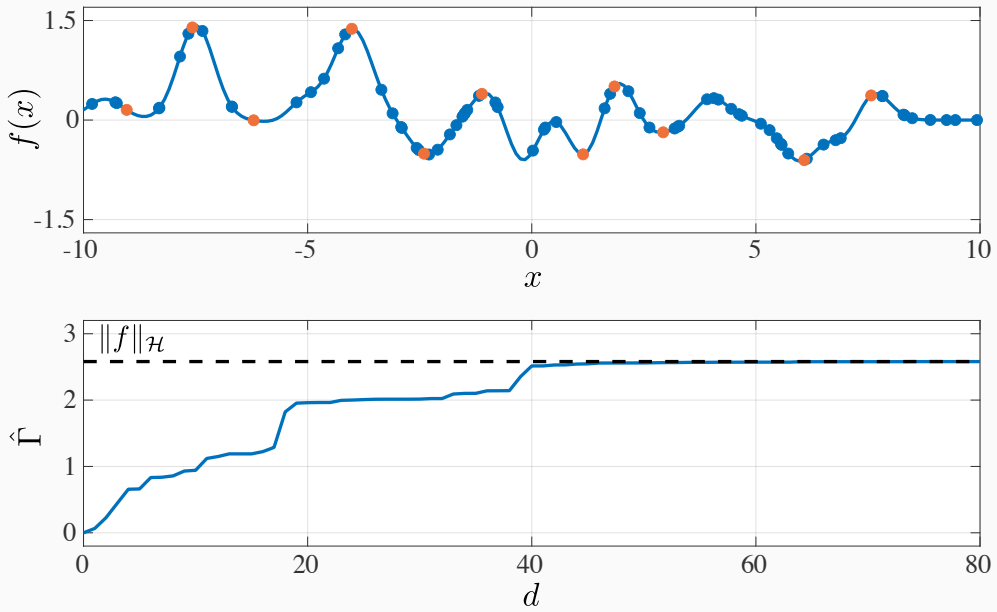


Figure 4: The orange samples alone were responsible for $> 90\%$ of the final value

Deterministic error bounds

You can also define first a nominal model

- A plain interpolant
- A kernel ridge regression (KRR) model
- A minimum-norm support-vector regression (SVR) model

and then derive deterministic closed-form error-bounds for them.

Bounds centered around the nominal predictions

Always more conservative than the optimal algorithm-independent bounds

Deterministic error bounds

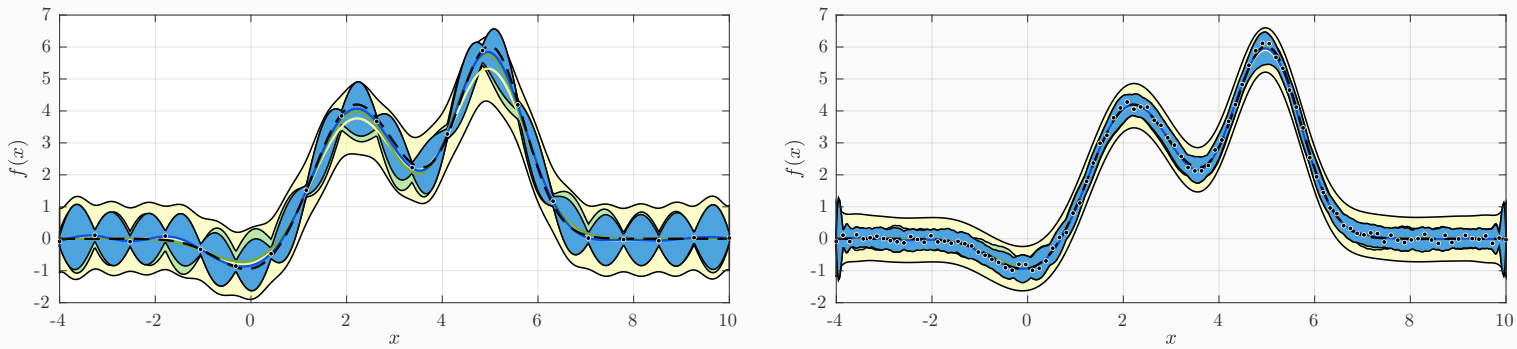


Figure 5: A comparison of some closed-form bounds (KRR and SVR)

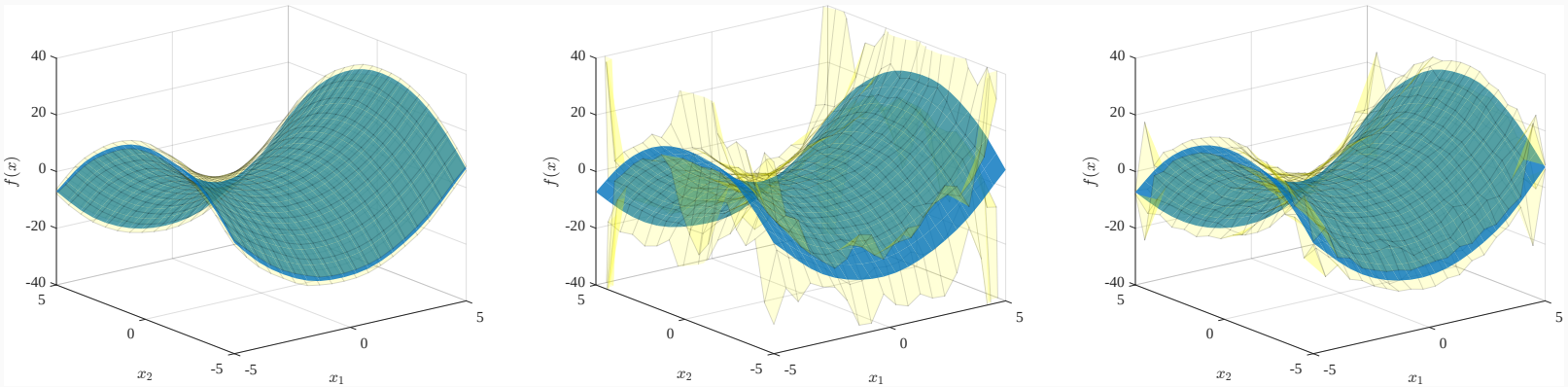


Figure 6: The effects of the dataset distribution (regular is always better ☺)

Deterministic error bounds

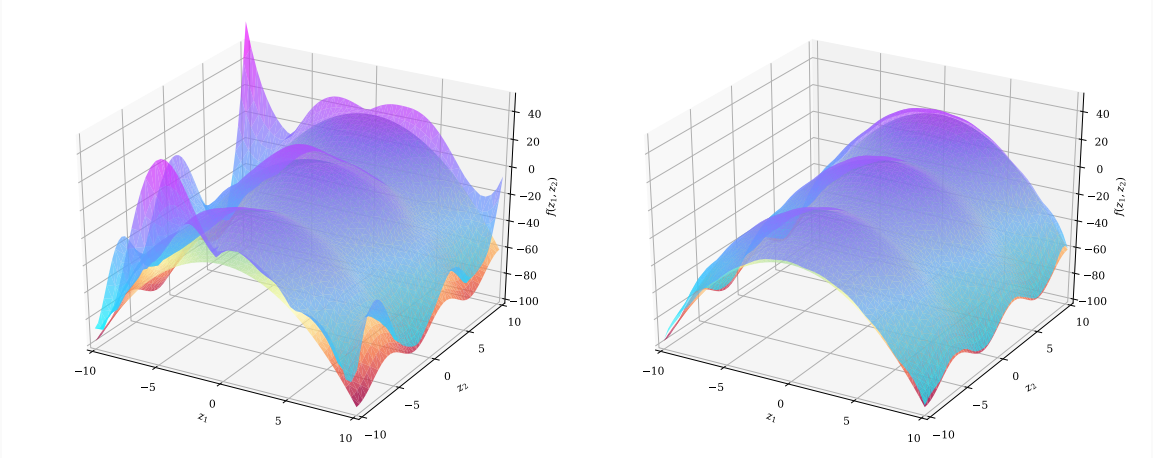


Figure 7: Hénon map and the optimal bounds with 100 samples and $\bar{\delta} = 1$

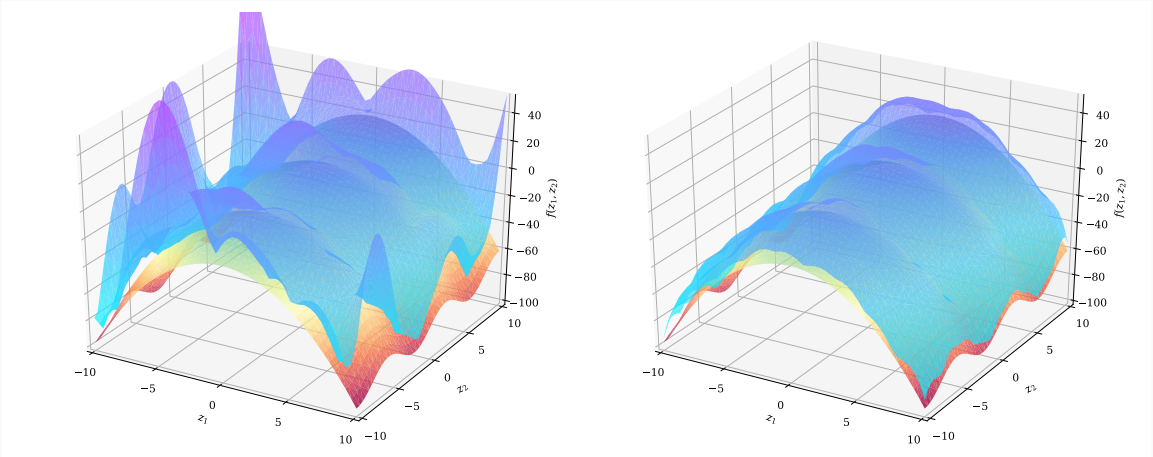


Figure 8: Hénon map and the optimal bounds with 100 samples and $\bar{\delta} = 5$

Koopman based GP

Dynamics: $x^+ = f(x)$, **Koopman Operator**: $\mathcal{K}g := g \circ f(\cdot)$

A Koopman operator models the evolution of a function driven by system dynamics f and it is a **linear** operator

fact: A GP remains a GP undergoing a linear operator.

$\Longrightarrow$ if $f \sim \mathcal{GP}(\mu(\cdot), k(\cdot))$, then $\mathcal{K}f$ is a Gaussian process.

Main contribution: Learn the distribution of the Koopman operator from input-output data

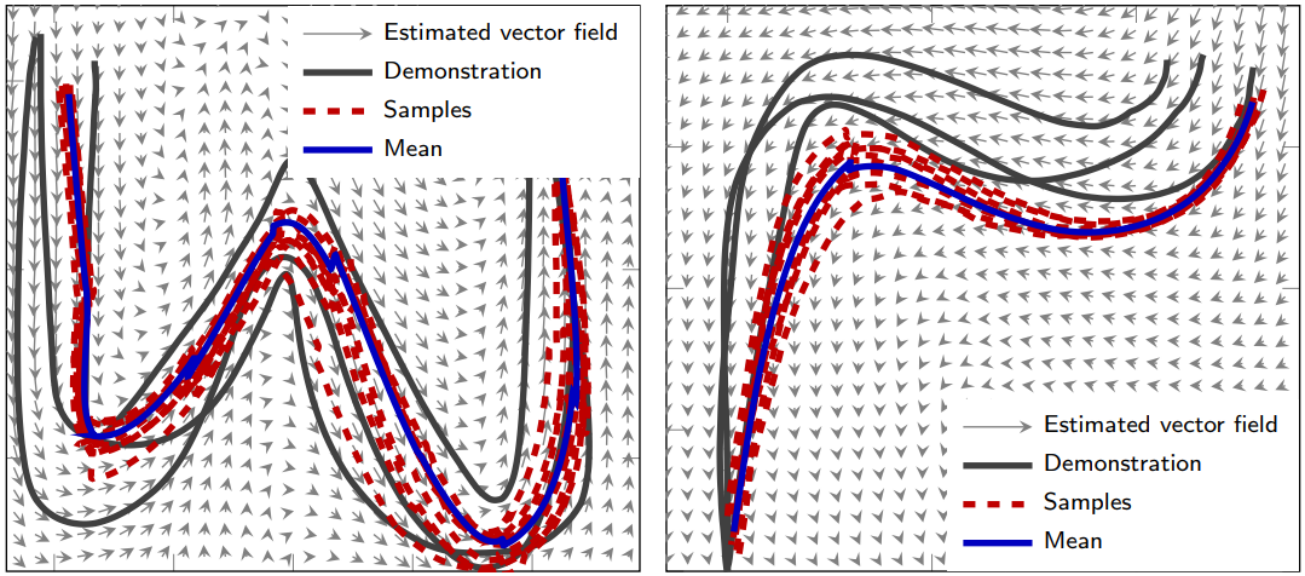


Figure 9: Learn to write characters from 3 demos

The main trick:

If you have the following linear equations

$$Ax = a$$

The solution of x satisfies

$$x^\top (A^\top A)x - 2x^\top A^\top a + a^\top a$$

.

If each column of A is a datapoint, then entries of $A^\top A$ and $A^\top a$ are inner products of datapoints. Therefore, we can apply the **kernel trick**.

Main contribution: Apply this trick to the fundamental lemma and build the theory around this.

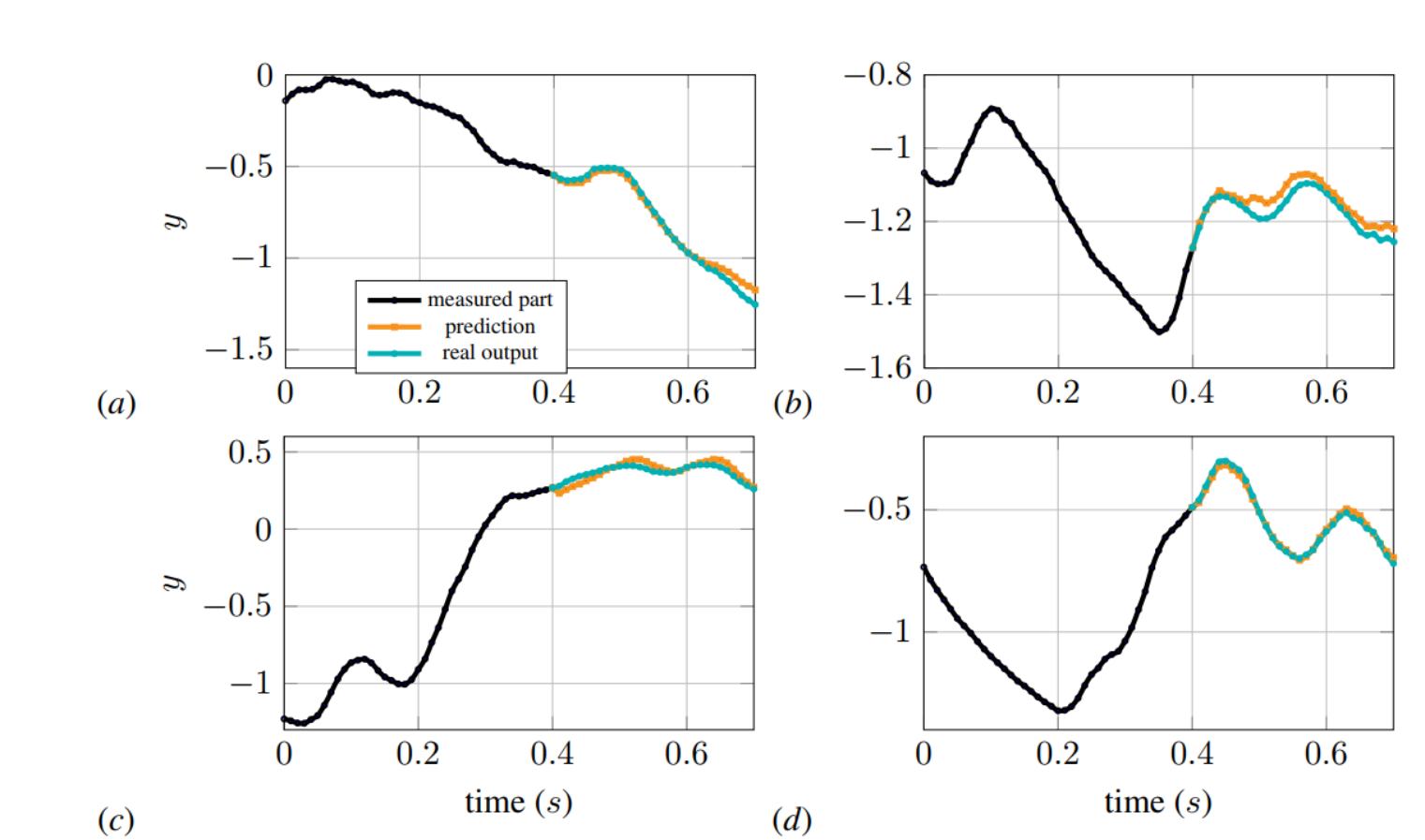


Figure 10: Open-loop prediction of a bi-linear motor dynamics