Approximate Inference, The Right Way

John Cunningham

PN School — 27 March 2023

Outline

- The Promise of Probabilistic Machine Learning
- Gaussian Process Introduction
- Scaling Gaussian Processes, and Implications
- Approximate Gaussian Process Inference, The Right Way
- iterGP as Probabilistic Numerics
- Broader Implications

Probabilistic Machine Learning

- Fundamental premise: *treat **all** unknown quantities as random variables.* Bayes Rule does the rest

belief about the state of the world

$p(f)$

how data arises from a state of the world

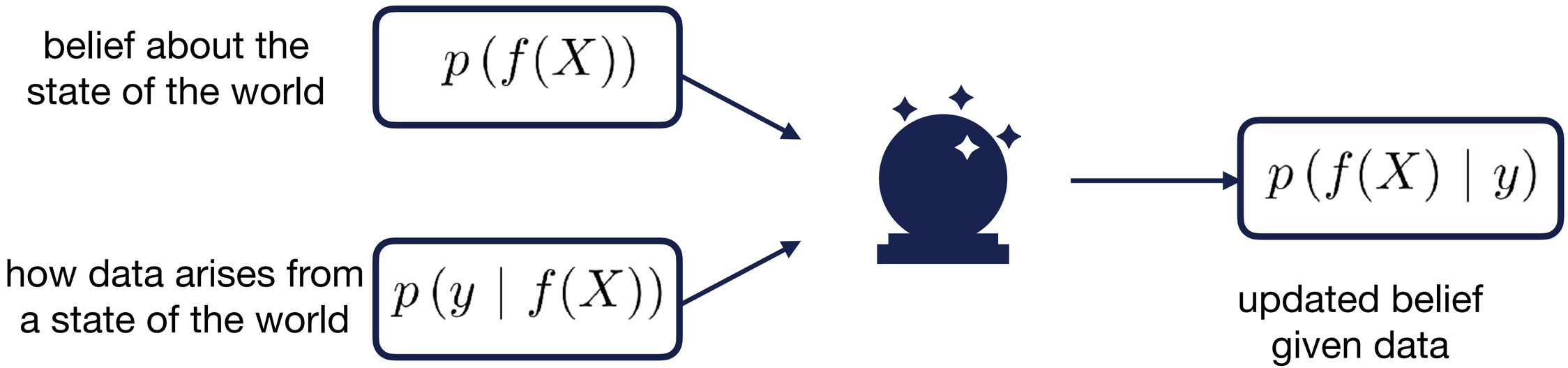
$p(y|f)$

$p(f|y)$

updated belief given data

Probabilistic Machine Learning

- Fundamental premise: *treat **all** unknown quantities as random variables.* Bayes Rule does the rest



Approximate Inference

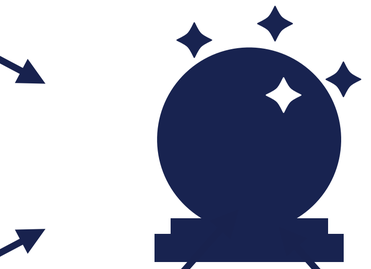
- But this is hard! As soon as you learn the magic of Bayes Rule, you learn that we need to approximately solve it.

belief about the state of the world

$p(f)$

how data arises from a state of the world

$p(y|f)$



$\approx p(f|y)$

updated belief given data

PYRO

Approximate Inference

- But this is hard! As soon as you learn the magic of Bayes Rule, you learn that we need to approximately solve it.

belief about the state of the world

$p(f)$

how data arises from a state of the world

$p(y|f)$

$\approx p(f|y)$

updated belief given data

Approximate Inference

- The crystal ball is actually a computing machine, making assumptions and choices of its own, but we have not accounted for them! All our assumptions, all our unknowns… we were supposed to reason about them probabilistically.

belief about the state of the world

$p(f)$

how data arises from a state of the world

$p(y|f)$

$\approx p(f|y)$

updated belief given data (and some other stuff?)

Approximate Inference

- This statement may seem inherent in the definition of “approximate”, but it is not! The entire point of this talk is to pay off this claim.

belief about the state of the world

$p(f)$

how data arises from a state of the world

$p(y|f)$

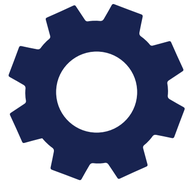
$\approx p(f|y)$

updated belief given data

today: how to rationalize approximate computation probabilistically

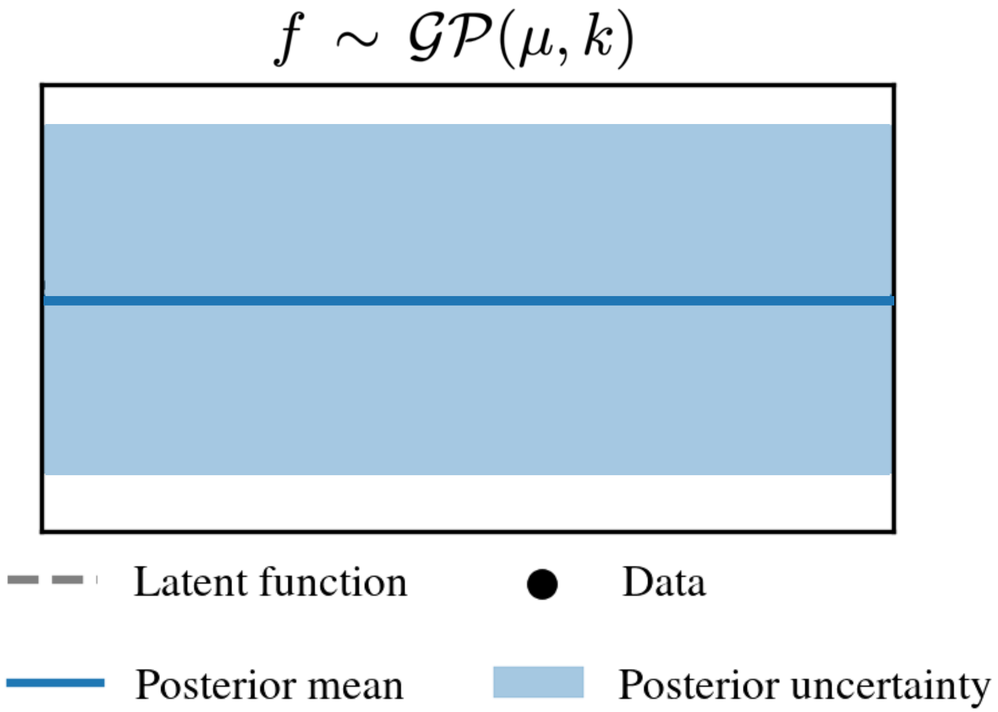
Outline

- The Promise of Probabilistic Machine Learning
- **Gaussian Process Introduction**
- Scaling Gaussian Processes, and Implications
- Approximate Gaussian Process Inference, The Right Way
- iterGP as Probabilistic Numerics
- Broader Implications



Gaussian Process Prior

- Setup:
  - Learn a function $h : \mathcal{X} \to \mathcal{Y}$
  - Training inputs $\boldsymbol{X} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n) \in \mathbb{R}^{n \times d}$
  - Training outputs $\boldsymbol{y} = (y_1, \ldots, y_n)^\intercal \in \mathbb{R}^n$



- Gaussian Process: $f \sim \mathcal{GP}(\mu, k)$
  - Mean function $\mu : \mathbb{R}^d \to \mathbb{R}$
  - Covariance kernel $k : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$
  - Joint Gaussian for all $\boldsymbol{X}$: $\mathbf{f} = (f(\boldsymbol{x}_1), \ldots, f(\boldsymbol{x}_n))^\intercal \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{K})$

$\boldsymbol{K}_{ij} = k(\boldsymbol{x}_i, \boldsymbol{x}_j)$

$\boldsymbol{\mu}_j = \mu(\boldsymbol{x}_j)$

Gaussian Process Posterior

- With likelihood: $\boldsymbol{y} \mid \mathbf{f} \sim \mathcal{N}(\mathbf{f}, \sigma^2 \boldsymbol{I})$
- And test inputs: $\boldsymbol{X}_\diamond$

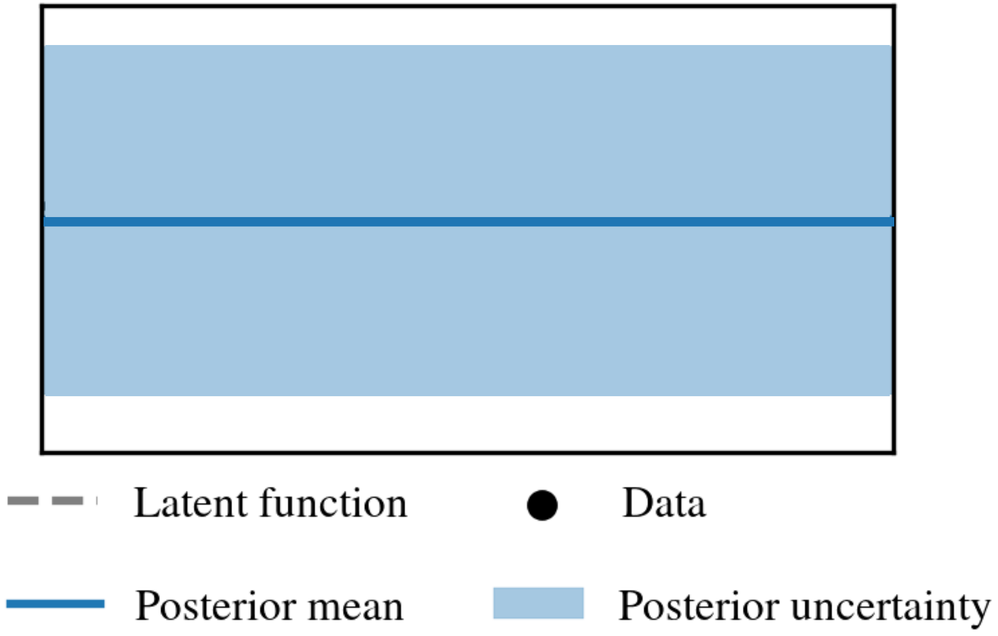
- Induces the posterior:

$$\mathbf{f}_\diamond \sim \mathcal{N}(\mu_*(\boldsymbol{X}_\diamond), k_*(\boldsymbol{X}_\diamond, \boldsymbol{X}_\diamond))$$

  - where: $\mu_*(\cdot) = \mu(\cdot) + k(\cdot, \boldsymbol{X})\, \hat{\boldsymbol{K}}^{-1}(\boldsymbol{y} - \boldsymbol{\mu})$

$$k_*(\cdot, \cdot) = k(\cdot, \cdot) - k(\cdot, \boldsymbol{X})\hat{\boldsymbol{K}}^{-1}k(\boldsymbol{X}, \cdot)$$

$$\hat{\boldsymbol{K}} := \boldsymbol{K} + \sigma^2 \boldsymbol{I} \in \mathbb{R}^{n \times n}$$



Gaussian Process Posterior

- With likelihood: $\boldsymbol{y} \mid \mathbf{f} \sim \mathcal{N}(\mathbf{f}, \sigma^2 \boldsymbol{I})$
- And test inputs: $\boldsymbol{X}_\diamond$

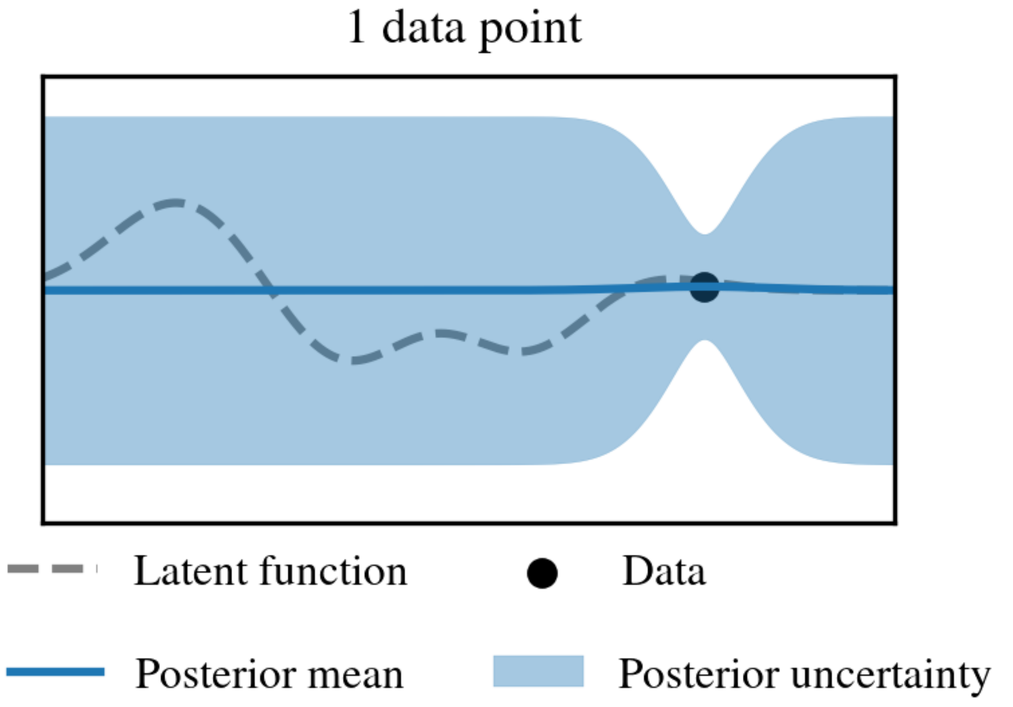
- Induces the posterior:

$$\mathbf{f}_\diamond \sim \mathcal{N}(\mu_*(\boldsymbol{X}_\diamond), k_*(\boldsymbol{X}_\diamond, \boldsymbol{X}_\diamond))$$

  - where: $\mu_*(\cdot) = \mu(\cdot) + k(\cdot, \boldsymbol{X})\, \hat{\boldsymbol{K}}^{-1}(\boldsymbol{y} - \boldsymbol{\mu})$

$$k_*(\cdot, \cdot) = k(\cdot, \cdot) - k(\cdot, \boldsymbol{X})\hat{\boldsymbol{K}}^{-1}k(\boldsymbol{X}, \cdot)$$

$$\hat{\boldsymbol{K}} := \boldsymbol{K} + \sigma^2 \boldsymbol{I} \in \mathbb{R}^{n \times n}$$



Gaussian Process Posterior

- With likelihood: $\boldsymbol{y} \mid \mathbf{f} \sim \mathcal{N}(\mathbf{f}, \sigma^2 \boldsymbol{I})$
- And test inputs: $\boldsymbol{X}_\diamond$

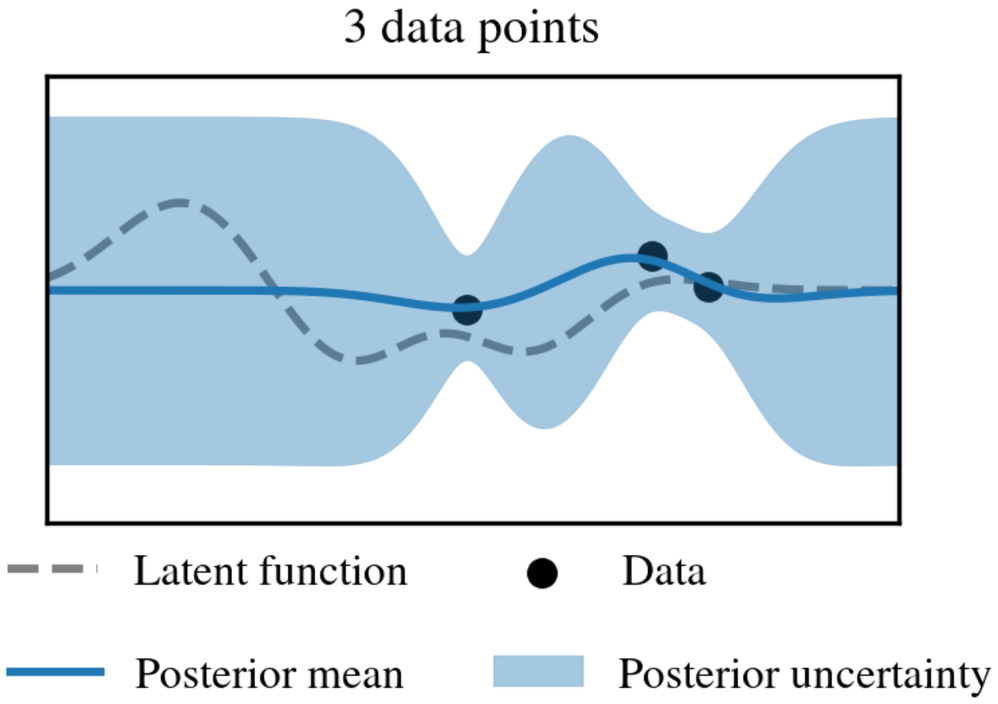
- Induces the posterior:

$$\mathbf{f}_\diamond \sim \mathcal{N}(\mu_*(\boldsymbol{X}_\diamond), k_*(\boldsymbol{X}_\diamond, \boldsymbol{X}_\diamond))$$

  - where: $\mu_*(\cdot) = \mu(\cdot) + k(\cdot, \boldsymbol{X})\, \hat{\boldsymbol{K}}^{-1}(\boldsymbol{y} - \boldsymbol{\mu})$

$$k_*(\cdot, \cdot) = k(\cdot, \cdot) - k(\cdot, \boldsymbol{X})\hat{\boldsymbol{K}}^{-1}k(\boldsymbol{X}, \cdot)$$

$$\hat{\boldsymbol{K}} := \boldsymbol{K} + \sigma^2 \boldsymbol{I} \in \mathbb{R}^{n \times n}$$



Gaussian Process Posterior

- With likelihood: $\boldsymbol{y} \mid \mathbf{f} \sim \mathcal{N}(\mathbf{f}, \sigma^2 \boldsymbol{I})$
- And test inputs: $\boldsymbol{X}_\diamond$

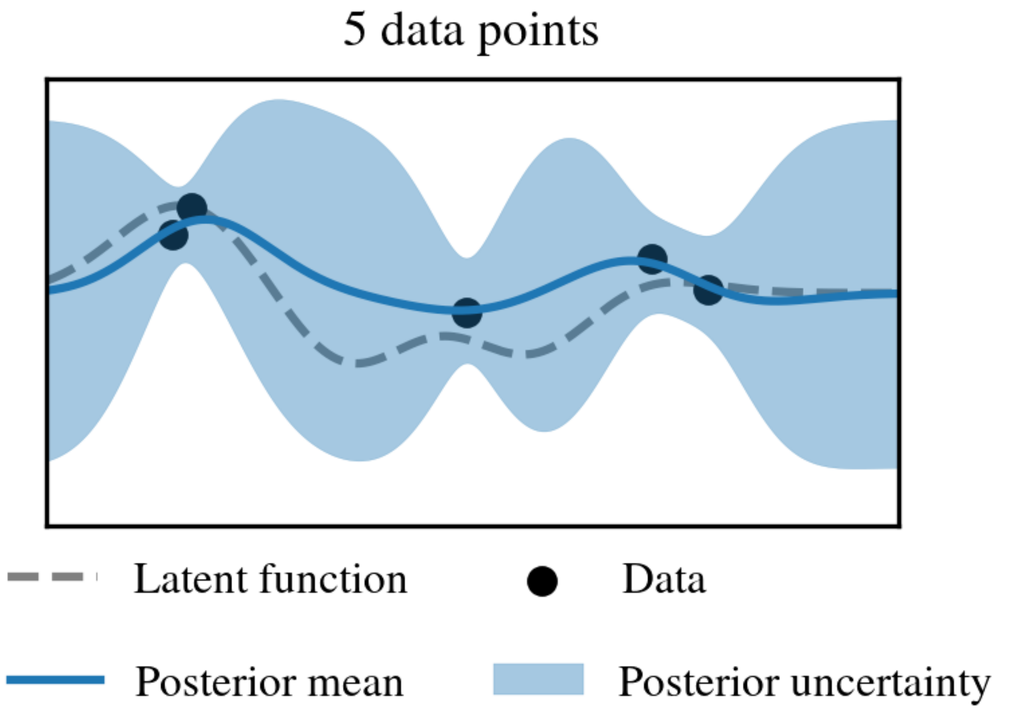
- Induces the posterior:

$$\mathbf{f}_\diamond \sim \mathcal{N}(\mu_*(\boldsymbol{X}_\diamond), k_*(\boldsymbol{X}_\diamond, \boldsymbol{X}_\diamond))$$

  - where: $\mu_*(\cdot) = \mu(\cdot) + k(\cdot, \boldsymbol{X})\, \hat{\boldsymbol{K}}^{-1}(\boldsymbol{y} - \boldsymbol{\mu})$

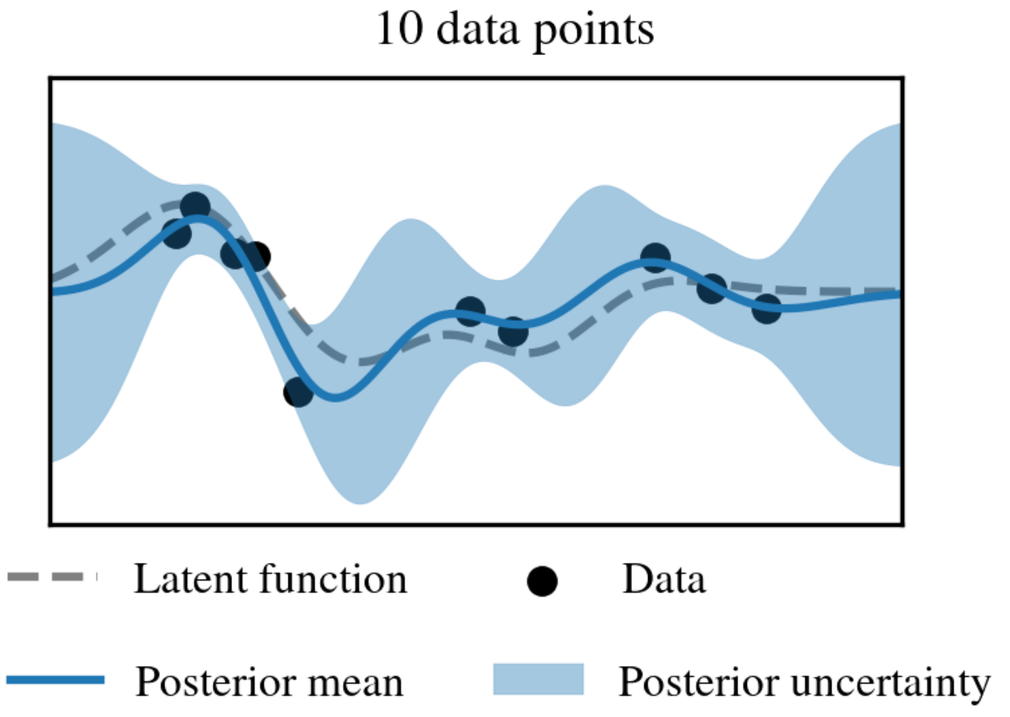
$$k_*(\cdot, \cdot) = k(\cdot, \cdot) - k(\cdot, \boldsymbol{X})\hat{\boldsymbol{K}}^{-1}k(\boldsymbol{X}, \cdot)$$

$$\hat{\boldsymbol{K}} := \boldsymbol{K} + \sigma^2 \boldsymbol{I} \in \mathbb{R}^{n \times n}$$



Gaussian Process Posterior

- With likelihood: $\boldsymbol{y} \mid \mathbf{f} \sim \mathcal{N}(\mathbf{f}, \sigma^2 \boldsymbol{I})$
- And test inputs: $\boldsymbol{X}_\diamond$



- Induces the posterior:

$$\mathbf{f}_\diamond \sim \mathcal{N}(\mu_*(\boldsymbol{X}_\diamond), k_*(\boldsymbol{X}_\diamond, \boldsymbol{X}_\diamond))$$

  - where: $\mu_*(\cdot) = \mu(\cdot) + k(\cdot, \boldsymbol{X})\, \hat{\boldsymbol{K}}^{-1}(\boldsymbol{y} - \boldsymbol{\mu})$

$$k_*(\cdot, \cdot) = k(\cdot, \cdot) - k(\cdot, \boldsymbol{X})\hat{\boldsymbol{K}}^{-1}k(\boldsymbol{X}, \cdot)$$

$$\hat{\boldsymbol{K}} := \boldsymbol{K} + \sigma^2 \boldsymbol{I} \in \mathbb{R}^{n \times n}$$

- We will have considerable work to deal with this linear solve

GP Weight Space View

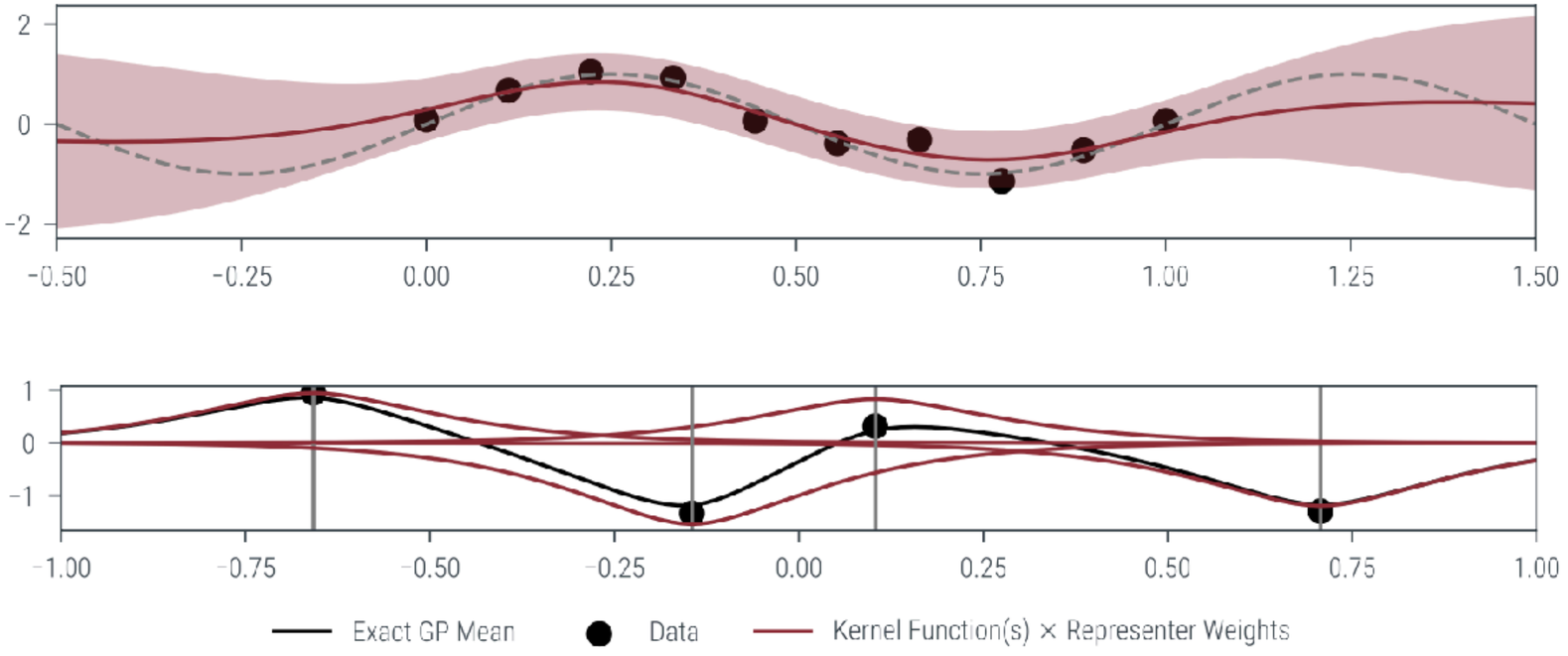
- Let us also consider the *representer weights* $v_* = \hat{K}^{-1}(y - \mu)$

$$\mu_*(\cdot) = \mu(\cdot) + k(\cdot, X)\overbrace{\hat{K}^{-1}(y - \mu)}^{v_*}$$

- View as kernel basis function regression: place a kernel shaped bump at each input, and the representer weights are the regression coefficients that map that onto the GP posterior mean.

- We are going to spend time reasoning over these weights

- (If we think of linear solves probabilistically, then a posterior on our representer weights will provide exactly what we need…)



[Bosch et al 2022]

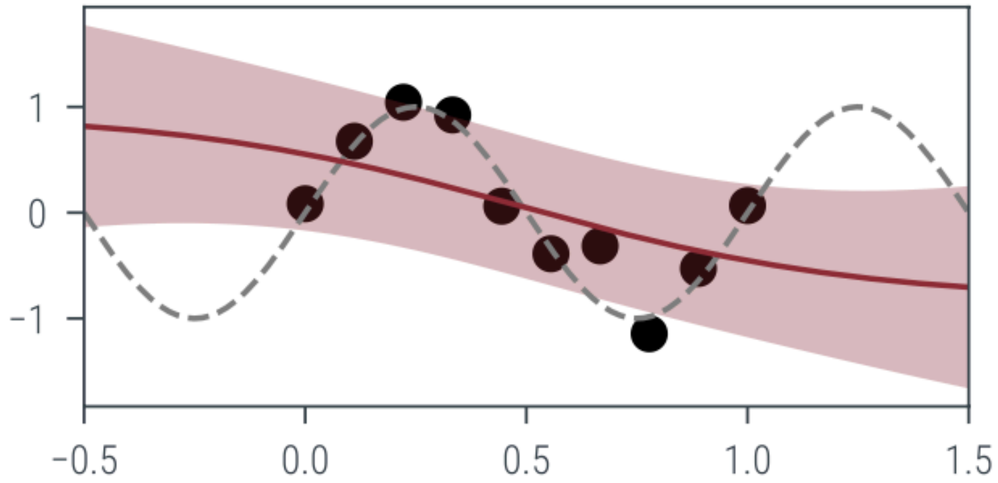
For Completeness: Learning GP Hyperparameters

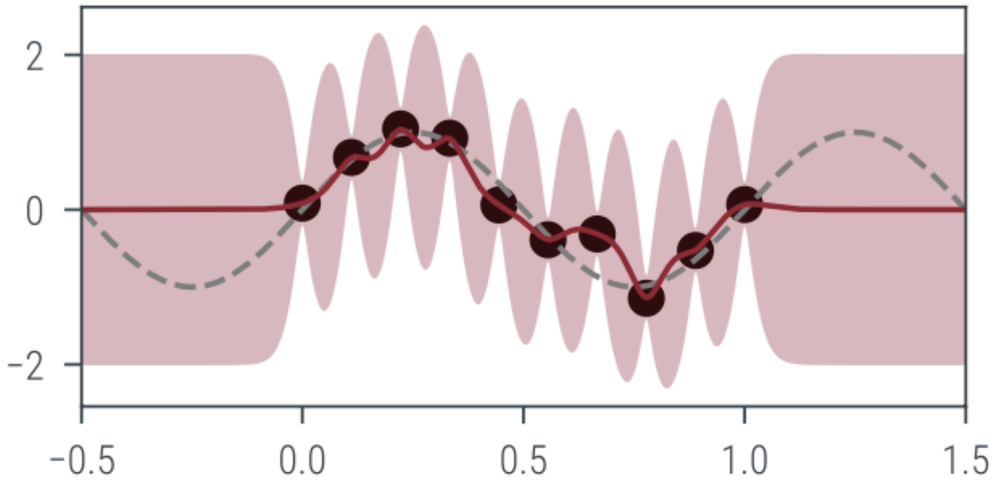
- Of course, a huge part of GP is learning the hyperparams: model selection or training
- There's a ton of great work on how to do this, and how to scale it too.

[Gardner et al 2018]
(and many others)

$$\arg\max_{\theta} \mathcal{L}(\theta) = \arg\max_{\theta} -\frac{1}{2}(\boldsymbol{y}-\boldsymbol{\mu})^{\intercal}\hat{\boldsymbol{K}}^{-1}(\boldsymbol{y}-\boldsymbol{\mu}) - \frac{1}{2}\log|\hat{\boldsymbol{K}}|$$

"data fit" "model complexity"

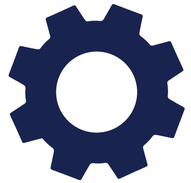




[Bosch et al 2022]

Outline

- The Promise of Probabilistic Machine Learning
- Gaussian Process Introduction
- **Scaling Gaussian Processes, and Implications**
- Approximate Gaussian Process Inference, The Right Way
- iterGP as Probabilistic Numerics
- Broader Implications



Enter Approximate Inference

- The core GP object (as far as computation is concerned): $\hat{\boldsymbol{K}} := \boldsymbol{K} + \sigma^2 \boldsymbol{I} \in \mathbb{R}^{n \times n}$
- It appears over and over:

$$\mu_*(\cdot) = \mu(\cdot) + k(\cdot, \boldsymbol{X})\, \hat{\boldsymbol{K}}^{-1}(\boldsymbol{y} - \boldsymbol{\mu})$$

$$k_*(\cdot, \cdot) = k(\cdot, \cdot) - k(\cdot, \boldsymbol{X}) \hat{\boldsymbol{K}}^{-1} k(\boldsymbol{X}, \cdot)$$

$$\arg\max_{\theta} -\frac{1}{2}(\boldsymbol{y} - \boldsymbol{\mu})^\intercal \hat{\boldsymbol{K}}^{-1}(\boldsymbol{y} - \boldsymbol{\mu}) - \frac{1}{2}\log|\hat{\boldsymbol{K}}|$$

- Generally these computations are all cubic in time (and quadratic in storage)
  - Much literature thus to deal with this cubic scaling
  - (And note that in cases of special structure, notably d=1,2,3, this cost can be much reduced)

[Karvonen and Sarkka 2016 IEEE]
[Loper et al 2021 JMLR]
[Greengard et al 2022 arXiv]

Fast Inference 1: Conjugate Gradients

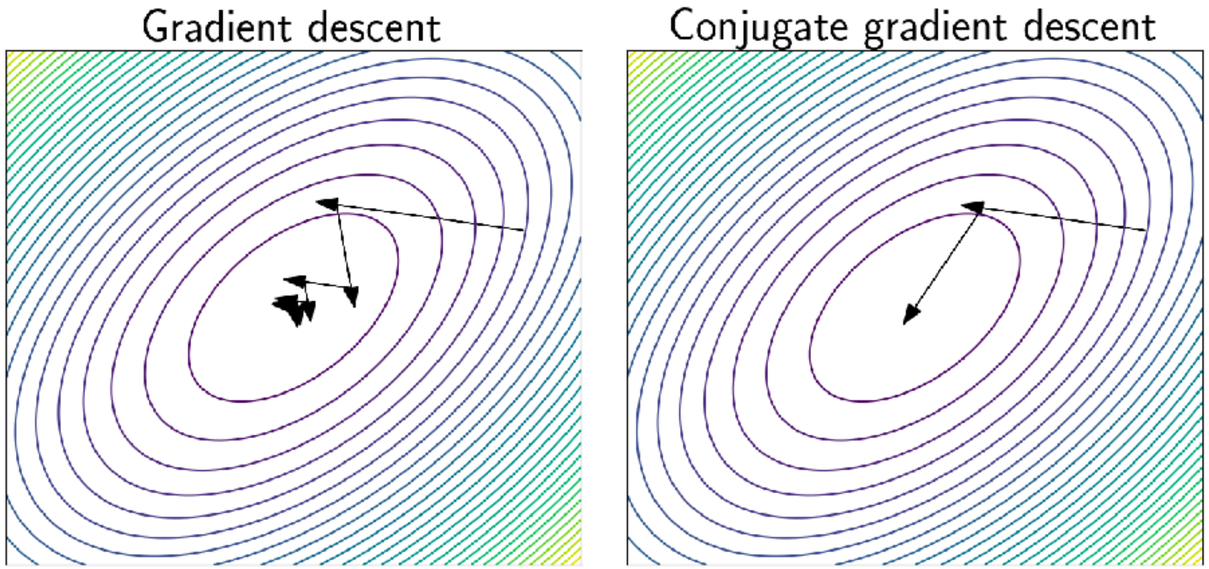
- View linear solves as an optimization:

$$\min_{\boldsymbol{x}} \frac{1}{2}\boldsymbol{x}^\intercal \hat{\boldsymbol{K}}\boldsymbol{x} + \boldsymbol{x}^\intercal(\boldsymbol{y}-\boldsymbol{\mu})$$

- Gradient descent is slow, so instead take *conjugate* gradient steps, namely steps in $\hat{\boldsymbol{K}}$ norm.

- Guaranteed to converge in n steps, but in practice converges to high precision much faster

- As with many (all) other fast linear solvers, it operates only with forward multiplies of form $\hat{\boldsymbol{K}}\boldsymbol{x}$



[Andy Jones]

Fast Inference 1: Conjugate Gradients

- View linear solves as an optimization:

$$\min_{\boldsymbol{x}} \frac{1}{2}\boldsymbol{x}^\intercal \hat{\boldsymbol{K}}\boldsymbol{x} + \boldsymbol{x}^\intercal(\boldsymbol{y}-\boldsymbol{\mu})$$

- Gradient descent is slow, so instead take *conjugate* gradient steps, namely steps in $\hat{\boldsymbol{K}}$ norm (via a clever recursion).

- Guaranteed to converge in n steps, but in practice converges to high precision much faster

- As with many (all) other fast linear solvers, it operates only with forward multiplies of form $\hat{\boldsymbol{K}}\boldsymbol{x}$

Algorithm S3: Preconditioned Conjugate Gradient Method [38]

Input: kernel matrix $\hat{\boldsymbol{K}}$, labels $\boldsymbol{y}$, prior mean $\boldsymbol{\mu}$, preconditioner $\hat{\boldsymbol{P}}$
Output: representer weights $\boldsymbol{v}_i \approx \hat{\boldsymbol{K}}^{-1}(\boldsymbol{y}-\boldsymbol{\mu})$

procedure CG($\hat{\boldsymbol{K}}, \boldsymbol{y}-\boldsymbol{\mu}, \hat{\boldsymbol{P}}$)
  $\boldsymbol{v}_0 \leftarrow \boldsymbol{0}$
  $\boldsymbol{s}_0 \leftarrow \boldsymbol{0}$
  while $\|\boldsymbol{r}_i\|_2 > \max(\delta_{\text{rtol}}\|\boldsymbol{y}\|_2, \delta_{\text{atol}})$ **and** $i < i_{\max}$ **do**
    $\boldsymbol{r}_{i-1} \leftarrow (\boldsymbol{y}-\boldsymbol{\mu}) - \hat{\boldsymbol{K}}\boldsymbol{v}_{i-1}$
    $\boldsymbol{s}_i \leftarrow \hat{\boldsymbol{P}}^{-1}\boldsymbol{r}_{i-1} - \frac{(\hat{\boldsymbol{P}}^{-1}\boldsymbol{r}_{i-1})^\intercal \hat{\boldsymbol{K}}\boldsymbol{s}_{i-1}}{\boldsymbol{s}_{i-1}^\intercal \boldsymbol{K}\boldsymbol{s}_{i-1}}\boldsymbol{s}_{i-1}$
    $\boldsymbol{v}_i \leftarrow \boldsymbol{v}_{i-1} + \frac{(\hat{\boldsymbol{P}}^{-1}\boldsymbol{r}_{i-1})^\intercal \boldsymbol{r}_{i-1}}{\boldsymbol{s}_i^\intercal \hat{\boldsymbol{K}}\boldsymbol{s}_i}\boldsymbol{s}_i$
  return $\boldsymbol{v}$

[Wenger et al 2022 NeurIPS]
[Cutajar et al 2016 ICML]
.
.
.
[Cunningham et al 2007 ICML]

Fast Inference 1: Conjugate Gradients

- (For completeness) CG is well used for log likelihood computation also:

Algorithm 1: log-Marginal Likelihood

Input: $\boldsymbol{y}$ (labels), $\hat{\boldsymbol{K}}$ (kernel matrix), $\hat{\boldsymbol{P}}$ (preconditioner), ℓ (# of random STE vectors), m (# of CG iterations)

procedure LOGMARGLIKELIHOOD($\boldsymbol{y}, \hat{\boldsymbol{K}}, \hat{\boldsymbol{P}}, \ell, m$)
- $\boldsymbol{u} \leftarrow \mathrm{CG}(\hat{\boldsymbol{K}}, \boldsymbol{y}, \hat{\boldsymbol{P}}, m)$ ▷ $\approx \hat{\boldsymbol{K}}^{-1}\boldsymbol{y}$
- $\tau^{\log}_{\hat{\boldsymbol{P}}} \leftarrow \log\det(\hat{\boldsymbol{P}})$
- **for** $i = 1, \ldots, \ell$ **do**
  - $\boldsymbol{z}_i \leftarrow \tilde{\boldsymbol{z}}_i / \|\tilde{\boldsymbol{z}}_i\|_2$ for rand. vector $\tilde{\boldsymbol{z}}_i$
  - $\boldsymbol{T} \leftarrow \mathrm{CG}(\hat{\boldsymbol{K}}, \boldsymbol{z}_i, \hat{\boldsymbol{P}}, m)$ ▷ equiv. to LANCZOS
  - $[\boldsymbol{W}, \boldsymbol{\lambda}] \leftarrow \mathrm{EIGENDECOMP}(\boldsymbol{T})$ ▷ $\boldsymbol{T}$ tridiagonal
  - $\omega_j \leftarrow (\boldsymbol{e}_1^\intercal \boldsymbol{w}_j)^2$ for $j = 0, \ldots, m$ ▷ quad. weights
  - $\gamma_i \leftarrow \sum_{j=0}^{m} \omega_j \log(\lambda_j)$ ▷ $\approx \boldsymbol{z}_i^\intercal \boldsymbol{\Delta}_{\log} \boldsymbol{z}_i$
- $\tau^{\log}_{*} \leftarrow \tau^{\log}_{\hat{\boldsymbol{P}}} + \frac{n}{\ell}\sum_{i=1}^{\ell}\gamma_i$ ▷ $\approx \log\det(\hat{\boldsymbol{K}})$
- **return** $-\frac{1}{2}(\boldsymbol{y}^\intercal \boldsymbol{u} + \tau^{\log}_{*} + n\log(2\pi))$ ▷ $\approx \mathcal{L}(\boldsymbol{\theta})$

Algorithm 2: Derivative of the log-Marginal Likelihood

Input: $\boldsymbol{y}$ (labels), $\hat{\boldsymbol{K}}$ (kernel matrix), $\hat{\boldsymbol{P}}$ (preconditioner), ℓ (# of random STE vectors), m (# of CG iterations), $\frac{\partial \hat{\boldsymbol{K}}}{\partial \theta}$ / $\frac{\partial \hat{\boldsymbol{P}}}{\partial \theta}$ (functions for computing kernel / preconditioner derivatives)

procedure DERIVATIVE($\boldsymbol{y}, \hat{\boldsymbol{K}}, \frac{\partial \hat{\boldsymbol{K}}}{\partial \theta}, \hat{\boldsymbol{P}}, \frac{\partial \hat{\boldsymbol{P}}}{\partial \theta}, \ell, m$)
- $\boldsymbol{u} \leftarrow \mathrm{CG}(\hat{\boldsymbol{K}}, \boldsymbol{y}, \hat{\boldsymbol{P}}, m)$ ▷ $\approx \hat{\boldsymbol{K}}^{-1}\boldsymbol{y}$
- $\tau^{\mathrm{inv}\partial}_{\hat{\boldsymbol{P}}} \leftarrow \mathrm{tr}\big(\hat{\boldsymbol{P}}^{-1}\frac{\partial \hat{\boldsymbol{P}}}{\partial \theta}\big)$
- **for** $i = 1, \ldots, \ell$ **do**
  - $\boldsymbol{z}_i \leftarrow \tilde{\boldsymbol{z}}_i / \|\tilde{\boldsymbol{z}}_i\|_2$ for rand. vector $\tilde{\boldsymbol{z}}_i$
  - $\boldsymbol{w}_i \leftarrow \mathrm{CG}(\hat{\boldsymbol{K}}, \frac{\partial \hat{\boldsymbol{K}}}{\partial \theta}\boldsymbol{z}_i, \hat{\boldsymbol{P}}, m)$ ▷ $\approx \hat{\boldsymbol{K}}^{-1}\frac{\partial \hat{\boldsymbol{K}}}{\partial \theta}\boldsymbol{z}_i$
  - $\tilde{\boldsymbol{w}}_i \leftarrow \hat{\boldsymbol{P}}^{-1}\frac{\partial \hat{\boldsymbol{P}}}{\partial \theta}\boldsymbol{z}_i$
  - $\gamma_i \leftarrow \boldsymbol{z}_i^\intercal(\boldsymbol{w}_i - \tilde{\boldsymbol{w}}_i)$ ▷ $\approx \boldsymbol{z}_i^\intercal \boldsymbol{\Delta}_{\mathrm{inv}\partial} \boldsymbol{z}_i$
- $\tau^{\mathrm{inv}\partial}_{*} \leftarrow \tau^{\mathrm{inv}\partial}_{\hat{\boldsymbol{P}}} + \frac{n}{\ell}\sum_{i=1}^{\ell}\gamma_i$ ▷ $\approx \mathrm{tr}\big(\boldsymbol{K}^{-1}\frac{\partial \boldsymbol{K}}{\partial \theta}\big)$.
- **return** $\frac{1}{2}(\boldsymbol{u}^\intercal \frac{\partial \boldsymbol{K}}{\partial \theta}\boldsymbol{u} - \tau^{\mathrm{inv}\partial}_{*})$ ▷ $\approx \frac{\partial}{\partial \theta}\mathcal{L}(\boldsymbol{\theta})$

[Wenger et al 2022 ICML]

Fast Inference 2: Inducing Point Methods

- *Inducing points* have been used under many names and variations
  - SoR, Nystrom, SVGP, DTC,…
  - These methods differ in their training and posterior covariance assumptions
  - Detailing all of them is a few lectures in its own right; here I will lay out the essentials

- Choose a set of locations $Z \in \mathbb{R}^{m \times d}$

- Posterior then uses the approximation $\boldsymbol{K}_{XX} \approx \boldsymbol{K}_{XZ}\boldsymbol{K}_{ZZ}^{-1}\boldsymbol{K}_{ZX}$
  - Well-chosen inducing points make should be a good low rank approximation

- While methods differ in posterior covariance, they share a posterior mean [Wild et al 2023]

$$\mu(\cdot) = k(\cdot, Z)\boldsymbol{K}_{ZZ}^{-1}\boldsymbol{K}_{ZX}\boldsymbol{K}_{XZ}(\boldsymbol{K}_{ZX}(\boldsymbol{K}_{XZ}\boldsymbol{K}_{ZZ}^{-1}\boldsymbol{K}_{ZX} + \sigma^2 I)\boldsymbol{K}_{XZ})^{-1}\boldsymbol{K}_{ZX}(\boldsymbol{y} - \boldsymbol{\mu})$$

- TL;DR: all inversions take place in the inducing point space:
  - Meaningful cost savings
  - Tendency to be overconfident, especially where inducing points are placed.

[Seeger and Williams 2001 NIPS]

.
.
.

[Wu et al 2022 ICML]
[Wu et al 2021 AISTATS]

Approximate Inference

- The crystal ball is actually a computing machine, making assumptions and choices of its own, but we have not accounted for them. All our assumptions, all our unknowns … we were supposed to reason about them probabilistically!

belief about the state of the world

$p(f)$

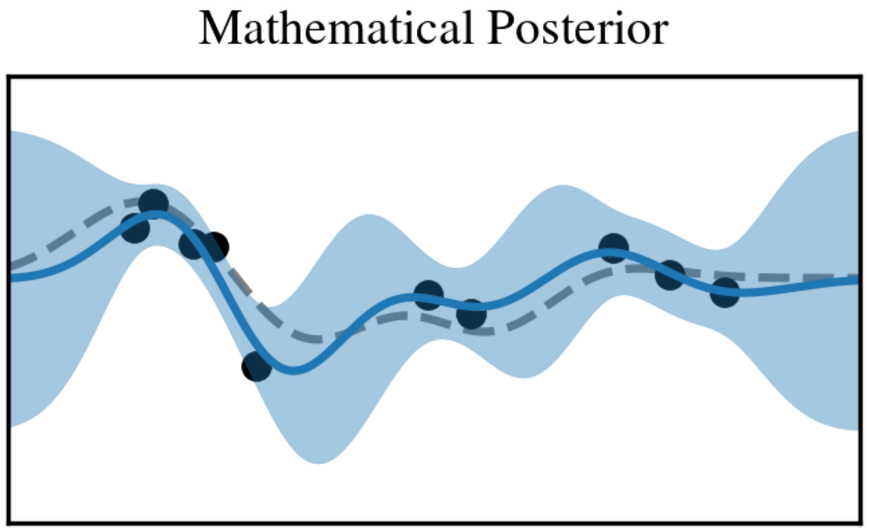
how data arises from a state of the world

$p(y|f)$

$\approx p(f|y)$

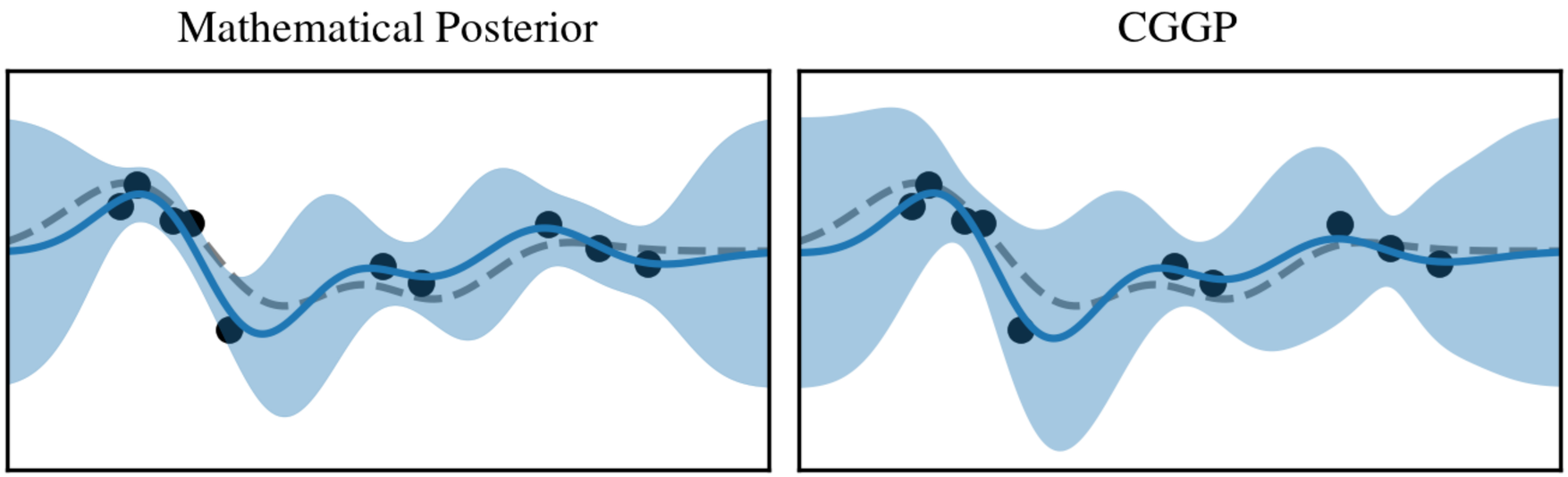
updated belief given data

Approximate Inference in Practice



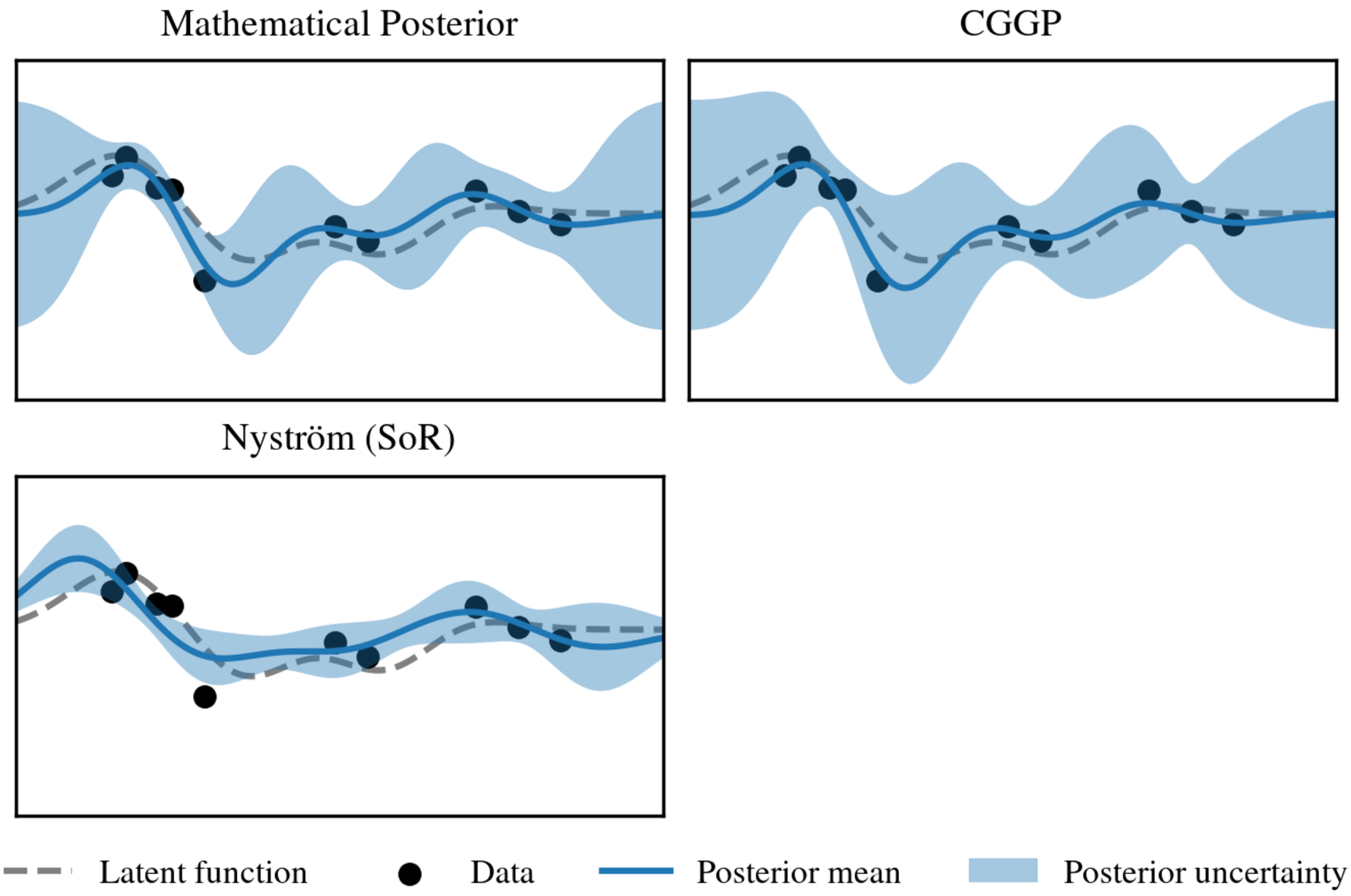
Latent function Data Posterior mean Posterior uncertainty

Approximate Inference in Practice

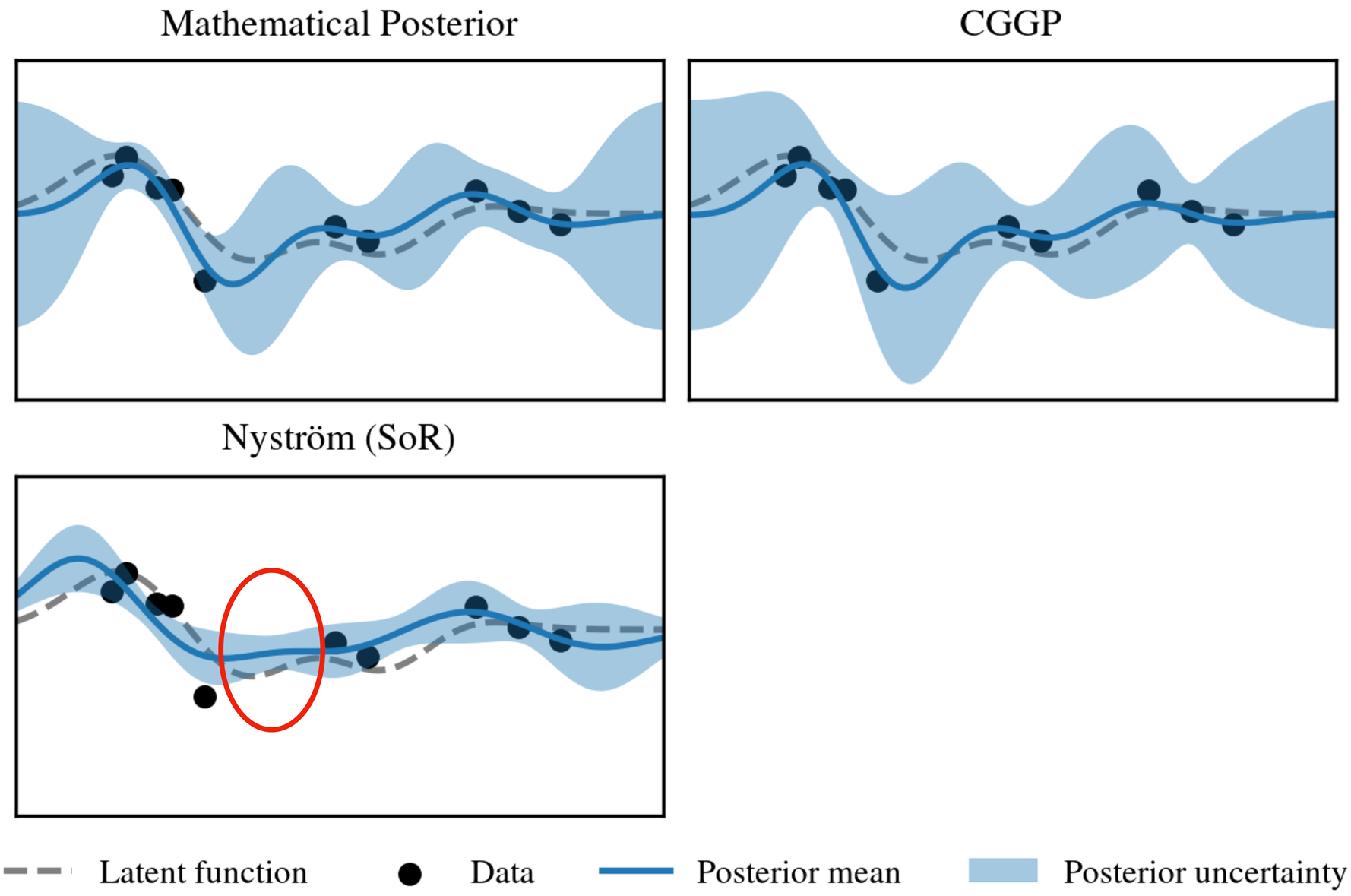


Latent function Data Posterior mean Posterior uncertainty

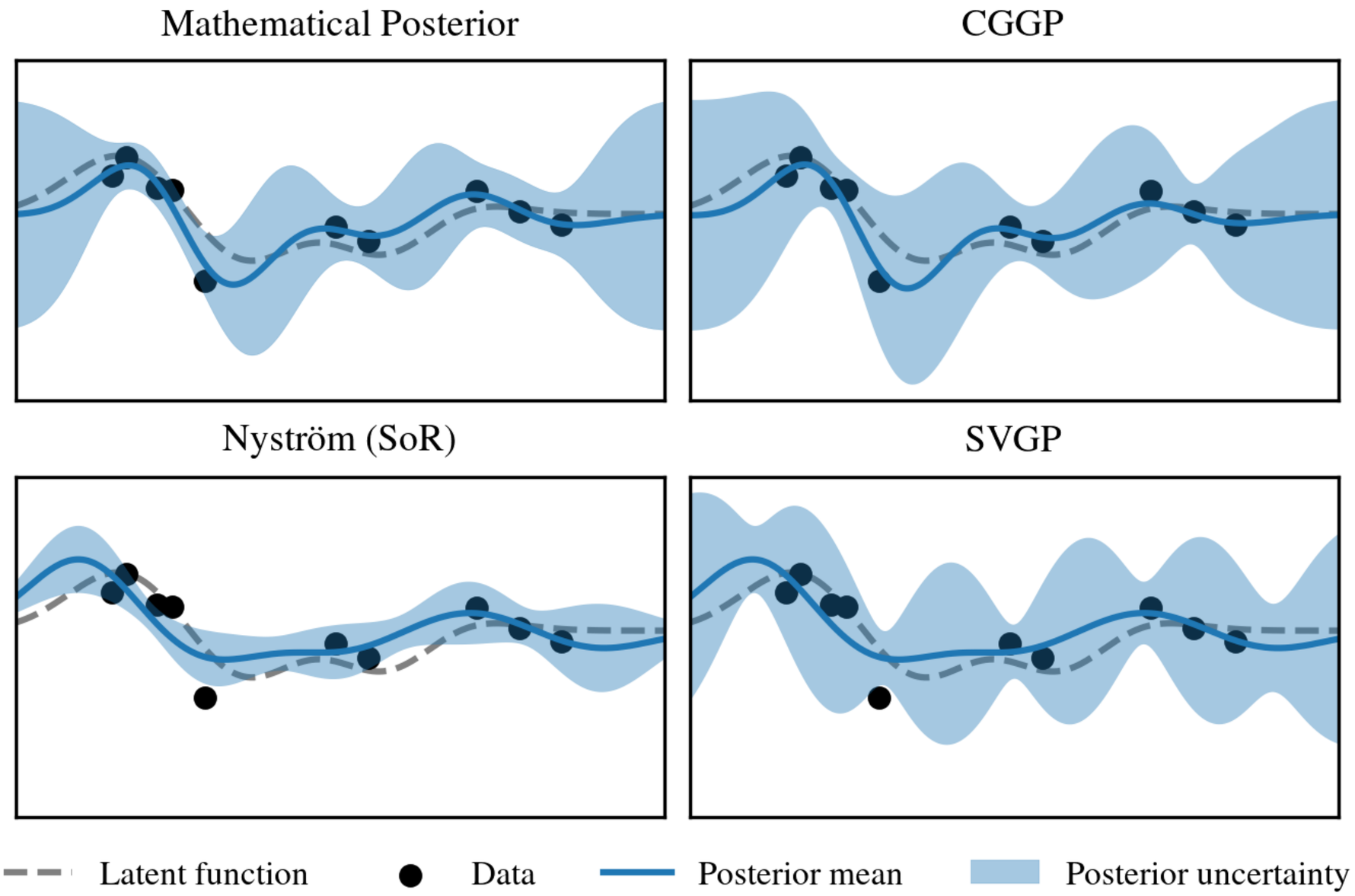
Approximate Inference in Practice



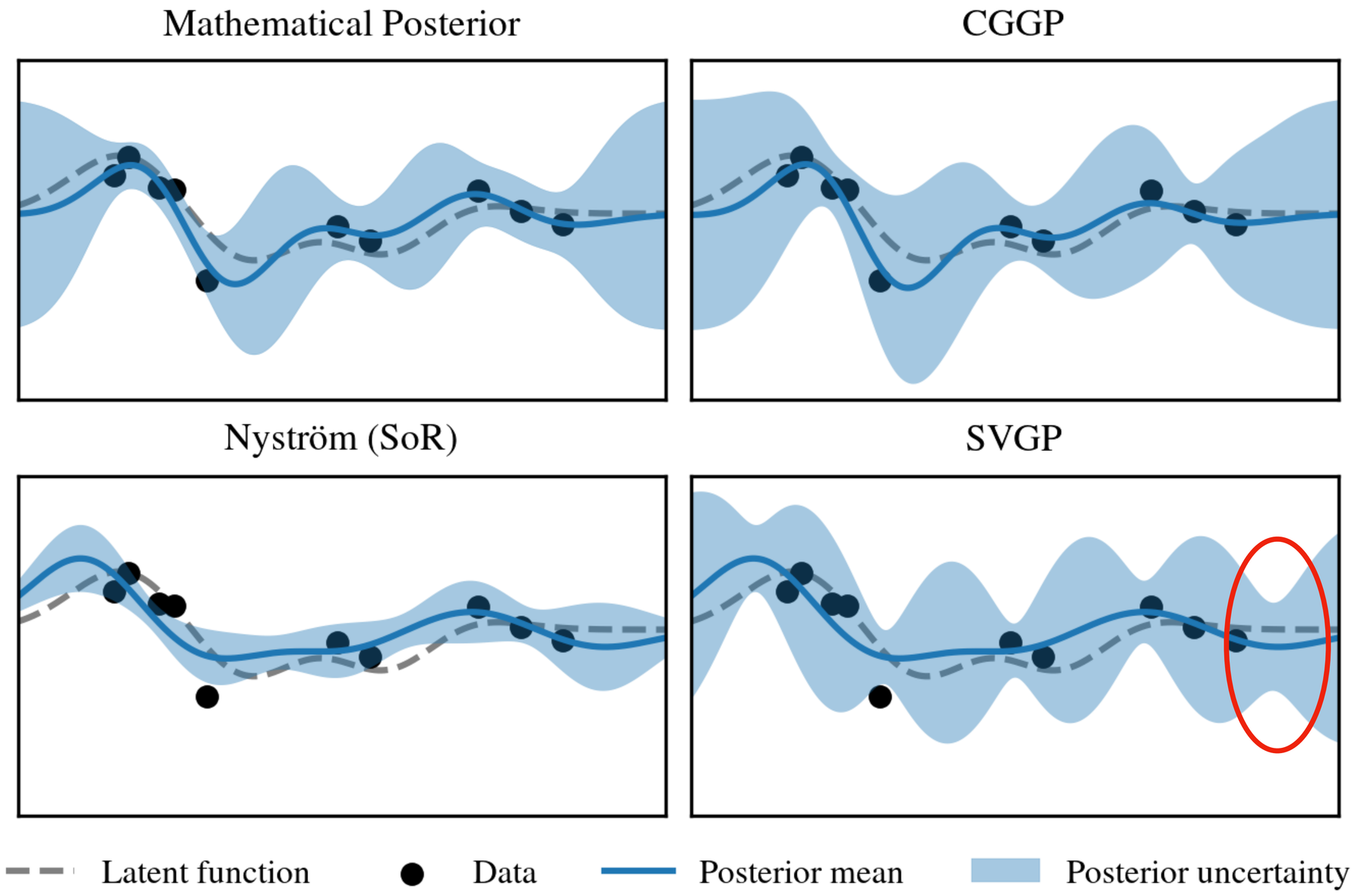
Approximate Inference in Practice



Approximate Inference in Practice

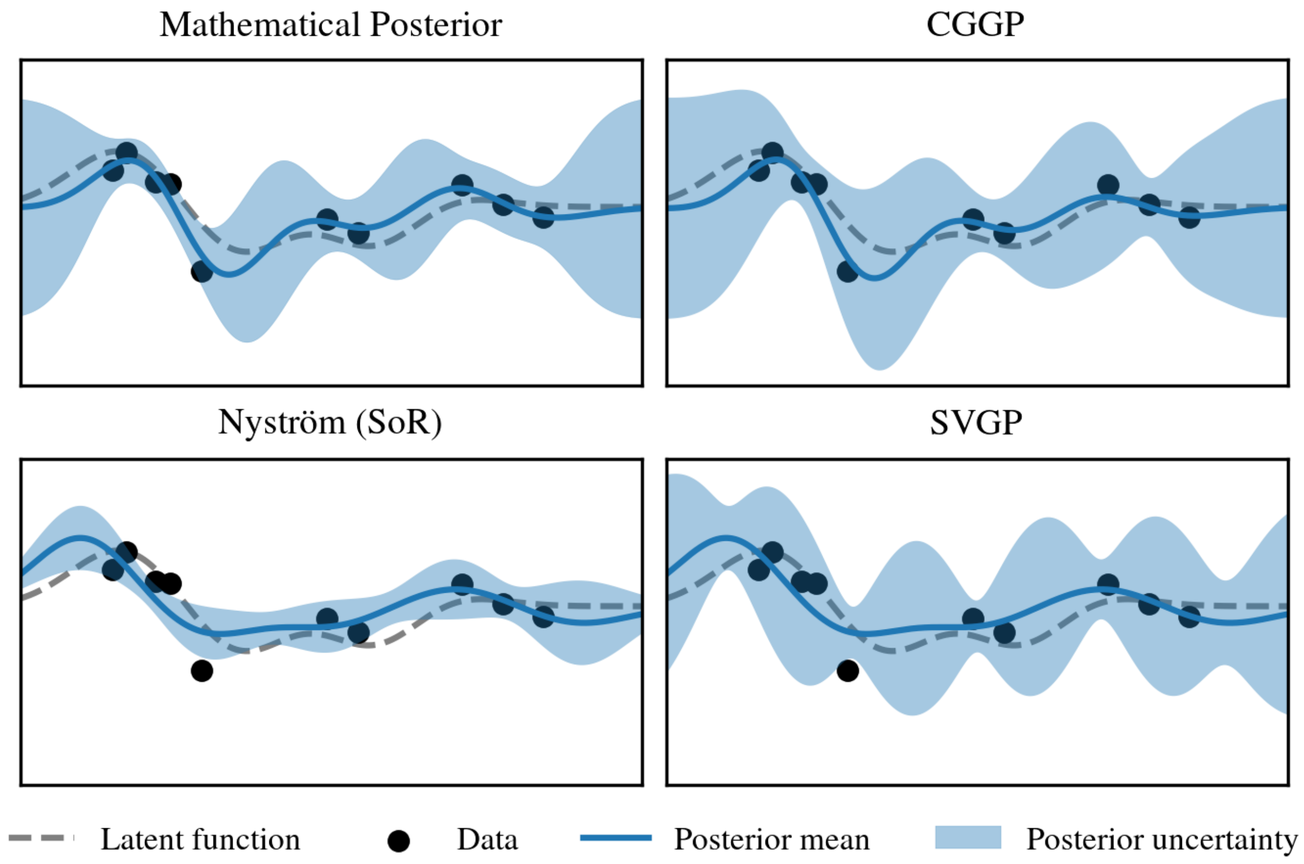


Approximate Inference in Practice



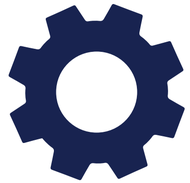
Approximate Inference in Practice

- The point: the approximation method is making strong (and different) statements about what you know
- But this effect is ignored…



Outline

- The Promise of Probabilistic Machine Learning
- Gaussian Process Introduction
- Scaling Gaussian Processes, and Implications
- **Approximate Gaussian Process Inference, The Right Way**
- iterGP as Probabilistic Numerics
- Broader Implications



Approximate Inference as Exact Inference

- Apparently approximate inference is exact inference, but under some different model

belief about the state of the world

$p(f)$

how data arises from a state of the world

$p(y|f)$

$\approx p(f|y)$

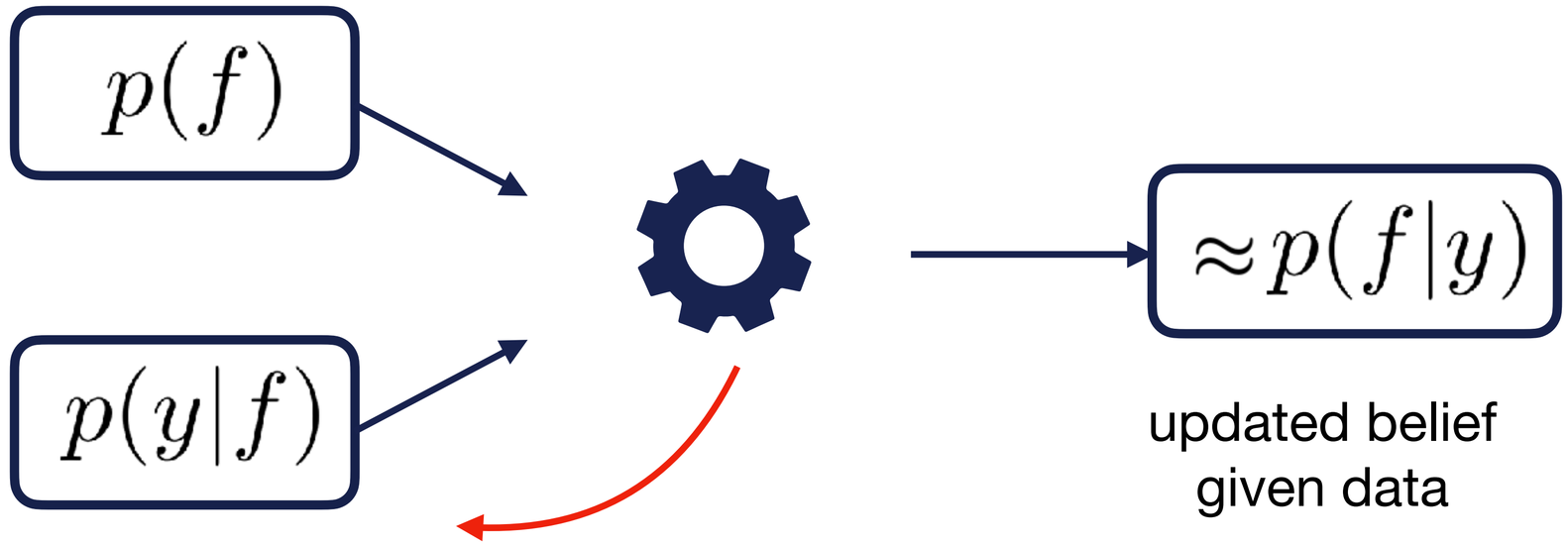
updated belief given data

Approximate Inference as Exact Inference

- Apparently approximate inference is exact inference, but under some different model

belief about the state of the world

how data arises from a state of the world

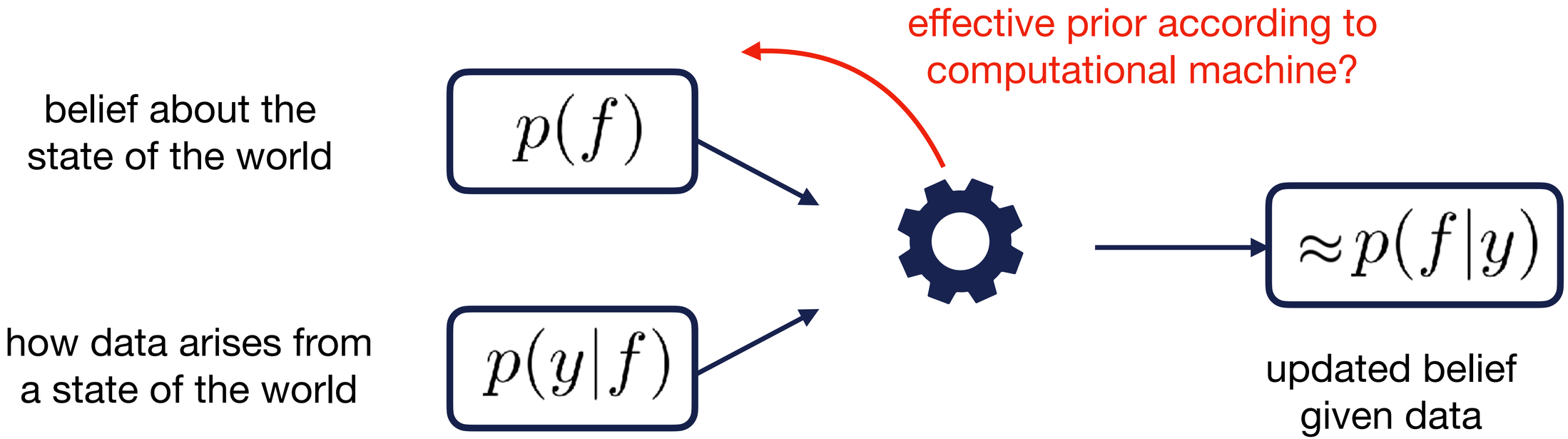


updated belief given data

effective likelihood according to the computational machine?

Approximate Inference as Exact Inference

- Apparently approximate inference is exact inference, but under some different model



Approximate Inference as Exact Inference

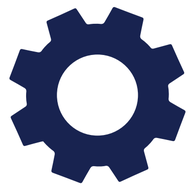
- Apparently approximate inference is exact inference, but under some different **data**

belief about the state of the world

$p(f)$

how data arises from a state of the world

$p(y|f)$



$\approx p(f|y)$

updated belief given effective data

effective **dataset** induced by the computation

[Wenger et al NeurIPS 2022]

Effective Dataset

- Iterative numerical methods (for GP) take linear combinations of data
- Let us define the set of actions taken by a given (approximate) solver as

$$\boldsymbol{S}_i = [\boldsymbol{s}_1\ \boldsymbol{s}_2\ \ldots\ \boldsymbol{s}_i] \in \mathbb{R}^{n \times i}$$

- This should feel plausible; consider the following actions:

Actions $\boldsymbol{s}_i$	Classic Analog
$\boldsymbol{e}_i$	(partial) Cholesky
$\mathrm{ev}_i(\hat{\boldsymbol{K}})$	(partial) EVD / SVD
$\boldsymbol{s}_i^{\mathrm{PCG}}$ or $\hat{\boldsymbol{P}}^{-1}\boldsymbol{r}_i$	(preconditioned) CG
$k(\boldsymbol{X}, \boldsymbol{z}_i)$	$\approx$ Nyström (SoR, DTC), SVGP

- (We will make this rigorous shortly, for now we just establish the connection)

Gaussian Process Posterior

- With likelihood: $\boldsymbol{y} \mid \mathbf{f} \sim \mathcal{N}(\mathbf{f}, \sigma^2 \boldsymbol{I})$
- And test inputs: $\boldsymbol{X}_\diamond$

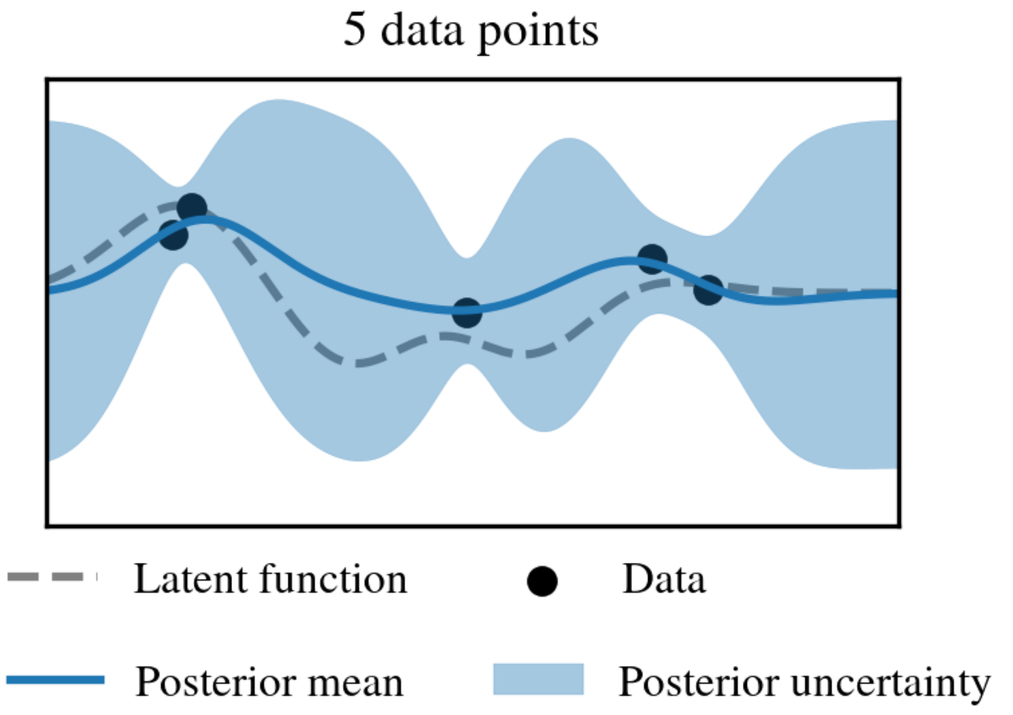
- Induces the posterior:

$$\mathbf{f}_\diamond \sim \mathcal{N}(\mu_*(\boldsymbol{X}_\diamond), k_*(\boldsymbol{X}_\diamond, \boldsymbol{X}_\diamond))$$

  - where: $\mu_*(\cdot) = \mu(\cdot) + k(\cdot, \boldsymbol{X})\, \hat{\boldsymbol{K}}^{-1}(\boldsymbol{y} - \boldsymbol{\mu})$

$$k_*(\cdot, \cdot) = k(\cdot, \cdot) - k(\cdot, \boldsymbol{X})\hat{\boldsymbol{K}}^{-1}k(\boldsymbol{X}, \cdot)$$

$$\hat{\boldsymbol{K}} := \boldsymbol{K} + \sigma^2 \boldsymbol{I} \in \mathbb{R}^{n \times n}$$



Gaussian Process Posterior

- With likelihood: $\boldsymbol{S}_i^\intercal \boldsymbol{y} \mid \mathbf{f} \sim \mathcal{N}(\boldsymbol{S}_i^\intercal \mathbf{f}, \sigma^2 \boldsymbol{S}_i^\intercal \boldsymbol{S}_i)$
- And test inputs: $\boldsymbol{X}_\diamond$
- Induces the posterior:

$$\mathbf{f}_\diamond \sim \mathcal{N}(\mathbf{f}_\diamond; \mu_i(\boldsymbol{X}_\diamond), k_i(\boldsymbol{X}_\diamond, \boldsymbol{X}_\diamond))$$

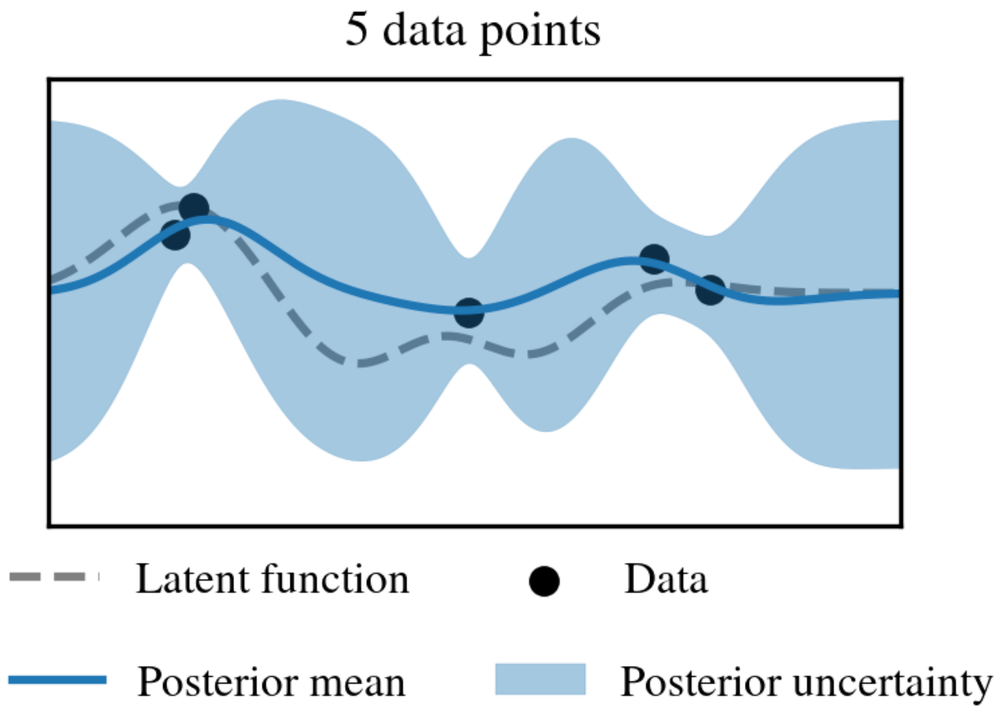
  - where:

$$\mu_i(\cdot) = \mu(\cdot) + k(\cdot, \boldsymbol{X})\boldsymbol{C}_i(\boldsymbol{y} - \boldsymbol{\mu})$$

$$k_i(\cdot, \cdot) = k(\cdot, \cdot) - k(\cdot, \boldsymbol{X})\boldsymbol{C}_i k(\boldsymbol{X}, \cdot)$$

$$\boldsymbol{C}_i = \boldsymbol{S}_i(\boldsymbol{S}_i^\intercal \hat{\boldsymbol{K}} \boldsymbol{S}_i)^{-1} \boldsymbol{S}_i^\intercal$$

$$\hat{\boldsymbol{K}} := \boldsymbol{K} + \sigma^2 \boldsymbol{I} \in \mathbb{R}^{n \times n}$$



Combined Uncertainty

- *Combined Uncertainty* admits a clean decomposition:

$$\underbrace{k(\cdot,\cdot) - k(\cdot,\boldsymbol{X})\boldsymbol{C}_i k(\boldsymbol{X},\cdot)}_{\text{combined uncertainty}} = \underbrace{k(\cdot,\cdot) - k(\cdot,\boldsymbol{X})\hat{\boldsymbol{K}}^{-1}k(\boldsymbol{X},\cdot)}_{\text{mathematical uncertainty}} + \underbrace{k(\cdot,\boldsymbol{X})\boldsymbol{\Sigma}_i k(\boldsymbol{X},\cdot)}_{\text{computational uncertainty}}$$

$$\boldsymbol{C}_i = \boldsymbol{S}_i(\boldsymbol{S}_i^\intercal \hat{\boldsymbol{K}}\boldsymbol{S}_i)^{-1}\boldsymbol{S}_i^\intercal \qquad \hat{\boldsymbol{K}} := \boldsymbol{K} + \sigma^2\boldsymbol{I}$$

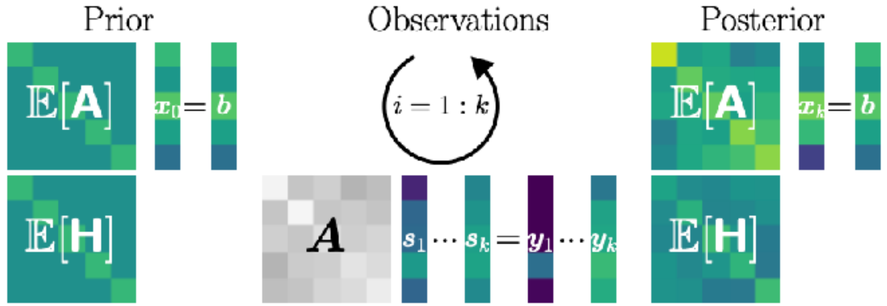
Computational uncertainty turns out to be exactly the uncertainty on the representer weights

- Very satisfying outcome:
  - Our combined (effective) uncertainty is a result of both finite data and finite compute
  - We call *mathematical uncertainty* the exact posterior with the given data
  - Computational uncertainty is a direct use of Probabilistic Numerics!

Interlude: so is this a PN story or a GP story?

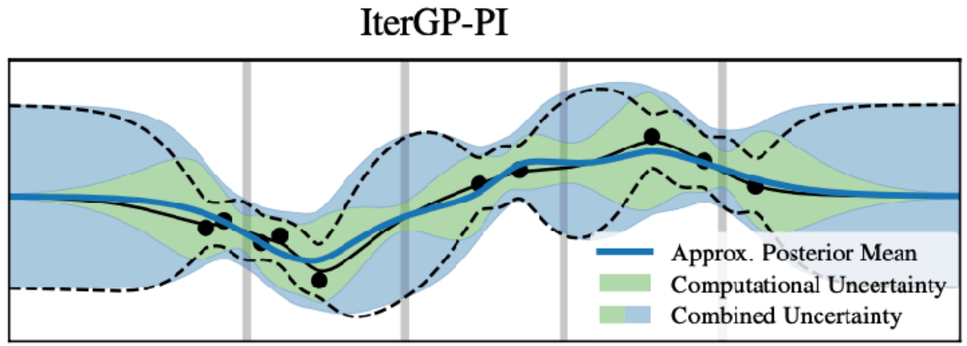
- Both. What I am telling you today builds on huge amounts GP approximate inference work and a significant amount of PN work, two most notably:

PN: linear solvers can be seen as probabilistic inference methods



[Wenger and Hennig NeurIPS 2020]

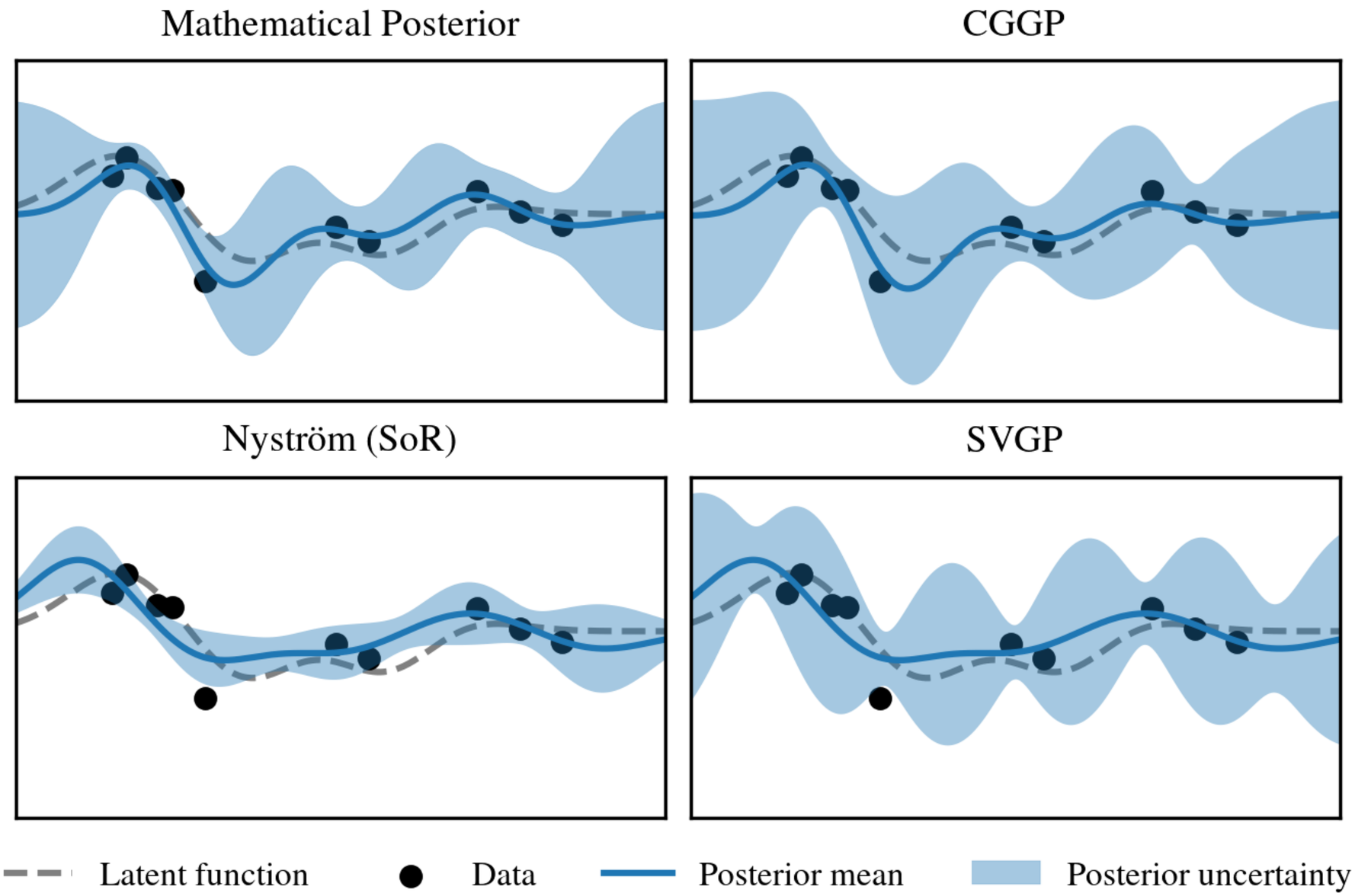
GP: inference is all linear solves, from which arises the effective dataset



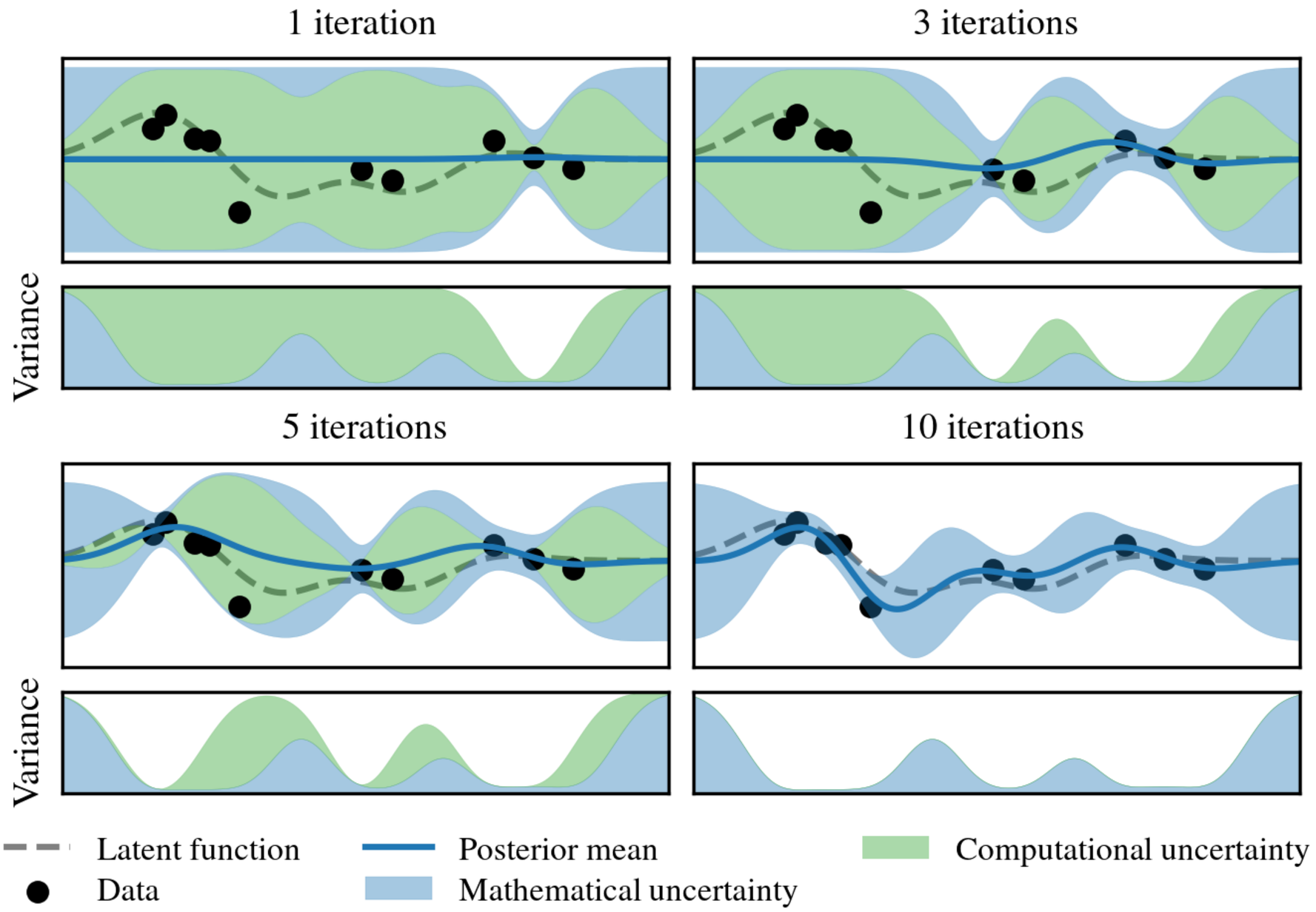
[Wenger et al NeurIPS 2022]

- *IterGP* is the family of methods producing combined GP uncertainty from iterative solvers

Old Way: Approximate Inference in Practice



New Way: Sequentially Updating Data Points as IterGP

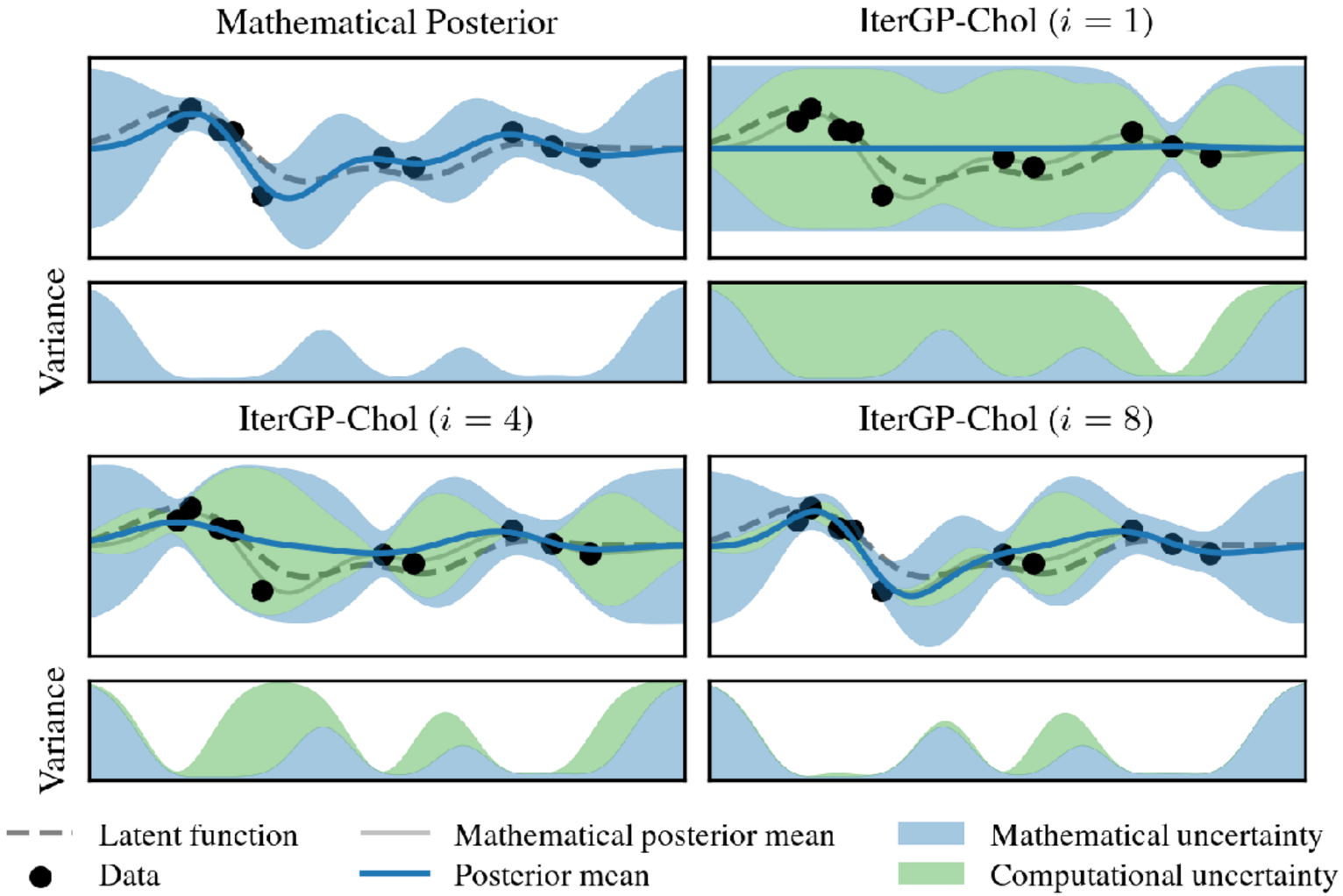


New Way: IterGP-Cholesky

Method	Actions s_i	Classic Analog
IterGP-Chol	e_i	(partial) Cholesky

- Recall:

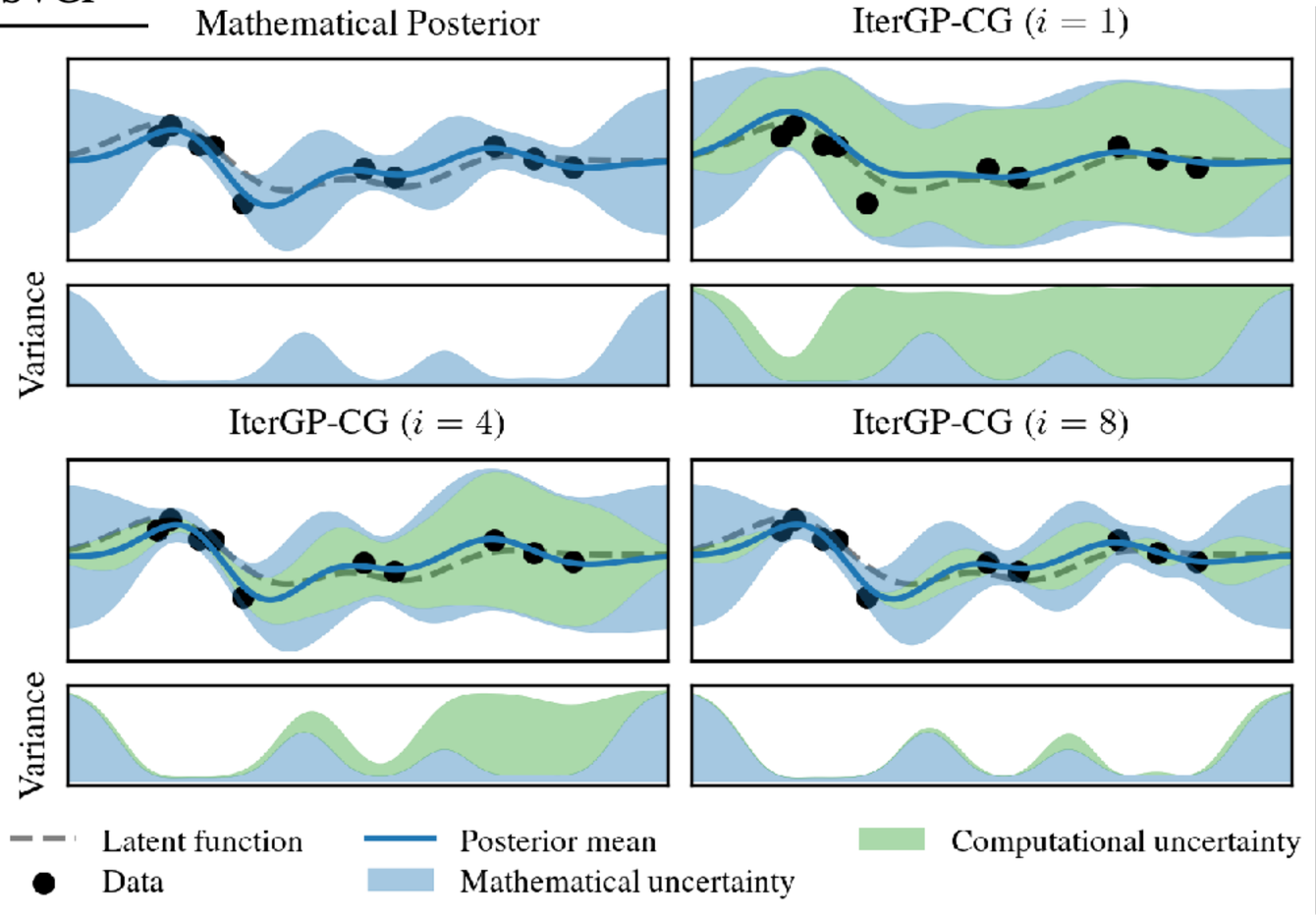
$$S_i^\intercal y \mid \mathbf{f} \sim \mathcal{N}(S_i^\intercal \mathbf{f}, \sigma^2 S_i^\intercal S_i)$$



New Way: IterGP-CG

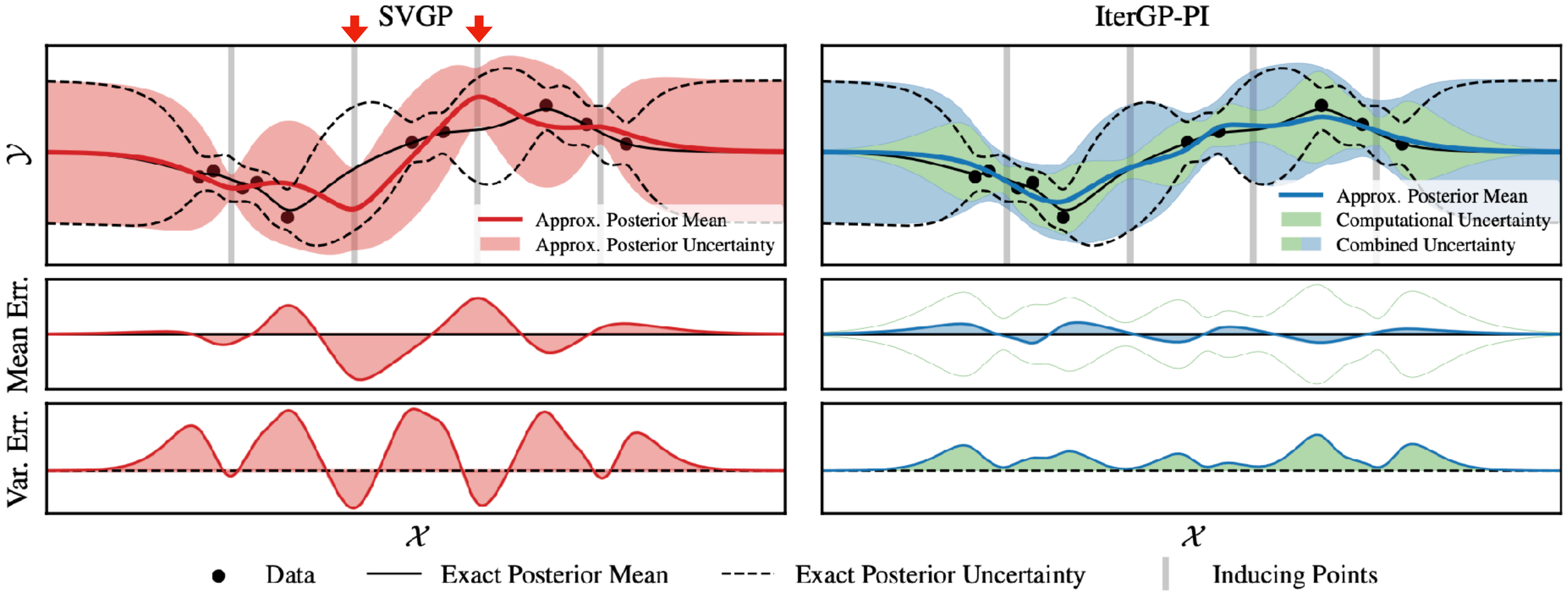
Method	Actions s_i	Classic Analog
IterGP-Chol	e_i	(partial) Cholesky
IterGP-PBR	$\mathrm{ev}_i(\hat{K})$	(partial) EVD / SVD
IterGP-CG	s_i^{PCG} or $\hat{P}^{-1}r_i$	(preconditioned) CG
IterGP-PI	$k(X, z_i)$	$\approx$ Nyström (SoR, DTC), SVGP

- Notice:
  - Computational uncertainty adds to mathematical uncertainty… of course it should
  - Untouched data (eg I=1 at right) has high combined uncertainty —> prior



Compare: iterGP-PseudoInput vs SVGP

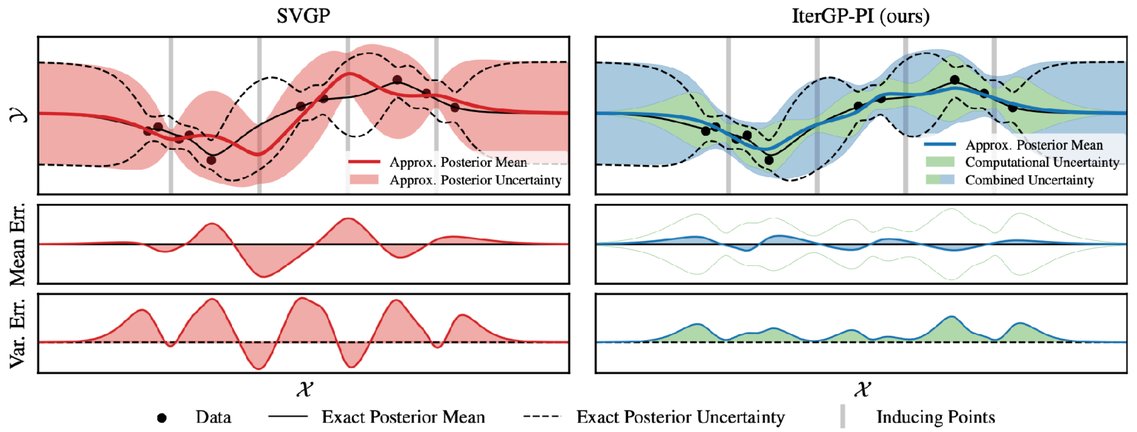
- Combined uncertainty reliably overestimates the truth —> this is desirable (and correct)!
- SVGP is overconfident at its inducing points, combined uncertainty corrects this.
- Overconfidence can produce large mean errors (⬇)



Compare: iterGP-PseudoInput vs SVGP

- Looking at posterior means is instructive:
  - Same as before, but for comparison let

$$q(\cdot,\cdot) = k(\cdot,\boldsymbol{Z})\boldsymbol{K}_{\boldsymbol{Z}\boldsymbol{Z}}^{-1}k(\boldsymbol{Z},\cdot)$$



- Then SVGP has posterior mean:

$$\mu_{\mathrm{SVGP}}(\cdot) = q(\cdot,\boldsymbol{X})\boldsymbol{K}_{\boldsymbol{X}\boldsymbol{Z}}(\boldsymbol{K}_{\boldsymbol{Z}\boldsymbol{X}}(q(\boldsymbol{X},\boldsymbol{X}) + \sigma^2\boldsymbol{I})\boldsymbol{K}_{\boldsymbol{X}\boldsymbol{Z}})^{-1}\boldsymbol{K}_{\boldsymbol{Z}\boldsymbol{X}}(\boldsymbol{y}-\boldsymbol{\mu})$$

- And IterGP-PI has posterior mean (for actions. $k(\boldsymbol{X},\boldsymbol{z}_i)$, recall $\boldsymbol{C}_i = \boldsymbol{S}_i(\boldsymbol{S}_i^\intercal\hat{\boldsymbol{K}}\boldsymbol{S}_i)^{-1}\boldsymbol{S}_i^\intercal$)

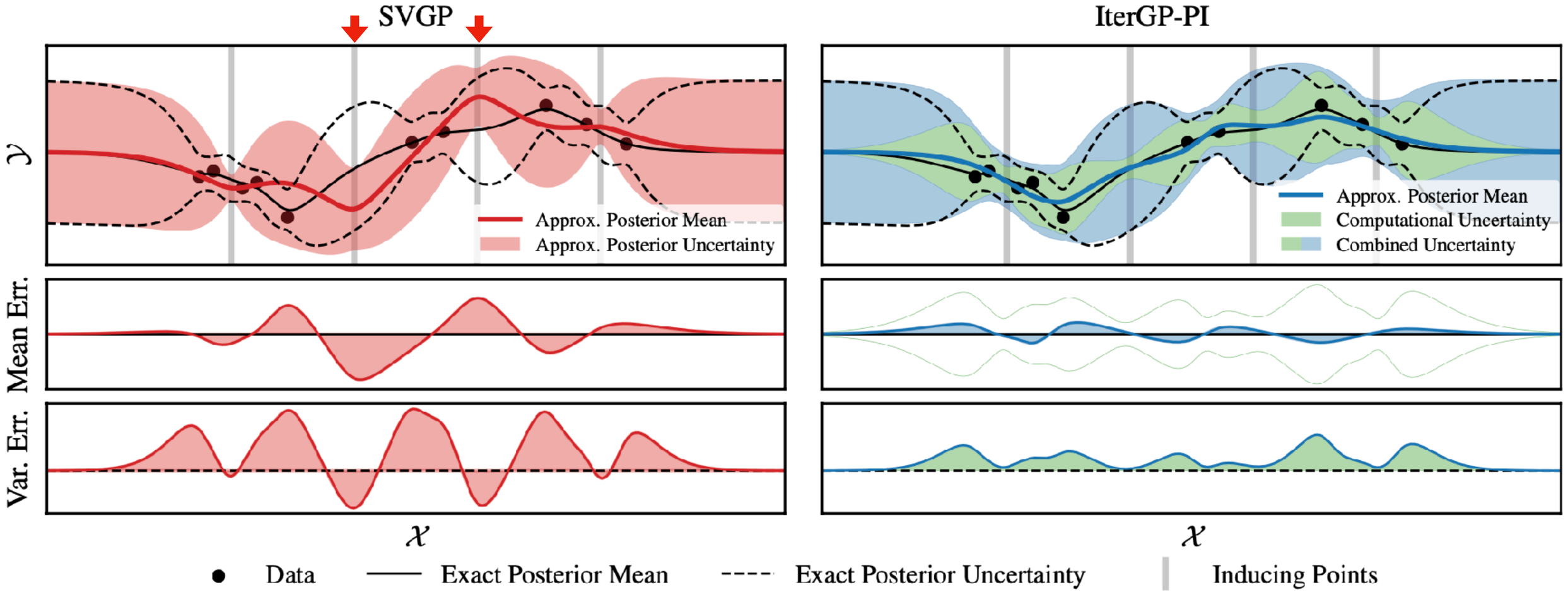
$$\mu_i(\cdot) = k(\cdot,\boldsymbol{X})\boldsymbol{K}_{\boldsymbol{X}\boldsymbol{Z}}(\boldsymbol{K}_{\boldsymbol{Z}\boldsymbol{X}}(k(\boldsymbol{X},\boldsymbol{X}) + \sigma^2\boldsymbol{I})\boldsymbol{K}_{\boldsymbol{X}\boldsymbol{Z}})^{-1}\boldsymbol{K}_{\boldsymbol{Z}\boldsymbol{X}}(\boldsymbol{y}-\boldsymbol{\mu})$$

- Speedup in SVGP comes at cost of overconfidence, since $q(\boldsymbol{X},\boldsymbol{X}) \preceq k(\boldsymbol{X},\boldsymbol{X})$

  - SVGP has too strong belief in representer weights, leading to potentially large errors even with a good variational fit. Computational uncertainty reduces that confidence.

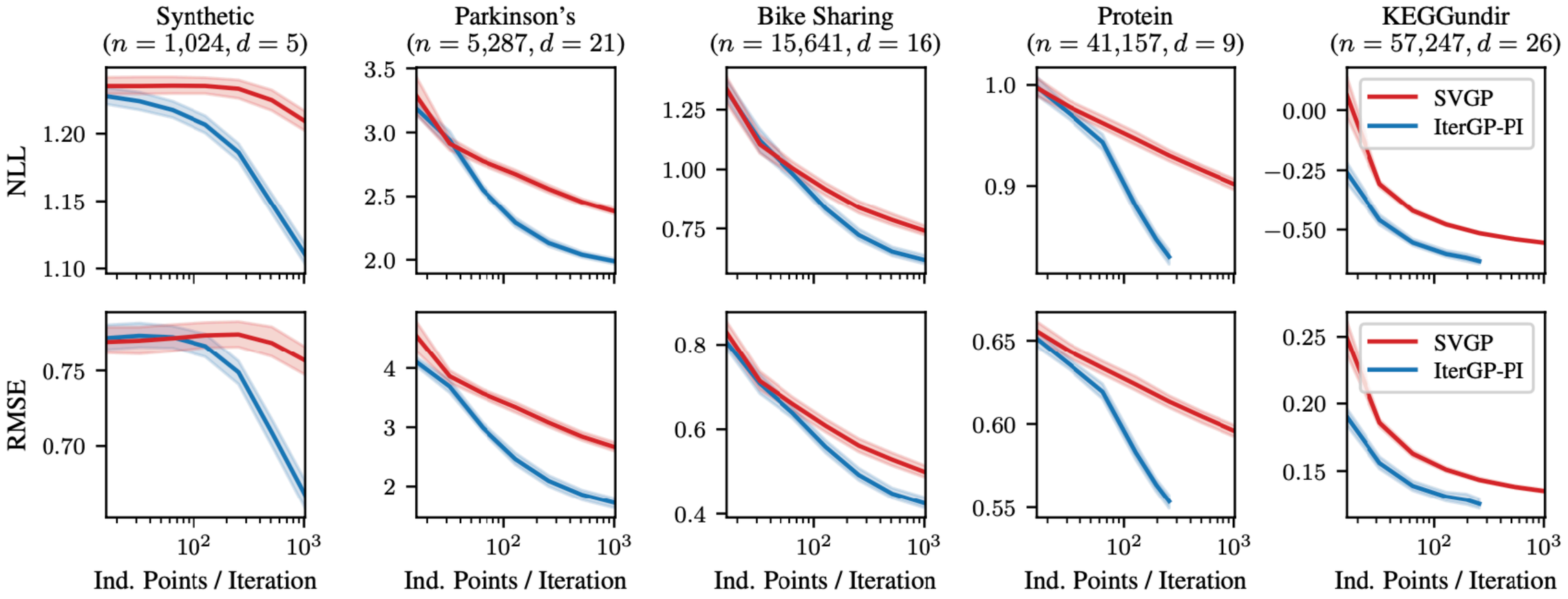
Compare: iterGP-PseudoInput vs SVGP

- Combined uncertainty reliably overestimates the truth —> this is desirable (and correct)!
- SVGP is overconfident at its inducing points, combined uncertainty corrects this.
- Overconfidence can produce large mean errors (⬇)



Computational Uncertainty Matters

- IterGP achieves better generalization with a smaller number of inducing points (vs SVGP)



- Note however that this is a statement about use of inducing points, not performance per flop

Approximate Inference as Exact Inference (on Different Data)

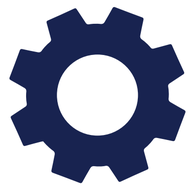
- Result: if you update your GP via matrix-vector multiplication, then the combined uncertainty of the IterGP algorithm is precisely the correct object to capture your belief.

belief about the state of the world

$p(f)$

how data arises from a state of the world

$p(y|f)$



iterGP captures effective **dataset** due to computation

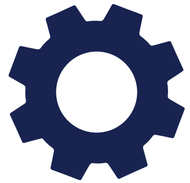
$p(f|y)$

updated belief given effective data

[Wenger et al NeurIPS 2022]

Outline

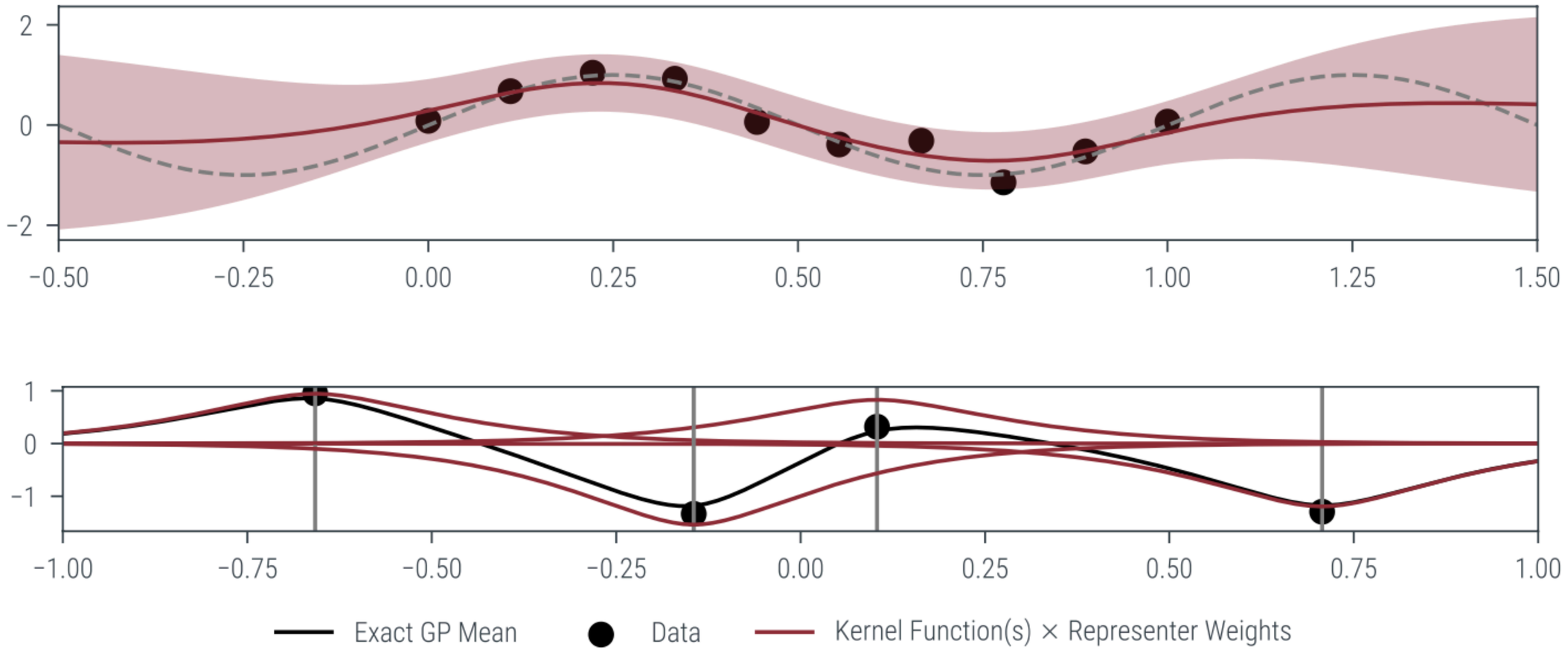
- The Promise of Probabilistic Machine Learning
- Gaussian Process Introduction
- Scaling Gaussian Processes, and Implications
- Approximate Gaussian Process Inference, The Right Way
- **iterGP as Probabilistic Numerics**
- Broader Implications



Return to GP Weight Space View

- Let us again consider the *representer weights* $\boldsymbol{v}_* = \hat{\boldsymbol{K}}^{-1}(\boldsymbol{y} - \boldsymbol{\mu})$

$$\mu_*(\cdot) = \mu(\cdot) + k(\cdot, \boldsymbol{X}) \overbrace{\hat{\boldsymbol{K}}^{-1}(\boldsymbol{y} - \boldsymbol{\mu})}^{\boldsymbol{v}_*}$$



[Bosch et al 2022]

Learning Representer Weights

- Consider the GP latent, conditioned on the representer weights

$$p(\mathbf{f}_\diamond \mid \boldsymbol{v}_*) = \mathcal{N}(\mu(\boldsymbol{X}_\diamond) + k(\boldsymbol{X}_\diamond, \boldsymbol{X})\boldsymbol{v}_*, \;\; k_*(\boldsymbol{X}_\diamond, \boldsymbol{X}_\diamond))$$

- First: suppose representer weights are known (linear solve is done!), we recover **posterior**:

$$\mathbf{f}_\diamond \sim \mathcal{N}(\mu_*(\boldsymbol{X}_\diamond), k_*(\boldsymbol{X}_\diamond, \boldsymbol{X}_\diamond))$$

$$\mu_*(\cdot) = \mu(\cdot) + k(\cdot, \boldsymbol{X})\overbrace{\hat{\boldsymbol{K}}^{-1}(\boldsymbol{y} - \boldsymbol{\mu})}^{\boldsymbol{v}_*} \qquad k_*(\cdot, \cdot) = k(\cdot, \cdot) - k(\cdot, \boldsymbol{X})\hat{\boldsymbol{K}}^{-1}k(\boldsymbol{X}, \cdot)$$

- Second: suppose representer weights are unknown (no solve yet!), we recover **prior**:

$$p(\boldsymbol{v}_*) = \mathcal{N}(\boldsymbol{v}_*; \mathbf{0}, \hat{\boldsymbol{K}}^{-1}) \qquad \int p(\mathbf{f}_\diamond \mid \boldsymbol{v}_*)p(\boldsymbol{v}_*)\, d\boldsymbol{v}_*$$

$$\mathbf{f}_\diamond \sim \mathcal{N}(\mu(\boldsymbol{X}_\diamond), k(\boldsymbol{X}_\diamond, \boldsymbol{X}_\diamond))$$

- This should already feel promising…

Learning Representer Weights (the details)

- Now the numerical method amounts to iteratively updating belief on representer weights:
  - Assume an existing belief: $p(\boldsymbol{v}_*) = \mathcal{N}(\boldsymbol{v}_*; \boldsymbol{v}_{i-1}, \boldsymbol{\Sigma}_{i-1})$
  - The error/residual of that belief is: $\boldsymbol{r}_{i-1} = (\boldsymbol{y} - \boldsymbol{\mu}) - \hat{\boldsymbol{K}}\boldsymbol{v}_{i-1}$
  - …this makes sense; the true representer weights are given by $\overbrace{\hat{\boldsymbol{K}}^{-1}(\boldsymbol{y} - \boldsymbol{\mu})}^{\boldsymbol{v}_*}$
  - Return of actions: $\alpha_i := \boldsymbol{s}_i^\intercal \boldsymbol{r}_{i-1} = \boldsymbol{s}_i^\intercal((\boldsymbol{y} - \boldsymbol{\mu}) - \hat{\boldsymbol{K}}\boldsymbol{v}_{i-1}) = \boldsymbol{s}_i^\intercal \hat{\boldsymbol{K}}(\boldsymbol{v}_* - \boldsymbol{v}_{i-1})$
  - Conditioning on this projected residual, we result in: $p(\boldsymbol{v}_*) = \mathcal{N}(\boldsymbol{v}_* \mid \boldsymbol{v}_i, \boldsymbol{\Sigma}_i)$

$$\boldsymbol{v}_i = \boldsymbol{v}_{i-1} + \underbrace{\boldsymbol{\Sigma}_{i-1}\hat{\boldsymbol{K}}\boldsymbol{s}_i}_{=:\boldsymbol{d}_i}\underbrace{(\boldsymbol{s}_i^\intercal \hat{\boldsymbol{K}}\boldsymbol{\Sigma}_{i-1}\hat{\boldsymbol{K}}\boldsymbol{s}_i)^{-1}}_{=:\eta_i}\underbrace{\boldsymbol{s}_i^\intercal \hat{\boldsymbol{K}}(\boldsymbol{v}_* - \boldsymbol{v}_{i-1})}_{=\alpha_i} = \boldsymbol{C}_i(\boldsymbol{y} - \boldsymbol{\mu})$$

$$\boldsymbol{\Sigma}_i = \boldsymbol{\Sigma}_{i-1} - \underbrace{\boldsymbol{\Sigma}_{i-1}\hat{\boldsymbol{K}}\boldsymbol{s}_i}_{=\boldsymbol{d}_i}\underbrace{(\boldsymbol{s}_i^\intercal \hat{\boldsymbol{K}}\boldsymbol{\Sigma}_{i-1}\hat{\boldsymbol{K}}\boldsymbol{s}_i)^{-1}}_{=\eta_i}\underbrace{\boldsymbol{s}_i^\intercal \hat{\boldsymbol{K}}\boldsymbol{\Sigma}_{i-1}}_{=\boldsymbol{d}_i^\intercal} = \hat{\boldsymbol{K}}^{-1} - \boldsymbol{C}_i.$$

$$\boldsymbol{C}_i := \textstyle\sum_{j=1}^{i} \frac{1}{\eta_j}\boldsymbol{d}_j\boldsymbol{d}_j^\intercal = \boldsymbol{S}_i(\boldsymbol{S}_i^\intercal \hat{\boldsymbol{K}}\boldsymbol{S}_i)^{-1}\boldsymbol{S}_i^\intercal$$

Learning Representer Weights (the details)

- NB this all is captured quite cleanly in an iterative (probabilistic) numerical method

- Accompanying theorems in the paper add considerable strength to these claims

- Matrix vector multiplies imply a computational cost of $\mathcal{O}(n^2 i)$

- Storage is linear as the full covariance matrix needs not be represented in memory

Algorithm 1: A Class of Computation-Aware Iterative Methods for GP Approximation

Input: prior mean function μ, prior covariance function / kernel k, training inputs $\boldsymbol{X}$, labels $\boldsymbol{y}$
Output: (combined) GP posterior $\mathcal{GP}(\mu_i, k_i)$

procedure ITERGP($\mu, k, \boldsymbol{X}, \boldsymbol{y}$)
  $(\mu_0, k_0) \leftarrow (\mu, k)$ ▷ Initialize mean and covariance function with prior.
  $\boldsymbol{\mu} \leftarrow \mu(\boldsymbol{X})$ ▷ Prior predictive mean.
  $\hat{\boldsymbol{K}} \leftarrow k(\boldsymbol{X}, \boldsymbol{X}) + \sigma^2 \boldsymbol{I}$ ▷ Prior predictive kernel matrix.
  while not STOPPINGCRITERION() **do** ▷ Stopping criterion.
    $\boldsymbol{s}_i \leftarrow$ POLICY() ▷ Select action via policy (see Table 1 for examples).
    $\boldsymbol{r}_{i-1} \leftarrow (\boldsymbol{y} - \boldsymbol{\mu}) - \hat{\boldsymbol{K}} \boldsymbol{v}_{i-1}$ ▷ Predictive residual.
    $\alpha_i \leftarrow \boldsymbol{s}_i^\intercal \boldsymbol{r}_{i-1}$ ▷ Observation via information operator.
    $\boldsymbol{d}_i \leftarrow \boldsymbol{\Sigma}_{i-1} \hat{\boldsymbol{K}} \boldsymbol{s}_i = (\boldsymbol{I} - \boldsymbol{C}_{i-1} \hat{\boldsymbol{K}}) \boldsymbol{s}_i$ ▷ Search direction.
    $\eta_i \leftarrow \boldsymbol{s}_i^\intercal \hat{\boldsymbol{K}} \boldsymbol{\Sigma}_{i-1} \hat{\boldsymbol{K}} \boldsymbol{s}_i = \boldsymbol{s}_i^\intercal \hat{\boldsymbol{K}} \boldsymbol{d}_i$ ▷ Normalization constant.
    $\boldsymbol{C}_i \leftarrow \boldsymbol{C}_{i-1} + \frac{1}{\eta_i} \boldsymbol{d}_i \boldsymbol{d}_i^\intercal$ ▷ Precision matrix approximation $\boldsymbol{C}_i \approx \hat{\boldsymbol{K}}^{-1}$.
    $\boldsymbol{Q}_i \leftarrow \boldsymbol{Q}_{i-1} + \frac{1}{\eta_i} \hat{\boldsymbol{K}} \boldsymbol{d}_i \boldsymbol{d}_i^\intercal \hat{\boldsymbol{K}}$ ▷ Kernel matrix approximation $\boldsymbol{Q}_i \approx \hat{\boldsymbol{K}}$.
    $\boldsymbol{v}_i \leftarrow \boldsymbol{v}_{i-1} + \frac{\alpha_i}{\eta_i} \boldsymbol{d}_i$ ▷ Representer weights estimate.
    $\boldsymbol{\Sigma}_i \leftarrow \boldsymbol{\Sigma}_0 - \boldsymbol{C}_i$ ▷ Computational representer weights uncertainty.
  $p(\boldsymbol{v}_*) \leftarrow \mathcal{N}(\boldsymbol{v}_*; \boldsymbol{v}_i, \boldsymbol{\Sigma}_i)$ ▷ Belief about representer weights.
  $\mu_i(\cdot) \leftarrow \mu(\cdot) + k(\cdot, \boldsymbol{X}) \boldsymbol{v}_i$ ▷ Approximate posterior mean function.
  $k_i(\cdot, \cdot) \leftarrow k(\cdot, \cdot) - k(\cdot, \boldsymbol{X}) \boldsymbol{C}_i k(\boldsymbol{X}, \cdot)$ ▷ Combined uncertainty.
  return $\mathcal{GP}(\mu_i, k_i)$

Greyed out quantities are *not* needed to compute the combined posterior and are only included for clarity of exposition.

[Wenger et al NeurIPS 2022]

Learning Representer Weights

- Notice what has happened:
  - Our belief on representer weights captures all computation $p(\boldsymbol{v}_*) = \mathcal{N}(\boldsymbol{v}_* \mid \boldsymbol{v}_i, \boldsymbol{\Sigma}_i)$
  - We have the conditional on the latent $p(\mathbf{f}_\diamond \mid \boldsymbol{v}_*) = \mathcal{N}(\mu(\boldsymbol{X}_\diamond) + k(\boldsymbol{X}_\diamond, \boldsymbol{X})\boldsymbol{v}_*,\ k_*(\boldsymbol{X}_\diamond, \boldsymbol{X}_\diamond))$
  - And voila! $p(\mathbf{f}_\diamond) = \int p(\mathbf{f}_\diamond \mid \boldsymbol{v}_*)p(\boldsymbol{v}_*)\, d\boldsymbol{v}_* = \mathcal{N}(\mathbf{f}_\diamond; \mu_i(\boldsymbol{X}_\diamond), k_i(\boldsymbol{X}_\diamond, \boldsymbol{X}_\diamond))$

$$\mu_i(\cdot) = \mu(\cdot) + k(\cdot, \boldsymbol{X})\boldsymbol{C}_i(\boldsymbol{y} - \boldsymbol{\mu})$$

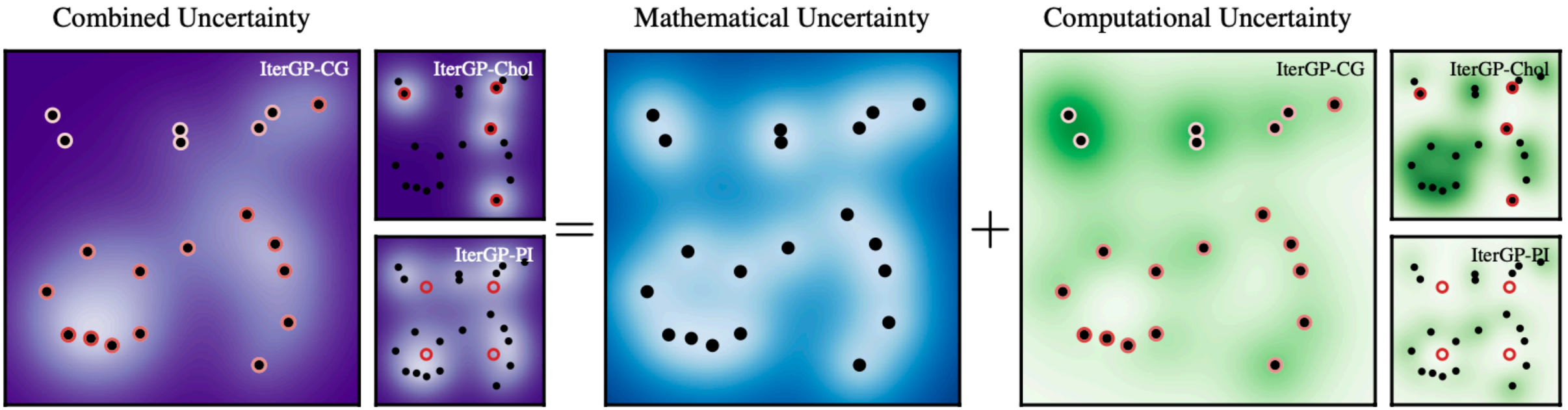
$$k_i(\cdot,\cdot) = \underbrace{k(\cdot,\cdot) - k(\cdot,\boldsymbol{X})\hat{\boldsymbol{K}}^{-1}k(\boldsymbol{X},\cdot)}_{\text{mathematical uncertainty}} + \underbrace{k(\cdot,\boldsymbol{X})\boldsymbol{\Sigma}_i k(\boldsymbol{X},\cdot)}_{\text{computational uncertainty}} = \underbrace{k(\cdot,\cdot) - k(\cdot,\boldsymbol{X})\boldsymbol{C}_i k(\boldsymbol{X},\cdot)}_{\text{combined uncertainty}}$$

- This is of course exactly the form we ended up with earlier $\boldsymbol{S}_i^\intercal \boldsymbol{y} \mid \mathbf{f} \sim \mathcal{N}(\boldsymbol{S}_i^\intercal \mathbf{f}, \sigma^2 \boldsymbol{S}_i^\intercal \boldsymbol{S}_i)$
  - …paying off the claim that “Effective Dataset” == “Representer Weight Belief Update”

Learning Representer Weights

$$\underbrace{k(\cdot,\cdot) - k(\cdot,\boldsymbol{X})\boldsymbol{C}_i k(\boldsymbol{X},\cdot)}_{\text{combined uncertainty}} = \underbrace{k(\cdot,\cdot) - k(\cdot,\boldsymbol{X})\hat{\boldsymbol{K}}^{-1}k(\boldsymbol{X},\cdot)}_{\text{mathematical uncertainty}} + \underbrace{k(\cdot,\boldsymbol{X})\boldsymbol{\Sigma}_i k(\boldsymbol{X},\cdot)}_{\text{computational uncertainty}}$$

$$\boldsymbol{C}_i = \boldsymbol{S}_i(\boldsymbol{S}_i^\intercal \hat{\boldsymbol{K}}\boldsymbol{S}_i)^{-1}\boldsymbol{S}_i^\intercal \qquad \hat{\boldsymbol{K}} := \boldsymbol{K} + \sigma^2\boldsymbol{I} \qquad p(\boldsymbol{v}_*) = \mathcal{N}(\boldsymbol{v}_* \mid \boldsymbol{v}_i, \boldsymbol{\Sigma}_i)$$



Learning Representer Weights

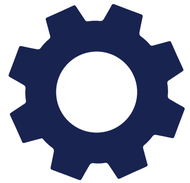
$$\underbrace{k(\cdot,\cdot) - k(\cdot,\boldsymbol{X})\boldsymbol{C}_i k(\boldsymbol{X},\cdot)}_{\text{combined uncertainty}} = \underbrace{k(\cdot,\cdot) - k(\cdot,\boldsymbol{X})\hat{\boldsymbol{K}}^{-1}k(\boldsymbol{X},\cdot)}_{\text{mathematical uncertainty}} + \underbrace{k(\cdot,\boldsymbol{X})\boldsymbol{\Sigma}_i k(\boldsymbol{X},\cdot)}_{\text{computational uncertainty}}$$

$$\boldsymbol{C}_i = \boldsymbol{S}_i(\boldsymbol{S}_i^\intercal \hat{\boldsymbol{K}}\boldsymbol{S}_i)^{-1}\boldsymbol{S}_i^\intercal \qquad \hat{\boldsymbol{K}} := \boldsymbol{K} + \sigma^2\boldsymbol{I} \qquad p(\boldsymbol{v}_*) = \mathcal{N}(\boldsymbol{v}_* \mid \boldsymbol{v}_i, \boldsymbol{\Sigma}_i)$$

- Parting thoughts:
  - Combined uncertainty is tractable and tells us exactly what data we actually consumed
  - Mathematical uncertainty is revealed to be conceptual (sure it's there, but only at cubic cost)
  - Computational uncertainty then is exactly the uncertainty on the representer weights, which carry all computational updates.
- Here then "Approximate GP Inference the Right Way" and "PN treatment of representer weights" are shown to be **identical**.

Outline

- The Promise of Probabilistic Machine Learning
- Gaussian Process Introduction
- Scaling Gaussian Processes, and Implications
- Approximate Gaussian Process Inference, The Right Way
- iterGP as Probabilistic Numerics
- **Broader Implications**



Takeaways

- Approximate inference **should** be taken into account in the Bayesian framework
  - iterGP does so for Gaussian Processes
  - Absent that, computational uncertainty is untracked, and may dominate

- **Data is as data does:**
  - In a very concrete sense, iterGP shows that data only “exists” to the extent that it is ingested by the compute/solver (note: this is next-level PN thinking)
  - Note then for example the practical ease then of iterGP online (actions on new data)

- **Challenge:** What other inference settings admit (tractable) inference of combined uncertainty?

- Active learning / BO foreshadow: here is a means to precisely trade off the cost of collecting another data point vs running more compute on existing data.
  - Combined uncertainty is **exact** regardless of how much compute you do (but of course limited by how much you do)

Thanks

- Questions?

- Special thanks to Jonathan, who has led iterGP (and who will be leading the tutorial this afternoon)

- Jonathan Wenger
- Geoff Pleiss
- Luhuan Wu
- Jacob Gardner
- Marvin Pförtner
- Philipp Hennig
- Dan Biderman
- Andres Potapczynski
- Kelly Buchanan
- Taiga Abe
- Others…

COLUMBIA UNIVERSITY
DEPARTMENT OF STATISTICS

SIMONS
FOUNDATION

COLUMBIA | ZUCKERMAN INSTITUTE
Mortimer B. Zuckerman Mind Brain Behavior Institute

ALFRED P. SLOAN
FOUNDATION

NIH National Institutes of Health

GATSBY

THE
KAVLI
FOUNDATION

NSF

THE McKNIGHT FOUNDATION