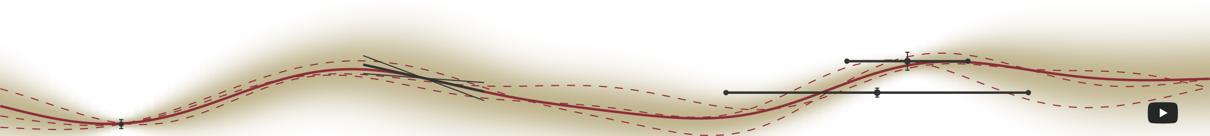
Numerics of Machine Learning
Lecture 04
Computation-Aware GP Inference

Jonathan Wenger

10 November 2022

EBERHARD KARLS UNIVERSITÄT TÜBINGEN

Faculty of Science
Department of Computer Science
Chair for the Methods of Machine Learning



Where are we in the course?

- Last week: Contemporary way of solving linear systems for GP regression on large datasets
- This week: Probabilistic numerics approach to (approximate) GP regression

Today

- **Learning to approximate GPs** with probabilistic numerics.
- Quantifying approximation error probabilistically.
- Iterative numerical methods for GPs as active learning agents.
- Philosophical connections between data and computation.
- **Exact uncertainty quantification for GPs in (sub-)quadratic time.**

Recap: Scalable GP Approximations

Recap: Gaussian Process Regression

An archetypical supervised machine learning model.

Goal: Learn an unknown function $f_* : \mathbb{R}^d \to \mathbb{R}$ from a training dataset of example input-output pairs.

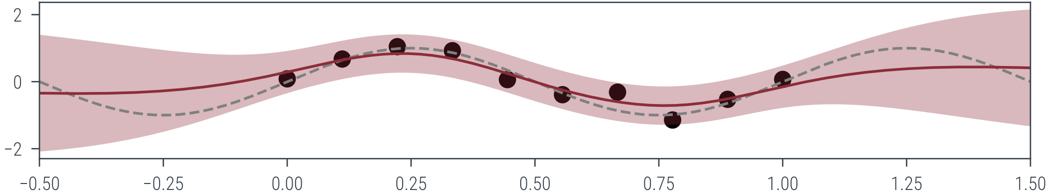
$$f \sim \mathcal{GP}(\mu, k)$$

$$y \mid f(X) \sim \mathcal{N}(f(X), \sigma^2 I)$$

$$f \mid X, y \sim \mathcal{GP}(\mu_{\text{post}}, k_{\text{post}})$$

$$\mu_{\text{post}}(x) = \mu(x) + k(x, X)(k(X, X) + \sigma^2 I)^{-1}(y - \mu(X))$$

$$k_{\text{post}}(x_0, x_1) = k(x_0, x_1) - k(x_0, X)(k(X, X) + \sigma^2 I)^{-1} k(X, x_1)$$



Recap: Gaussian Process Inference via the Partial Cholesky

Performing Gaussian process inference with a learned inverse approximation via the partial Cholesky decomposition.

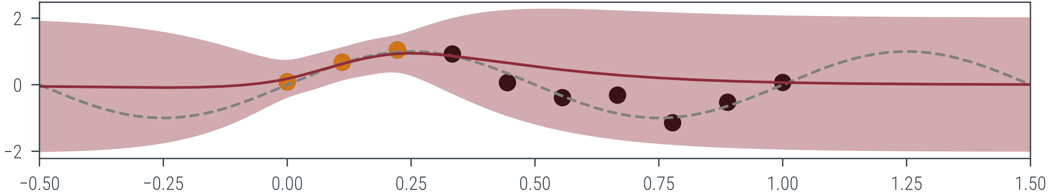
$$f \sim \mathcal{GP}(\mu, k)$$

$$y \mid f(X) \sim \mathcal{N}(f(X), \sigma^2 I)$$

$$f \mid X, y \sim \mathcal{GP}(\mu_{\text{post}}, k_{\text{post}})$$

$$\mu_{\text{post}}(x) = \mu(x) + k(x, X) C_i (y - \mu(X))$$

$$k_{\text{post}}(x_0, x_1) = k(x_0, x_1) - k(x_0, X) C_i k(X, x_1)$$



Recap: Learning to Invert the Kernel Matrix

The Cholesky decomposition as a learning algorithm for the inverse kernel matrix.

Algorithm Cholesky with Inverse Approximation

Input: spd matrix A

Output: lower triangular L_i, s.t. $L_i L_i^\top \approx A$, low-rank $C_i \approx A^{-1}$

```
procedure CHOLESKY($A$)
    $A' \leftarrow A$, $C_0 = 0$
    for $i \in \{1, \dots, n\}$ do
        $s_i \leftarrow e_i$                                              // Action
        $d_i \leftarrow (I - C_{i-1} A) s_i$
        $\eta_i \leftarrow s_i^\top A d_i = e_i^\top A' e_i = \|e_i\|_{A'}^2$        // Norm. constant
        $l_i \leftarrow A \frac{1}{\sqrt{\eta_i}} d_i$                                // Matrix observation
        $C_i \leftarrow C_{i-1} + \frac{1}{\eta_i} d_i d_i^\top$                          // Inverse estimate
        $A' \leftarrow A - L_i L_i^\top = A(A^{-1} - C_i)A = A(I - C_i A)$
        $L_i = (L_{i-1} \quad l_i)$
    end for
    return $L_i$, $C_i$
end procedure
```

Goal: (Low-rank) Approximation $C_i \approx A^{-1}$

Observation: Matrix approx. → inverse approx.?

$$L_i L_i^\top \approx A$$

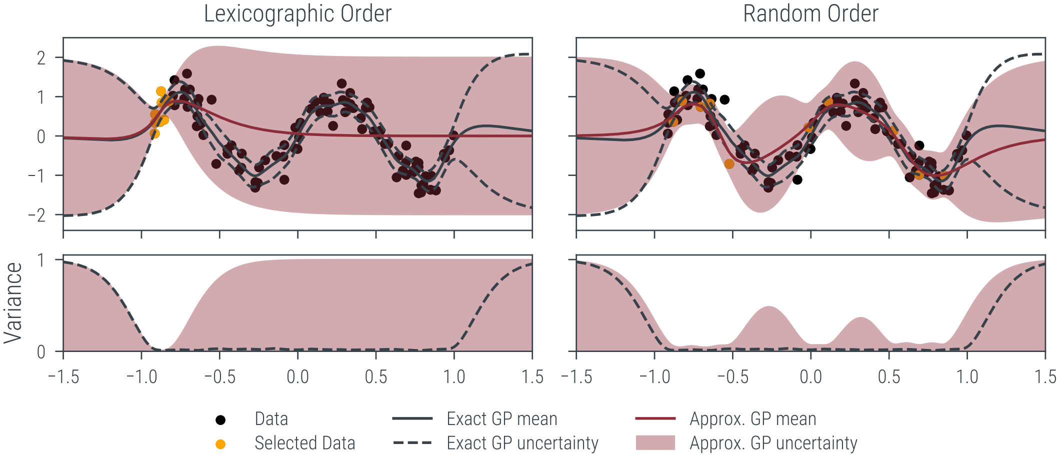
$$\underbrace{(A^{-1} L_i)(A^{-1} L_i)^\top}_{= C_i} \approx A^{-1}$$

Computational complexity: $\#\text{flops} \in \mathcal{O}(in^2)$

Cholesky can be seen as an iterative learning algorithm for the kernel matrix and its inverse.

Recap: Interpreting the Pivoting Strategy as Active Learning

In each iteration the partial Cholesky selects a datapoint as a pivot via its action.



The selection of datapoints, i.e. choice of actions $\boldsymbol{s}_i$, matters a lot for convergence.

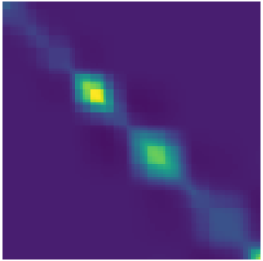
Recap: Can we find better actions?

Why restrict ourselves to just unit vectors to probe the matrix residual?

Partial Cholesky

$$A'e_i = A(I - C_{i-1}A)s_i = Ad_i$$

$=$

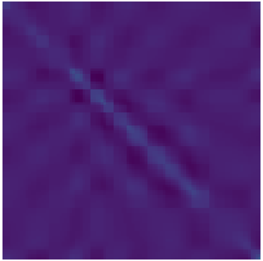


$$\begin{pmatrix} 0 \\ \vdots \\ 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$

Other Method?

$$A'e_i = A(I - C_{i-1}A)s_i = Ad_i$$

$=$



$$\begin{pmatrix} * \\ \vdots \\ * \\ * \\ * \\ \vdots \\ * \end{pmatrix}$$

Can we learn the kernel matrix (inverse) in a more efficient way via different actions?

Recap: Method of Conjugate Gradients

Efficiently solving linear systems with positive definite system matrix via matrix-vector multiplies.

Goal: Approximately solve linear system $Ax = b$ with few matrix-vector multiplies.

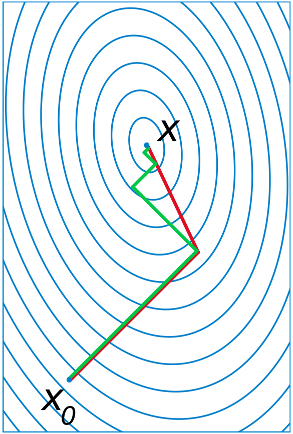
Idea: Rephrase as quadratic optimization problem and optimize. Let

$$f(x) = \frac{1}{2} x^\intercal A x - b^\intercal x$$

then $\nabla f(x) = 0 \iff Ax = b \iff r(x) := b - Ax = 0$.

Question: How should we optimize?

1. Gradient descent: Follow $d_i = r(x_i) = -\nabla f(x_i)$ s.t. $\langle d_i, d_j \rangle = 0$.
2. Conjugate direction method: Follow d_i s. t. $\langle d_i^\intercal d_j \rangle_A = d_i^\intercal A d_j = 0$ for $i \neq j$.
   $\implies$ convergence in at most n steps.
3. Conjugate gradient method: First step $d_0 = r(x_0)$.



Oleg Alexandrov, commons.wikimedia.org/w/index.php?curid=2267598

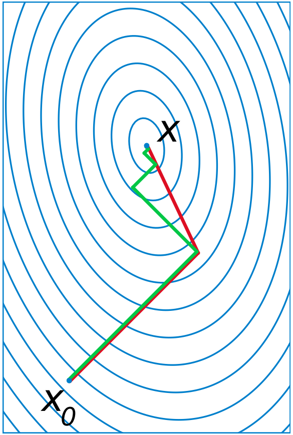
Recap: Algorithm: Method of Conjugate Gradients

We can interpret CG as a learning algorithm for the matrix inverse as well.

Algorithm CG with Inverse Approximation

Input: spd matrix A, vector b, initial guess x_0
Output: approximate solution $x_i \approx A^{-1}b$, low-rank $C_i \approx A^{-1}$

```
procedure CG($A, b, x_0$)
    while $||r_i||_2 > \max(\delta_{\text{rtol}}||b||_2, \delta_{\text{atol}})$ do
        $r_{i-1} \leftarrow b - Ax_{i-1}$                                   // Residual
        $s_i \leftarrow r_{i-1}$                                            // Action
        $\alpha_i \leftarrow s_i^\top r_{i-1}$                              // Observation
        $d_i \leftarrow (I - C_{i-1}A)s_i$                                  // Search direction
        $\eta_i \leftarrow s_i^\top A d_i = d_i^\top A d_i$                 // Norm. constant
        $C_i \leftarrow C_{i-1} + \frac{1}{\eta_i} d_i d_i^\top$            // Inverse estimate
        $x_i \leftarrow x_{i-1} + \frac{\alpha_i}{\eta_i} d_i = C_i b$      // Solution estimate
    end while
    return $x_i, C_i$
end procedure
```



Oleg Alexandrov, commons.wikimedia.org/w/index.php?curid=2267598

Recap: Algorithm: Method of Conjugate Gradients

We can interpret CG as a learning algorithm for the matrix inverse as well.

Algorithm CG with Inverse Approximation

Input: spd matrix A, vector b, initial guess x_0
Output: approximate solution $x_i \approx A^{-1}b$, low-rank $C_i \approx A^{-1}$

```
procedure CG($A, b, x_0$)
    while $\|r_i\|_2 > \max(\delta_{\text{rtol}}\|b\|_2, \delta_{\text{atol}})$ do
        $r_{i-1} \leftarrow b - Ax_{i-1}$                                  // Residual
        $s_i \leftarrow r_{i-1}$                                           // Action
        $\alpha_i \leftarrow s_i^\top r_{i-1}$                             // Observation
        $d_i \leftarrow (I - C_{i-1}A)s_i$                                 // Search direction
        $\eta_i \leftarrow s_i^\top A d_i = d_i^\top A d_i$                // Norm. constant
        $C_i \leftarrow C_{i-1} + \frac{1}{\eta_i} d_i d_i^\top$           // Inverse estimate
        $x_i \leftarrow x_{i-1} + \frac{\alpha_i}{\eta_i} d_i = C_i b$     // Solution estimate
    end while
    return $x_i, C_i$
end procedure
```

Algorithm Cholesky with Inverse Approximation

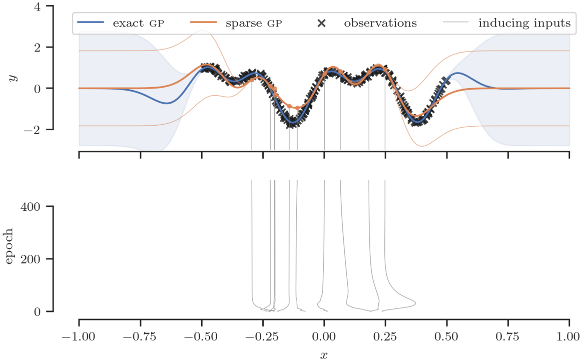
Input: spd matrix A
Output: lower triangular L_i, s.t. $L_i L_i^\top \approx A$, low-rank $C_i \approx A^{-1}$

```
procedure CHOLESKY($A$)
    $A' \leftarrow A$, $C_0 = 0$
    for $i \in \{1, \dots, n\}$ do
        $s_i \leftarrow e_i$                                                  // Action
        $d_i \leftarrow (I - C_{i-1}A)s_i$
        $\eta_i \leftarrow s_i^\top A d_i = e_i^\top A' e_i = \|e_i\|_{A'}^2$,   // Norm. constant
        $l_i \leftarrow A \frac{1}{\sqrt{\eta_i}} d_i$                        // Matrix observation
        $C_i \leftarrow C_{i-1} + \frac{1}{\eta_i} d_i d_i^\top$              // Inverse estimate
        $A' \leftarrow A - L_i L_i^\top = A(A^{-1} - C_i)A = A(I - C_i A)$
        $L_i = (L_{i-1} \quad l_i)$
    end for
    return $L_i, C_i$
end procedure
```

Recap: Stochastic Variational Gaussian Processes

Titsias. *Variational learning of inducing variables in sparse Gaussian processes*, AISTATS, 2009.
Hensman, Fusi, Lawrence. *Gaussian Processes for Big Data*, UAI, 2013

Idea: Linear time GP approximation via inducing points.



Source: https://tiao.io/post/sparse-variational-gaussian-processes/

Can we design a method where we can trust the UQ *no matter how much computation we've done*?

Exact UQ for GP approximation with arbitrary amounts of compute.

Computation-aware GP Inference

Coming Up

Posterior and Computational Uncertainty in Gaussian Processes

Jonathan Wenger, Geoff Pleiss, Marvin Pförtner, Philipp Hennig and John Cunningham

- IterGP: new class of GP approximations *accounting for computational uncertainty*.
- IterGP instances extend classic methods (Cholesky, CG, Nyström, ...).
- Strong theoretical guarantees.
- Modeling computational uncertainty either *saves computation* or *improves generalization*.

Paper	arXiv	`https://arxiv.org/abs/2107.00243`
Implementation		`https://github.com/JonathanWenger/itergp`

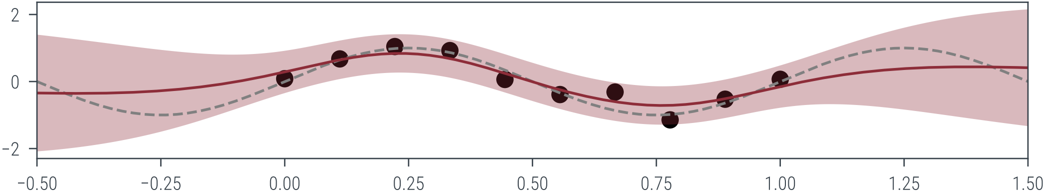
Representer Weights

A second look at Gaussian process regression.

$$f \mid X, y \sim \mathcal{GP}(\mu_{\text{post}}, k_{\text{post}})$$

$$\mu_{\text{post}}(x) = \mu(x) + k(x, X)(k(X, X) + \sigma^2 I)^{-1}(y - \mu(X))$$

$$k_{\text{post}}(x_0, x_1) = k(x_0, x_1) - k(x_0, X)(k(X, X) + \sigma^2 I)^{-1} k(X, x_1)$$



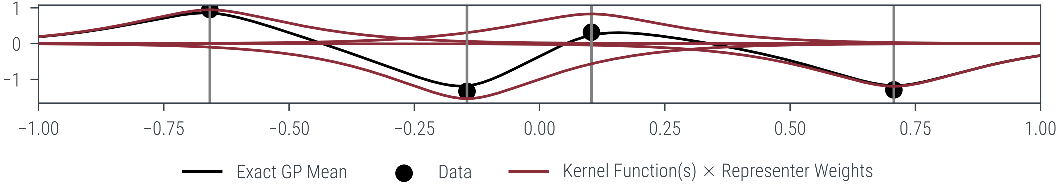
Representer Weights

A second look at Gaussian process regression.

$$f \mid X, y \sim \mathcal{GP}(\mu_{\text{post}}, k_{\text{post}})$$

$$\mu_{\text{post}}(x) = \mu(x) + k(x, X)(k(X, X) + \sigma^2 I)^{-1}(y - \mu(X))$$

$$= \mu(x) + k(x, X) \underbrace{v_*}_{\text{representer weights}} = \mu(x) + \sum_{j=1}^{n} k(x, x_j)(v_*)_j$$



The posterior mean is a linear combination of kernel functions centered at datapoints.

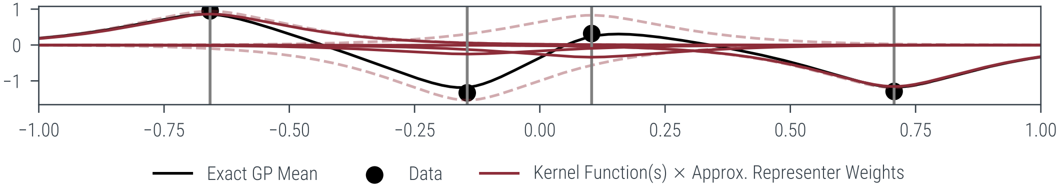
Approximating Representer Weights

Iterative linear solvers approximate the representer weights.

Observation: Iterative linear solvers (e.g. CG) approximate the representer weights $v_i \approx v_* = \hat{K}^{-1} y$.

$$\mu_{\text{post}}(x) = \mu(x) + k(x, X) \underbrace{v_*}_{\text{representer weights}} = \mu(x) + \sum_{j=1}^{n} k(x, x_j)(v_*)_j$$

$$\approx \mu(x) + k(x, X) v_i$$



Can we quantify the approximation error $||v_* - v_i||$ in the representer weights *probabilistically*?.

Interlude: Gaussians provide the **Linear Algebra of Inference**

If all joints are Gaussian and all observations are linear, all posteriors are Gaussian.

- products of Gaussians are Gaussians

$$\begin{aligned}&\mathcal{N}(x;a,A)\mathcal{N}(x;b,B)\\ &= \mathcal{N}(x;c,C)\mathcal{N}(a;b,A+B)\end{aligned}$$

$$C := (A^{-1}+B^{-1})^{-1} \quad c := C(A^{-1}a+B^{-1}b)$$

- linear projections of Gaussians are Gaussians

$$\begin{aligned}&p(z) = \mathcal{N}(z;\mu,\Sigma)\\ \Rightarrow\quad &p(Az) = \mathcal{N}(Az, A\mu, A\Sigma A^\intercal)\end{aligned}$$

- marginals of Gaussians are Gaussians

$$\int \mathcal{N}\left[\begin{pmatrix}x\\y\end{pmatrix};\begin{pmatrix}\mu_x\\\mu_y\end{pmatrix},\begin{pmatrix}\Sigma_{xx} & \Sigma_{xy}\\ \Sigma_{yx} & \Sigma_{yy}\end{pmatrix}\right]dy = \mathcal{N}(x;\mu_x,\Sigma_{xx})$$

- (linear) conditionals of Gaussians are Gaussians

$$\begin{aligned}p(x \mid y) &= \frac{p(x,y)}{p(y)}\\ &= \mathcal{N}\left(x;\mu_x+\Sigma_{xy}\Sigma_{yy}^{-1}(y-\mu_y), \Sigma_{xx}-\Sigma_{xy}\Sigma_{yy}^{-1}\Sigma_{yx}\right)\end{aligned}$$

Bayesian inference becomes linear algebra

If $p(x) = \mathcal{N}(x;\mu,\Sigma)$ and $p(y \mid x) = \mathcal{N}(y;A^\intercal x+b,\Lambda)$, then

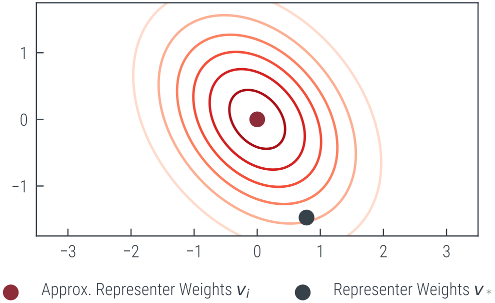
$$p(B^\intercal x+c \mid y) = \mathcal{N}[B^\intercal x+c; B^\intercal\mu+c+B^\intercal\Sigma A(A^\intercal\Sigma A+\Lambda)^{-1}(y-A^\intercal\mu-b), B^\intercal\Sigma B-B^\intercal\Sigma A(A^\intercal\Sigma A+\Lambda)^{-1}A^\intercal\Sigma B]$$

Probabilistic Linear Solvers

Learning the solution of linear systems while quantifying computational uncertainty.

Goal: Quantify approximation error when solving $v_* = \hat{K}^{-1} y$ probabilistically, i.e. $v_* \sim \mathcal{N}(v_i, \Sigma_i)$.

Prior: $v_* \sim \mathcal{N}(v_0, \Sigma_0)$



Probabilistic Linear Solvers

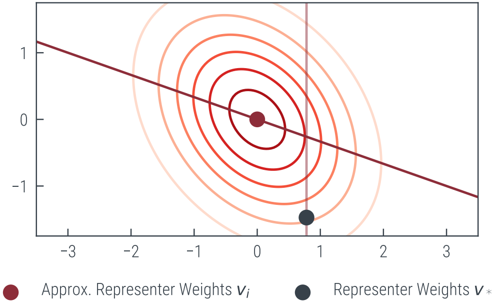
Learning the solution of linear systems while quantifying computational uncertainty.

Goal: Quantify approximation error when solving $v_* = \hat{K}^{-1} y$ probabilistically, i.e. $v_* \sim \mathcal{N}(v_i, \Sigma_i)$.

Prior: $v_* \sim \mathcal{N}(v_0, \Sigma_0)$

Likelihood: Observe representer weights indirectly via matrix-vector multiplication with the residual:

$$\alpha_i := s_i^\top r_{i-1} = s_i^\top((y - \boldsymbol{\mu}) - \hat{K} v_{i-1}) = s_i^\top \hat{K}(v_* - v_{i-1})$$



Probabilistic Linear Solvers

Learning the solution of linear systems while quantifying computational uncertainty.

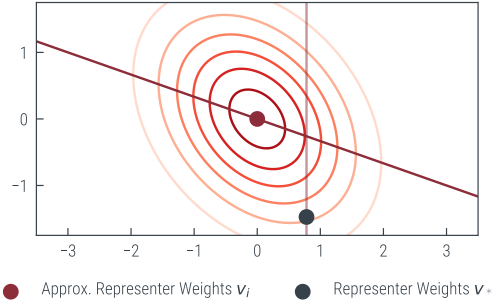
Goal: Quantify approximation error when solving $v_* = \hat{K}^{-1} y$ probabilistically, i.e. $v_* \sim \mathcal{N}(v_i, \Sigma_i)$.

Prior: $v_* \sim \mathcal{N}(v_0, \Sigma_0)$

Likelihood: Observe representer weights indirectly via matrix-vector multiplication with the residual:

$$\alpha_i := s_i^\intercal r_{i-1} = s_i^\intercal ((y - \boldsymbol{\mu}) - \hat{K} v_{i-1}) = s_i^\intercal \hat{K}(v_* - v_{i-1})$$

Posterior: Affine Gaussian inference!



Probabilistic Linear Solvers

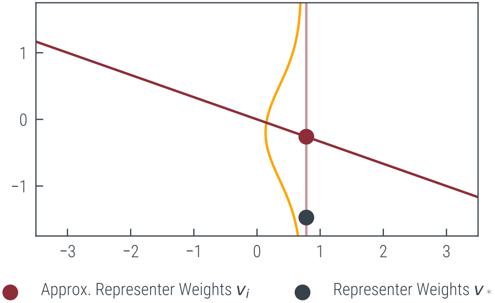
Learning the solution of linear systems while quantifying computational uncertainty.

Goal: Quantify approximation error when solving $v_* = \hat{K}^{-1} y$ probabilistically, i.e. $v_* \sim \mathcal{N}(v_i, \boldsymbol{\Sigma}_i)$.

Prior: $v_* \sim \mathcal{N}(v_0, \boldsymbol{\Sigma}_0)$

Likelihood: Observe representer weights indirectly via matrix-vector multiplication with the residual:

$$\alpha_i := s_i^\intercal r_{i-1} = s_i^\intercal((y - \boldsymbol{\mu}) - \hat{K} v_{i-1}) = s_i^\intercal \hat{K}(v_* - v_{i-1})$$



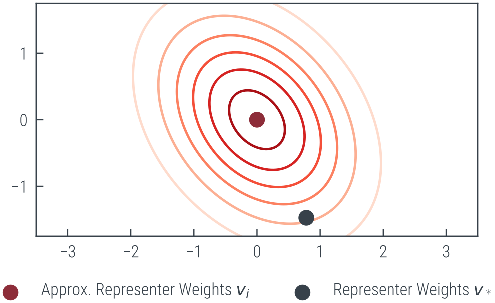
Posterior: $v_* \mid \alpha_i \sim \mathcal{N}(v_i, \boldsymbol{\Sigma}_i)$, where

$$\begin{aligned} v_i &= v_{i-1} + \overbrace{\boldsymbol{\Sigma}_{i-1}\hat{K}s_i}^{=:d_i}(\overbrace{s_i^\intercal \hat{K}\boldsymbol{\Sigma}_{i-1}\hat{K}s_i}^{=:\eta_i})^{-1}\overbrace{s_i^\intercal \hat{K}(v_* - v_{i-1})}^{=\alpha_i} \\ &= C_i(y - \boldsymbol{\mu}) \\ \boldsymbol{\Sigma}_i &= \boldsymbol{\Sigma}_{i-1} - \overbrace{\boldsymbol{\Sigma}_{i-1}\hat{K}s_i}^{=d_i}(\overbrace{s_i^\intercal \hat{K}\boldsymbol{\Sigma}_{i-1}\hat{K}s_i}^{=\eta_i})^{-1}\overbrace{s_i^\intercal \hat{K}\boldsymbol{\Sigma}_{i-1}}^{=d_i^\intercal} \\ &= \boldsymbol{\Sigma}_0 - C_i = \boldsymbol{\Sigma}_0 - \sum_{j=1}^{i}\frac{1}{\eta_j}d_j d_j^\intercal = \boldsymbol{\Sigma}_0 - S_i(S_i^\intercal \hat{K} S_i)^{-1} S_i^\intercal \end{aligned}$$

Choosing the “Right” Linear Solver Prior

The GP makes assumptions about the representer weights.

Problem: How to choose the linear solver prior?



Choosing the “Right” Linear Solver Prior

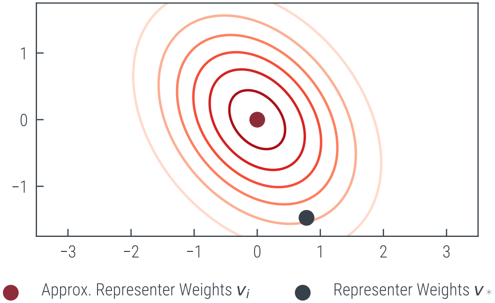
The GP makes assumptions about the representer weights.

Problem: How to choose the linear solver prior?

Remember: $\boldsymbol{y} = f(\boldsymbol{X}) + \boldsymbol{\varepsilon}$, where $\boldsymbol{\varepsilon} \sim \mathcal{N}(\boldsymbol{0}, \sigma^2 \boldsymbol{I})$.

Gaussian process prior $f \sim \mathcal{GP}(0, k)$ gives:

$$\boldsymbol{y} \sim \mathcal{N}(\boldsymbol{0}, k(\boldsymbol{X}, \boldsymbol{X}) + \sigma^2 \boldsymbol{I}) = \mathcal{N}(\boldsymbol{0}, \hat{\boldsymbol{K}})$$



Choosing the “Right” Linear Solver Prior

The GP makes assumptions about the representer weights.

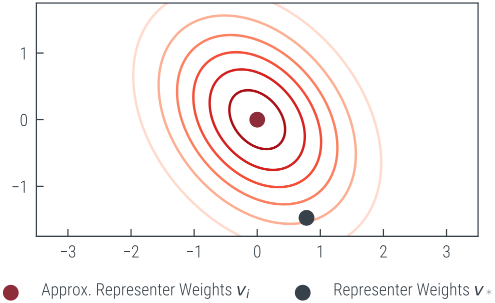
Problem: How to choose the linear solver prior?

Remember: $y = f(X) + \varepsilon$, where $\varepsilon \sim \mathcal{N}(\mathbf{0}, \sigma^2 I)$.

Gaussian process prior $f \sim \mathcal{GP}(0, k)$ gives:

$$y \sim \mathcal{N}(\mathbf{0}, k(X, X) + \sigma^2 I) = \mathcal{N}(\mathbf{0}, \hat{K})$$

$$v_* = \hat{K}^{-1} y \sim \mathcal{N}(\mathbf{0}, \hat{K}^{-1}) = \mathcal{N}(v_0, \Sigma_0)$$



● Approx. Representer Weights v_i ● Representer Weights v_*

Chicken & Egg Problem: How can we get a probabilistic error estimate for $v_i \approx v_*$, if we need $\hat{K}^{-1}$ for it?

Computation-Aware Gaussian Process Inference

Combining mathematical and computational uncertainty.

Gaussian Processes

Mathematical posterior: $\boldsymbol{f}_\diamond \mid \boldsymbol{v}_* \sim \mathcal{N}(\mu_*(\boldsymbol{X}_\diamond), k_*(\boldsymbol{X}_\diamond, \boldsymbol{X}_\diamond))$, s.t.

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

Learning the Representer Weights

Infer representer weights via probabilistic linear solver: $p(\boldsymbol{v}_*) = \mathcal{N}(\boldsymbol{v}_*; \boldsymbol{v}_i, \boldsymbol{\Sigma}_i)$, s.t.

$$\boldsymbol{v}_i = \boldsymbol{C}_i(\boldsymbol{y} - \boldsymbol{\mu}) \quad \text{and} \quad \boldsymbol{\Sigma}_i = \boldsymbol{\Sigma}_0 - \boldsymbol{C}_i = \hat{\boldsymbol{K}}^{-1} - \boldsymbol{C}_i$$

Combined Uncertainty

Marginal distribution: $p(\boldsymbol{f}_\diamond) = \int p(\boldsymbol{f}_\diamond \mid \boldsymbol{v}_*)p(\boldsymbol{v}_*)\, d\boldsymbol{v}_* = \mathcal{N}(\boldsymbol{f}_\diamond; \mu_i(\boldsymbol{X}_\diamond), k_i(\boldsymbol{X}_\diamond, \boldsymbol{X}_\diamond))$, s.t.

$$\begin{aligned} \mu_i(\cdot) &= \mu(\cdot) + k(\cdot,\boldsymbol{X})\boldsymbol{v}_i \\ k_i(\cdot,\cdot) &= \underbrace{k(\cdot,\cdot) - k(\cdot,\boldsymbol{X})\hat{\boldsymbol{K}}^{-1}k(\boldsymbol{X},\cdot)}_{\text{mathematical uncertainty 🔵}} + \underbrace{k(\cdot,\boldsymbol{X})\boldsymbol{\Sigma}_i k(\boldsymbol{X},\cdot)}_{\text{computational uncertainty 🟢}} = \underbrace{k(\cdot,\cdot) - k(\cdot,\boldsymbol{X})\boldsymbol{C}_i k(\boldsymbol{X},\cdot)}_{\text{combined uncertainty 🟣}} \end{aligned}$$

Probabilistic Quantification of Approximation Error

The covariance can be interpreted as a squared error.

Combined Uncertainty

Belief about the true function is captured by $f \sim \mathcal{GP}(\mu_i, k_i)$, s.t.

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

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

Remember: $\operatorname{Cov}(f(\boldsymbol{x}), f(\boldsymbol{x})) = \mathbb{E}[(f(\boldsymbol{x}) - \mathbb{E}[f(\boldsymbol{x})])^2] = \mathbb{E}[(f(\boldsymbol{x}) - \mu(\boldsymbol{x}))^2]$

Probabilistic Quantification of Approximation Error

The covariance can be interpreted as a squared error.

Combined Uncertainty

Belief about the true function is captured by $f \sim \mathcal{GP}(\mu_i, k_i)$, s.t.

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

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

Remember: $\operatorname{Cov}(f(x), f(x)) = \mathbb{E}[(f(x) - \mathbb{E}[f(x)])^2] = \mathbb{E}[(f(x) - \mu(x))^2]$

$$k_{\text{post}}(x,x) = \underbrace{k(x,x) - k(x,\boldsymbol{X})\hat{\boldsymbol{K}}^{-1}k(\boldsymbol{X},x)}_{\text{mathematical uncertainty } \bullet} = \mathbb{E}[(f(x) - \mu_*(x))^2]$$

Probabilistic Quantification of Approximation Error

The covariance can be interpreted as a squared error.

Combined Uncertainty

Belief about the true function is captured by $f \sim \mathcal{GP}(\mu_i, k_i)$, s.t.

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

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

Remember: $\operatorname{Cov}(f(x), f(x)) = \mathbb{E}[(f(x) - \mathbb{E}[f(x)])^2] = \mathbb{E}[(f(x) - \mu(x))^2]$

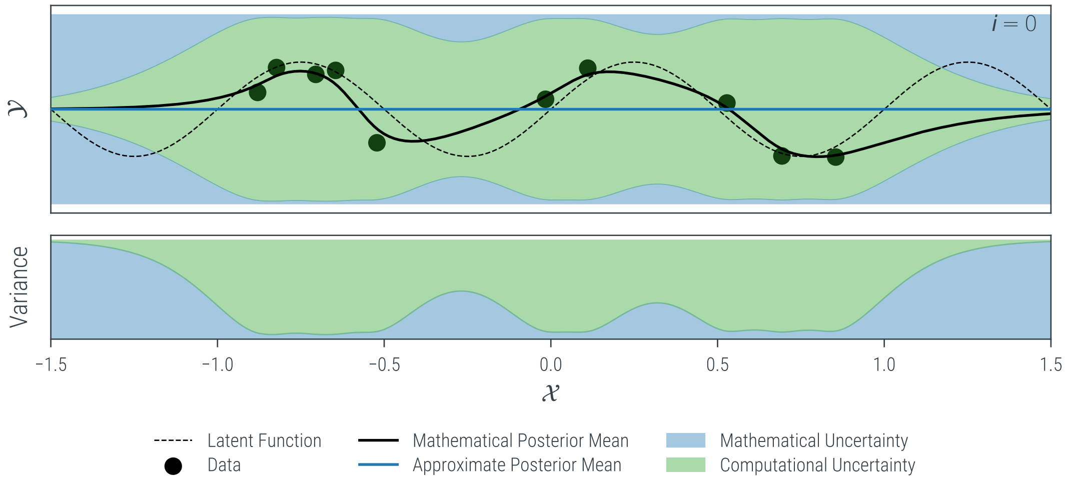
$$k_{\text{post}}(x,x) = \underbrace{k(x,x) - k(x,\boldsymbol{X})\hat{\boldsymbol{K}}^{-1}k(\boldsymbol{X},x)}_{\text{mathematical uncertainty 🔵}} = \mathbb{E}[(f(x) - \mu_*(x))^2]$$

$$k_i^{\text{comp}}(x,x) = \underbrace{k(x,\boldsymbol{X})\boldsymbol{\Sigma}_i k(\boldsymbol{X},x)}_{\text{computational uncertainty 🟢}} \underbrace{=}_{\boldsymbol{\Sigma}_i = \operatorname{Cov}(\boldsymbol{v}_*) = \mathbb{E}[(\boldsymbol{v}_* - \boldsymbol{v}_i)^2]} \mathbb{E}[(\mu_*(x) - \mu_i(x))^2]$$

Computation-Aware GP Inference Illustrated

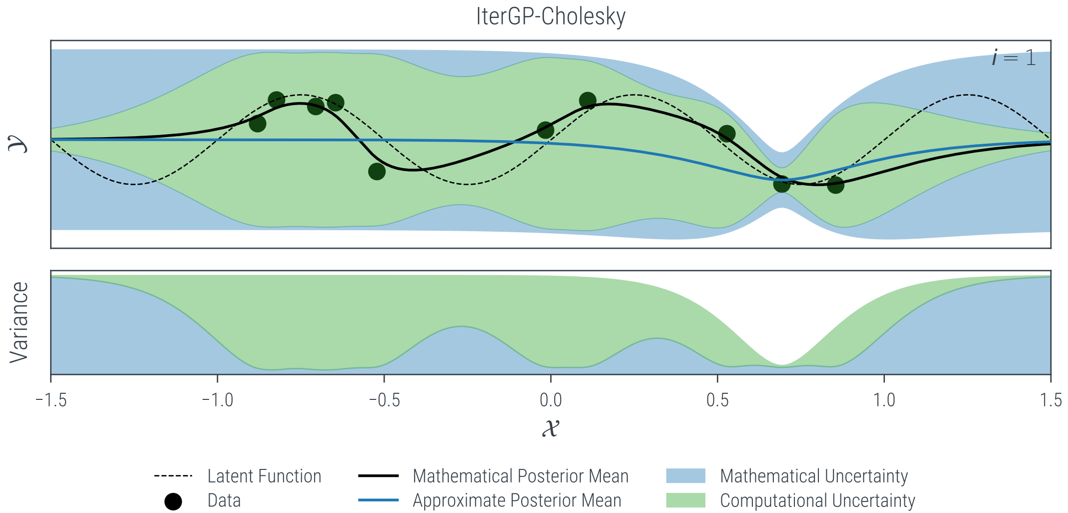
Interpreting computational and combined uncertainty as error quantification.

IterGP-Cholesky



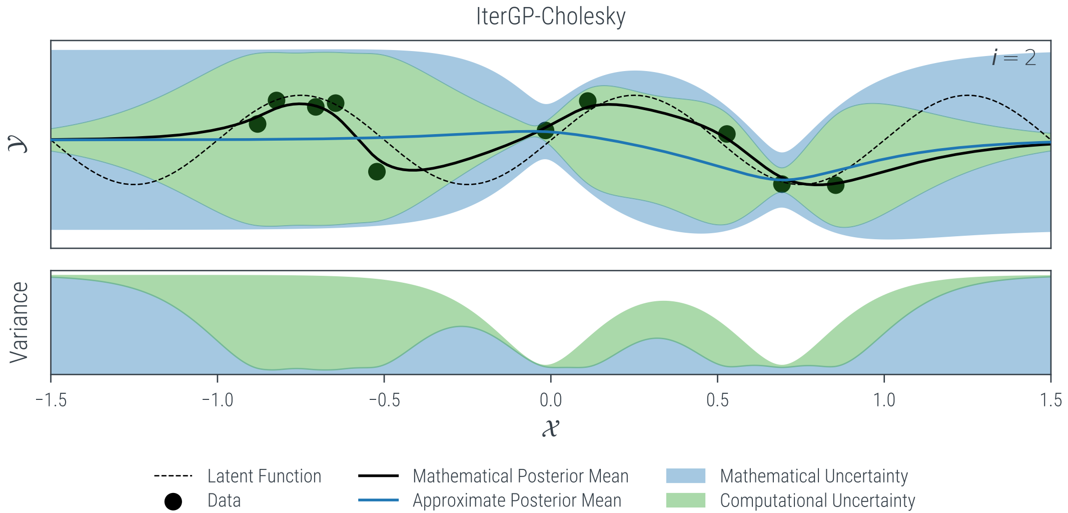
Computation-Aware GP Inference Illustrated

Interpreting computational and combined uncertainty as error quantification.



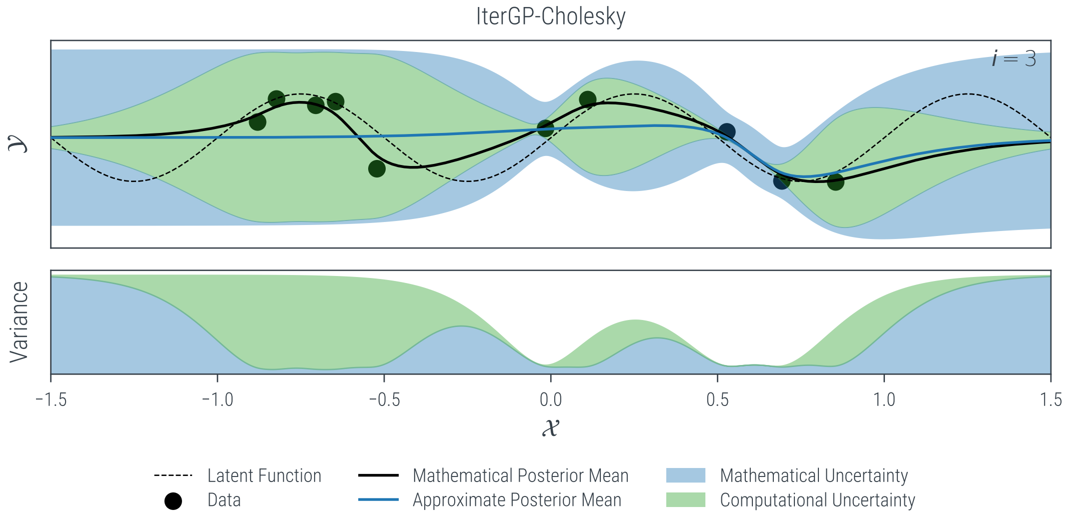
Computation-Aware GP Inference Illustrated

Interpreting computational and combined uncertainty as error quantification.



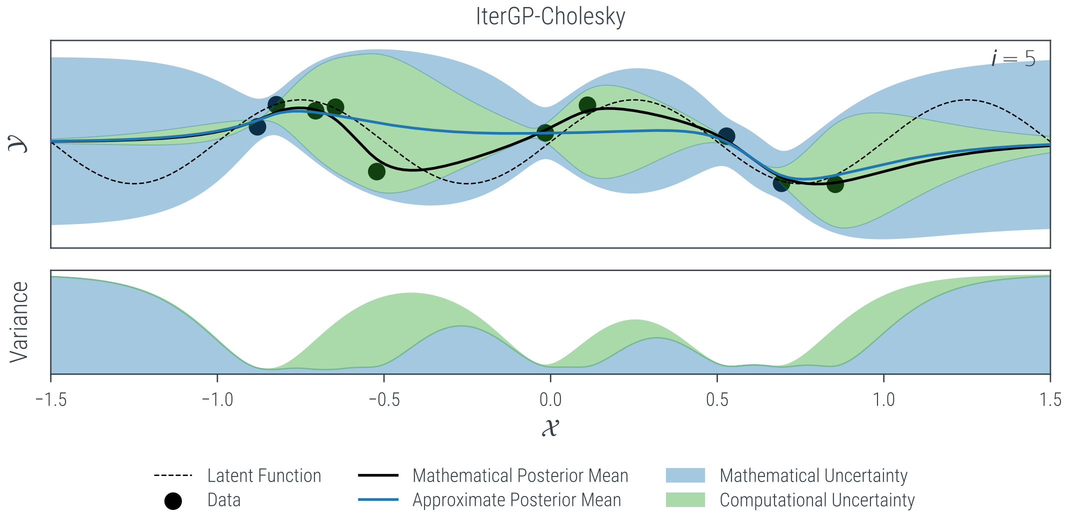
Computation-Aware GP Inference Illustrated

Interpreting computational and combined uncertainty as error quantification.



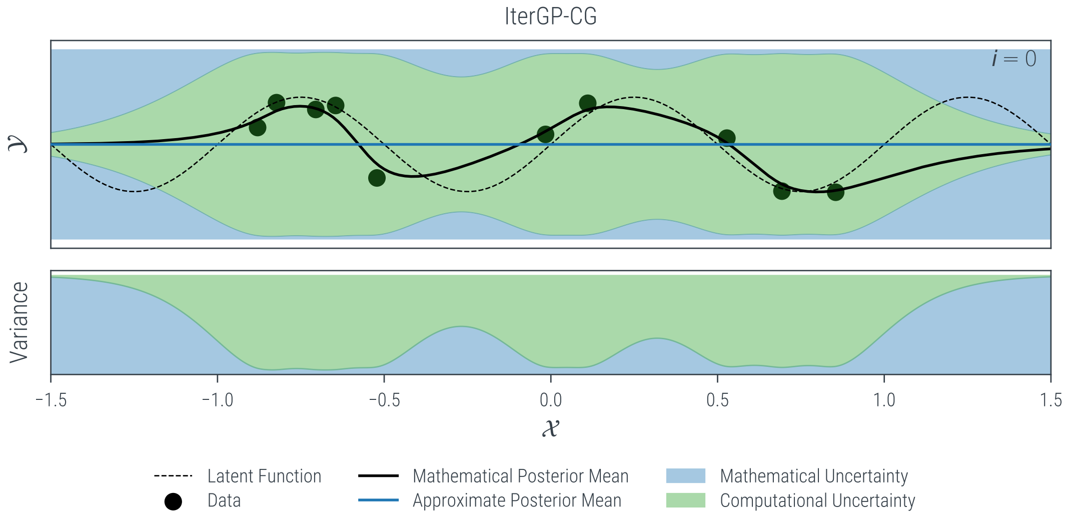
Computation-Aware GP Inference Illustrated

Interpreting computational and combined uncertainty as error quantification.



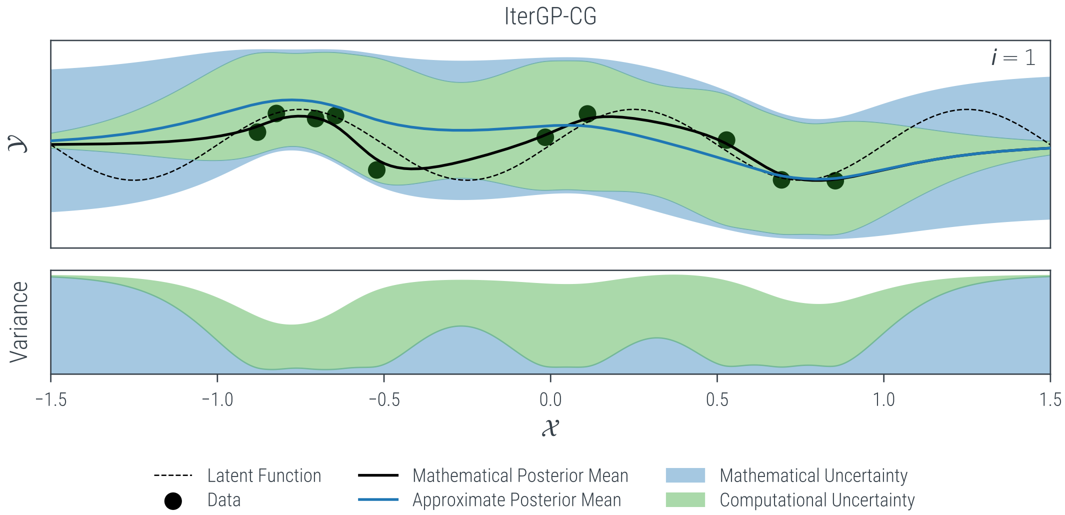
Computation-Aware GP Inference Illustrated

Interpreting computational and combined uncertainty as error quantification.



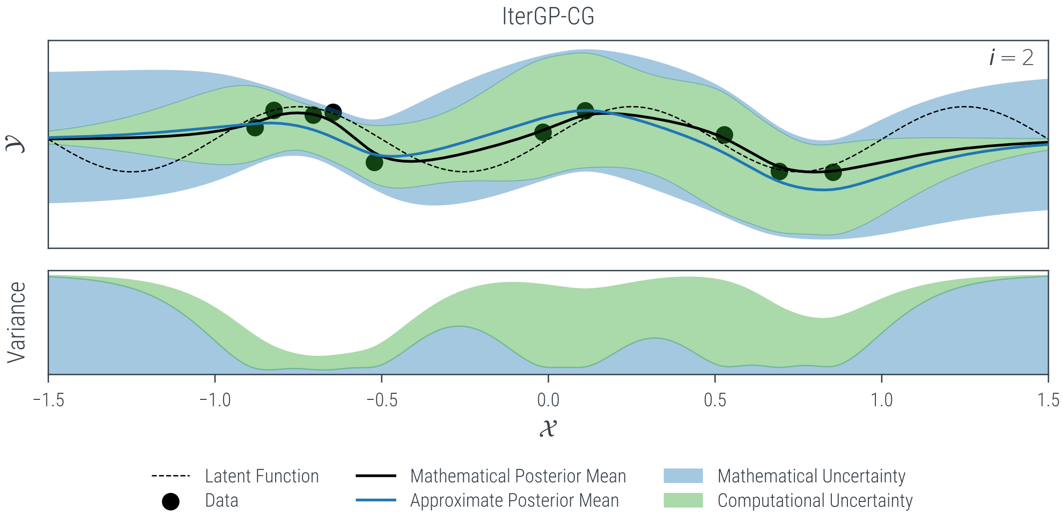
Computation-Aware GP Inference Illustrated

Interpreting computational and combined uncertainty as error quantification.



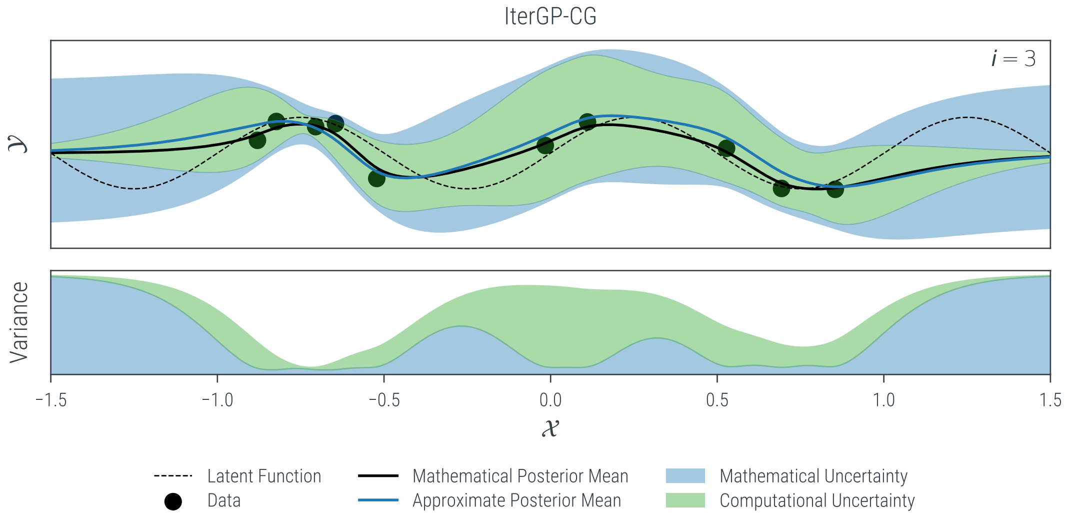
Computation-Aware GP Inference Illustrated

Interpreting computational and combined uncertainty as error quantification.



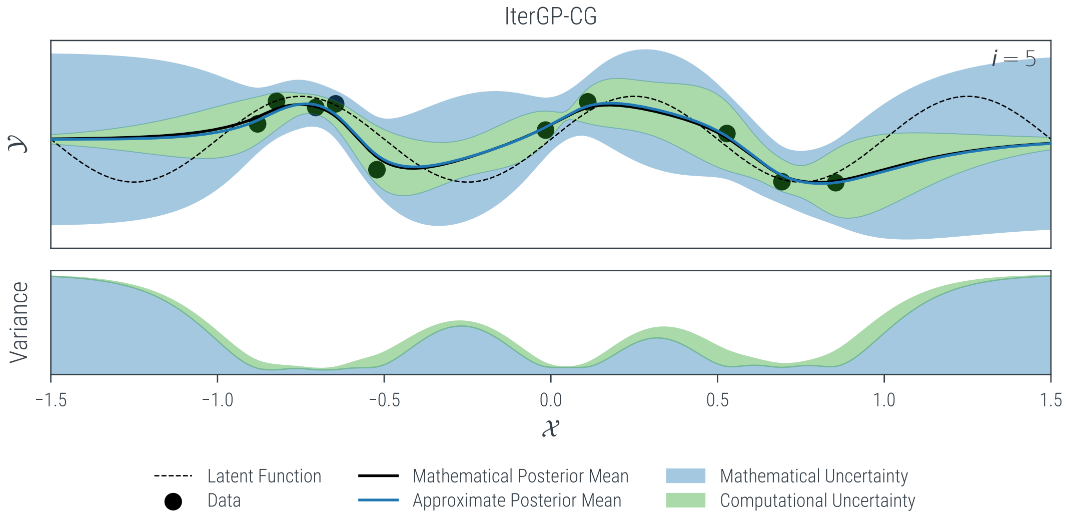
Computation-Aware GP Inference Illustrated

Interpreting computational and combined uncertainty as error quantification.



Computation-Aware GP Inference Illustrated

Interpreting computational and combined uncertainty as error quantification.



Algorithm: IterGP

A class of computation-aware iterative methods for GP approximation.

Input: prior mean μ, prior kernel k, training data $\boldsymbol{X}, \boldsymbol{y}$

Output: (combined) GP posterior $\mathcal{GP}(\mu_i, k_i)$

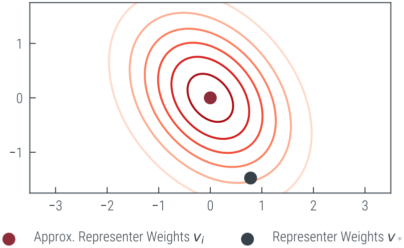
```
procedure IterGP($\mu, k, \boldsymbol{X}, \boldsymbol{y}$)
    $(\mu_0, k_0) \leftarrow (\mu, k)$
    $\boldsymbol{\mu} \leftarrow \mu(\boldsymbol{X})$
    $\hat{K} \leftarrow k(\boldsymbol{X}, \boldsymbol{X}) + \sigma^2 I$
    while not StoppingCriterion() do
        $s_i \leftarrow$ Policy()                          // Select action via policy.
        $r_{i-1} \leftarrow (\boldsymbol{y} - \boldsymbol{\mu}) - \hat{K} v_{i-1}$              // Predictive residual.
        $\alpha_i \leftarrow s_i^\intercal r_{i-1}$                     // Observation of linear solver.
        $d_i \leftarrow \Sigma_{i-1} \hat{K} s_i = (I - C_{i-1} \hat{K}) s_i$          // Search direction.
        $\eta_i \leftarrow s_i^\intercal \hat{K} \Sigma_{i-1} \hat{K} s_i = s_i^\intercal \hat{K} d_i$       // Normalization constant.
        $C_i \leftarrow C_{i-1} + \frac{1}{\eta_i} d_i d_i^\intercal$   // Inverse approx. $C_i \approx \hat{K}^{-1}$.
        $v_i \leftarrow v_{i-1} + \frac{\alpha_i}{\eta_i} d_i$               // Representer weights estimate.
        $\Sigma_i \leftarrow \Sigma_0 - C_i$                // Computational rep. w. uncertainty.
    end while
    $p(v_*) \leftarrow \mathcal{N}(v_*; v_i, \Sigma_i)$  // Belief about representer weights.
    $\mu_i(\cdot) \leftarrow \mu(\cdot) + k(\cdot, \boldsymbol{X}) v_i$          // Approximate posterior mean.
    $k_i(\cdot,\cdot) \leftarrow k(\cdot,\cdot) - k(\cdot, \boldsymbol{X}) C_i k(\boldsymbol{X}, \cdot)$  // Combined uncertainty.
    return $\mathcal{GP}(\mu_i, k_i)$
end procedure
```

Initialize representer weights belief

$$v_0 = 0$$

$$C_0 = 0$$

$$\Sigma_0 = \Sigma_0 - C_0 = \hat{K}^{-1}$$



Algorithm: IterGP

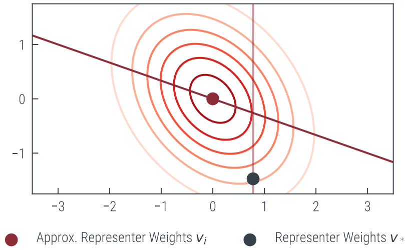
A class of computation-aware iterative methods for GP approximation.

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

```
procedure IterGP($\mu, k, \boldsymbol{X}, \boldsymbol{y}$)
    $(\mu_0, k_0) \leftarrow (\mu, k)$
    $\boldsymbol{\mu} \leftarrow \mu(\boldsymbol{X})$
    $\hat{K} \leftarrow k(\boldsymbol{X}, \boldsymbol{X}) + \sigma^2 I$
    while not StoppingCriterion() do
        $s_i \leftarrow$ Policy()                                          // Select action via policy.
        $r_{i-1} \leftarrow (\boldsymbol{y} - \boldsymbol{\mu}) - \hat{K} v_{i-1}$                       // Predictive residual.
        $\alpha_i \leftarrow s_i^\top r_{i-1}$                               // Observation of linear solver.
        $d_i \leftarrow \Sigma_{i-1} \hat{K} s_i = (I - C_{i-1} \hat{K}) s_i$           // Search direction.
        $\eta_i \leftarrow s_i^\top \hat{K} \Sigma_{i-1} \hat{K} s_i = s_i^\top \hat{K} d_i$        // Normalization constant.
        $C_i \leftarrow C_{i-1} + \frac{1}{\eta_i} d_i d_i^\top$   // Inverse approx. $C_i \approx \hat{K}^{-1}$.
        $v_i \leftarrow v_{i-1} + \frac{\alpha_i}{\eta_i} d_i$                    // Representer weights estimate.
        $\Sigma_i \leftarrow \Sigma_0 - C_i$                  // Computational rep. w. uncertainty.
    end while
    $p(v_*) \leftarrow \mathcal{N}(v_*; v_i, \Sigma_i)$ // Belief about representer weights.
    $\mu_i(\cdot) \leftarrow \mu(\cdot) + k(\cdot, \boldsymbol{X}) v_i$          // Approximate posterior mean.
    $k_i(\cdot,\cdot) \leftarrow k(\cdot,\cdot) - k(\cdot, \boldsymbol{X}) C_i k(\boldsymbol{X}, \cdot)$  // Combined uncertainty.
    return $\mathcal{GP}(\mu_i, k_i)$
end procedure
```

Select action via policy

$$s_i = \textsc{Policy}()$$



Algorithm: IterGP

A class of computation-aware iterative methods for GP approximation.

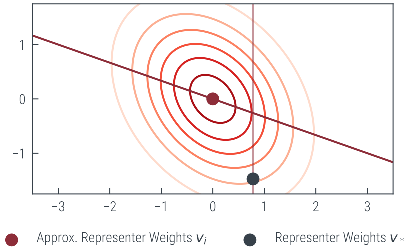
```
Input: prior mean $\mu$, prior kernel $k$, training data $X, y$
Output: (combined) GP posterior $\mathcal{GP}(\mu_i, k_i)$
procedure ITERGP($\mu, k, X, y$)
    $(\mu_0, k_0) \leftarrow (\mu, k)$
    $\boldsymbol{\mu} \leftarrow \mu(X)$
    $\hat{K} \leftarrow k(X, X) + \sigma^2 I$
    while not STOPPINGCRITERION() do
        $s_i \leftarrow$ POLICY()                                   // Select action via policy.
        $r_{i-1} \leftarrow (y - \boldsymbol{\mu}) - \hat{K} v_{i-1}$          // Predictive residual.
        $\alpha_i \leftarrow s_i^\top r_{i-1}$                        // Observation of linear solver.
        $d_i \leftarrow \boldsymbol{\Sigma}_{i-1} \hat{K} s_i = (I - C_{i-1} \hat{K}) s_i$      // Search direction.
        $\eta_i \leftarrow s_i^\top \hat{K} \boldsymbol{\Sigma}_{i-1} \hat{K} s_i = s_i^\top \hat{K} d_i$     // Normalization constant.
        $C_i \leftarrow C_{i-1} + \frac{1}{\eta_i} d_i d_i^\top$   // Inverse approx. $C_i \approx \hat{K}^{-1}$.
        $v_i \leftarrow v_{i-1} + \frac{\alpha_i}{\eta_i} d_i$            // Representer weights estimate.
        $\boldsymbol{\Sigma}_i \leftarrow \boldsymbol{\Sigma}_0 - C_i$            // Computational rep. w. uncertainty.
    end while
    $p(v_*) \leftarrow \mathcal{N}(v_*; v_i, \boldsymbol{\Sigma}_i)$ // Belief about representer weights.
    $\mu_i(\cdot) \leftarrow \mu(\cdot) + k(\cdot, X) v_i$          // Approximate posterior mean.
    $k_i(\cdot, \cdot) \leftarrow k(\cdot, \cdot) - k(\cdot, X) C_i k(X, \cdot)$  // Combined uncertainty.
    return $\mathcal{GP}(\mu_i, k_i)$
end procedure
```

Observe projected residual

$$\alpha_i = s_i^\top r_{i-1} = (\hat{K} s_i)^\top (v_* - v_{i-1})$$

Compute search direction

$$d_i = (I - C_{i-1} \hat{K}) s_i$$



Algorithm: IterGP

A class of computation-aware iterative methods for GP approximation.

Input: prior mean μ, prior kernel k, training data X, y
Output: (combined) GP posterior $\mathcal{GP}(\mu_i, k_i)$

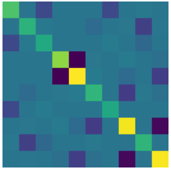
```
procedure IterGP($\mu, k, X, y$)
    $(\mu_0, k_0) \leftarrow (\mu, k)$
    $\boldsymbol{\mu} \leftarrow \mu(X)$
    $\hat{K} \leftarrow k(X, X) + \sigma^2 I$
    while not StoppingCriterion() do
        $s_i \leftarrow$ Policy()                                   // Select action via policy.
        $r_{i-1} \leftarrow (y - \boldsymbol{\mu}) - \hat{K} v_{i-1}$                 // Predictive residual.
        $\alpha_i \leftarrow s_i^\top r_{i-1}$                        // Observation of linear solver.
        $d_i \leftarrow \boldsymbol{\Sigma}_{i-1} \hat{K} s_i = (I - C_{i-1} \hat{K}) s_i$          // Search direction.
        $\eta_i \leftarrow s_i^\top \hat{K} \boldsymbol{\Sigma}_{i-1} \hat{K} s_i = s_i^\top \hat{K} d_i$      // Normalization constant.
        $C_i \leftarrow C_{i-1} + \frac{1}{\eta_i} d_i d_i^\top$  // Inverse approx. $C_i \approx \hat{K}^{-1}$.
        $v_i \leftarrow v_{i-1} + \frac{\alpha_i}{\eta_i} d_i$               // Representer weights estimate.
        $\boldsymbol{\Sigma}_i \leftarrow \boldsymbol{\Sigma}_0 - C_i$           // Computational rep. w. uncertainty.
    end while
    $p(v_*) \leftarrow \mathcal{N}(v_*; v_i, \boldsymbol{\Sigma}_i)$ // Belief about representer weights.
    $\mu_i(\cdot) \leftarrow \mu(\cdot) + k(\cdot, X) v_i$           // Approximate posterior mean.
    $k_i(\cdot, \cdot) \leftarrow k(\cdot, \cdot) - k(\cdot, X) C_i k(X, \cdot)$  // Combined uncertainty.
    return $\mathcal{GP}(\mu_i, k_i)$
end procedure
```

Update precision matrix approximation

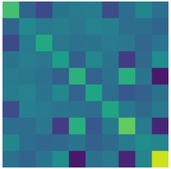
$$C_i = C_{i-1} + \frac{1}{\eta_i} d_i d_i^\top$$

$$= \begin{pmatrix} | & & | \\ d_1 & \cdots & d_i \\ | & & | \end{pmatrix} \begin{pmatrix} \frac{1}{\eta_1} & & \\ & \ddots & \\ & & \frac{1}{\eta_i} \end{pmatrix} \begin{pmatrix} - d_1^\top - \\ \vdots \\ - d_i^\top - \end{pmatrix}$$

Precision Matrix



Precision Matrix Approx.



Algorithm: IterGP

A class of computation-aware iterative methods for GP approximation.

```
Input: prior mean $\mu$, prior kernel $k$, training data $X, y$
Output: (combined) GP posterior $\mathcal{GP}(\mu_i, k_i)$
procedure IterGP($\mu, k, X, y$)
    $(\mu_0, k_0) \leftarrow (\mu, k)$
    $\boldsymbol{\mu} \leftarrow \mu(X)$
    $\hat{K} \leftarrow k(X, X) + \sigma^2 I$
    while not StoppingCriterion() do
        $s_i \leftarrow$ Policy()                                        // Select action via policy.
        $r_{i-1} \leftarrow (y - \boldsymbol{\mu}) - \hat{K} v_{i-1}$                      // Predictive residual.
        $\alpha_i \leftarrow s_i^\top r_{i-1}$                             // Observation of linear solver.
        $d_i \leftarrow \boldsymbol{\Sigma}_{i-1} \hat{K} s_i = (I - C_{i-1} \hat{K}) s_i$             // Search direction.
        $\eta_i \leftarrow s_i^\top \hat{K} \boldsymbol{\Sigma}_{i-1} \hat{K} s_i = s_i^\top \hat{K} d_i$      // Normalization constant.
        $C_i \leftarrow C_{i-1} + \frac{1}{\eta_i} d_i d_i^\top$  // Inverse approx. $C_i \approx \hat{K}^{-1}$.
        $v_i \leftarrow v_{i-1} + \frac{\alpha_i}{\eta_i} d_i$                   // Representer weights estimate.
        $\boldsymbol{\Sigma}_i \leftarrow \boldsymbol{\Sigma}_0 - C_i$                  // Computational rep. w. uncertainty.
    end while
    $p(v_*) \leftarrow \mathcal{N}(v_*; v_i, \boldsymbol{\Sigma}_i)$ // Belief about representer weights.
    $\mu_i(\cdot) \leftarrow \mu(\cdot) + k(\cdot, X) v_i$          // Approximate posterior mean.
    $k_i(\cdot, \cdot) \leftarrow k(\cdot, \cdot) - k(\cdot, X) C_i k(X, \cdot)$  // Combined uncertainty.
    return $\mathcal{GP}(\mu_i, k_i)$
end procedure
```

Update precision matrix approximation

$$C_i = C_{i-1} + \frac{1}{\eta_i} d_i d_i^\top$$

$$= \begin{pmatrix} | & & | \\ d_1 & \cdots & d_i \\ | & & | \end{pmatrix} \begin{pmatrix} \frac{1}{\eta_1} & & \\ & \ddots & \\ & & \frac{1}{\eta_i} \end{pmatrix} \begin{pmatrix} - d_1^\top - \\ \vdots \\ - d_i^\top - \end{pmatrix}$$

Update representer weights belief

$$v_i = C_i (y - \boldsymbol{\mu}) = v_{i-1} + \frac{\alpha_i}{\eta_i} d_i$$

$$\Sigma_i = \Sigma_i - C_i$$

What about the partial Cholesky and CG?

Connection to Other GP Approximations

Connection to Other GP Approximation Methods

IterGP extends the most commonly used GP approximations to include computational uncertainty, with at most quadratic cost.

Method	Actions s_i	Classic Analog
IterGP-Cholesky	e_i	(partial) Cholesky / subset of data
IterGP-EVD	$\mathrm{ev}_i(\hat{K})$	(partial) eigenvalue decomp.
IterGP-CG	s_i^{PCG} or $\hat{P}^{-1} r_i$	(preconditioned) CG
IterGP-PseudoInput	$k(X, z_i)$	$\approx$ SVGP

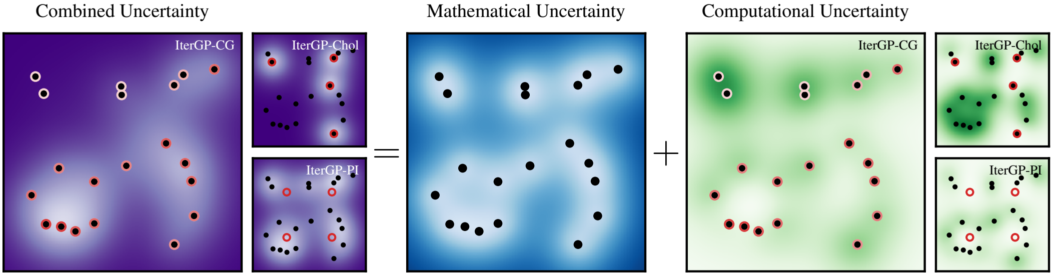
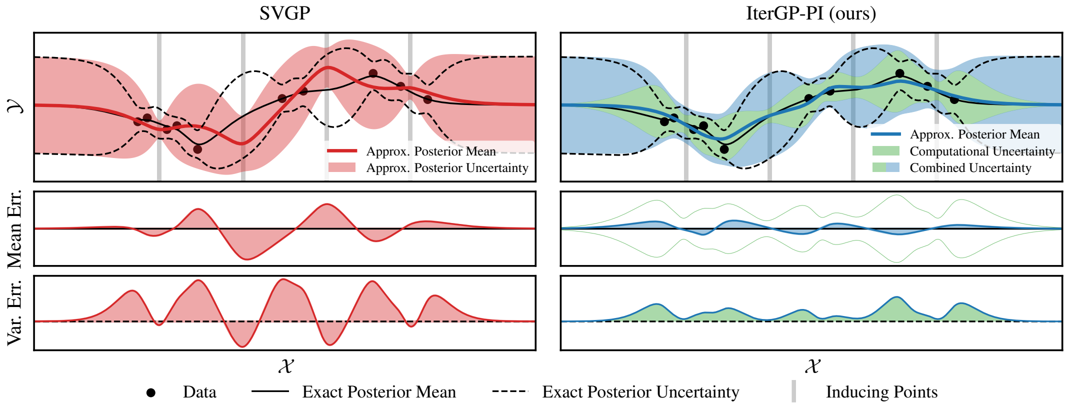


Figure: Computational uncertainty is small where there either is no data (●) or computation was “targeted” (○).

Connection to SVGP

One can construct a similar approximation to SVGP with proper uncertainty quantification.



IterGP-PseudoInput has complexity $\mathcal{O}(n^2 i)$. Are we restricted to quadratic time?

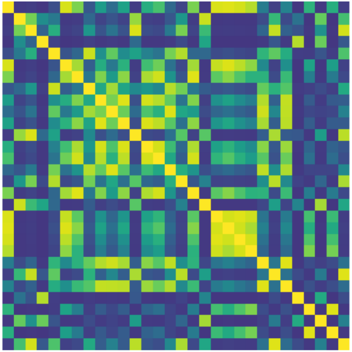
The Cost of Combined Uncertainty Quantification

Taking a second look at the computational complexity of IterGP.

Question: How costly is one iteration of IterGP for a specific policy?

Kernel matrix $\hat{K}$ appears in three ways:

- Observation: $\alpha_i \leftarrow \boldsymbol{s}_i^\intercal ((\boldsymbol{y} - \boldsymbol{\mu}) - \hat{K}\boldsymbol{v}_{i-1})$
- Search direction: $\boldsymbol{d}_i \leftarrow (\boldsymbol{I} - \boldsymbol{C}_{i-1}\hat{K})\boldsymbol{s}_i$
- Normalization const.: $\eta_i \leftarrow \boldsymbol{s}_i^\intercal \hat{K}\boldsymbol{d}_i$



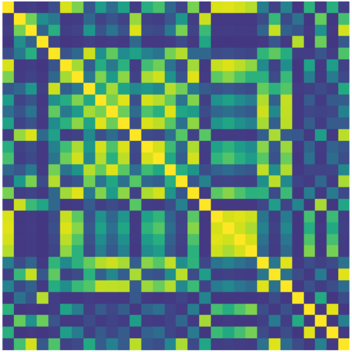
The Cost of Combined Uncertainty Quantification

Taking a second look at the computational complexity of IterGP.

Question: How costly is one iteration of IterGP for a specific policy?

Kernel matrix $\hat{K}$ appears in three ways:

- Observation: $\alpha_i \leftarrow s_i^\mathsf{T}((y - \mu) - \hat{K}v_{i-1}) \implies s_i^\mathsf{T}\hat{K}d_j$
- Search direction: $d_i \leftarrow (I - C_{i-1}\hat{K})s_i$
- Normalization const.: $\eta_i \leftarrow s_i^\mathsf{T}\hat{K}d_i$



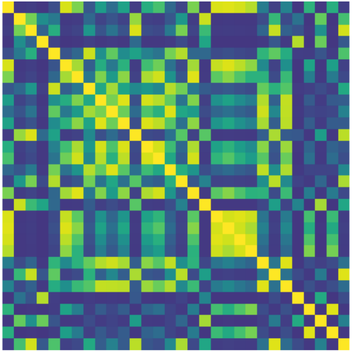
The Cost of Combined Uncertainty Quantification

Taking a second look at the computational complexity of IterGP.

Question: How costly is one iteration of IterGP for a specific policy?

Kernel matrix $\hat{K}$ appears in three ways:

- Observation: $\alpha_i \leftarrow \boldsymbol{s}_i^\top((\boldsymbol{y} - \boldsymbol{\mu}) - \hat{K}\boldsymbol{v}_{i-1}) \implies \boldsymbol{s}_i^\top \hat{K}\boldsymbol{d}_i$
- Search direction:
  $\boldsymbol{d}_i \leftarrow (I - C_{i-1}\hat{K})\boldsymbol{s}_i = \boldsymbol{s}_i - \sum_{j=1}^{i} \frac{1}{\eta_j}\boldsymbol{d}_j\boldsymbol{d}_j^\top \hat{K}\boldsymbol{s}_i \implies \boldsymbol{s}_i^\top \hat{K}\boldsymbol{d}_i$
- Normalization const.: $\eta_i \leftarrow \boldsymbol{s}_i^\top \hat{K}\boldsymbol{d}_i$



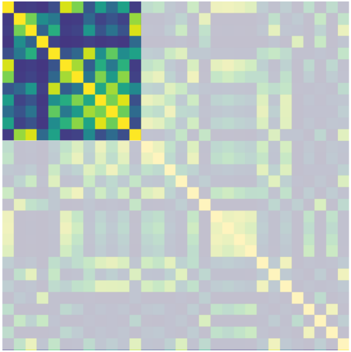
The Cost of Combined Uncertainty Quantification

Taking a second look at the computational complexity of IterGP.

Question: How costly is one iteration of IterGP for a specific policy?

Kernel matrix $\hat{K}$ appears in three ways:

- Observation: $\alpha_i \leftarrow s_i^\top((y-\mu)-\hat{K}v_{i-1}) \implies s_i^\top \hat{K} d_j$
- Search direction:
  $d_i \leftarrow (I - C_{i-1}\hat{K})s_i = s_i - \sum_{j=1}^{i} \frac{1}{\eta_j} d_j d_j^\top \hat{K} s_i \implies s_i^\top \hat{K} d_j$
- Normalization const.: $\eta_i \leftarrow s_i^\top \hat{K} d_i$



Idea: Choose i actions with at most $\ell \ll n$ non-zero entries. $\implies \mathcal{O}(\ell^2)$ per iteration!

We only operate on the data that we target with computation → arbitrary computation cost!

Working with Infinite Data

For IterGP it does not matter how large the dataset is, or whether we have it stored on our machine.

Theorem (Online GP Approximation with IterGP)

Let $n, n' \in \mathbb{N}$ and consider training data sets $\mathbf{X} \in \mathbb{R}^{n \times d}, \mathbf{y} \in \mathbb{R}^n$ and $\mathbf{X}' \in \mathbb{R}^{n' \times d}, \mathbf{y}' \in \mathbb{R}^{n'}$. Consider two sequences of actions $(\mathbf{s}_i)_{i=1}^n \in \mathbb{R}^n$ and $(\tilde{\mathbf{s}}_i)_{i=1}^{n+n'} \in \mathbb{R}^{n+n'}$ such that for all $i \in \{1, \dots, n\}$, it holds that

$$\tilde{\mathbf{s}}_i = \begin{pmatrix} \mathbf{s}_i \\ 0 \end{pmatrix} \tag{1}$$

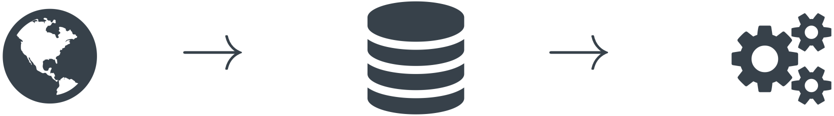
Then the posterior returned by IterGP for the dataset $(\mathbf{X}, \mathbf{y})$ using actions $\mathbf{s}_i$ is identical to the posterior returned by IterGP for the extended dataset using actions $\tilde{\mathbf{s}}_i$, i.e. it holds for any $i \in \{1, \dots, n\}$, that

$$\textit{I}\textsc{ter}\textit{GP}(\mu, k, \mathbf{X}, \mathbf{y}, (\mathbf{s}_i)_i) = (\mu_i, k_i) = (\tilde{\mu}_i, \tilde{k}_i) = \textit{I}\textsc{ter}\textit{GP}\left(\mu, k, \begin{pmatrix} X \\ X' \end{pmatrix}, \begin{pmatrix} y \\ y' \end{pmatrix}, (\tilde{\mathbf{s}}_i)_i\right).$$

An Approximation or a Better Model?

An alternative view of IterGP as a better model for the way we do inference with a computer.

Observation: Only once we perform computation on data, does it enter our prediction.

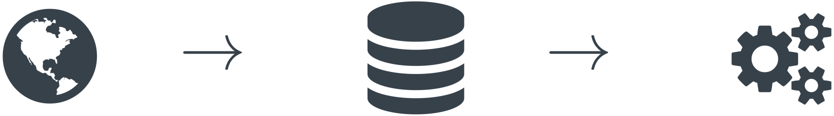


- The distinction between data and computation vanishes from this perspective.

An Approximation or a Better Model?

An alternative view of IterGP as a better model for the way we do inference with a computer.

Observation: Only once we perform computation on data, does it enter our prediction.



- The distinction between data and computation vanishes from this perspective.

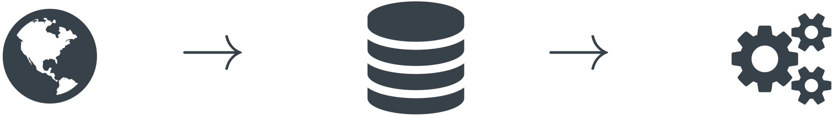
What if we modelled this situation with a Gaussian process?

$$f \sim \mathcal{GP}(\mu, k)$$

An Approximation or a Better Model?

An alternative view of IterGP as a better model for the way we do inference with a computer.

Observation: Only once we perform computation on data, does it enter our prediction.



- The distinction between data and computation vanishes from this perspective.

What if we modelled this situation with a Gaussian process?

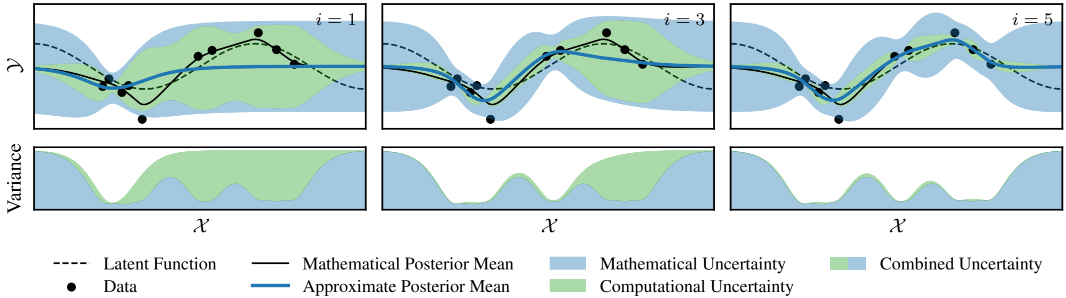
$$
\begin{aligned}
f &\sim \mathcal{GP}(\mu, k)\\
\tilde{y} \mid f(X) &\sim \mathcal{N}(S_i^\intercal f(X), \sigma^2 S_i^\intercal S_i)\\
f \mid X, \tilde{y} &\sim \mathcal{GP}(\mu_i, k_i)
\end{aligned}
$$

- IterGP's combined posterior is equivalent exact GP regression for linearly projected data.

How meaningful is computational and combined uncertainty?
Theoretical Analysis

Combined Uncertainty as Worst Case Error

The combined uncertainty is a tight worst case bound on the relative error to the latent function.



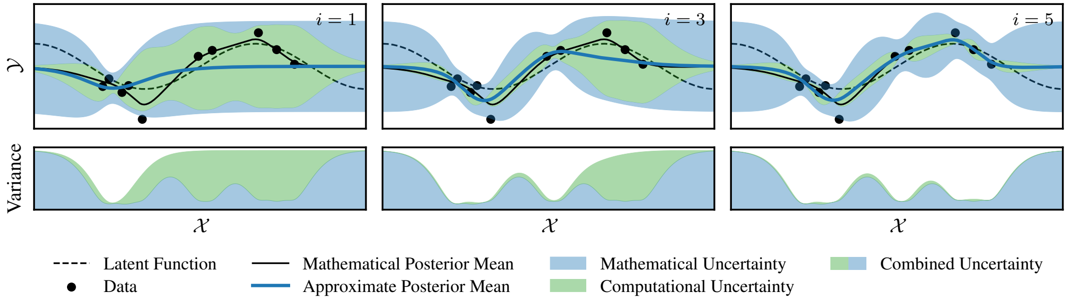
GP: $$\frac{|\text{Latent Function}(x) - \text{Math. Posterior Mean}(x)|}{\|\text{Latent Function}\|} \leq \text{Posterior Pred. Std. Deviation}(x)$$ 🔵

IterGP: $$\frac{|\text{Latent Function}(x) - \text{Approx. Posterior Mean}(x)|}{\|\text{Latent Function}\|} \leq \text{Combined Std. Deviation}(x)$$ 🔵 + 🟢

Exact uncertainty quantification in quadratic / linear / constant time!

Combined Uncertainty as Worst Case Error

The combined uncertainty is a tight worst case bound on the relative error to the latent function.



Theorem

$$\sup_{g\in\mathcal{H}_{k^\sigma}:\|g\|_{\mathcal{H}_{k^\sigma}}\le 1} \overbrace{\underbrace{g(\boldsymbol{x})-\mu_*^g(\boldsymbol{x})}_{\text{error of math. post. mean ●}} + \underbrace{\mu_*^g(\boldsymbol{x})-\mu_i^g(\boldsymbol{x})}_{\text{computational error ●}}}^{\text{error of approximate posterior mean ●}} = \sqrt{k_i(\boldsymbol{x},\boldsymbol{x})+\sigma^2}, \quad \textit{and} \tag{2}$$

$$\sup_{g\in\mathcal{H}_{k^\sigma}:\|g\|_{\mathcal{H}_{k^\sigma}}\le 1} \underbrace{\mu_*^g(\boldsymbol{x})-\mu_i^g(\boldsymbol{x})}_{\text{computational error ●}} = \sqrt{k_i^{\text{comp}}(\boldsymbol{x},\boldsymbol{x})} \tag{3}$$

Summary

- ▶ Approximate GPs by learning the representer weights.
- ▶ Can quantify approximation error *probabilistically*.
- ▶ Variants of IterGP defined via the policy learn actively.
- ▶ Distinction between data and computation vanishes.
- ▶ Exact UQ in arbitrary time with strong guarantees.

Please cite this course, as

```
@techreport{NoML22,
    title = {Numerics of Machine Learning},
    author = {N. Bosch and J. Grosse
    and P. Hennig and A. Kristiadi
    and M. Pförtner and J. Schmidt
    and F. Schneider and L. Tatzel
    and J. Wenger},
    series = {Lecture Notes in Machine Learning},
    year = {2022},
    institution = {Tübingen AI Center},
}
```

Next Week

- ▶ How to simulate, i.e. *learn* the dynamics of systems that follow (partially-known) physical laws.