Physics-informed machine learning for accelerated modeling and optimization of complex systems

Paris Perdikaris
Department of Mechanical Engineering
University of Pennsylvania
email: pgp@seas.upenn.edu

Supported by:

SLAC AI Seminar
October 2, 2020

Penn
University of Pennsylvania

U.S. DEPARTMENT OF ENERGY
INCITE
LEADERSHIP COMPUTING

BROWN

Argonne
NATIONAL LABORATORY

JÜLICH
FORSCHUNGSZENTRUM

*Simulation credit: Leopold Grinberg (IBM)

$\mathcal{O}(10^6)$ lines of code, ~20 PhD theses (1990-2010)

32k supercomputer nodes (IBM BG/P, 128k processors)

Finalist for the Gordon-Bell prize in supercomputing (2011, only US entry)

Simulation of one cardiac cycle took ~24hr

4 MW at $0.10/kWh is $400 an hour or about $3.5 million per year.

*Simulation credit: Leopold Grinberg (IBM)

Impressive computation, but still an approximation to the true underlying physics

Validity of results relies on accurate model calibration

Many sources of uncertainty: geometry, IC/BCs, rheology, material properties, etc.

$\mathcal{O}(10 - 10^2)$ parameters to be calibrated using clinical data

U.S. DEPARTMENT OF ENERGY
INCITE
LEADERSHIP COMPUTING

BROWN

Argonne
NATIONAL LABORATORY

JÜLICH
FORSCHUNGSZENTRUM

*Simulation credit: Leopold Grinberg (IBM)

$$\theta^* = \arg\min_{\theta} \mathcal{L}(\theta), \quad \mathcal{L}(\theta) := ||m_\theta(\boldsymbol{x}) - \boldsymbol{t}||$$

θ : Model parameters
x : Model inputs
t : Target quantities of interest

- High dimensional optimization
- Expensive, black-box loss function
- No gradients available

U.S. DEPARTMENT OF ENERGY
INCITE
LEADERSHIP COMPUTING

BROWN

Argonne
NATIONAL LABORATORY

JÜLICH
FORSCHUNGSZENTRUM

*Simulation credit: Leopold Grinberg (IBM)

$$\theta^* = \arg\min_{\theta} \mathcal{L}(\theta), \quad \mathcal{L}(\theta) : ||m_\theta(\boldsymbol{x}) - \boldsymbol{t}||$$

Big data vs small data → Dimensionality vs # of observations

U.S. DEPARTMENT OF ENERGY
INCITE
LEADERSHIP COMPUTING

BROWN

Argonne
NATIONAL LABORATORY

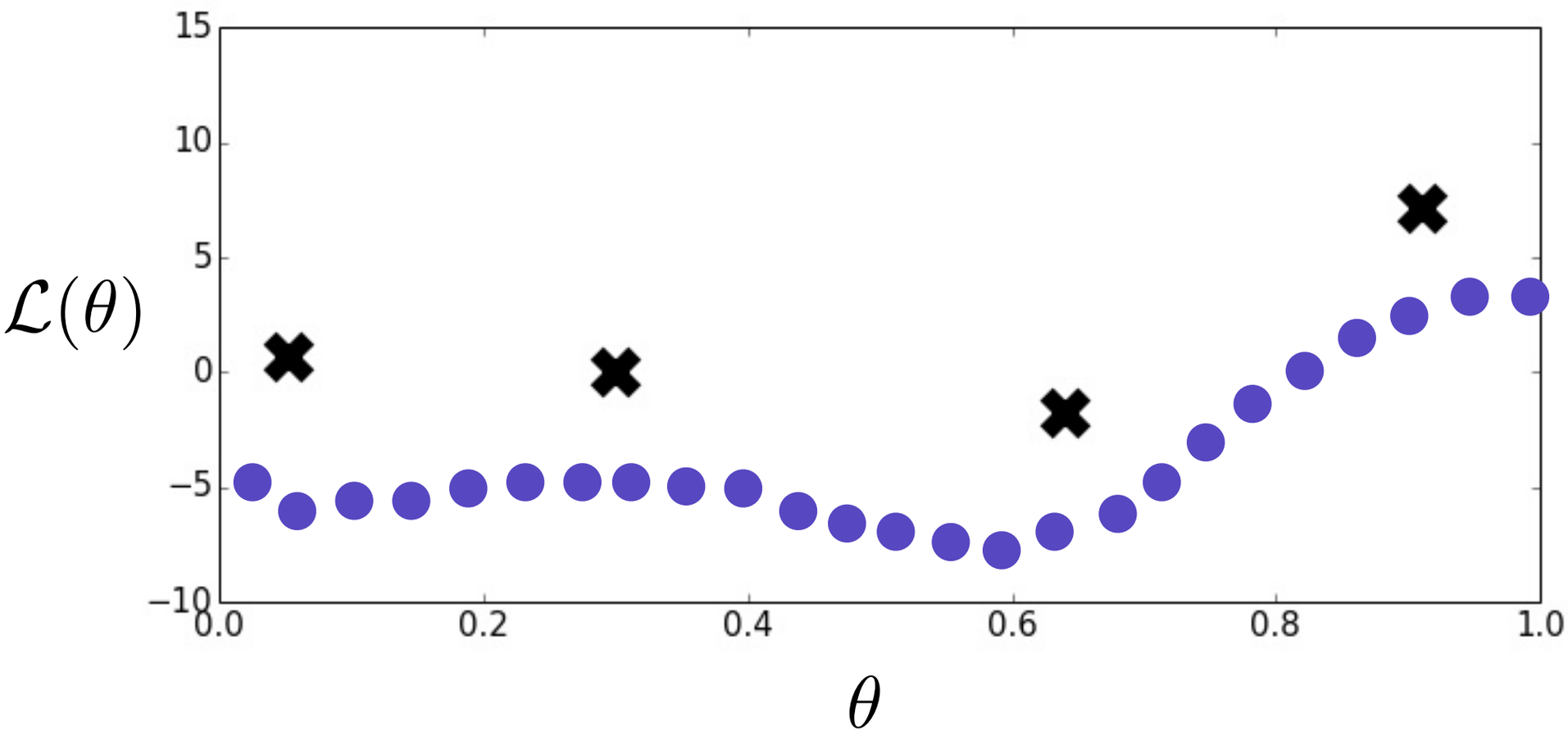
JÜLICH
FORSCHUNGSZENTRUM

*Simulation credit: Leopold Grinberg (IBM)

$$\theta^* = \arg\min_{\theta} \mathcal{L}(\theta), \quad \mathcal{L}(\theta) : ||m_\theta(\boldsymbol{x}) - \boldsymbol{t}||$$

Can we solve this problem using surrogate models that are cheaper to compute?

Optimization of expensive black-box functions



Where is the minimum of $\mathcal{L}(\theta)$?

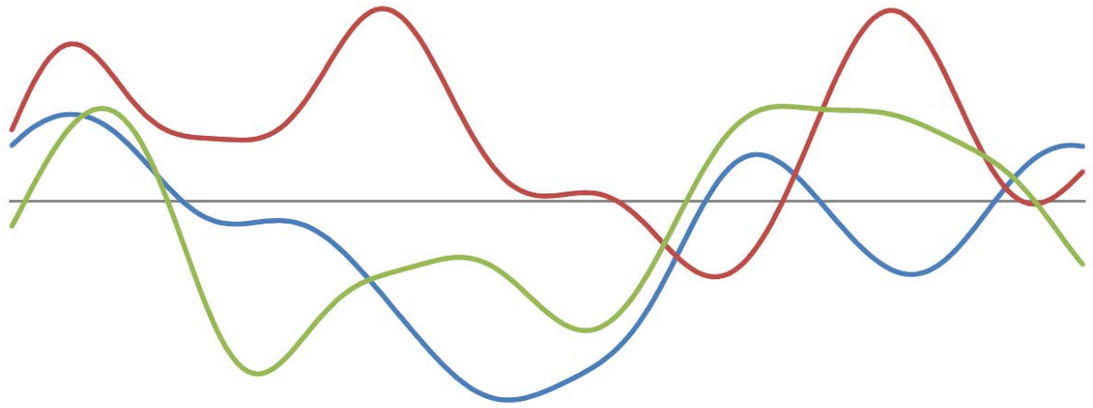
Where should we take our next evaluation?

How can we distill useful information from low-fidelity models to accelerate this process?

Data-driven modeling with Gaussian processes

Priors over functions:

$$f \sim \mathcal{GP}(\mu(x), K(\mathbf{x}, \mathbf{x}'; \theta))$$

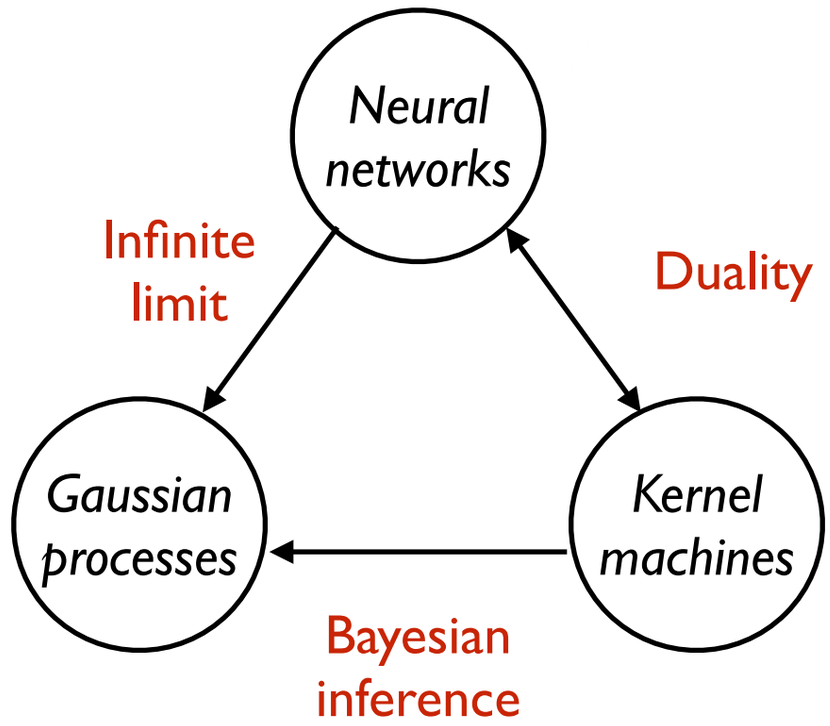


Samples from a GP prior

Marginalization:

$p(\mathbf{f}_A, \mathbf{f}_B) \sim \mathcal{N}(\boldsymbol{\mu}, \mathbf{K})$. Then:

$$p(\mathbf{f}_A) = \int_{\mathbf{f}_B} p(\mathbf{f}_A, \mathbf{f}_B) \mathrm{d}\mathbf{f}_B = \mathcal{N}(\boldsymbol{\mu}_A, \mathbf{K}_{AA})$$



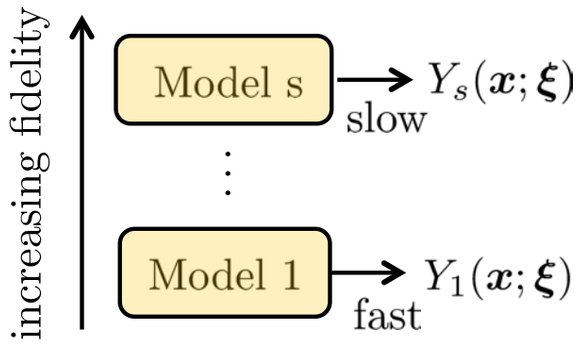
Posterior is also Gaussian:

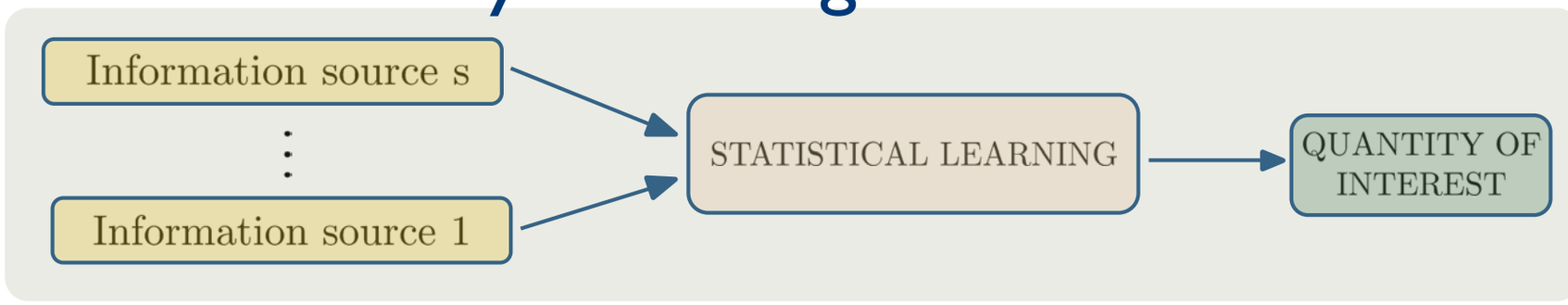
$p(\mathbf{f}_A, \mathbf{f}_B) \sim \mathcal{N}(\boldsymbol{\mu}, \mathbf{K})$. Then:

$$p(\mathbf{f}_A|\mathbf{f}_B) = \mathcal{N}(\boldsymbol{\mu}_A + \mathbf{K}_{AB}\mathbf{K}_{BB}^{-1}(\mathbf{f}_B - \boldsymbol{\mu}_B), \mathbf{K}_{AA} - \mathbf{K}_{AB}\mathbf{K}_{BB}^{-1}\mathbf{K}_{BA})$$

Rasmussen, C.E. and Williams C. Gaussian processes for machine learning (2006)

Multi-fidelity modeling





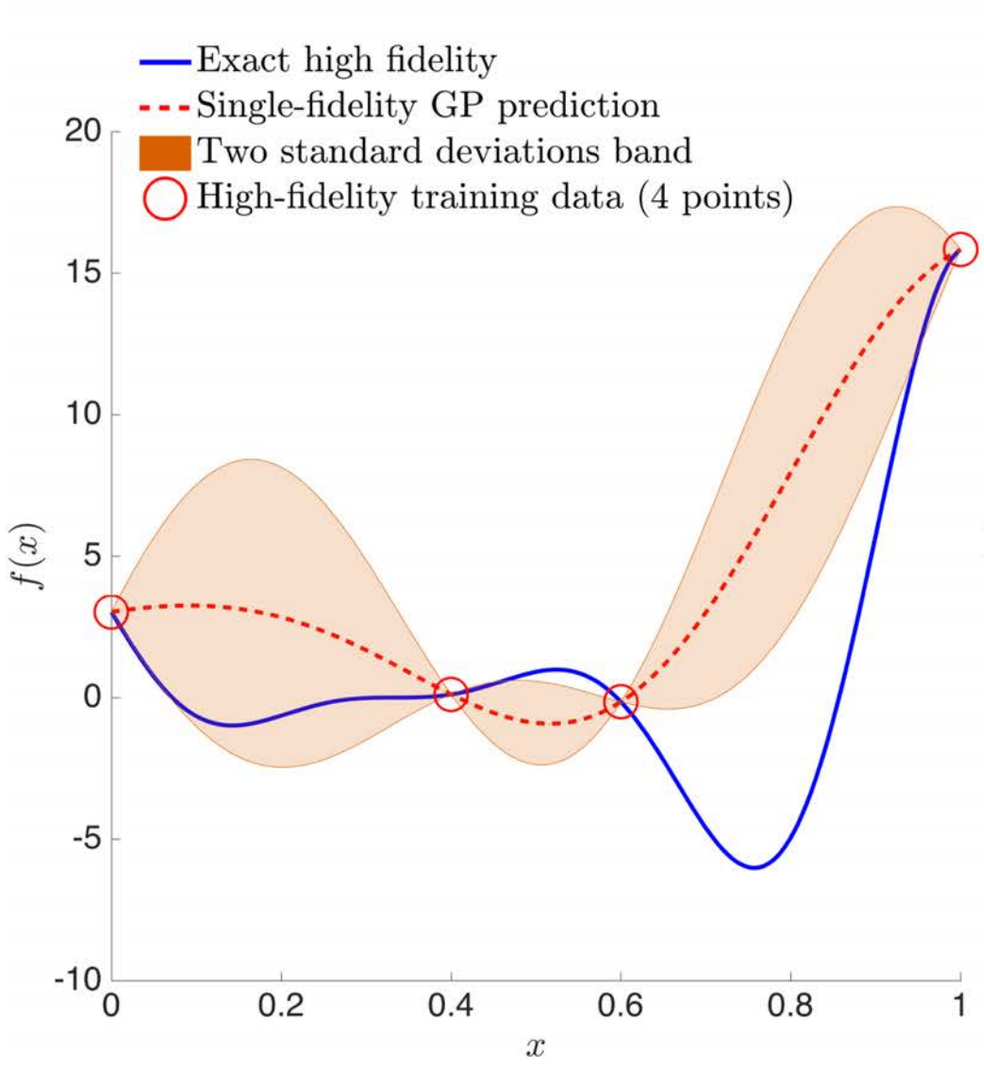
Number of runs is limited by time and computational resources

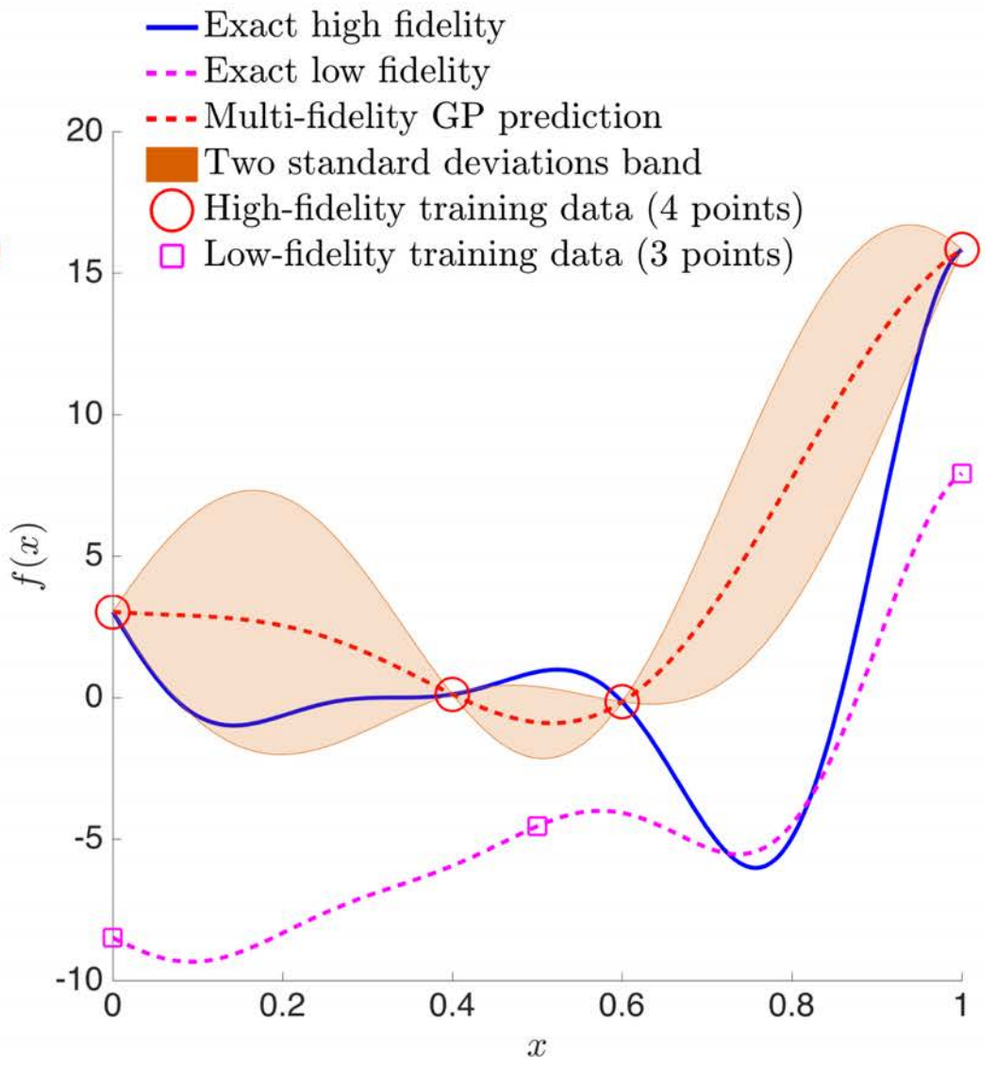
⇓

We cannot compute at all $(\boldsymbol{x};\boldsymbol{\xi})$

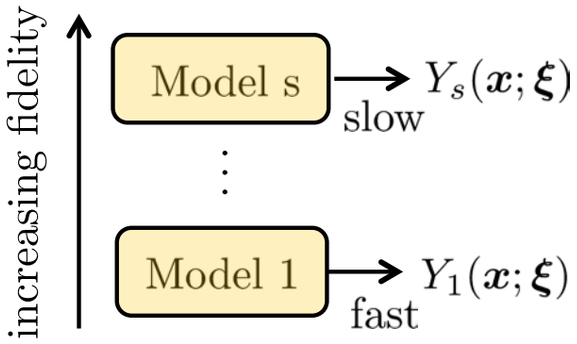
⇓

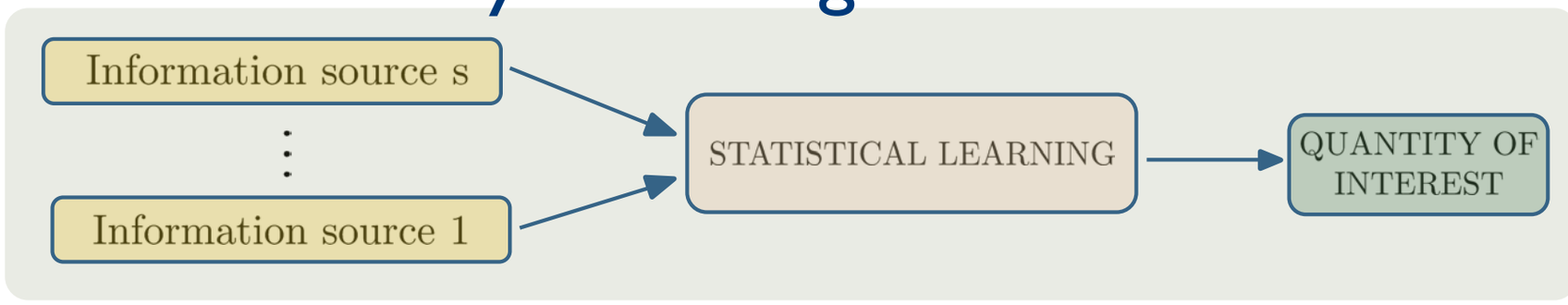
Prediction of $Z_i(\boldsymbol{x}) = \mathbb{E}[f(Y_i(\boldsymbol{x};\boldsymbol{\xi}))]$ is a problem of statistical inference





Multi-fidelity modeling





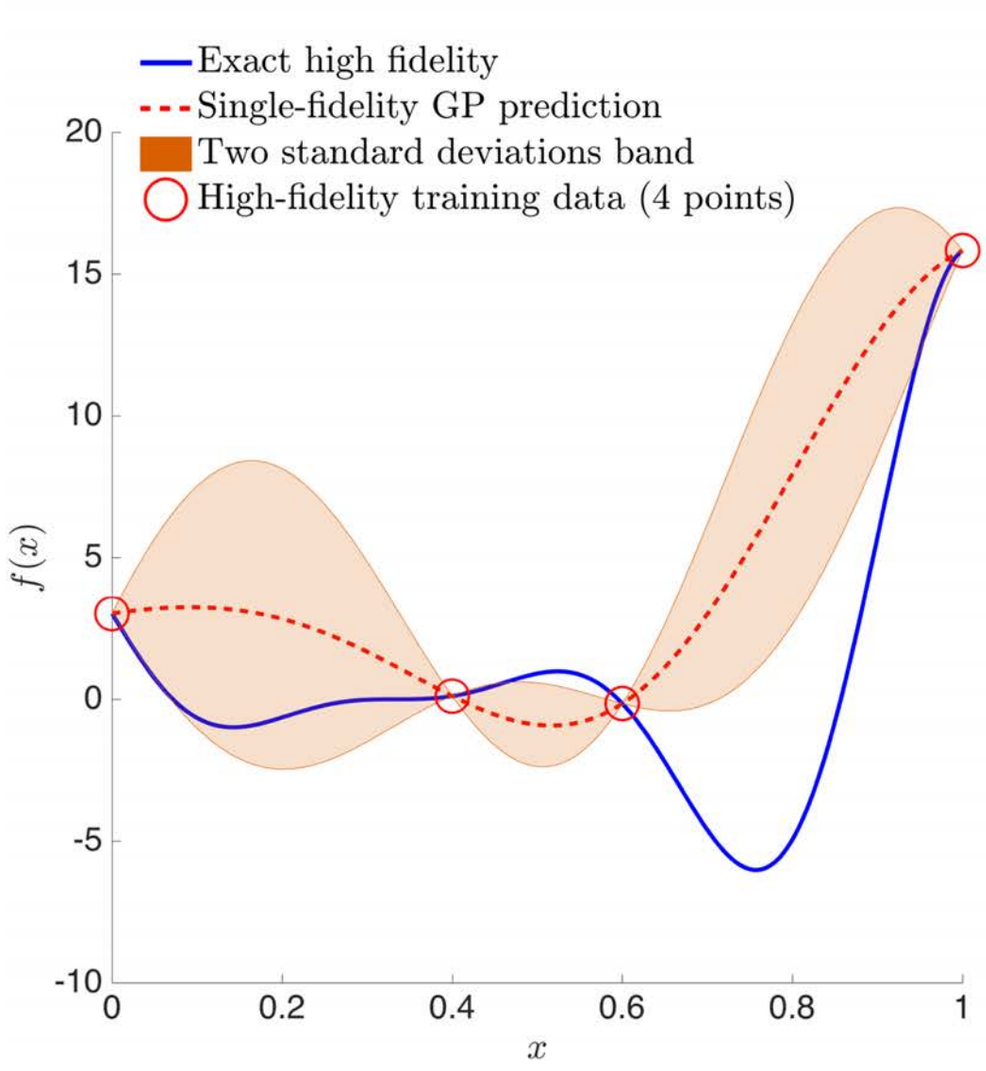
Number of runs is limited by time and computational resources

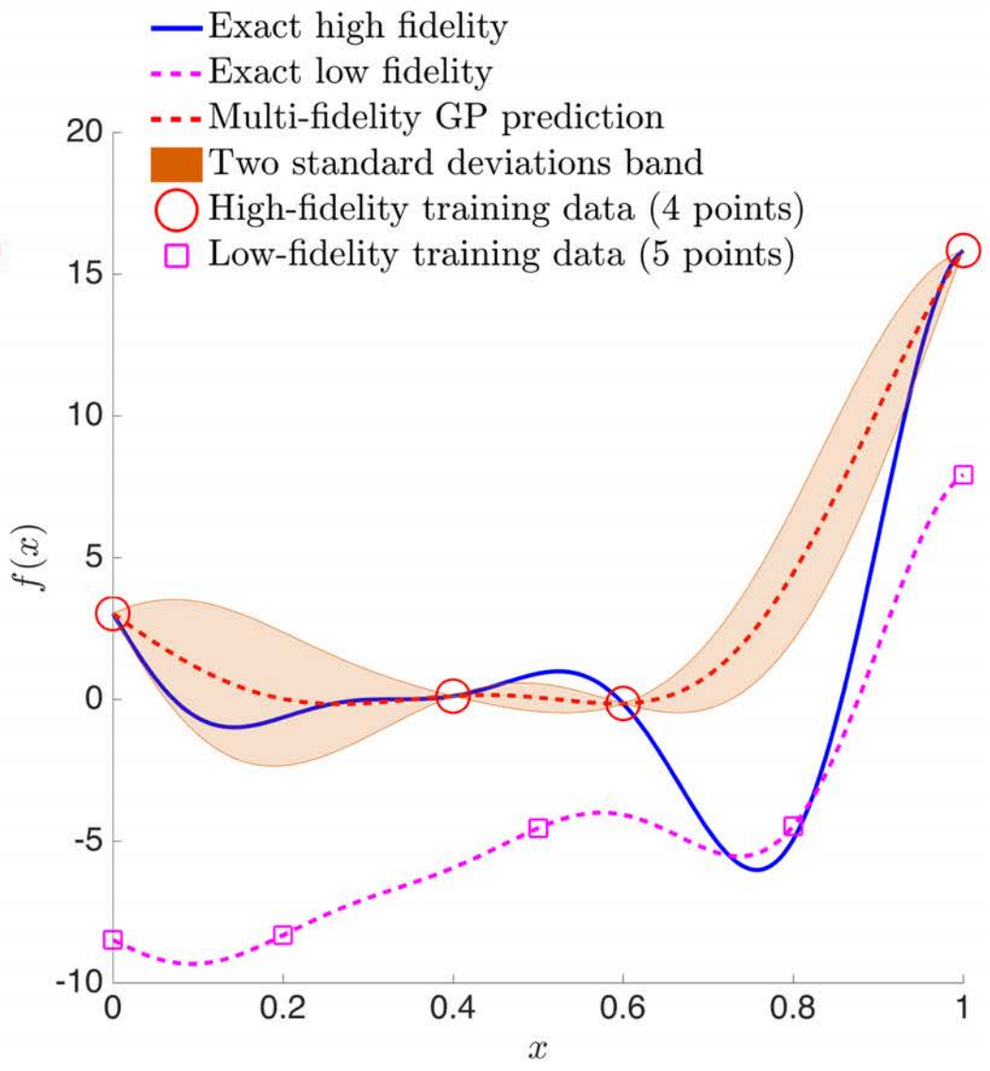
⇓

We cannot compute at all $(\boldsymbol{x};\boldsymbol{\xi})$

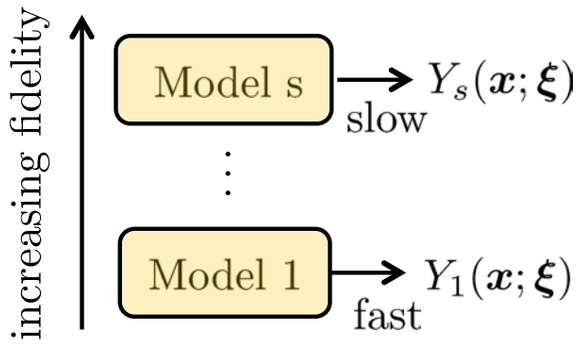
⇓

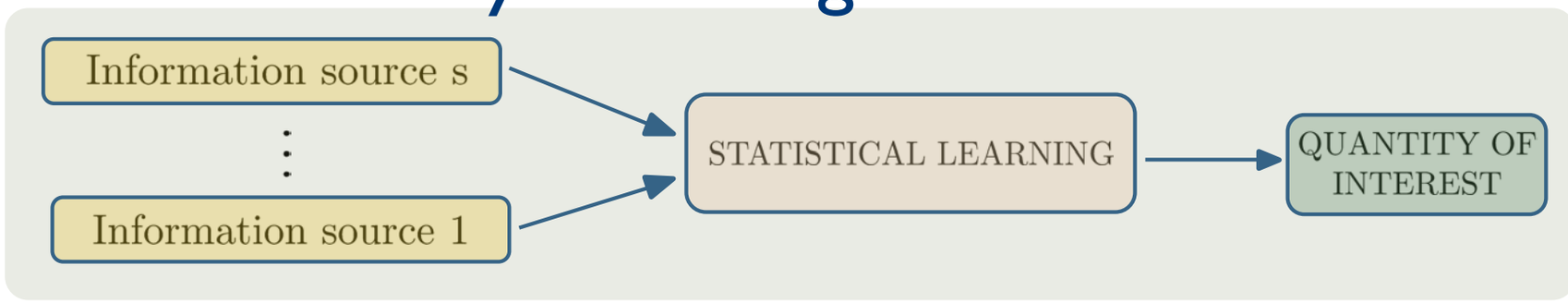
Prediction of $Z_i(\boldsymbol{x}) = \mathbb{E}[f(Y_i(\boldsymbol{x};\boldsymbol{\xi}))]$ is a problem of statistical inference





Multi-fidelity modeling





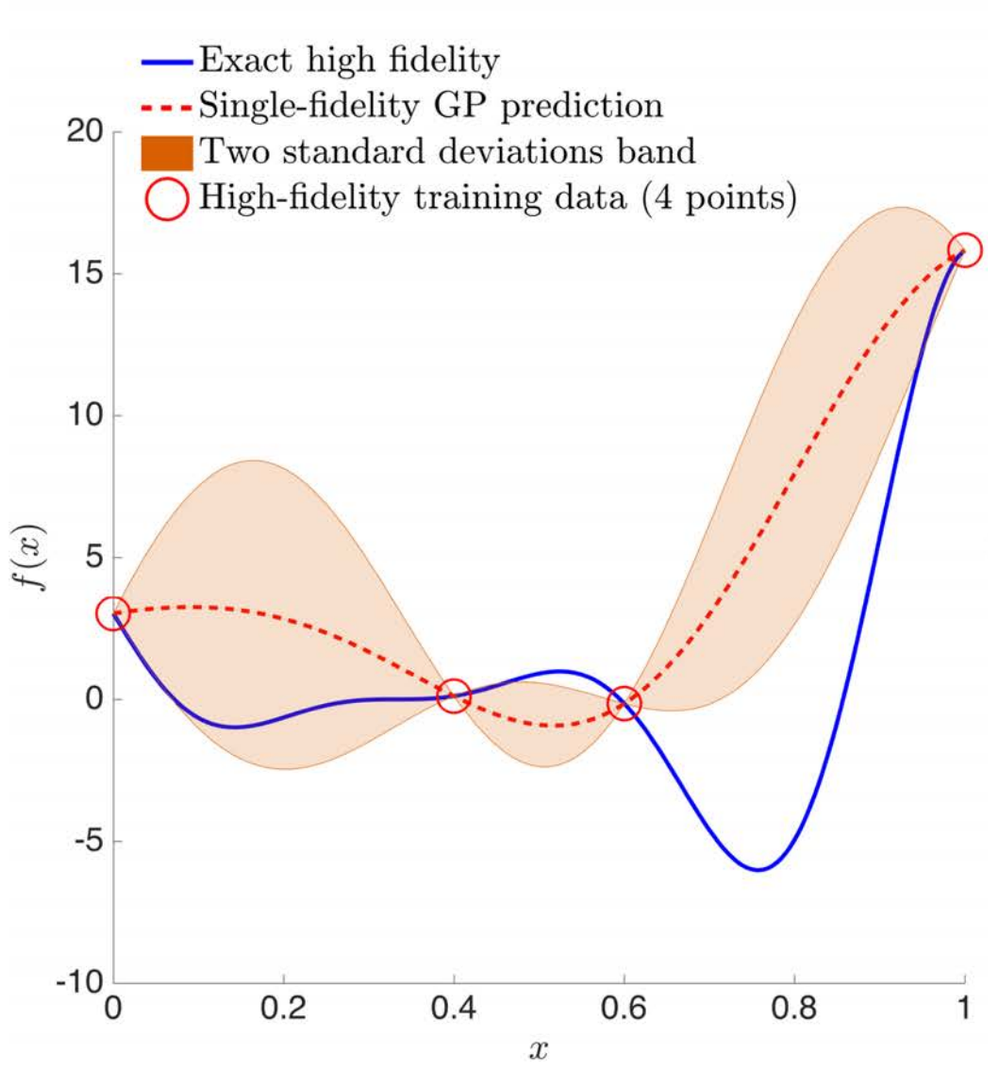
Number of runs is limited by time and computational resources

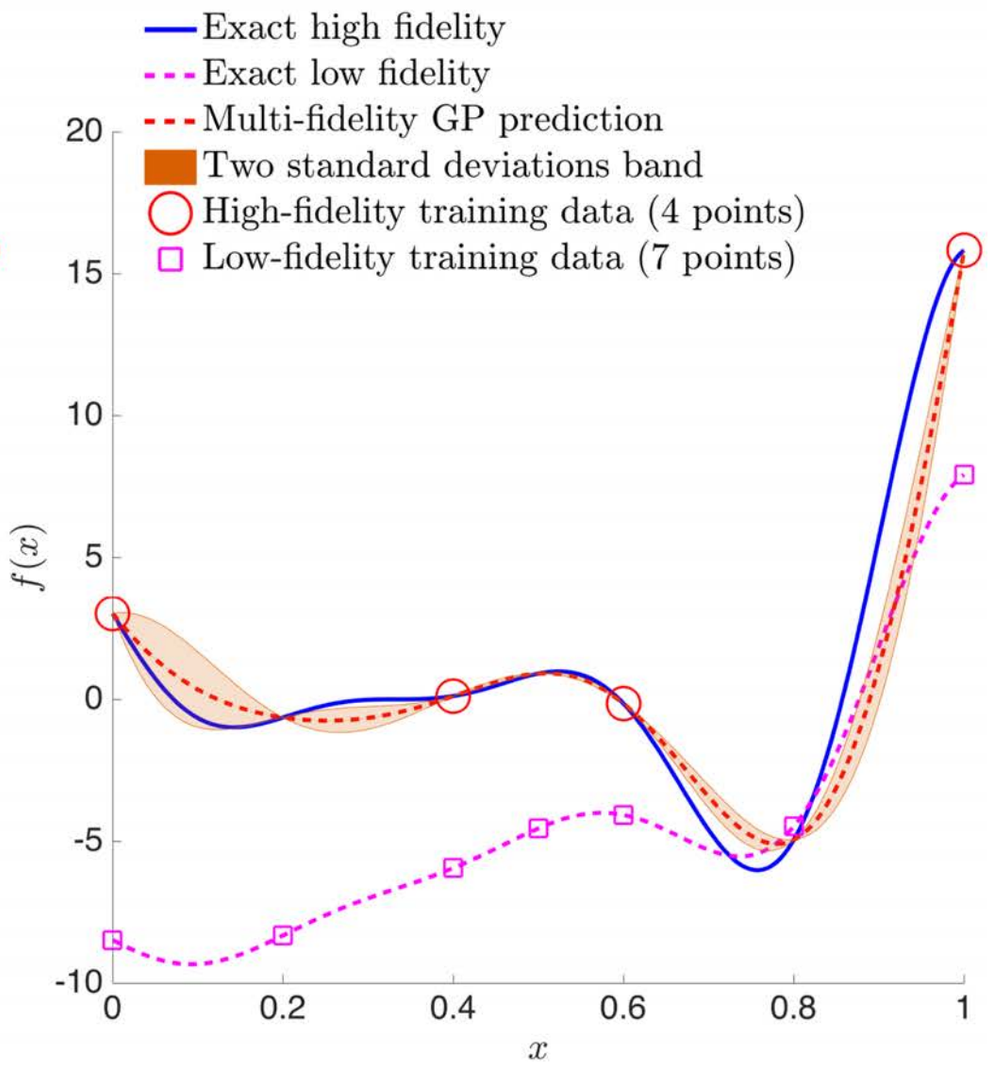
⇓

We cannot compute at all $(\boldsymbol{x};\boldsymbol{\xi})$

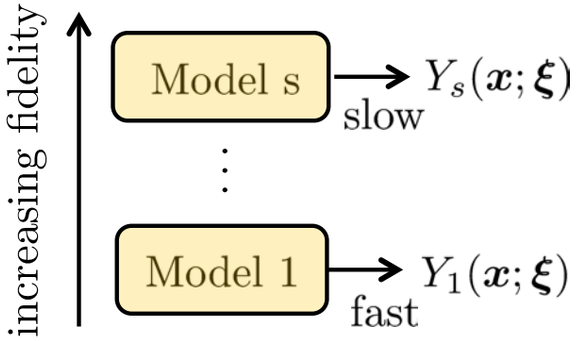
⇓

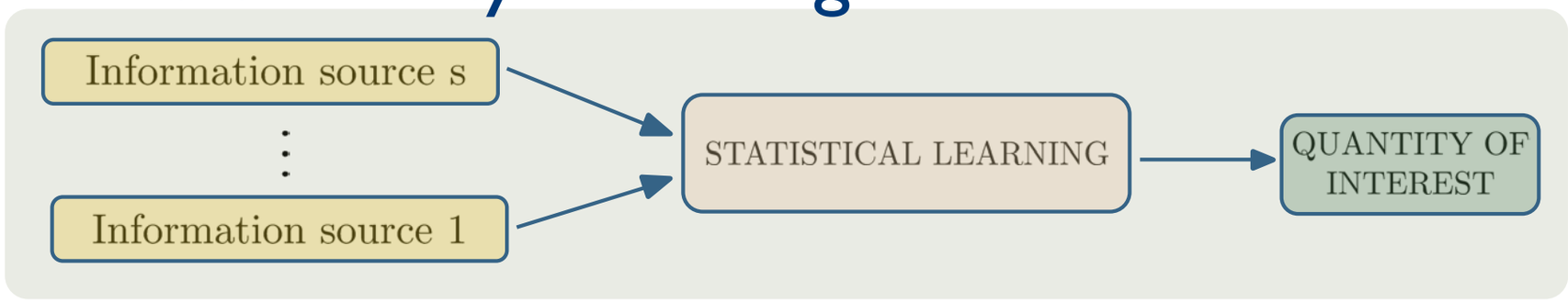
Prediction of $Z_i(\boldsymbol{x}) = \mathbb{E}[f(Y_i(\boldsymbol{x};\boldsymbol{\xi}))]$ is a problem of statistical inference





Multi-fidelity modeling





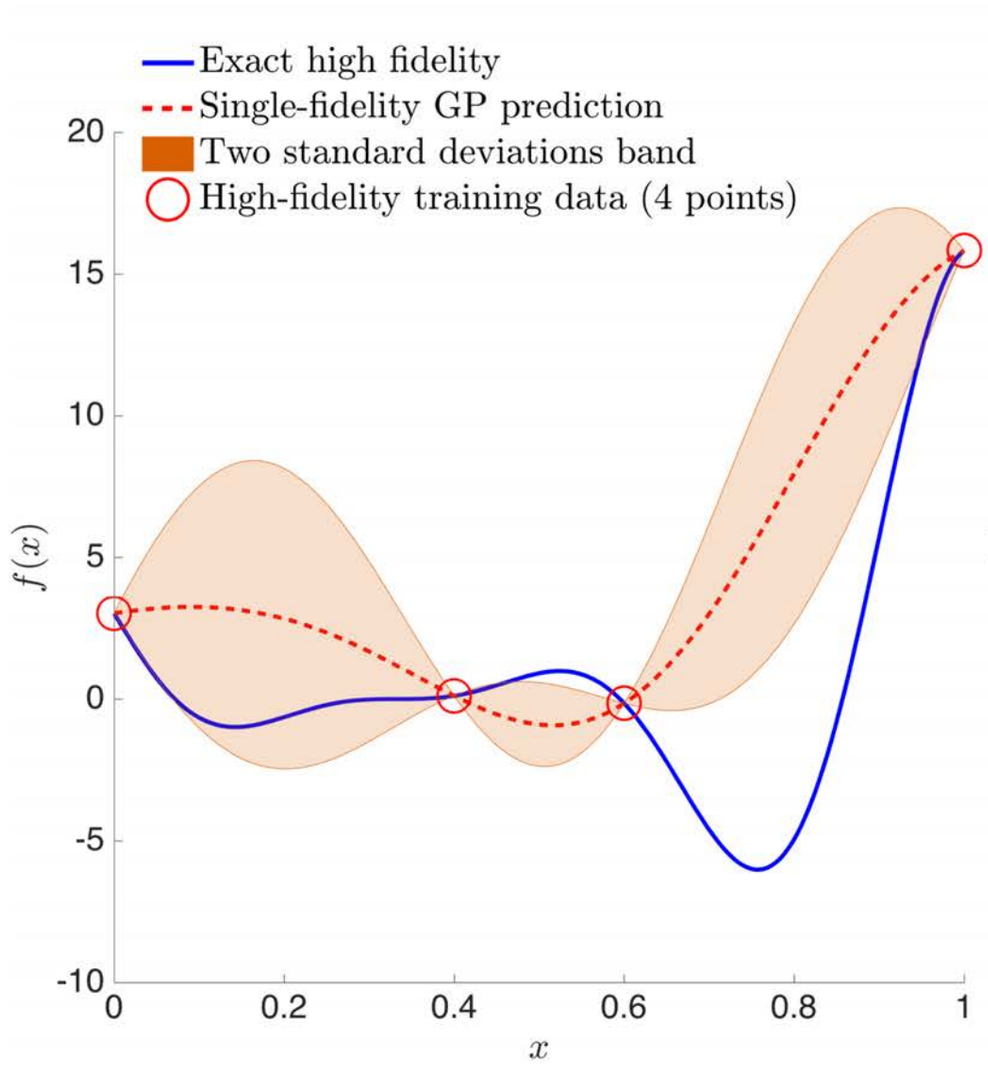
Number of runs is limited by time and computational resources

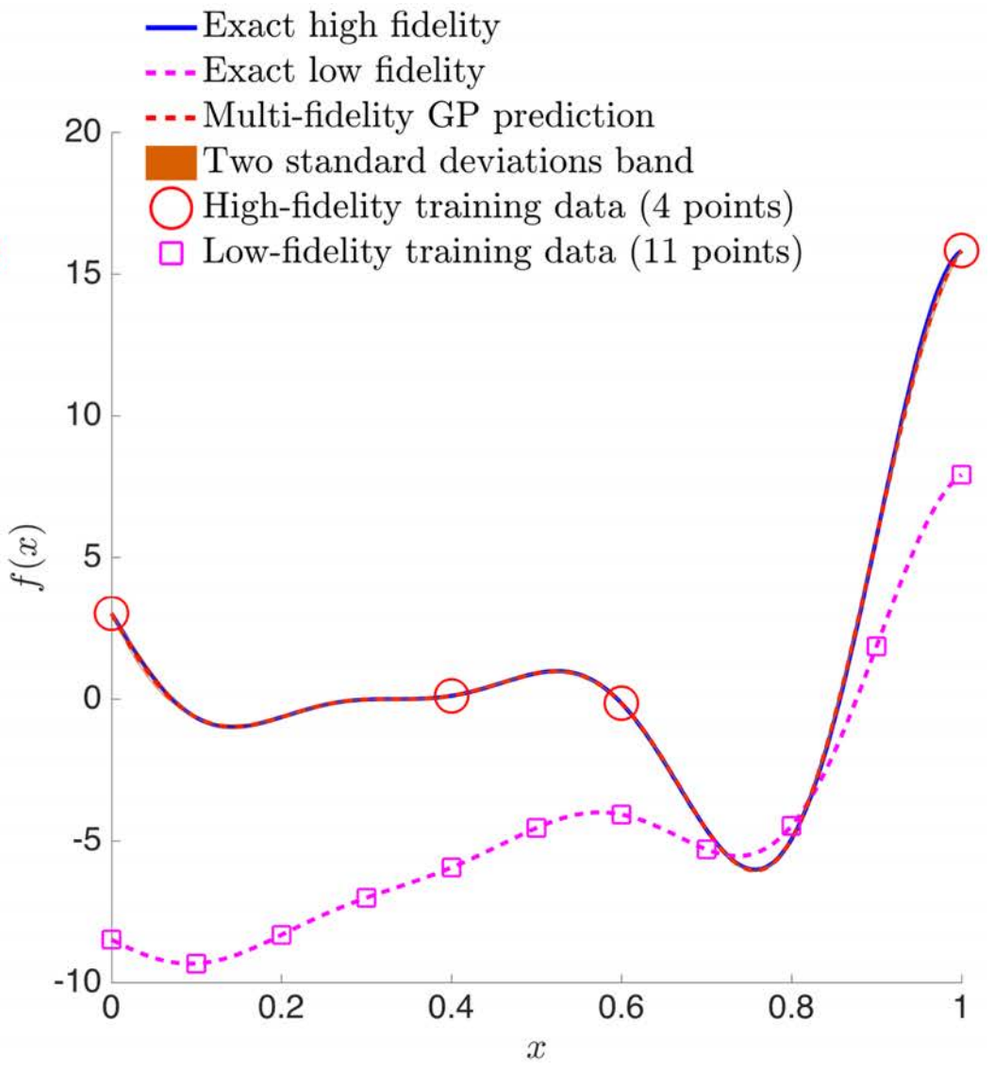
⇓

We cannot compute at all $(\boldsymbol{x};\boldsymbol{\xi})$

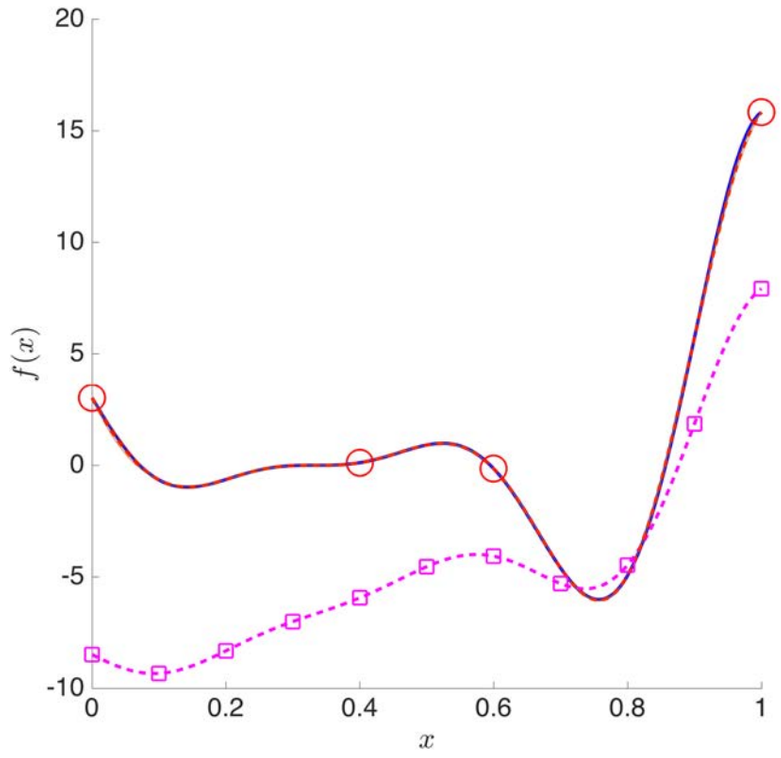
⇓

Prediction of $Z_i(\boldsymbol{x}) = \mathbb{E}[f(Y_i(\boldsymbol{x};\boldsymbol{\xi}))]$ is a problem of statistical inference





Multi-fidelity modeling



Multi-fidelity observations:

$$\boldsymbol{y}_L = f_L(\boldsymbol{x}_L) + \boldsymbol{\epsilon}_L$$
$$\boldsymbol{y}_H = f_H(\boldsymbol{x}_H) + \boldsymbol{\epsilon}_H$$

Probabilistic model:

$$f_H(\boldsymbol{x}) = \rho f_L(\boldsymbol{x}) + \delta(\boldsymbol{x})$$
$$f_L(\boldsymbol{x}) \sim \mathcal{GP}(0, k_L(\boldsymbol{x}, \boldsymbol{x}'; \theta_L))$$
$$\delta(\boldsymbol{x}) \sim \mathcal{GP}(0, k_H(\boldsymbol{x}, \boldsymbol{x}'; \theta_H))$$
$$\epsilon_L \sim \mathcal{N}(0, \sigma_{\epsilon_L}^2 \boldsymbol{I})$$
$$\epsilon_H \sim \mathcal{N}(0, \sigma_{\epsilon_H}^2 \boldsymbol{I})$$

Training:

$$\boldsymbol{y} = \begin{bmatrix} \boldsymbol{y}_L \\ \boldsymbol{y}_H \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \boldsymbol{0} \\ \boldsymbol{0} \end{bmatrix}, \begin{bmatrix} k_L(\boldsymbol{x}_L, \boldsymbol{x}'_L; \theta_L) + \sigma_{\epsilon_L}^2 \boldsymbol{I} & \rho k_L(\boldsymbol{x}_L, \boldsymbol{x}'_H; \theta_L) \\ \rho k_L(\boldsymbol{x}_H, \boldsymbol{x}'_L; \theta_L) & \rho^2 k_L(\boldsymbol{x}_H, \boldsymbol{x}'_H; \theta_L) + k_H(\boldsymbol{x}_H, \boldsymbol{x}'_H; \theta_H) + \sigma_{\epsilon_H}^2 \boldsymbol{I} \end{bmatrix} \right)$$

$$\boldsymbol{X} = \begin{bmatrix} \boldsymbol{x}_L \\ \boldsymbol{x}_H \end{bmatrix} \qquad -\log p(\boldsymbol{y}|\boldsymbol{X}, \theta_L, \theta_H, \rho, \sigma_{\epsilon_L}^2, \sigma_{\epsilon_H}^2) = \frac{1}{2}\log|\boldsymbol{K}| + \frac{1}{2}\boldsymbol{y}^T \boldsymbol{K}^{-1} \boldsymbol{y} - \frac{N_L + N_H}{2}\log 2\pi$$

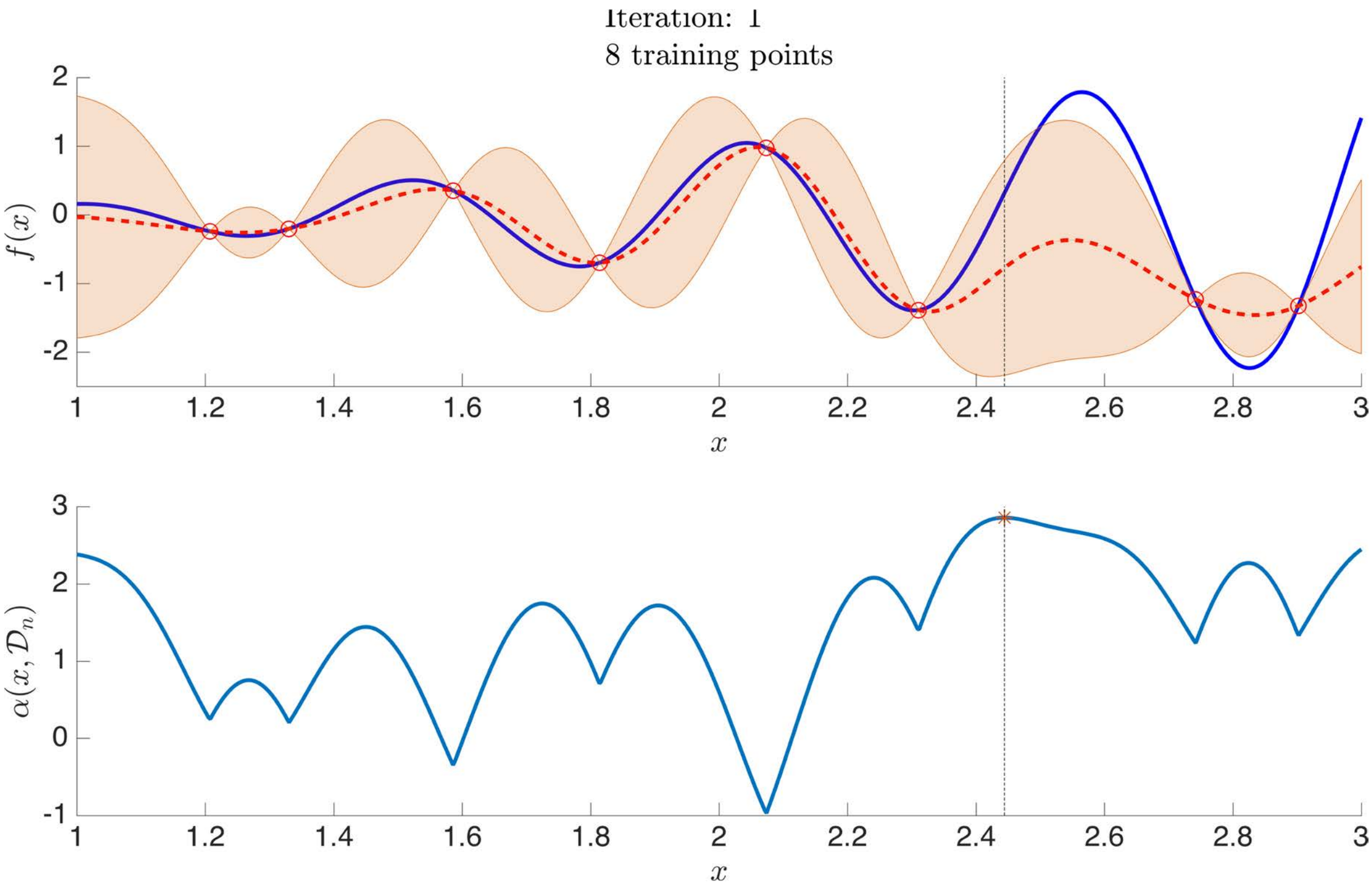
Prediction:

$$p(f(\boldsymbol{x}^*)|\boldsymbol{y}, \boldsymbol{X}, \boldsymbol{x}^*) \sim \mathcal{N}(f(\boldsymbol{x}^*)|\mu(\boldsymbol{x}^*), \sigma^2(\boldsymbol{x}^*))$$
$$\mu(\boldsymbol{x}^*) = \boldsymbol{k}(\boldsymbol{x}^*, \boldsymbol{X})\boldsymbol{K}^{-1}\boldsymbol{y}$$
$$\sigma(\boldsymbol{x}^*) = \boldsymbol{k}(\boldsymbol{x}^*, \boldsymbol{x}^*) - \boldsymbol{k}(\boldsymbol{x}^*, \boldsymbol{X})\boldsymbol{K}^{-1}\boldsymbol{k}(\boldsymbol{X}, \boldsymbol{x}^*)$$

M.C Kennedy, and A. O'Hagan. *Predicting the output from a complex computer code when fast approximations are available*, 2000.

Demo code: https://github.com/PredictiveIntelligenceLab/GPTutorial

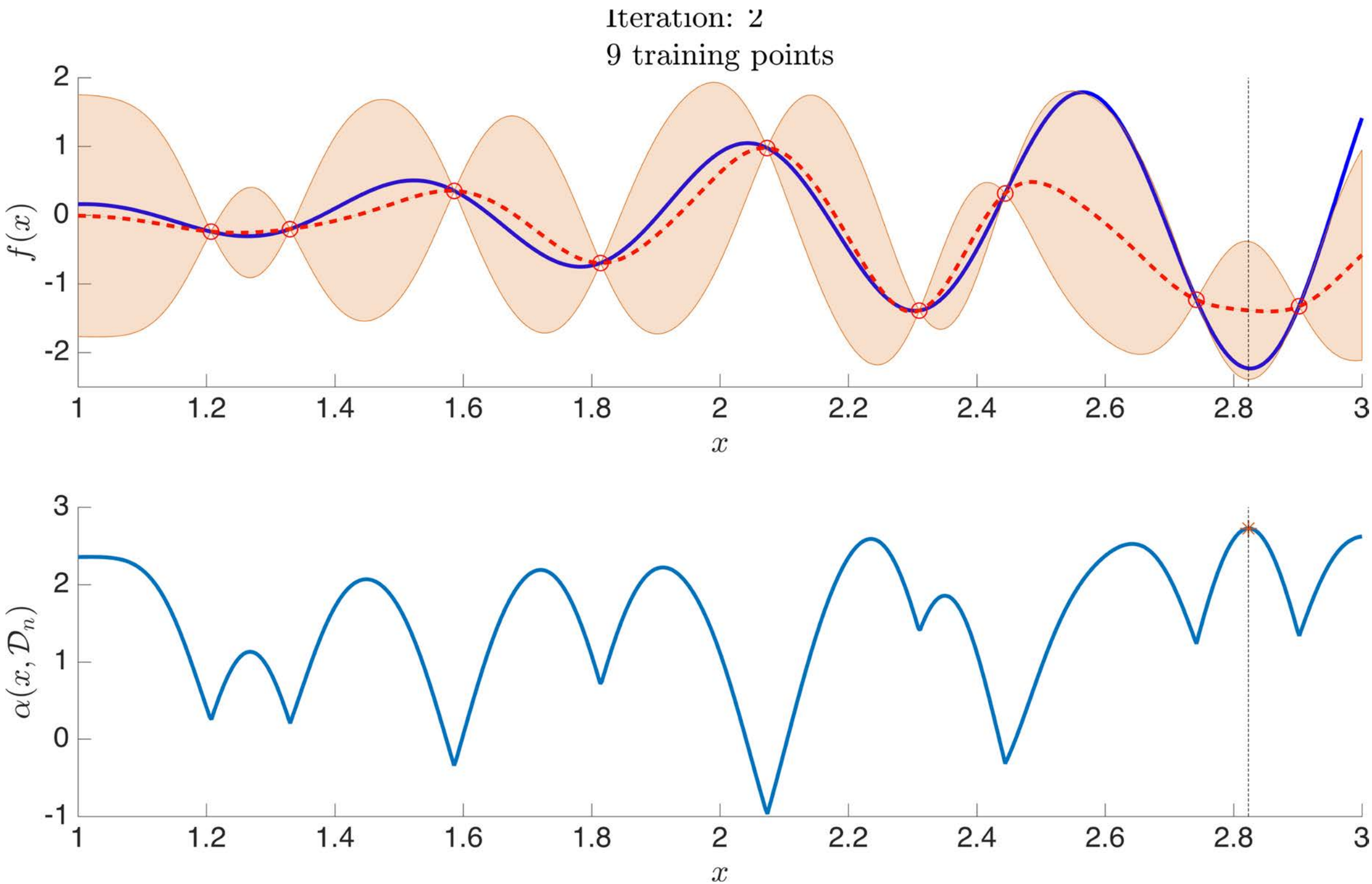
Bayesian optimization



Sample at the locations that minimize the lower super-quintile risk confidence bound of the posterior:

$$\boldsymbol{x}_{n+1} = \arg\min_{\boldsymbol{x}\in\mathbb{R}^d} \mu(\boldsymbol{x}) - \frac{\phi(\Phi^{-1}(\alpha))}{1-\alpha}\sigma(\boldsymbol{x})$$

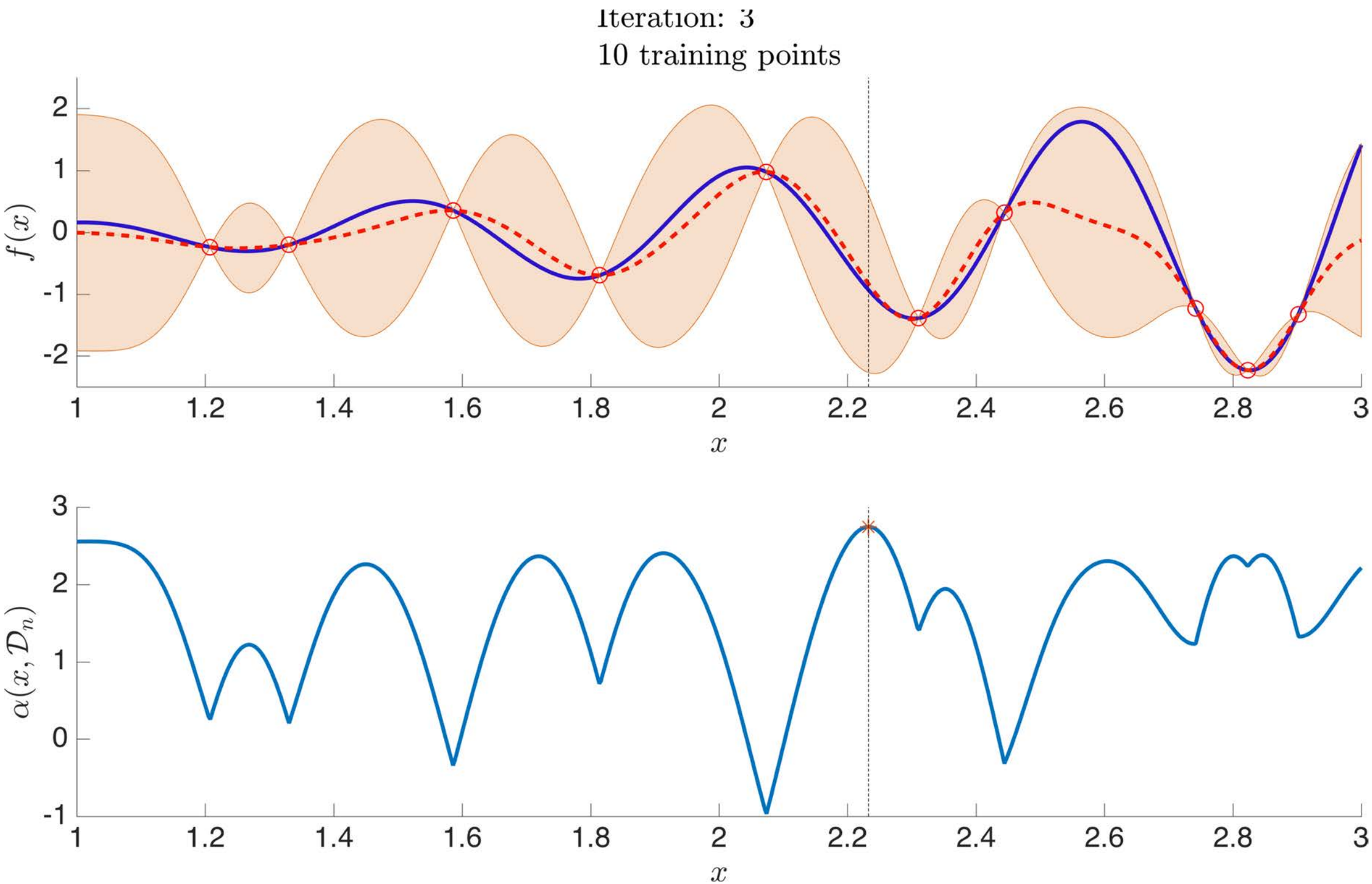
Bayesian optimization



Sample at the locations that minimize the lower super-quintile risk confidence bound of the posterior:

$$\boldsymbol{x}_{n+1} = \arg\min_{\boldsymbol{x} \in \mathbb{R}^d} \mu(\boldsymbol{x}) - \frac{\phi(\Phi^{-1}(\alpha))}{1-\alpha}\sigma(\boldsymbol{x})$$

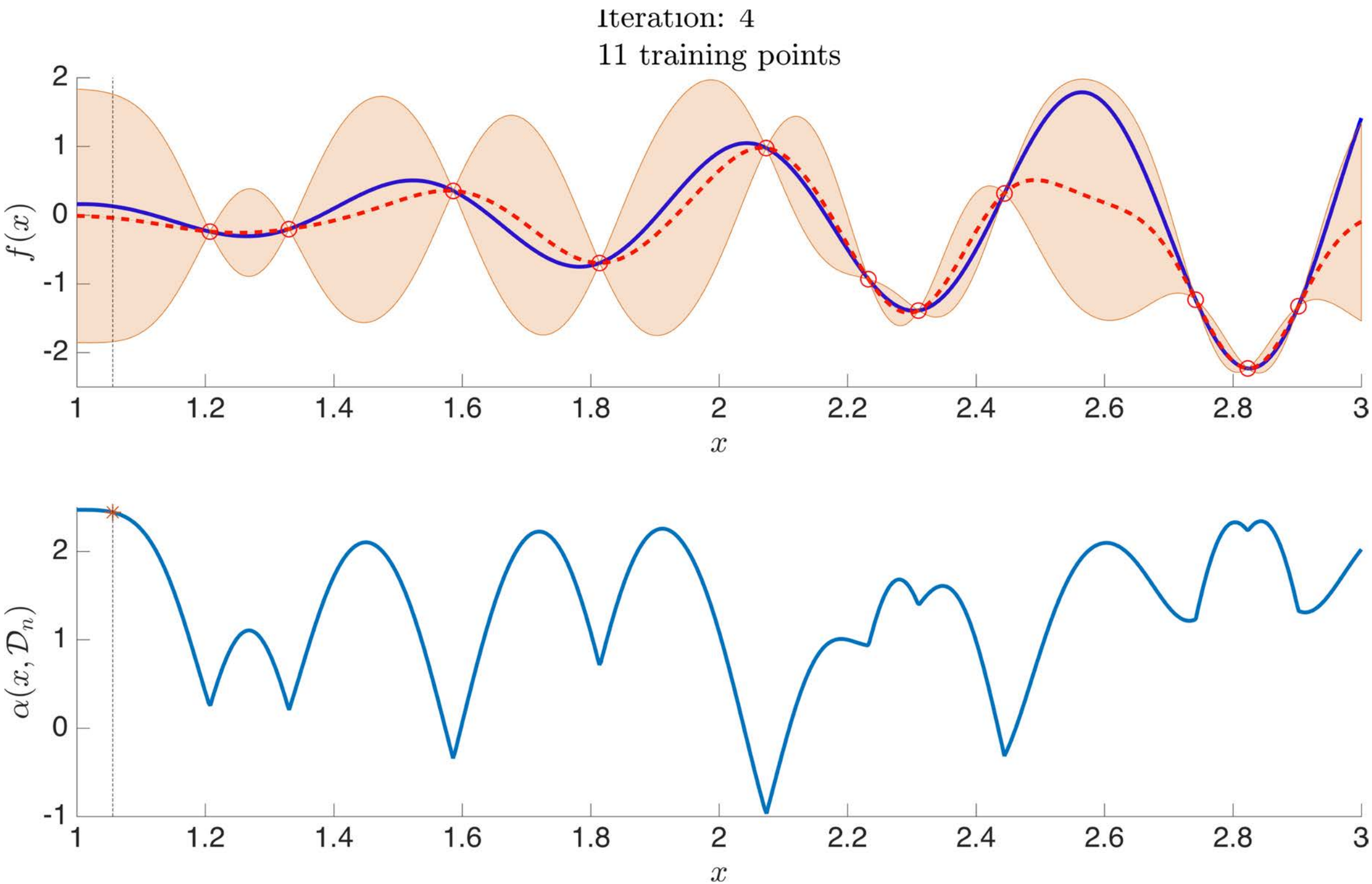
Bayesian optimization



Sample at the locations that minimize the lower super-quintile risk confidence bound of the posterior:

$$\boldsymbol{x}_{n+1} = \arg\min_{\boldsymbol{x}\in\mathbb{R}^d} \mu(\boldsymbol{x}) - \frac{\phi(\Phi^{-1}(\alpha))}{1-\alpha}\sigma(\boldsymbol{x})$$

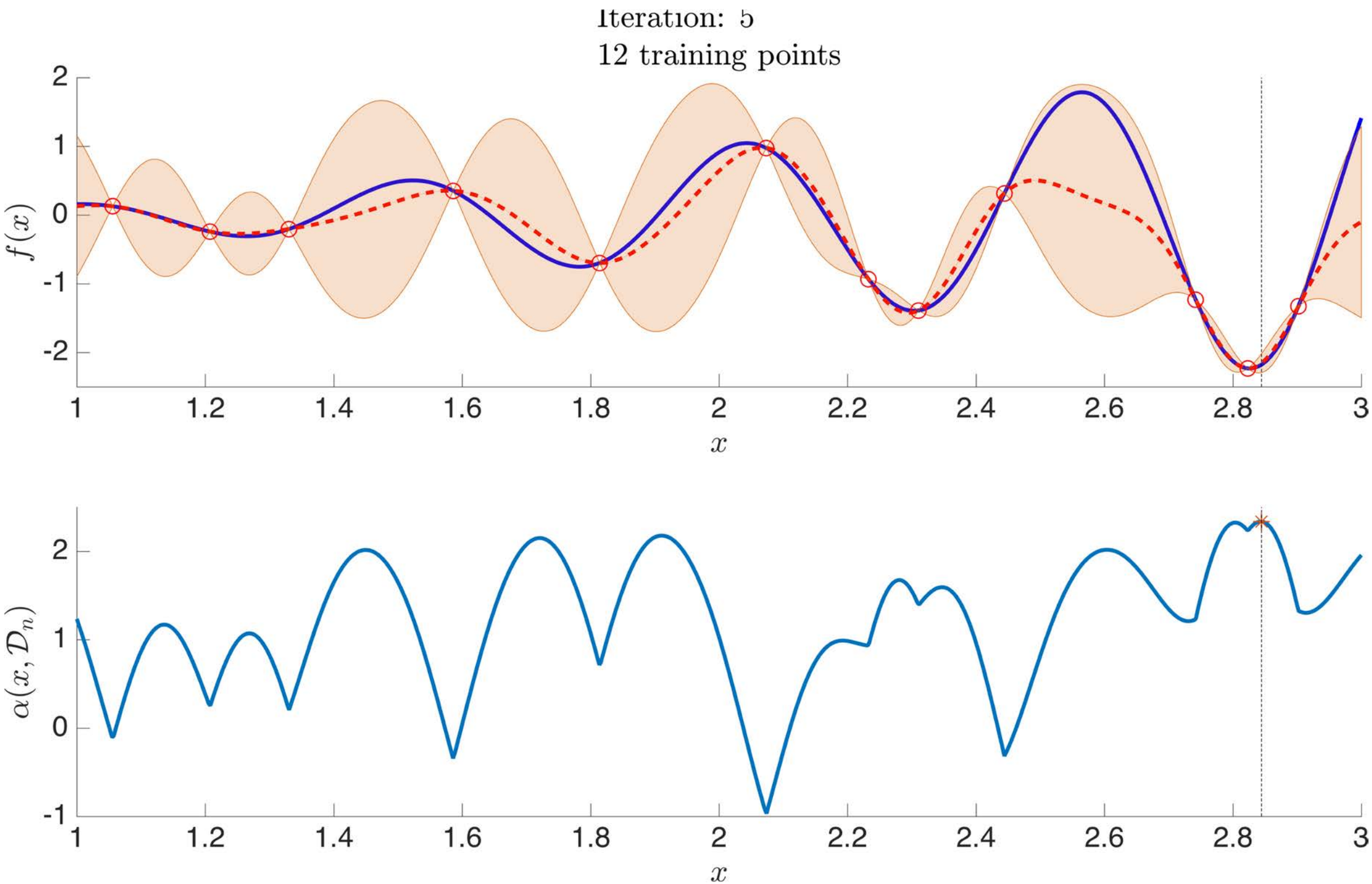
Bayesian optimization



Sample at the locations that minimize the lower super-quintile risk confidence bound of the posterior:

$$\boldsymbol{x}_{n+1} = \arg\min_{\boldsymbol{x} \in \mathbb{R}^d} \mu(\boldsymbol{x}) - \frac{\phi(\Phi^{-1}(\alpha))}{1-\alpha}\sigma(\boldsymbol{x})$$

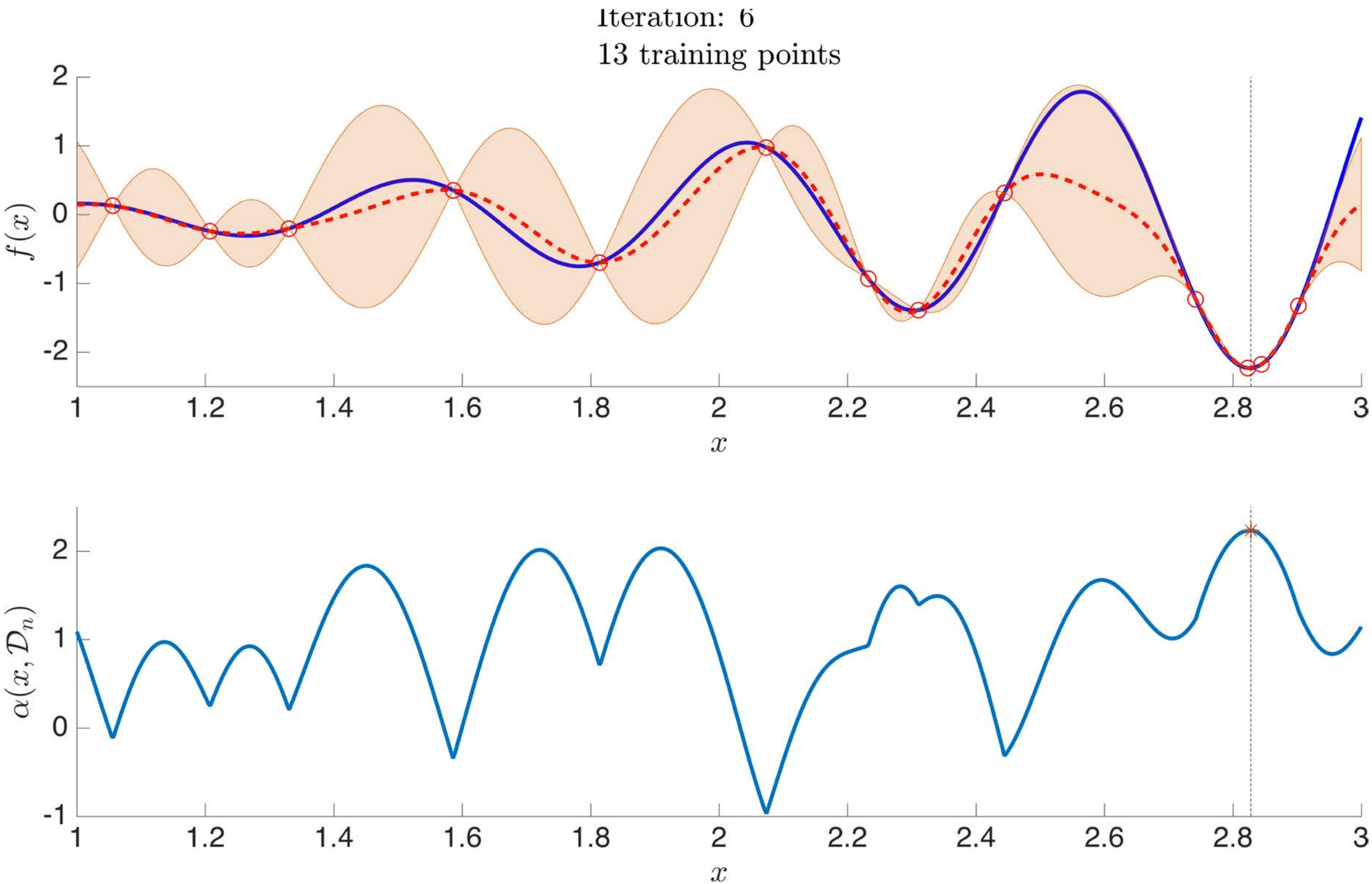
Bayesian optimization



Sample at the locations that minimize the lower super-quintile risk confidence bound of the posterior:

$$\boldsymbol{x}_{n+1} = \arg\min_{\boldsymbol{x}\in\mathbb{R}^d} \mu(\boldsymbol{x}) - \frac{\phi(\Phi^{-1}(\alpha))}{1-\alpha}\sigma(\boldsymbol{x})$$

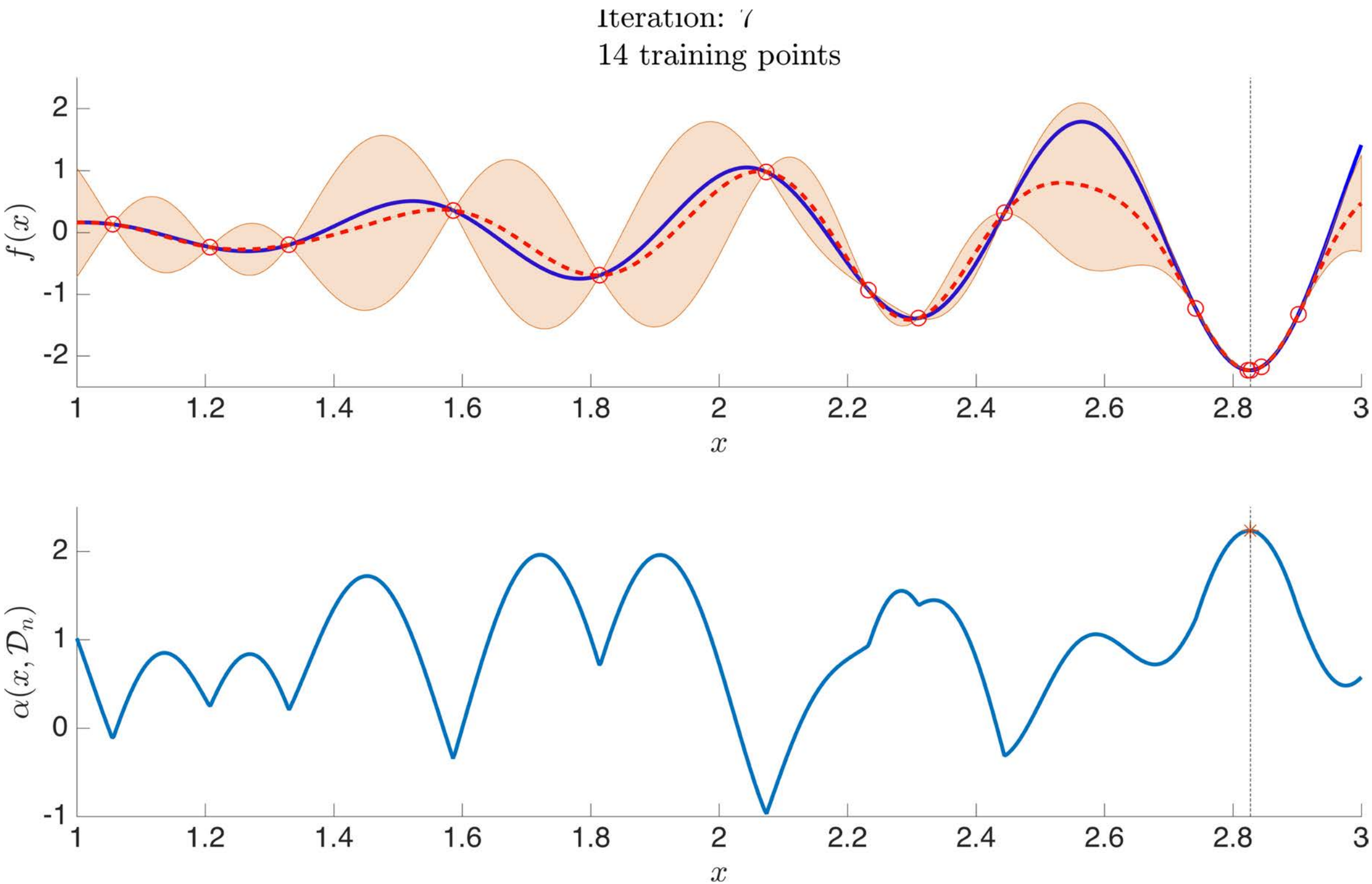
Bayesian optimization



Sample at the locations that minimize the lower super-quintile risk confidence bound of the posterior:

$$\boldsymbol{x}_{n+1} = \arg\min_{\boldsymbol{x}\in\mathbb{R}^d} \mu(\boldsymbol{x}) - \frac{\phi(\Phi^{-1}(\alpha))}{1-\alpha}\sigma(\boldsymbol{x})$$

Bayesian optimization

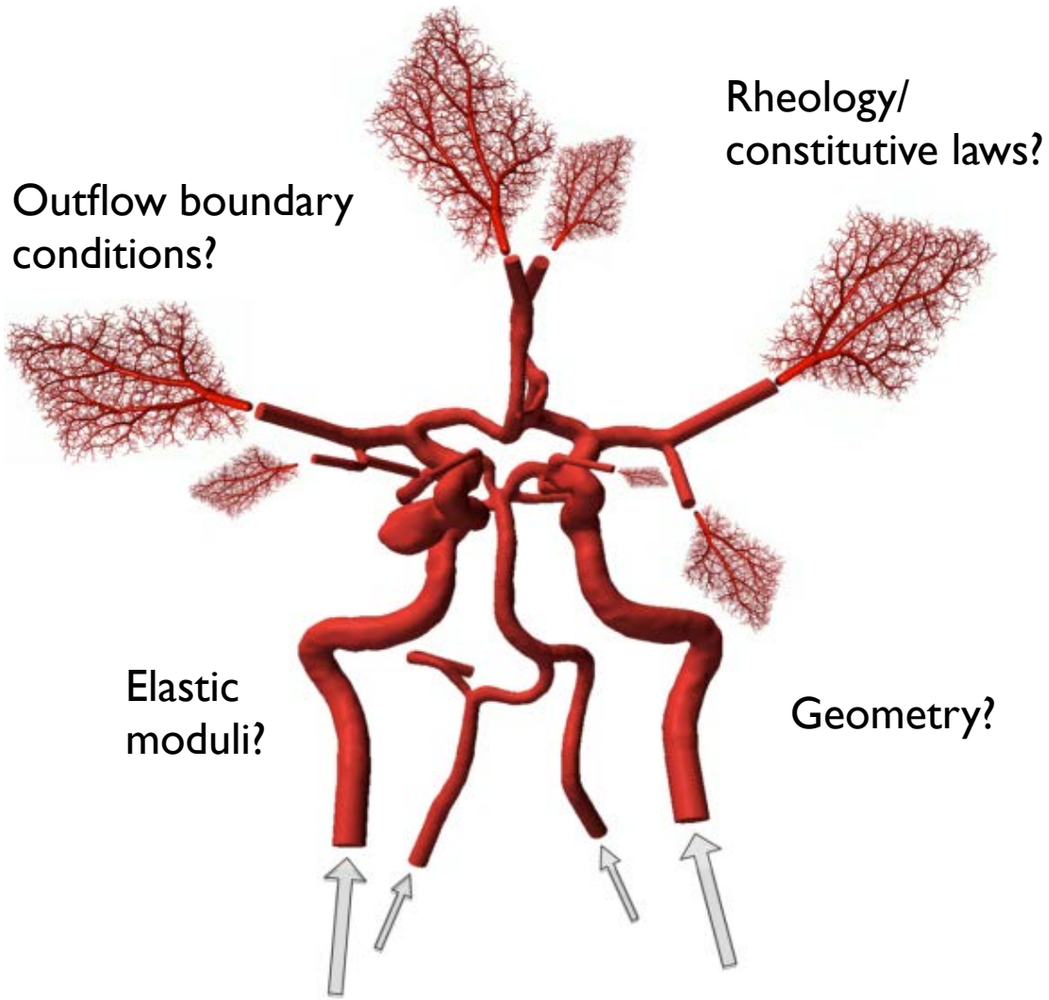


Sample at the locations that minimize the lower super-quintile risk confidence bound of the posterior:

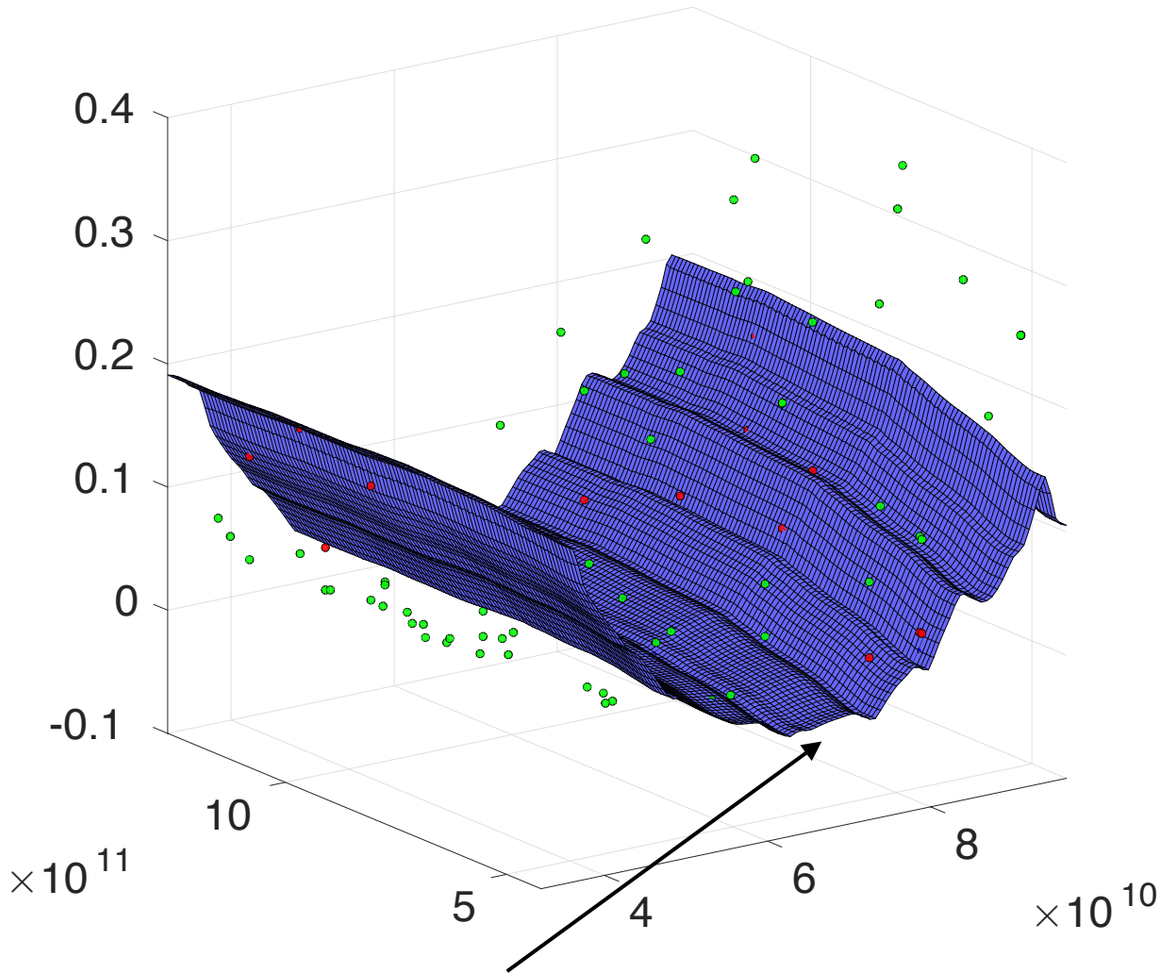
$$\boldsymbol{x}_{n+1} = \arg\min_{\boldsymbol{x}\in\mathbb{R}^d} \mu(\boldsymbol{x}) - \frac{\phi(\Phi^{-1}(\alpha))}{1-\alpha}\sigma(\boldsymbol{x})$$

Example application: *Calibration of blood flow simulations*

Goal: Calibrate a large-scale flow solver to match observed clinical data.



Models (e.g. 3D vs 1D,
continuum vs atomistic, etc.)



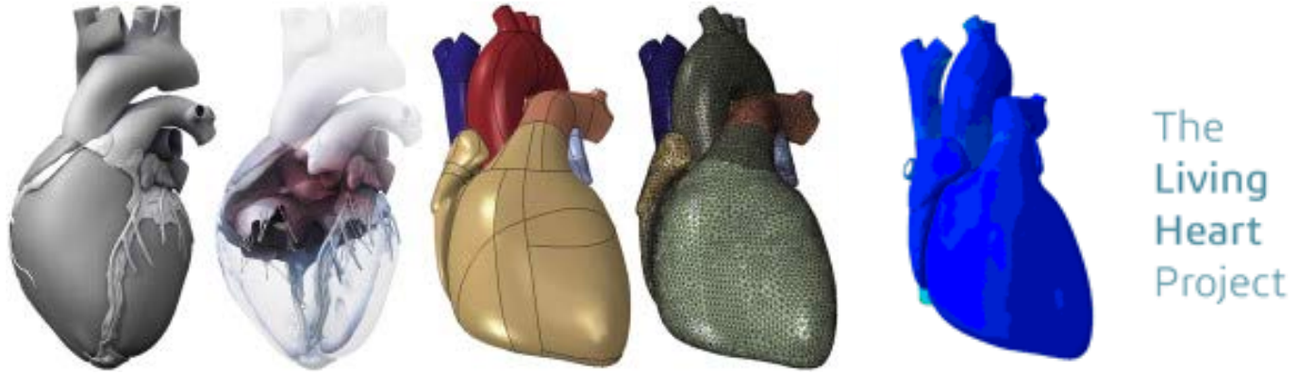
Decreased the relative error to 1e-3 after 3 iterations of BO, mainly sampling the lowest fidelity (cheapest) solver.

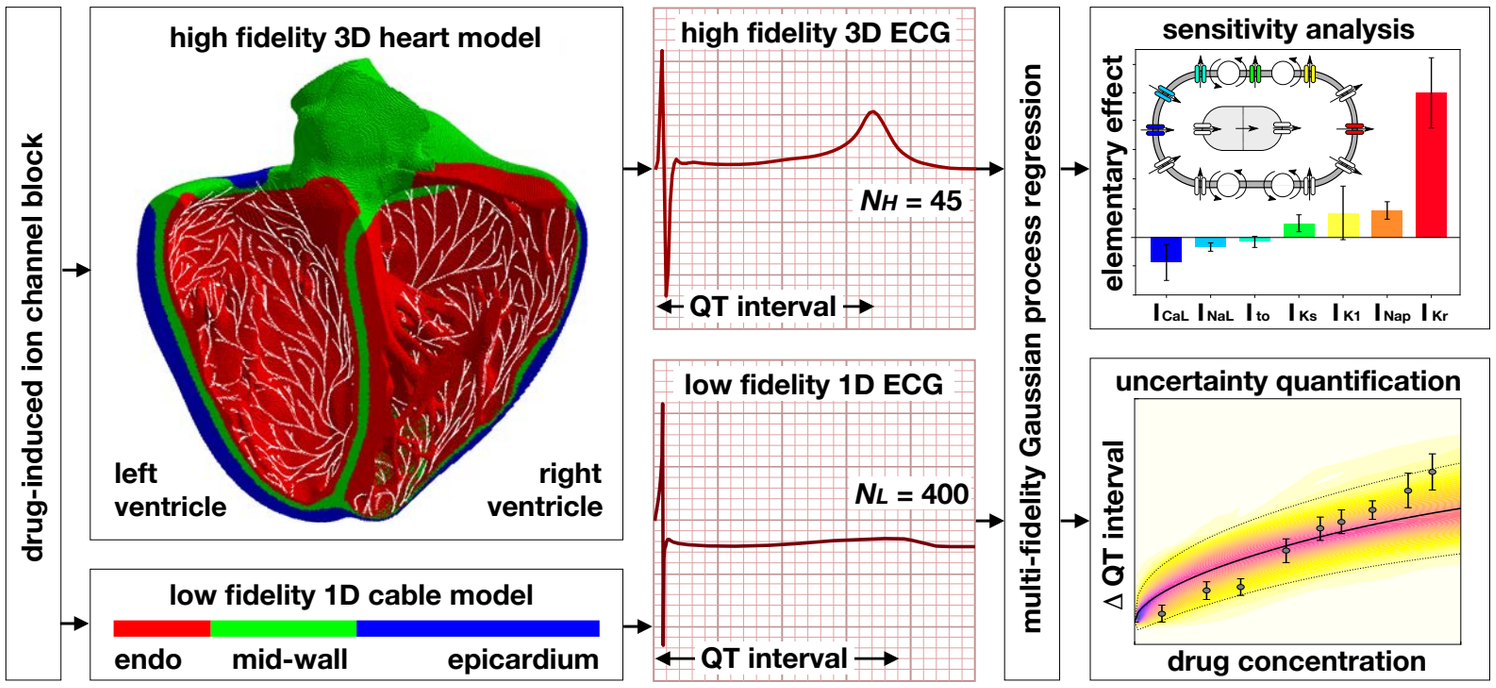
Multi-fidelity approach:

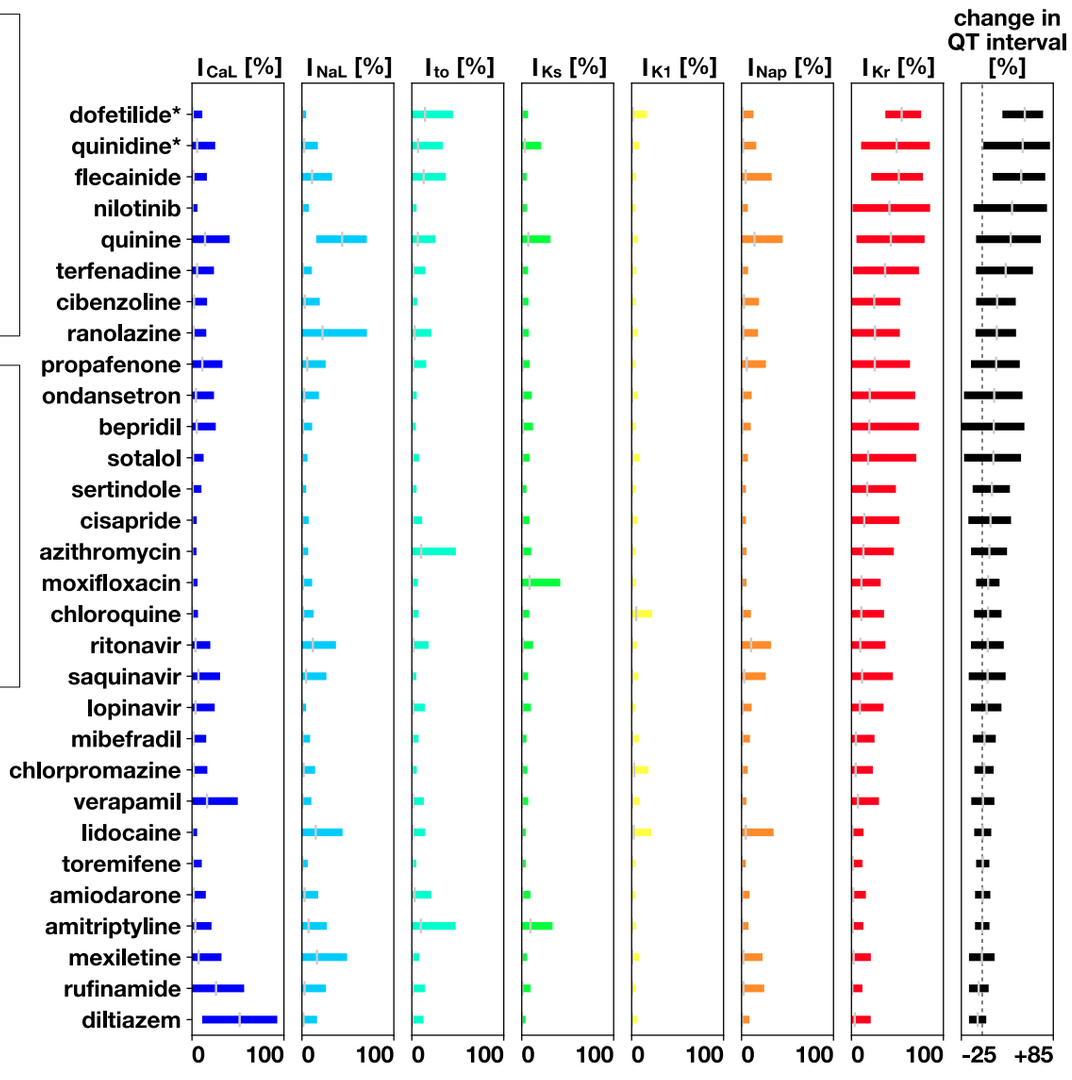
1.) 3D Navier-Stokes (spectral/hp elements, rigid artery) - high fidelity O(hrs)
2.) Non-linear 1D-FSI (DG, compliant artery) - intermediate fidelity O(mins)
3.) Linearized 1D-FSI solver around a (possibly) inaccurate reference state - low fidelity O(s)

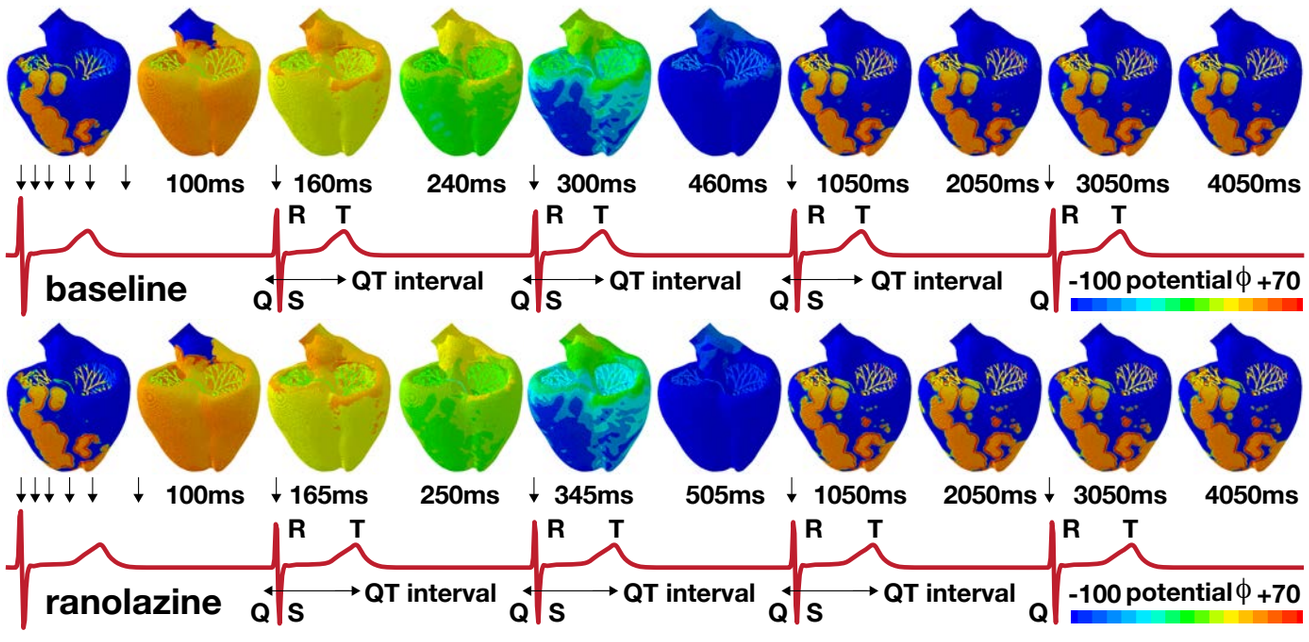
P. Perdikaris, and G.E Karniadakis. "Model inversion via multi-fidelity Bayesian optimization." J. R. Soc. Interface (2016)

Example application:
Assessing the risk of drug-induced cardiac arrhythmias





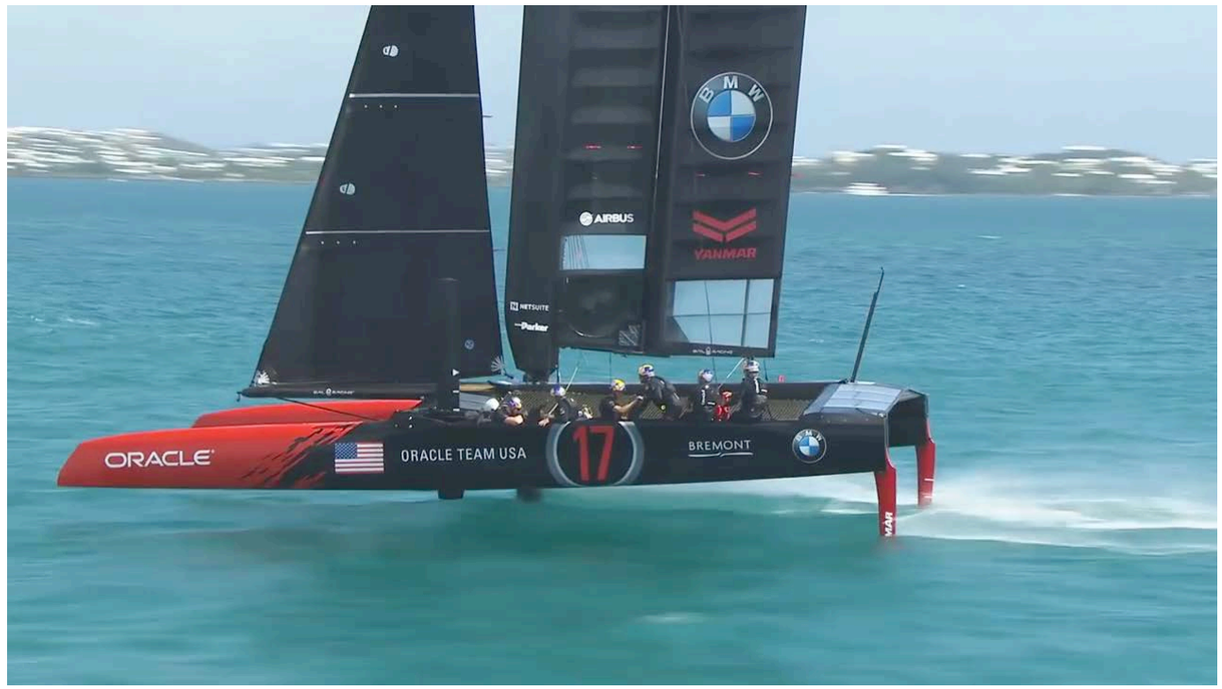


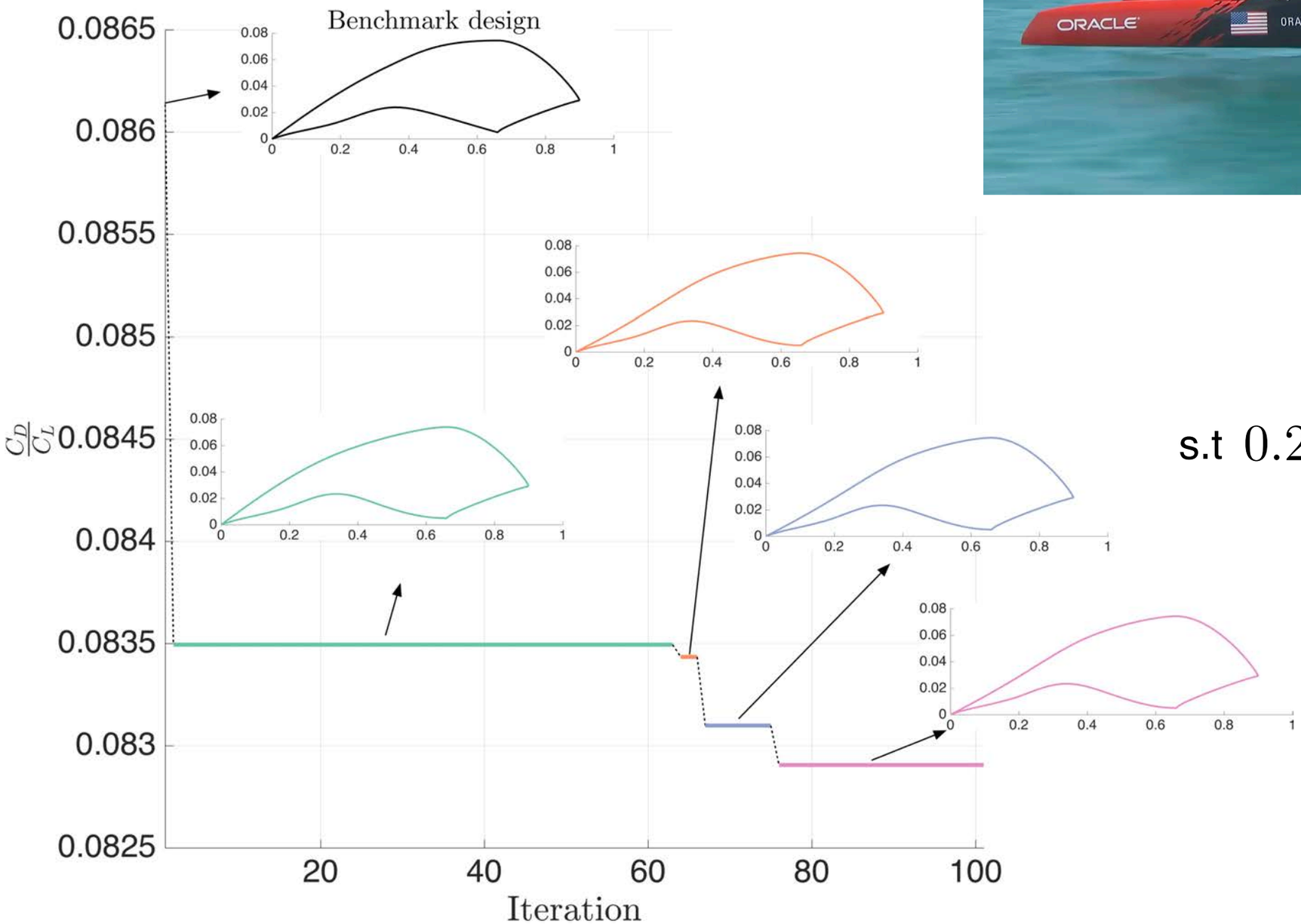


We demonstrated that drugs that block a specific potassium current most drastically prolong the QT interval and are therefore associated with the highest risk of torsades de pointes.

Costabal, F. S., Matsuno, K., Yao, J., Perdikaris, P., & Kuhl, E. (2019). Machine learning in drug development: Characterizing the effect of 30 drugs on the QT interval using Gaussian process regression, sensitivity analysis, and uncertainty quantification. Computer Methods in Applied Mechanics and Engineering, 348, 313-333.

Example application:
Shape optimization of super-cavitating hydrofoils





$$\min_{\mathbf{x}\in\mathcal{X}} \mathcal{R}_{0.9}\left(\frac{C_D}{C_L}(\boldsymbol{x};\boldsymbol{\xi})\right)$$

$$\text{s.t } 0.23 \leq \mathcal{R}_{0.5}(C_L(\boldsymbol{x};\boldsymbol{\xi})) \leq 0.26$$

$$\mathcal{R}_{0.5}(I_{\mathrm{x}}(\boldsymbol{x};\boldsymbol{\xi})) \geq 8.1 \cdot 10^{-6}$$

$$\mathcal{R}_{0.5}(T_{\mathrm{p}}(\boldsymbol{x};\boldsymbol{\xi})) \geq 0.00132$$

35 design variables
B-MF: 735 LF + 120 HF runs
GA-LF: ~50,000 LF runs

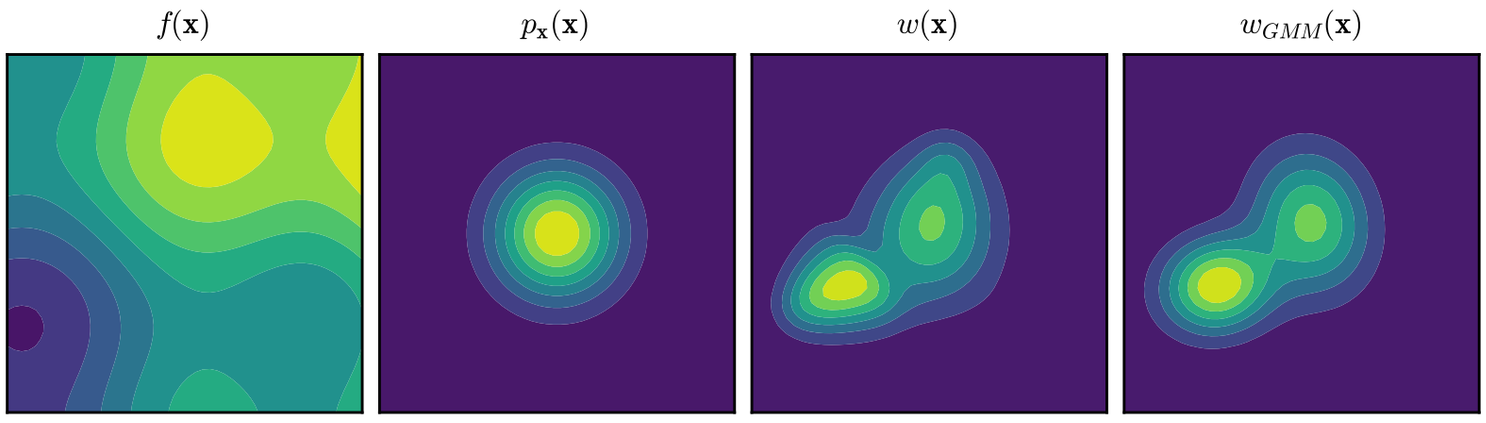
Bonfiglio, L., Perdikaris, P., Brizzolara, S., & Karniadakis, G. E. (2018). Multi-fidelity optimization of super-cavitating hydrofoils. Computer Methods in Applied Mechanics and Engineering.

Likelihood-weighted Bayesian optimization

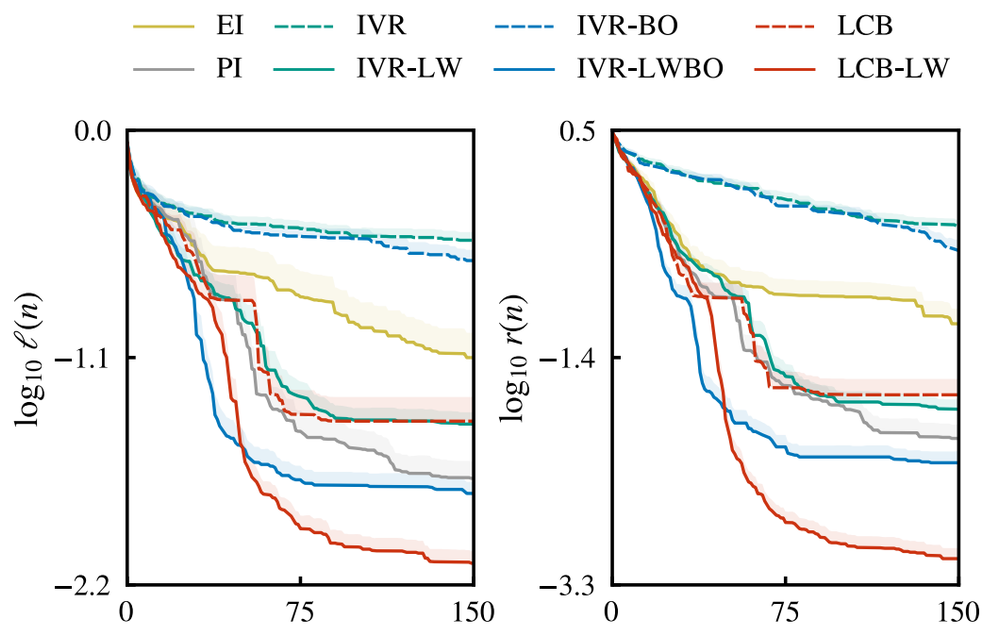
Key idea: Account for the importance of the output relative to the input.

$$\boldsymbol{x}_{n+1} = \arg\min_{\boldsymbol{x}} \alpha(\boldsymbol{x}; \mathcal{D}) := \mu(\boldsymbol{x}) - \kappa\sigma(\boldsymbol{x}) \frac{p_{\boldsymbol{x}}(\boldsymbol{x})}{p_{\mu}(\mu(\boldsymbol{x}))}$$

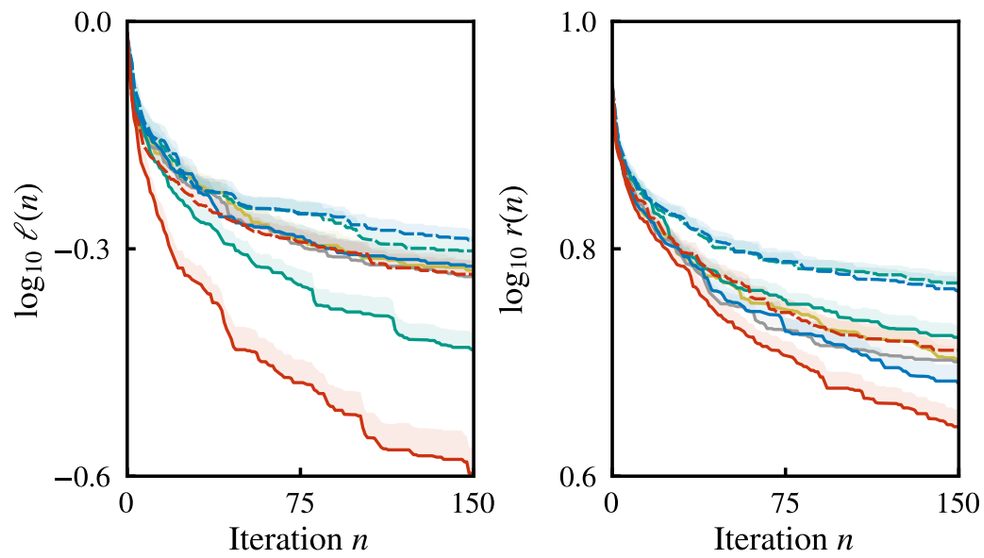
e.g. likelihood-weighted LCB acquisition function



- *The likelihood ratio guides the search algorithm towards regions of the input space where the objective function assumes abnormally large (or small) values.*
- *Largest gains achieved when the objective function assumes rare and extreme (i.e., abnormally large or small) output values, i.e. the conditional pdf of the output $p_{f|x}$ is heavy- tailed.*
- *Currently limited to scalar-output GP priors and kernels with analytical gradients (RBF).*



(e) 6-D Hartmann function



(f) 10-D Michalewicz function

Blanchard, A., & Sapsis, T. (2020). Bayesian Optimization with Output-Weighted Importance Sampling. arXiv preprint arXiv:2004.10599.

Integrated software

A Bayesian optimization library in JAX

https://github.com/PredictiveIntelligenceLab/JAX-BO

Key features:

- Automatic differentiation
- Vectorization/Parallelization
- XLA/JIT compilation to GPUs/TPUs
- Support for arbitrary GP priors and deep neural networks
- Support for fully Bayesian inference with HMC-NUTS sampling (via NumPyro)
- Support for multi-fidelity GPs with heterogeneous inputs
- Support for likelihood-weighted acquisition functions for Bayesian optimization

Currently under final rounds of development - - let me know if you'd like to try it out and contribute!

The fine prints…

- Low-fidelity sources can be inaccurate, but they need to be strongly correlated with the high-fidelity data.
- Automating simulation pipelines is challenging (mesh generation, pre/post-processing, etc.)
- Failed/unstable experiments/simulations can introduce outliers and data bias.
- Dealing with heterogeneous inputs is not straightforward.
- Scaling to high-dimensions and large data-sets is (to some extend) possible, but not straightforward.
- Prior specification/kernel design plays a massively important role.

- How to best incorporate domain knowledge and physical constraints?

$$\mathcal{L}_x u(x) = f(x)$$

e.g. linear constraints:

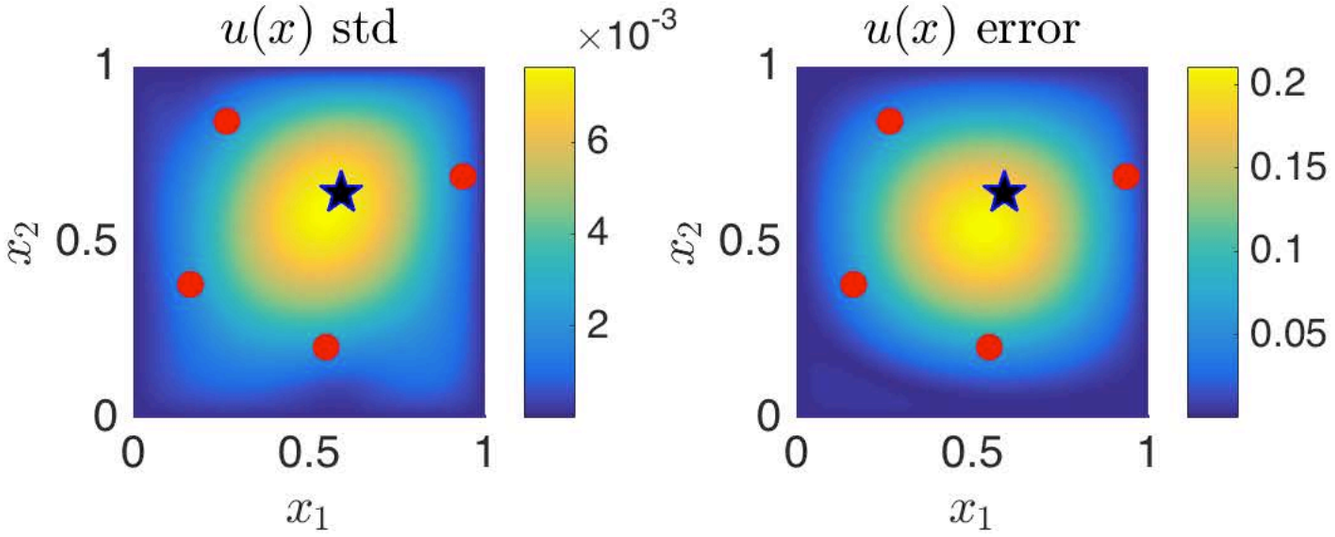
$$u(x) \sim \mathcal{GP}(0, g(x, x'; \theta)) \longrightarrow f(x) \sim \mathcal{GP}(0, k(x, x'; \theta))$$

$$k(x, x'; \theta) = \mathcal{L}_x \mathcal{L}_{x'} g(x, x'; \theta)$$

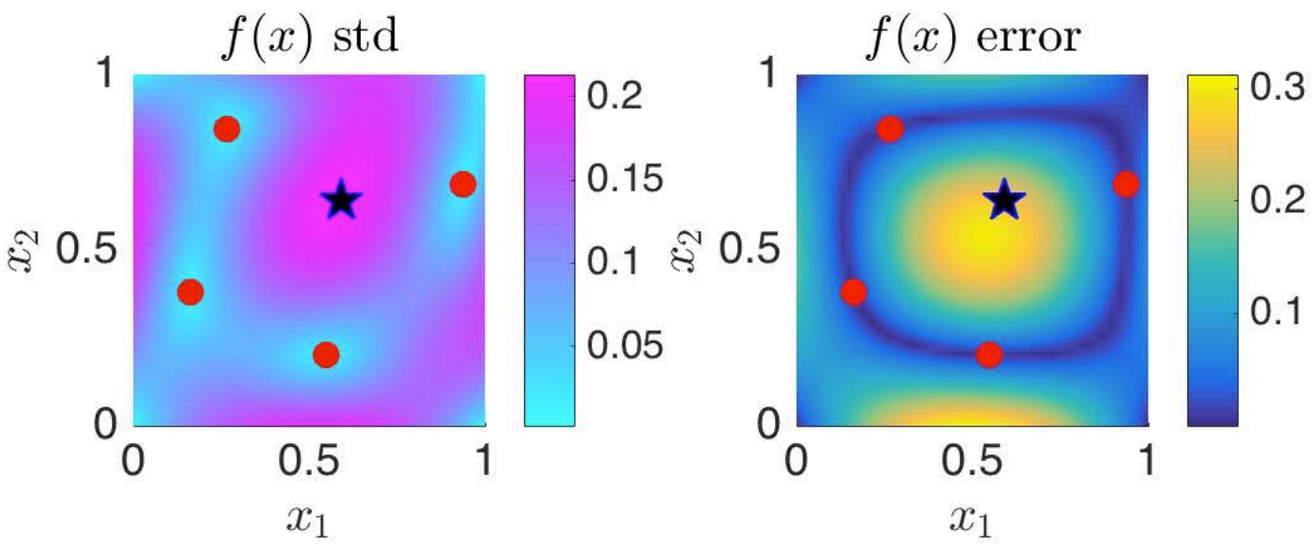
Raissi, M., Perdikaris, P., & Karniadakis, G. E. (2017). Inferring solutions of differential equations using noisy multi-fidelity data. Journal of Computational Physics, 335, 736-746.

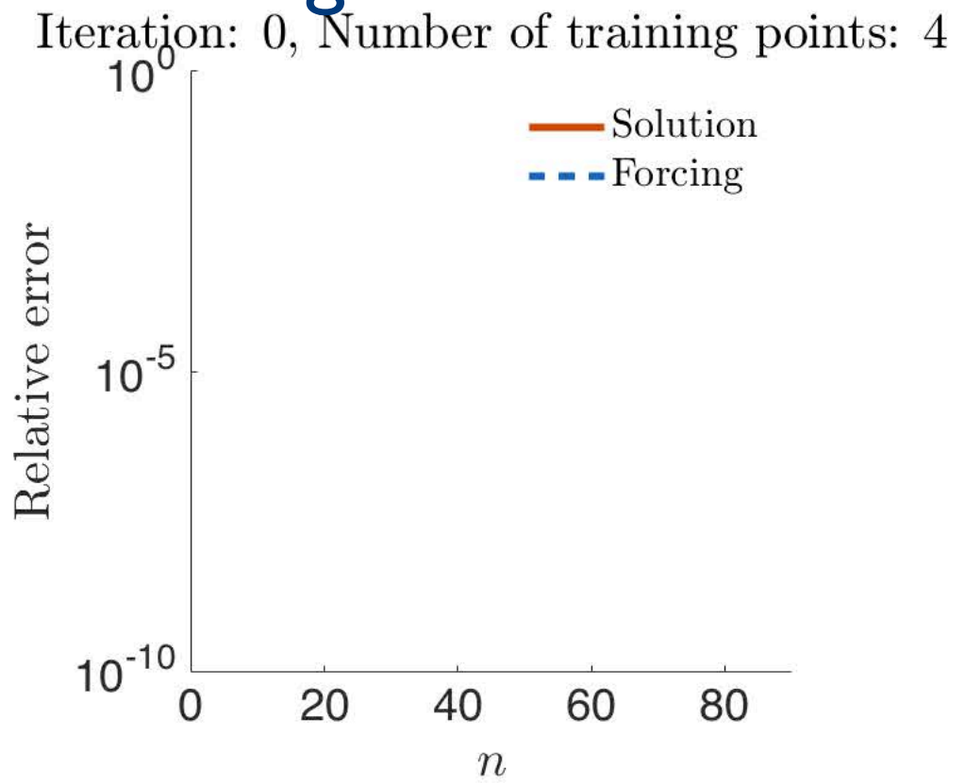
Raissi, M., Perdikaris, P., & Karniadakis, G. E. (2017). Machine learning of linear differential equations using Gaussian processes. Journal of Computational Physics, 348, 683-693.

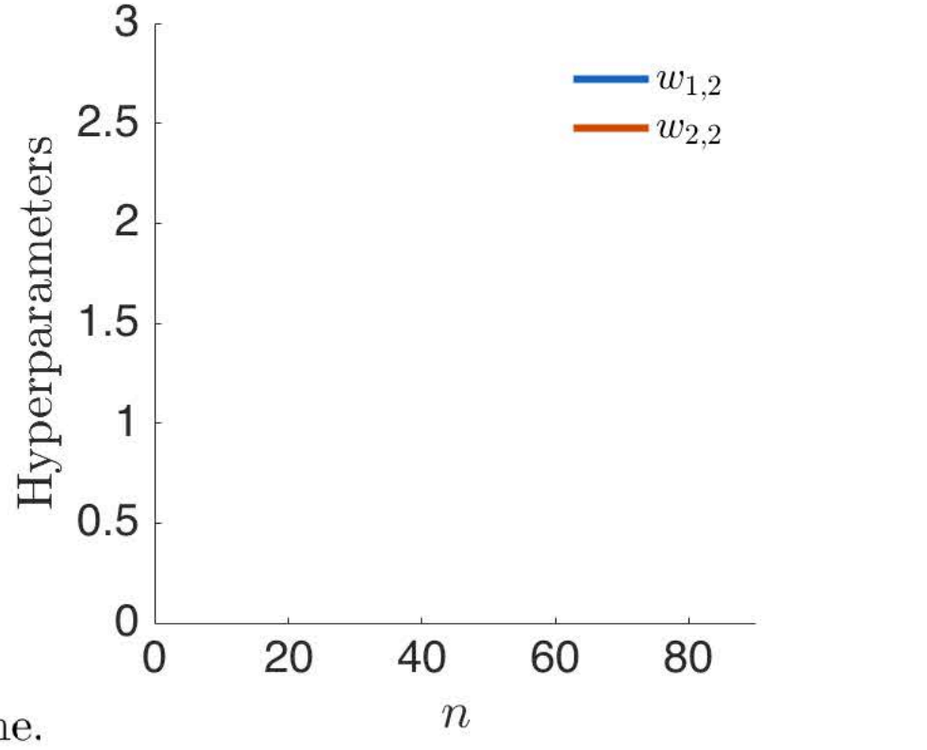
Solving PDEs via active learning



$$\frac{\partial^2 u}{\partial {x_1}^2} + \frac{\partial^2 u}{\partial {x_2}^2} = f(x_1, x_2)$$





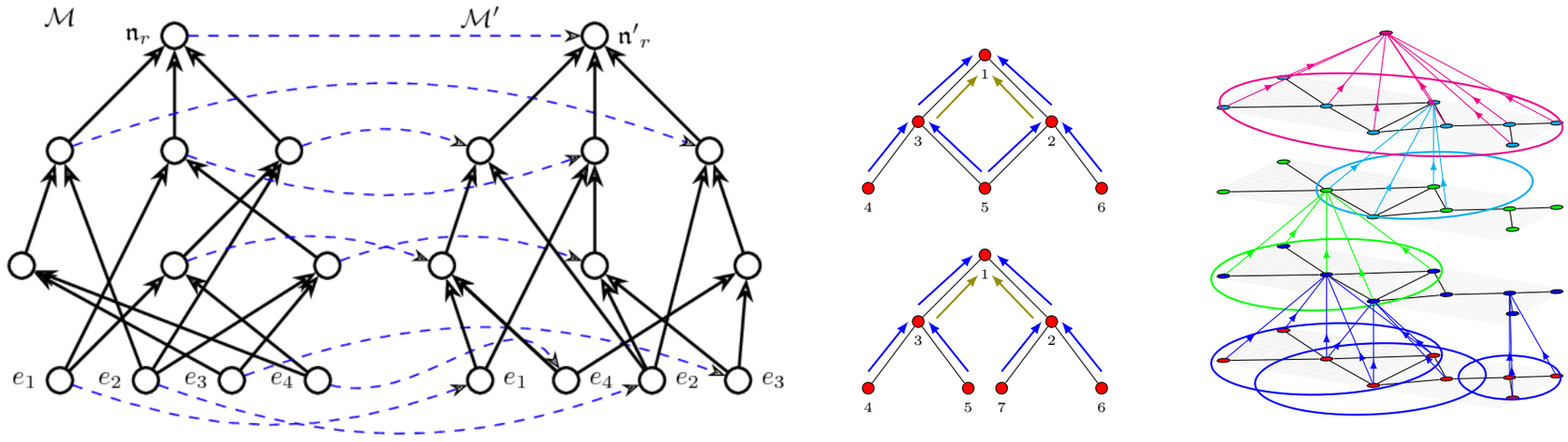


- denotes the training data actively collected by the scheme.
- ★ denotes the next sampling point suggested by the active learning scheme.

Raissi, M., Perdikaris, P., & Karniadakis, G. E. (2017). Inferring solutions of differential equations using noisy multi-fidelity data. Journal of Computational Physics, 335, 736-746.

Physics of AI: Two schools of thought

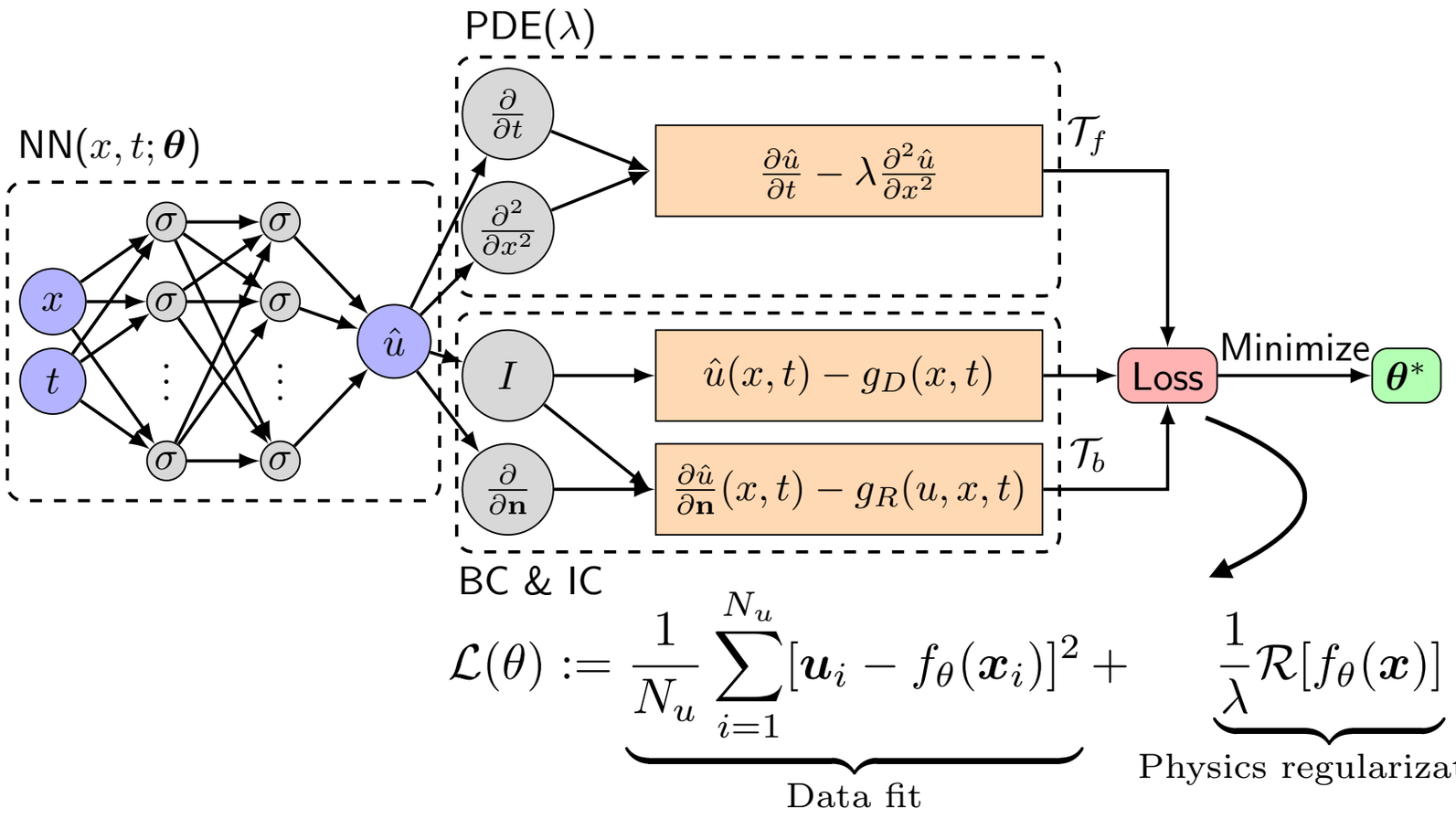
1. Physics is implicitly baked in specialized neural architectures with strong inductive biases (e.g. invariance to simple group symmetries).



**figures from Kondor, R., Son, H.T., Pan, H., Anderson, B., & Trivedi, S. (2018). Covariant compositional networks for learning graphs. arXiv preprint arXiv:1801.02144.*

2. Physics is explicitly imposed by constraining the output of conventional neural architectures with weak inductive biases.

Psichogios & Ungar, 1992
Lagaris et. al., 1998
Raissi et. al., 2019
Lu et. al., 2019
Zhu et. al., 2019



General formulation of PINNs

Physics-informed neural networks (PINNs) aim at inferring a continuous latent function $\boldsymbol{u}(\boldsymbol{x}, t)$ that arises as the solution to a system of nonlinear partial differential equations (PDE) of the general form

$$\begin{aligned}
&\boldsymbol{u}_t + \mathcal{N}_{\boldsymbol{x}}[\boldsymbol{u}] = 0, \quad \boldsymbol{x} \in \Omega, t \in [0, T] \\
&\boldsymbol{u}(\boldsymbol{x}, 0) = h(\boldsymbol{x}), \quad x \in \Omega \\
&\boldsymbol{u}(\boldsymbol{x}, t) = g(\boldsymbol{x}, t), \quad t \in [0, T], \quad \boldsymbol{x} \in \partial\Omega
\end{aligned}$$

We proceed by approximating $\boldsymbol{u}(\boldsymbol{x}, t)$ by a deep neural network $f_\theta(\boldsymbol{x}, t)$, and define the residual of the PDE as

$$\boldsymbol{r}_\theta(\boldsymbol{x}, t) := \frac{\partial}{\partial t} f_\theta(\boldsymbol{x}, t) + \mathcal{N}_{\boldsymbol{x}}[f_\theta(\boldsymbol{x}, t)]$$

The corresponding loss function is given by

$$\mathcal{L}(\theta) := \underbrace{\mathcal{L}_u(\theta)}_{\text{Data fit}} + \underbrace{\mathcal{L}_r(\theta)}_{\text{PDE residual}} + \underbrace{\mathcal{L}_{u_0}(\theta)}_{\text{ICs fit}} + \underbrace{\mathcal{L}_{u_b}(\theta)}_{\text{BCs fit}}$$

Training via stochastic gradient descent:

$$\theta_{n+1} = \theta_n - \eta \nabla_\theta \mathcal{L}(\theta_n)$$

**all gradients are computed via automatic differentiation*

Raissi, M., Perdikaris, P., & Karniadakis, G. E. (2019). Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. Journal of Computational Physics, 378, 686-707.

Lagaris, I. E., Likas, A., & Fotiadis, D. I. (1998). Artificial neural networks for solving ordinary and partial differential equations.

Psichogios, D. C., & Ungar, L. H. (1992). A hybrid neural network–first principles approach to process modeling.

Physics-informed Neural Networks

Example: Burgers' equation in 1D

$$
\begin{aligned}
&u_t + uu_x - (0.01/\pi)u_{xx} = 0, \quad x \in [-1, 1], \quad t \in [0, 1], \qquad (3)\\
&u(0, x) = -\sin(\pi x),\\
&u(t, -1) = u(t, 1) = 0.
\end{aligned}
$$

Let us define $f(t, x)$ to be given by

$$f := u_t + uu_x - (0.01/\pi)u_{xx},$$

```
def u(t, x):
    u = neural_net(tf.concat([t,x],1), weights, biases)
    return u
```

Correspondingly, the *physics informed neural network* $f(t, x)$ takes the form

```
def f(t, x):
    u = u(t, x)
    u_t = tf.gradients(u, t)[0]
    u_x = tf.gradients(u, x)[0]
    u_xx = tf.gradients(u_x, x)[0]
    f = u_t + u*u_x - (0.01/tf.pi)*u_xx
    return f
```

Physics-informed Neural Networks

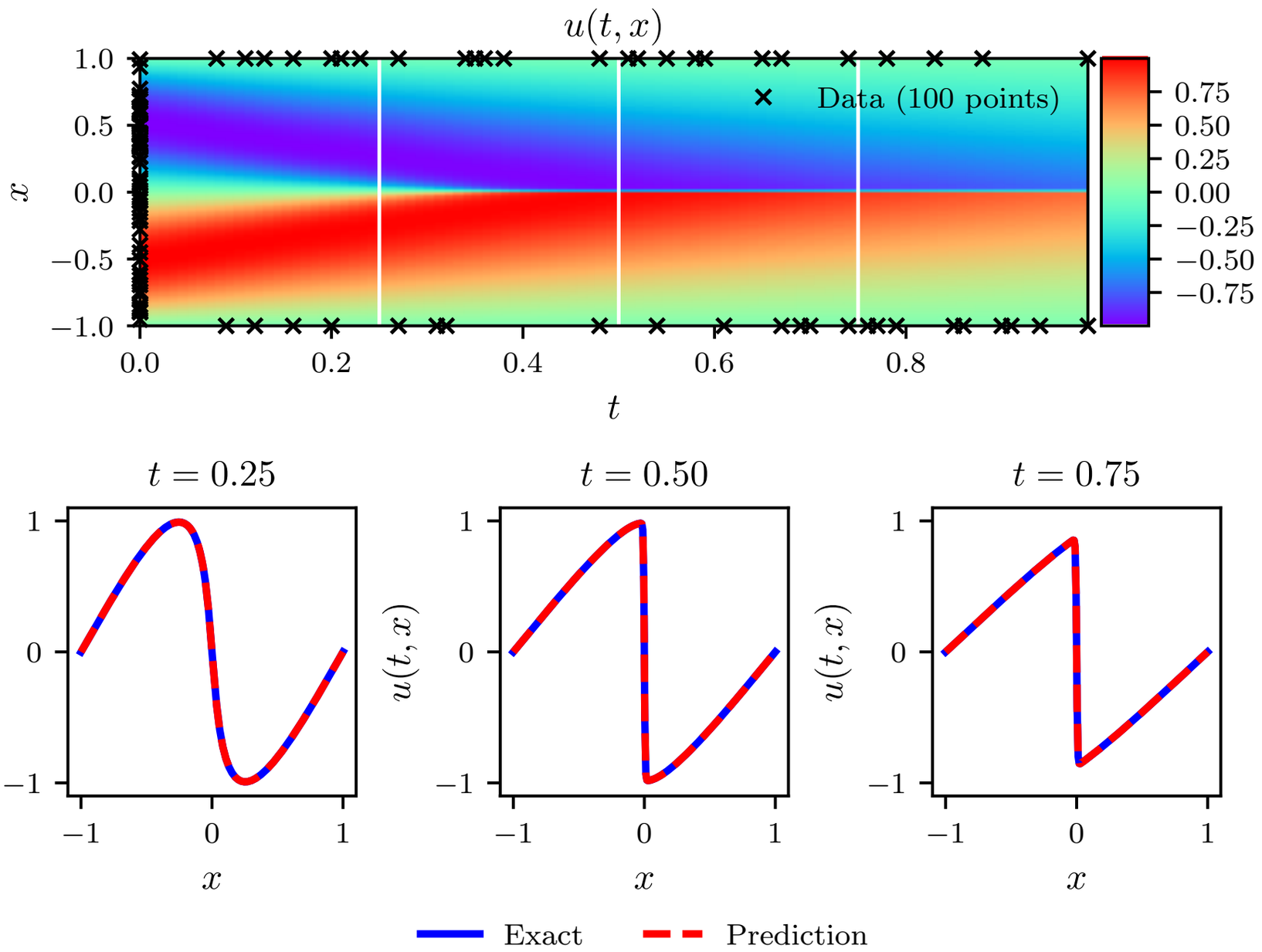
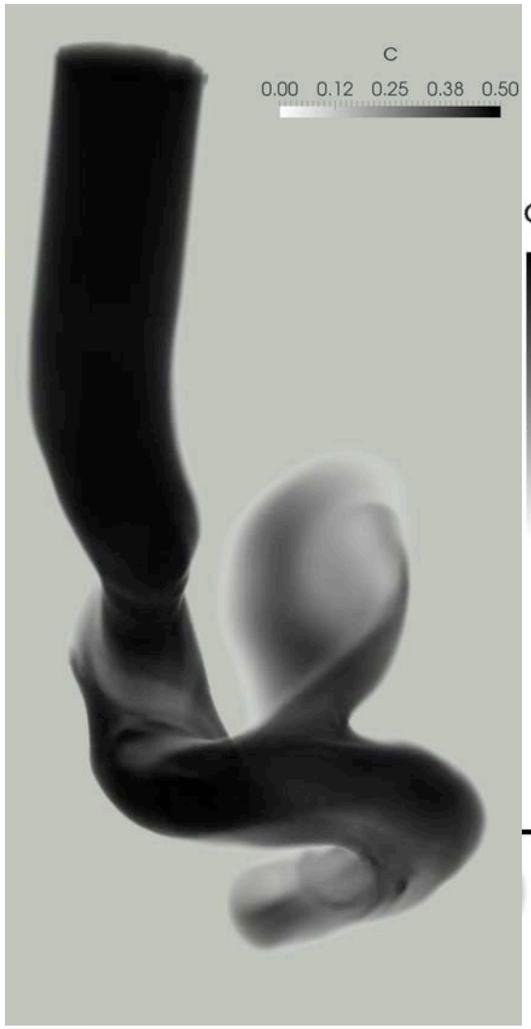
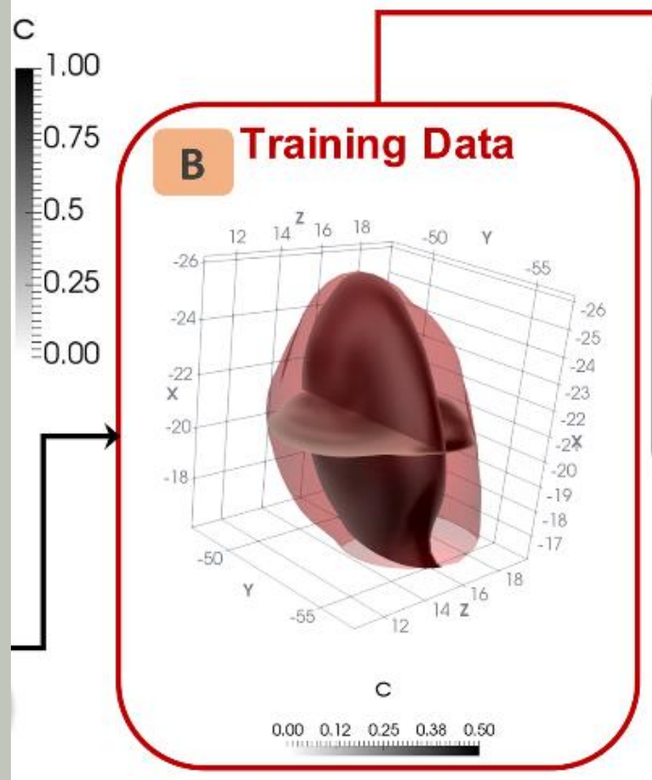
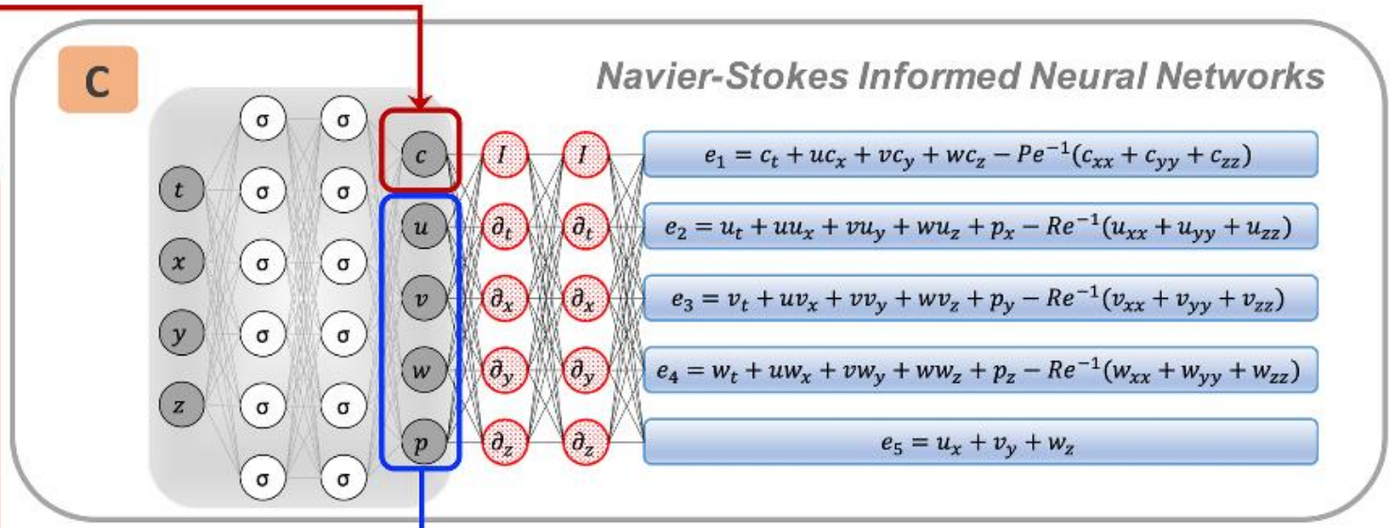


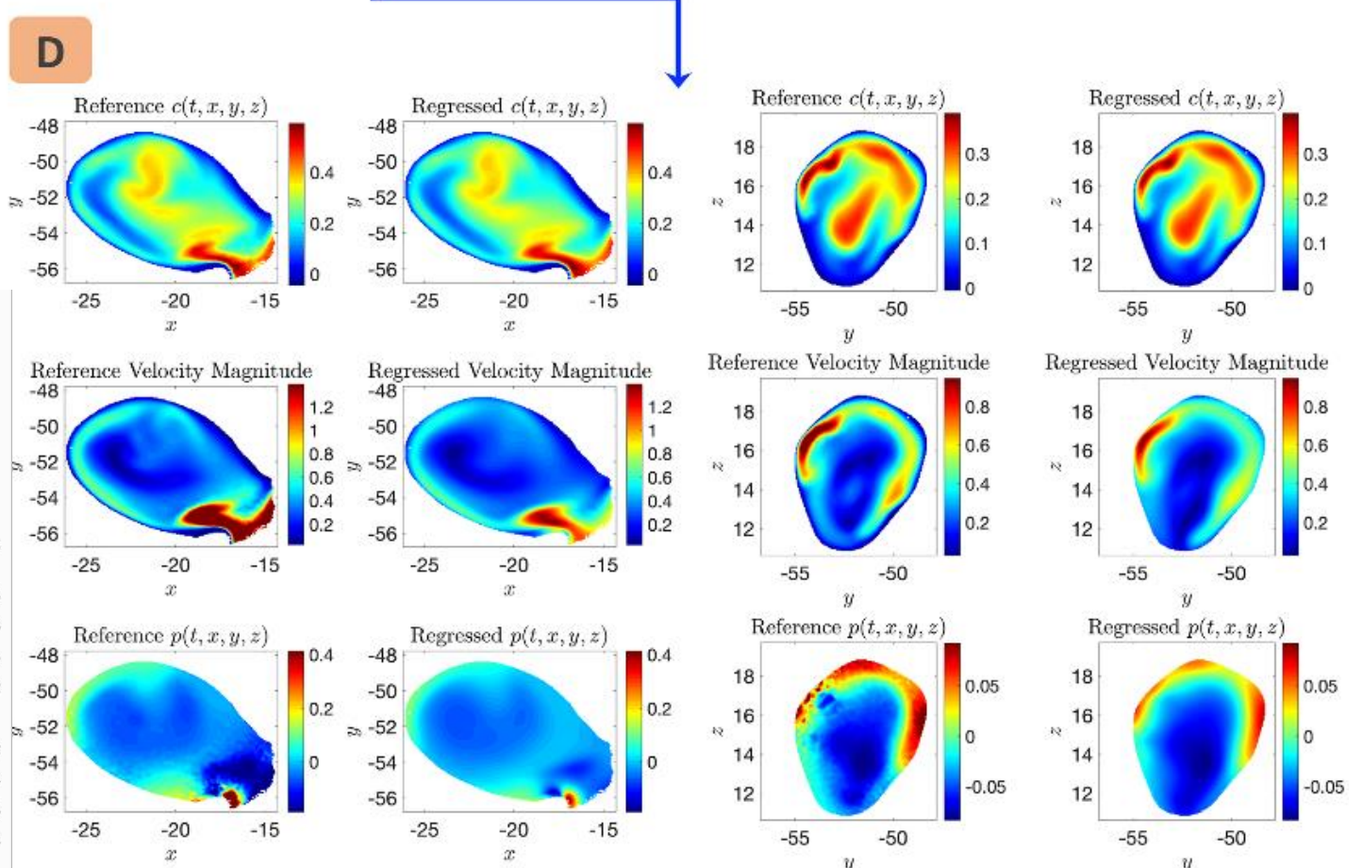
Figure 1: *Burgers' equation: Top:* Predicted solution $u(t,x)$ along with the initial and boundary training data. In addition we are using 10,000 collocation points generated using a Latin Hypercube Sampling strategy. *Bottom:* Comparison of the predicted and exact solutions corresponding to the three temporal snapshots depicted by the white vertical lines in the top panel. The relative $\mathcal{L}_2$ error for this case is $6.7 \cdot 10^{-4}$. Model training took approximately 60 seconds on a single NVIDIA Titan X GPU card.

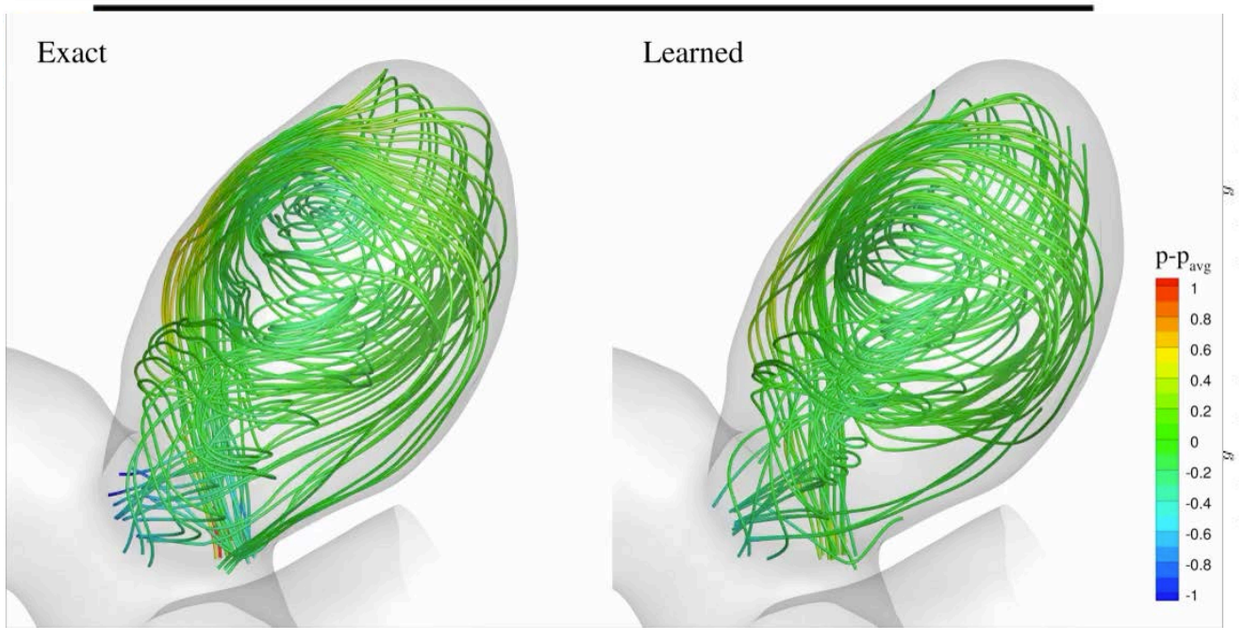
Physics-informed neural networks











Raissi, M., Yazdani, A., & Karniadakis, G. E. (2020). Hidden fluid mechanics: Learning velocity and pressure fields from flow visualizations. Science.

Physics as a prior in deep learning

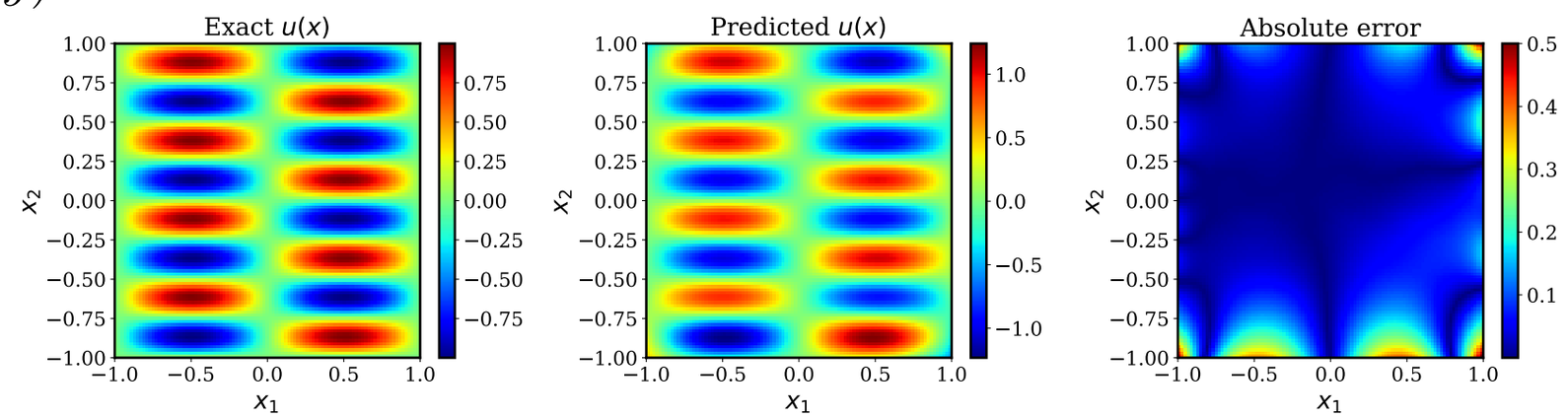
$$\mathcal{L}(\theta) := \underbrace{\frac{1}{N_u}\sum_{i=1}^{N_u}[\boldsymbol{u}_i - f_\theta(\boldsymbol{x}_i)]^2}_{\text{Data fit}} + \underbrace{\frac{1}{\lambda}\mathcal{R}[f_\theta(\boldsymbol{x})]}_{\text{Physics regularization}}$$

Recent results showcase remarkable promise, but failure looms even for the simplest problems…

Example: Helmholtz equation in 2D.

$$\Delta u(x,y) + k^2 u(x,y) = q(x,y), \quad (x,y) \in \Omega := (-1,1)$$
$$u(x,y) = h(x,y), \quad (x,y) \in \partial\Omega$$

where Δ is the Laplace operator.



20% error in the prediction of a fully-connected, 4-layer deep PINN model

An "unconventional" regularizer/prior that requires us to revisit standard deep learning practices:

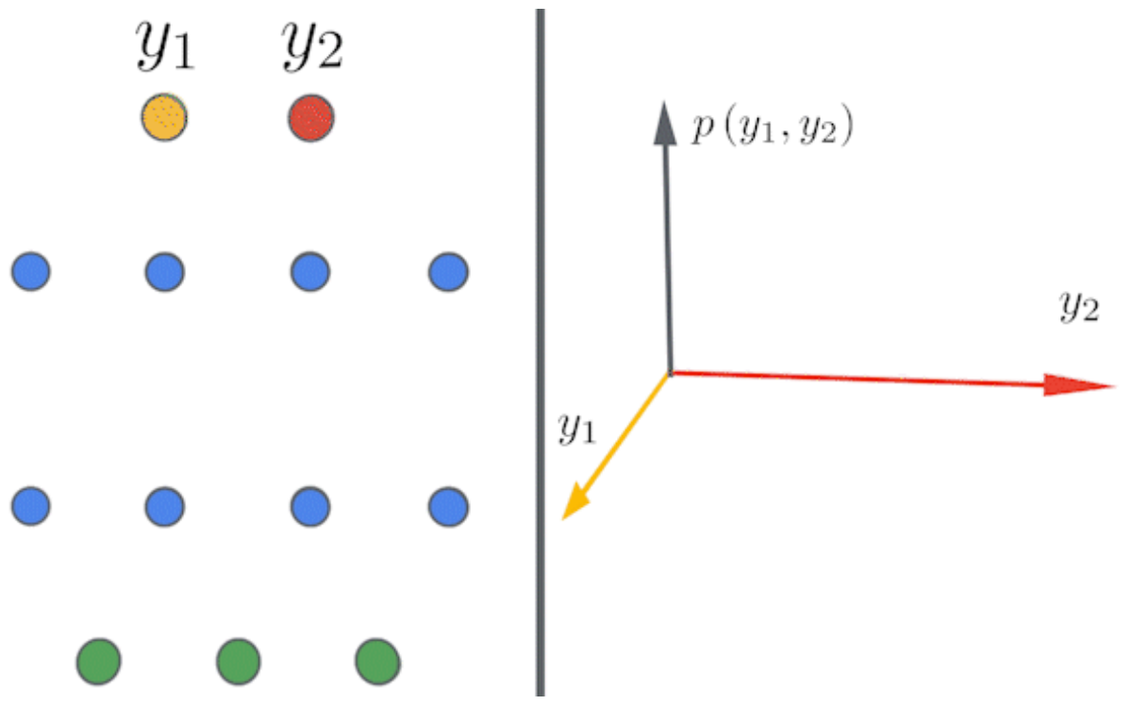
- loss function
- network initialization
- data normalization
- optimization
- network architecture

Despite some empirical success, we are still lacking a rigorous mathematical understanding on when, why and how physics-informed constraints can be effective in regularizing deep learning models.

Understanding the training dynamics of PINNs

Key concepts:

- Explore the connection between deep neural networks and kernel regression methods to elucidate the training dynamics of deep learning models.
- At initialization, fully-connected networks are equivalent to Gaussian processes in the infinite-width limit.
- The evolution of an infinite-width network during training can also be described by another kernel, the so-called Neural Tangent Kernel (NTK).



Lee, J., Bahri, Y., Novak, R., Schoenholz, S. S., Pennington, J., & Sohl-Dickstein, J. (2017). Deep neural networks as Gaussian processes.

Jacot, A., Gabriel, F., & Hongler, C. (2018). Neural tangent kernel: Convergence and generalization in neural networks. In Advances in neural information processing systems (pp. 8571-8580).

PINNs converge to GPs at the infinite-width limit for linear PDEs

Model Problem: 1D Poisson's equation

$$u_{xx}(x) = f(x), \quad \forall x \in [0,1]$$
$$u(x) = g(x), \quad x = 0, 1.$$

We proceed by approximating the solution $u(x)$ by a fully-connected neural network denoted by $u(x, \boldsymbol{\theta})$ with one hidden layer:

$$u(x, \boldsymbol{\theta}) = \frac{1}{\sqrt{N}} \boldsymbol{W}^{(1)} \cdot \sigma(\boldsymbol{W}^{(0)} x + \boldsymbol{b}^{(0)}) + \boldsymbol{b}^{(1)}$$

Then

$$u_{xx}(x, \boldsymbol{\theta}) = \frac{1}{\sqrt{N}} \boldsymbol{W}^{(1)} \cdot \left[\ddot{\sigma}(\boldsymbol{W}^{(0)} x + \boldsymbol{b}^{(0)}) \odot \boldsymbol{W}^{(0)} \odot \boldsymbol{W}^{(0)} \right]$$

Theorem: Assume that the activation function σ is smooth and has a bounded second order derivative $\ddot{\sigma}$. Then for a fully-connected neural network of one hidden layer at initialization,

$$u(x, \boldsymbol{\theta}) \xrightarrow{\mathcal{D}} \mathcal{GP}(0, \Sigma^{(1)}(x, x'))$$
$$u_{xx}(x, \boldsymbol{\theta}) \xrightarrow{\mathcal{D}} \mathcal{GP}(0, \Sigma^{(1)}_{xx}(x, x')),$$

as $N \to \infty$, where $\mathcal{D}$ means convergence in distribution and

$$\Sigma^{(1)}_{xx}(x, x') = \mathop{\mathbb{E}}_{u,v \sim \mathcal{N}(0,1)} \left[u^4 \ddot{\sigma}(ux + v) \ddot{\sigma}(ux' + v) \right].$$

- *By induction this result can be generalized to arbitrary deep fully-connected architectures and any linear PDE operator.*
- *Opens up a path for studying approximation accuracy and convergence rate of PINNs in the associated RKHS.*

The Neural Tangent Kernel (NTK) of PINNs

Consider the gradient flow:

$$\frac{d\boldsymbol{\theta}}{dt} = -\nabla \mathcal{L}(\boldsymbol{\theta}),$$

Lemma: Given the data points $\{\boldsymbol{x}_b^i, g(\boldsymbol{x}_b^i)\}_{i=1}^{N_b}, \{\boldsymbol{x}_r^i, f(\boldsymbol{x}_r^i)\}_{i=1}^{N_r}$, we have

$$\begin{bmatrix} \frac{du(\boldsymbol{x}_b, \boldsymbol{\theta}(t))}{dt} \\ \frac{d\mathcal{L}u(\boldsymbol{x}_r, \boldsymbol{\theta}(t))}{dt} \end{bmatrix} = - \begin{bmatrix} \boldsymbol{K}_{uu}(t) & \boldsymbol{K}_{ur}(t) \\ \boldsymbol{K}_{ru}(t) & \boldsymbol{K}_{rr}(t) \end{bmatrix} \cdot \begin{bmatrix} u(\boldsymbol{x}_b, \boldsymbol{\theta}(t)) - g(\boldsymbol{x}_b) \\ \mathcal{L}u(\boldsymbol{x}_r, \boldsymbol{\theta}(t)) - f(\boldsymbol{x}_r) \end{bmatrix},$$

where $\boldsymbol{K}_{ru}(t) = \boldsymbol{K}_{ur}^T(t)$ and $\boldsymbol{K}_{uu}(t) \in \mathbb{R}^{N_b \times N_b}$, $\boldsymbol{K}_{ur}(t) \in \mathbb{R}^{N_b \times N_r}$, and $\boldsymbol{K}_{rr}(t) \in \mathbb{R}^{N_r \times N_r}$ whose (i,j)-th entry is given by

$$(\boldsymbol{K}_{uu})_{ij}(t) = \left\langle \frac{du(\boldsymbol{x}_b^i, \boldsymbol{\theta}(t))}{d\boldsymbol{\theta}}, \frac{du(\boldsymbol{x}_b^j, \boldsymbol{\theta}(t))}{d\boldsymbol{\theta}} \right\rangle$$

$$(\boldsymbol{K}_{ur})_{ij}(t) = \left\langle \frac{du(\boldsymbol{x}_b^i, \boldsymbol{\theta}(t))}{d\boldsymbol{\theta}}, \frac{d\mathcal{L}u(\boldsymbol{x}_r^j, \boldsymbol{\theta}(t))}{d\boldsymbol{\theta}} \right\rangle$$

$$(\boldsymbol{K}_{rr})_{ij}(t) = \left\langle \frac{d\mathcal{L}u(\boldsymbol{x}_r^i, \boldsymbol{\theta}(t))}{d\boldsymbol{\theta}}, \frac{d\mathcal{L}u(\boldsymbol{x}_r^j, \boldsymbol{\theta}(t))}{d\boldsymbol{\theta}} \right\rangle$$

**this ODE system holds for any differential operator and any neural network architecture.*

The NTK of PINNs and its evolution under gradient descent

Model Problem: 1D Poisson's equation

$$u_{xx}(x) = f(x), \quad \forall x \in [0,1]$$
$$u(x) = g(x), \quad x = 0, 1.$$

Recall

$$\begin{bmatrix} \frac{du(\boldsymbol{x}_b, \boldsymbol{\theta}(t))}{dt} \\ \frac{du_{xx}(\boldsymbol{x}_r, \boldsymbol{\theta}(t))}{dt} \end{bmatrix} = - \begin{bmatrix} \boldsymbol{K}_{uu}(t) & \boldsymbol{K}_{ur}(t) \\ \boldsymbol{K}_{ru}(t) & \boldsymbol{K}_{rr}(t) \end{bmatrix} \cdot \begin{bmatrix} u(\boldsymbol{x}_b, \boldsymbol{\theta}(t)) - g(\boldsymbol{x}_b) \\ u_{xx}(\boldsymbol{x}_r, \boldsymbol{\theta}(t)) - f(\boldsymbol{x}_r) \end{bmatrix},$$

Theorem: For a physics-informed network with one hidden layer at initialization, and in the limit as the layer's width $N \to \infty$, the NTK $\boldsymbol{K}(t)$ of the PINNs model defined in equation above converges in probability to a deterministic limiting kernel, i.e,

$$\boldsymbol{K}(0) = \begin{bmatrix} \boldsymbol{K}_{uu}(0) & \boldsymbol{K}_{ur}(0) \\ \boldsymbol{K}_{ru}(0) & \boldsymbol{K}_{rr}(0) \end{bmatrix} \to \begin{bmatrix} \Theta_{uu}^{(1)} & \Theta_{ur}^{(1)} \\ \Theta_{ru}^{(1)} & \Theta_{rr}^{(1)} \end{bmatrix} := \boldsymbol{K}^*$$

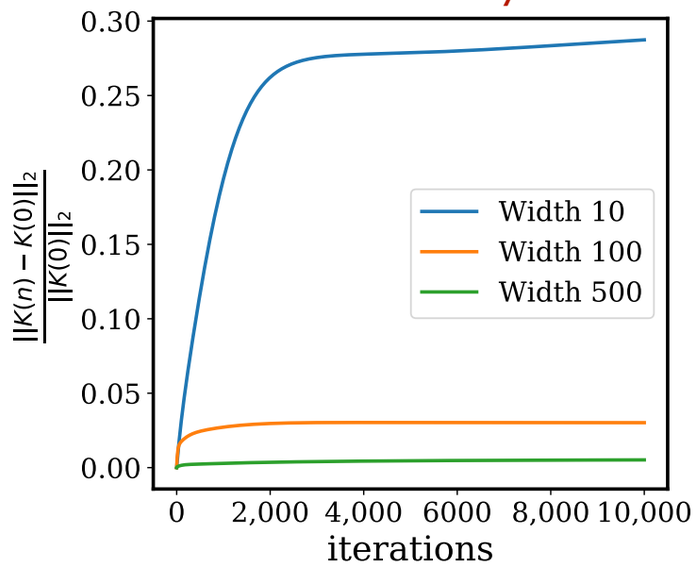
For any $T > 0$ satisfying some suitable assumptions, we have

$$\lim_{N \to \infty} \sup_{t \in [0,T]} \|\boldsymbol{K}(t) - \boldsymbol{K}(0)\|_2 = 0$$

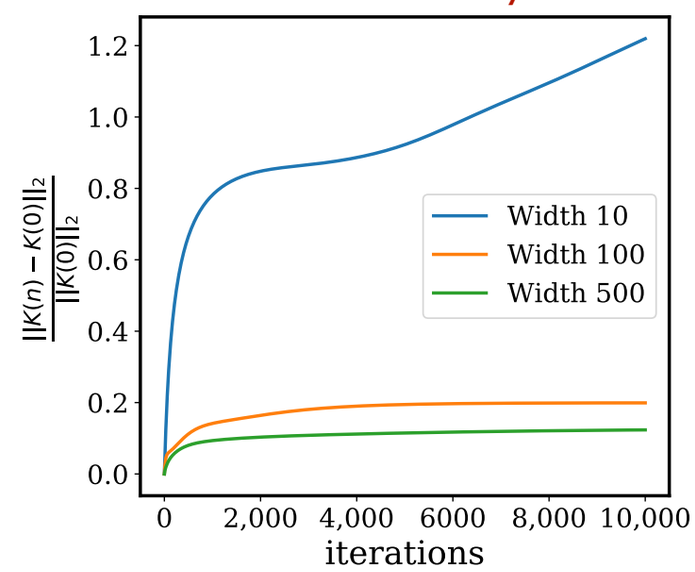
We conclude that, for sufficient wide fully-connected neural network of one hidden layer at any random initialization, the NTK of PINNs satisfies

$$\boldsymbol{K}(t) \approx \boldsymbol{K}(0) \approx \boldsymbol{K}^*, \quad \forall t \in [0,T]$$

One hidden layer



Three hidden layers



**in practice the NTK quickly converges to a deterministic kernel that remains constant during training for any neural architecture and any PDE operator.*

PINNs is equivalent to kernel regression

Model Problem: 1D Poisson's equation

$$u_{xx}(x) = f(x), \quad \forall x \in [0,1]$$
$$u(x) = g(x), \quad x = 0, 1.$$

$$\begin{bmatrix} \frac{du(\boldsymbol{x}_b, \boldsymbol{\theta}(t))}{dt} \\ \frac{du_{xx}(\boldsymbol{x}_r, \boldsymbol{\theta}(t))}{dt} \end{bmatrix} = -\boldsymbol{K}(t) \cdot \begin{bmatrix} u(\boldsymbol{x}_b, \boldsymbol{\theta}(t)) - g(\boldsymbol{x}_b) \\ u_{xx}(\boldsymbol{x}_r, \boldsymbol{\theta}(t)) - f(\boldsymbol{x}_r) \end{bmatrix}$$

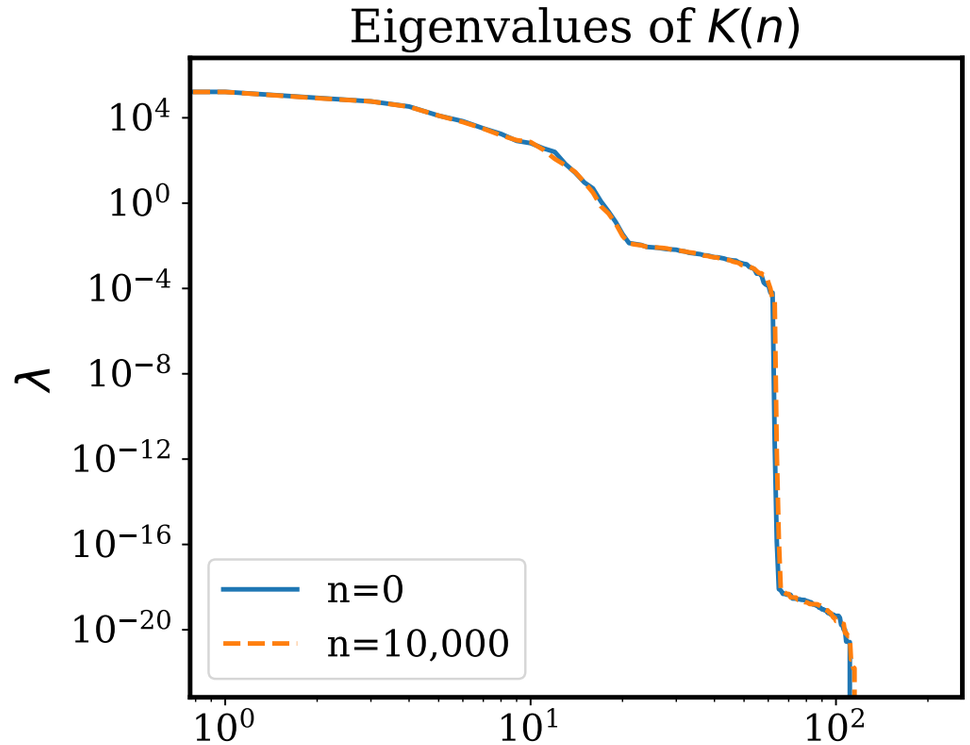
$$\approx -\boldsymbol{K}^* \cdot \begin{bmatrix} u(\boldsymbol{x}_b, \boldsymbol{\theta}(t)) - g(\boldsymbol{x}_b) \\ u_{xx}(\boldsymbol{x}_r, \boldsymbol{\theta}(t)) - f(\boldsymbol{x}_r). \end{bmatrix}$$

Assumption: If $\boldsymbol{K}^*$ is invertible, then

$$\begin{bmatrix} u(\boldsymbol{x}_{test}, \boldsymbol{\theta}(t)) \\ u_{xx}(\boldsymbol{x}_{test}, \boldsymbol{\theta}(t)) \end{bmatrix} \approx \boldsymbol{K}^*_{test} (\boldsymbol{K}^*)^{-1} \left(I - e^{-\boldsymbol{K}^* t} \right) \cdot \begin{bmatrix} g(\boldsymbol{x}_b) \\ f(\boldsymbol{x}_r) \end{bmatrix},$$

Letting $t \to \infty$ gives

$$\begin{bmatrix} u(\boldsymbol{x}_{test}, \boldsymbol{\theta}(\infty)) \\ u_{xx}(\boldsymbol{x}_{test}, \boldsymbol{\theta}(\infty)) \end{bmatrix} \approx \boldsymbol{K}^*_{test} (\boldsymbol{K}^*)^{-1} \cdot \begin{bmatrix} g(\boldsymbol{x}_b) \\ f(\boldsymbol{x}_r) \end{bmatrix}.$$



NTK spectrum at initialization, and after 10,000 gradient descent steps

- *From our experience, the NTK of PINNs is always degenerate, hence we may not be able to casually perform kernel regression predictions in practice.*
- *Additional regularization may help in obtaining an invertible NTK.*

Spectral bias in PINNs

Since the NTK of PINNs barely changes during training, we may rewrite

$$\begin{bmatrix} \frac{du(\boldsymbol{x}_b,\boldsymbol{\theta}(t))}{dt} \\ \frac{du_{xx}(\boldsymbol{x}_r,\boldsymbol{\theta}(t))}{dt} \end{bmatrix} \approx -\boldsymbol{K}(0) \begin{bmatrix} u(\boldsymbol{x}_b,\boldsymbol{\theta}(t)) - g(\boldsymbol{x}_b) \\ u_{xx}(\boldsymbol{x}_r,\boldsymbol{\theta}(t)) - f(\boldsymbol{x}_r) \end{bmatrix},$$

which leads to

$$\begin{bmatrix} \frac{du(\boldsymbol{x}_b,\boldsymbol{\theta}(t))}{dt} \\ \frac{du_{xx}(\boldsymbol{x}_r,\boldsymbol{\theta}(t))}{dt} \end{bmatrix} \approx \left(I - e^{-\boldsymbol{K}(0)t}\right) \cdot \begin{bmatrix} g(\boldsymbol{x}_b) \\ f(\boldsymbol{x}_r) \end{bmatrix}$$

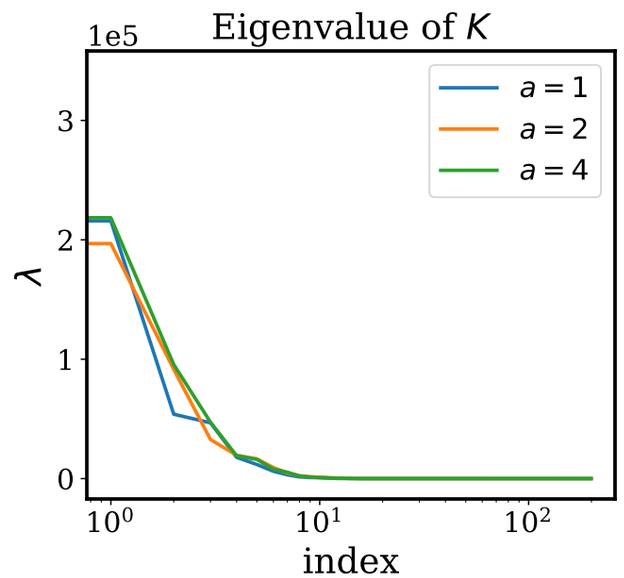
Recall $\boldsymbol{K}(0)$ is positive semi-definite, so $\boldsymbol{K}(0) = \boldsymbol{Q}^T\boldsymbol{\Lambda}\boldsymbol{Q}$, where $\boldsymbol{Q}$ is an orthogonal matrix and $\boldsymbol{\Lambda}$ is a diagonal matrix whose entries are the eigenvalues $\lambda_i \geq 0$ of $\boldsymbol{K}(0)$. Consequently,

$$\begin{aligned} \begin{bmatrix} \frac{du(\boldsymbol{x}_b,\boldsymbol{\theta}(t))}{dt} \\ \frac{du_{xx}(\boldsymbol{x}_r,\boldsymbol{\theta}(t))}{dt} \end{bmatrix} - \begin{bmatrix} g(\boldsymbol{x}_b) \\ f(\boldsymbol{x}_r) \end{bmatrix} &\approx \left(I - e^{-\boldsymbol{K}(0)t}\right) \cdot \begin{bmatrix} g(\boldsymbol{x}_b) \\ f(\boldsymbol{x}_r) \end{bmatrix} - \begin{bmatrix} g(\boldsymbol{x}_b) \\ f(\boldsymbol{x}_r) \end{bmatrix} \\ &\approx -\boldsymbol{Q}^T e^{-\boldsymbol{\Lambda}t}\boldsymbol{Q} \cdot \begin{bmatrix} g(\boldsymbol{x}_b) \\ f(\boldsymbol{x}_r) \end{bmatrix}, \end{aligned}$$

which is equivalent to

$$\boldsymbol{Q}\left(\begin{bmatrix} \frac{du(\boldsymbol{x}_b,\boldsymbol{\theta}(t))}{dt} \\ \frac{du_{xx}(\boldsymbol{x}_r,\boldsymbol{\theta}(t))}{dt} \end{bmatrix} - \begin{bmatrix} g(\boldsymbol{x}_b) \\ f(\boldsymbol{x}_r) \end{bmatrix}\right) \approx -e^{-\boldsymbol{\Lambda}t}\boldsymbol{Q} \cdot \begin{bmatrix} g(\boldsymbol{x}_b) \\ f(\boldsymbol{x}_r) \end{bmatrix}$$

- *The eigenvalues of the NTK characterize how fast the absolute training error decreases.*
- *Components of the target function that correspond to NTK eigenvectors with larger eigenvalues will be learned faster.*
- *For fully-connected networks, the eigenvectors corresponding to higher eigenvalues of the NTK matrix generally exhibit lower frequencies.*
- *In practice, we observe that the eigenvalues of the NTK of PINNs decay rapidly. This results in extremely slow convergence to the high-frequency components of the target function.*



Discrepancy of convergence rate in PINNs loss functions

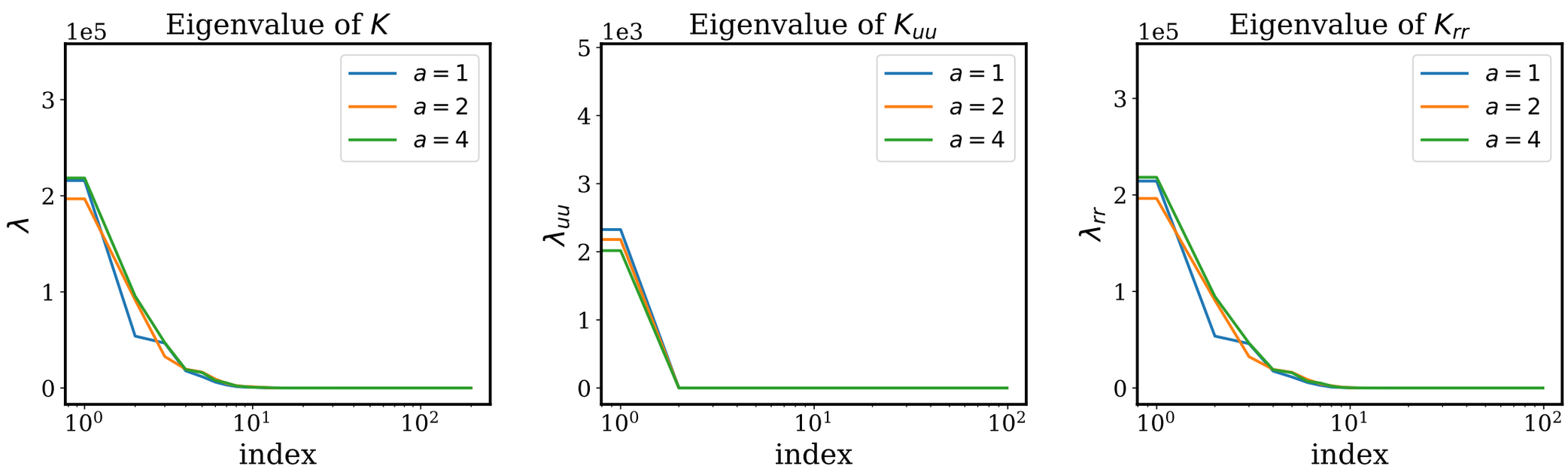
Definition 5.1. *For a positive semi-definite kernel matrix $\boldsymbol{K} \in \mathbb{R}^{n\times n}$, the average convergence rate c is defined as the mean of all its eigenvalues λ_i's, i.e.*

$$c = \frac{\sum_{i=1}^{n} \lambda_i}{n} = \frac{Tr(K)}{n}. \tag{5.4}$$

In particular, for any two kernel matrices $\boldsymbol{K}_1, \boldsymbol{K}_2$ with average convergence rate c_1 and c_2 respectively, we say that $\boldsymbol{K}_1$ dominates $\boldsymbol{K}_2$ if $c_1 \gg c_2$.

Beyond the spectral bias of fully-connected architectures, PINNs also suffer from a remarkable discrepancy of convergence rate in the different components of their loss function:

$$\mathcal{L}(\theta) := \underbrace{\mathcal{L}_u(\theta)}_{\text{Data fit}} + \underbrace{\mathcal{L}_r(\theta)}_{\text{PDE residual}} + \underbrace{\mathcal{L}_{u_0}(\theta)}_{\text{ICs fit}} + \underbrace{\mathcal{L}_{u_b}(\theta)}_{\text{BCs fit}}$$



- *The eigenvalues of Krr are much greater than Kuu, namely Krr dominates Kuu by definition 5.1.*
- *As a consequence, the PDE residual converges much faster than fitting the PDE boundary conditions, which may prevent the network from approximating the correct solution*

From theory to practice: Adaptive training for PINNs

$$\mathcal{L}(\boldsymbol{\theta}) = \frac{\lambda_b}{2N_b}\sum_{i=1}^{N_b}|u(\boldsymbol{x}_b^i,\boldsymbol{\theta}) - g(\boldsymbol{x}_b^i)|^2 + \frac{\lambda_r}{2N_r}\sum_{i=1}^{N_r}|r(\boldsymbol{x}_r^i,\boldsymbol{\theta})|^2$$

Practical insight: Adjusting the learning-rate (or, equivalently, the batch-size) can resolve the discrepancy in convergence rate of the total training error.

Algorithm 1: Adaptive weights for physics-informed neural networks

Consider a physics-informed neural network $u(\boldsymbol{x},\boldsymbol{\theta})$ with parameters $\boldsymbol{\theta}$ and a loss function

$$\mathcal{L}(\boldsymbol{\theta}) := \lambda_b\mathcal{L}_b(\boldsymbol{\theta}) + \lambda_r\mathcal{L}_r(\boldsymbol{\theta}),$$

where $\mathcal{L}_r(\boldsymbol{\theta})$ denotes the PDE residual loss and $\mathcal{L}_r(\theta)$ corresponds to boundary conditions. $\lambda_b = \lambda_r = 1$ are free parameters used to overcome the discrepancy between $\boldsymbol{K}_{uu}$ and $\boldsymbol{K}_{rr}$. Then use S steps of a gradient descent algorithm to update the parameters $\boldsymbol{\theta}$ as:

for $n = 1,\ldots,S$ **do**

(a) Compute λ_b and λ_r by

$$\lambda_b = \frac{\sum_{i=1}^{N_r+N_b}\lambda_i(n)}{\sum_{i=1}^{N_b}\lambda_i^{uu}(n)} = \frac{Tr(\boldsymbol{K}(n))}{Tr(\boldsymbol{K}_{uu}(n))} \tag{6.5}$$

$$\lambda_r = \frac{\sum_{i=1}^{N_r+N_b}\lambda_i(n)}{\sum_{i=1}^{N_r}\lambda_i^{rr}(n)} = \frac{Tr(\boldsymbol{K}(n))}{Tr(\boldsymbol{K}_{rr}(n))} \tag{6.6}$$

where $\lambda_i(n)$, λ_i^{uu} and $\lambda_i^{rr}(n)$ are eigenvalues of $\boldsymbol{K}(n)$, $\boldsymbol{K}_{uu}(n)$, $\boldsymbol{K}_{rr}(n)$ at n-th iteration.

(b) Update the parameters θ via gradient descent

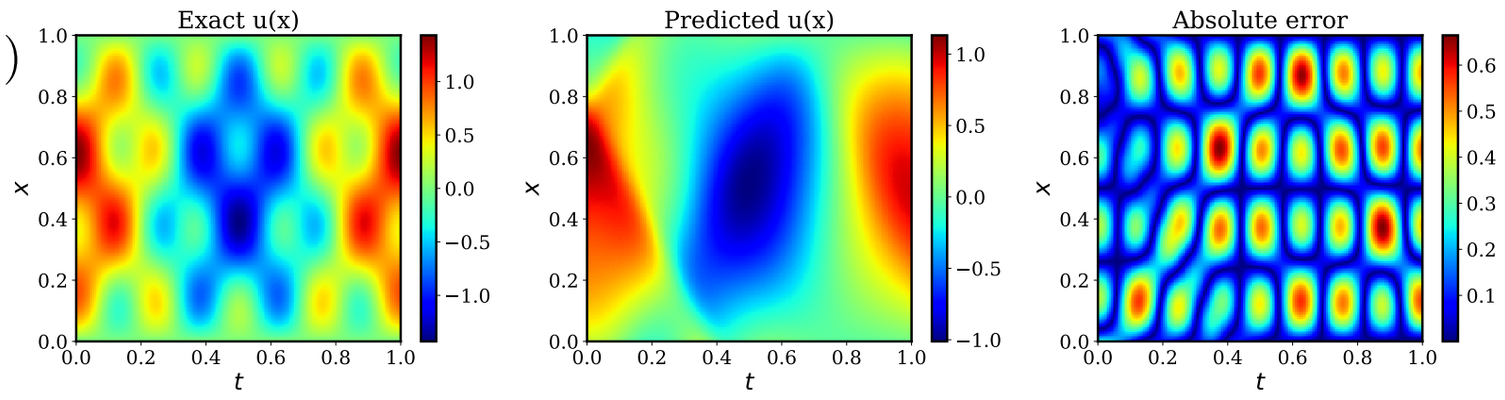
$$\boldsymbol{\theta}_{n+1} = \boldsymbol{\theta}_n - \eta\nabla_{\boldsymbol{\theta}}\mathcal{L}(\boldsymbol{\theta}_n) \tag{6.7}$$

end

Wang, S., Yu, X., & Perdikaris, P. (2020). When and why PINNs fail to train: A neural tangent kernel perspective.

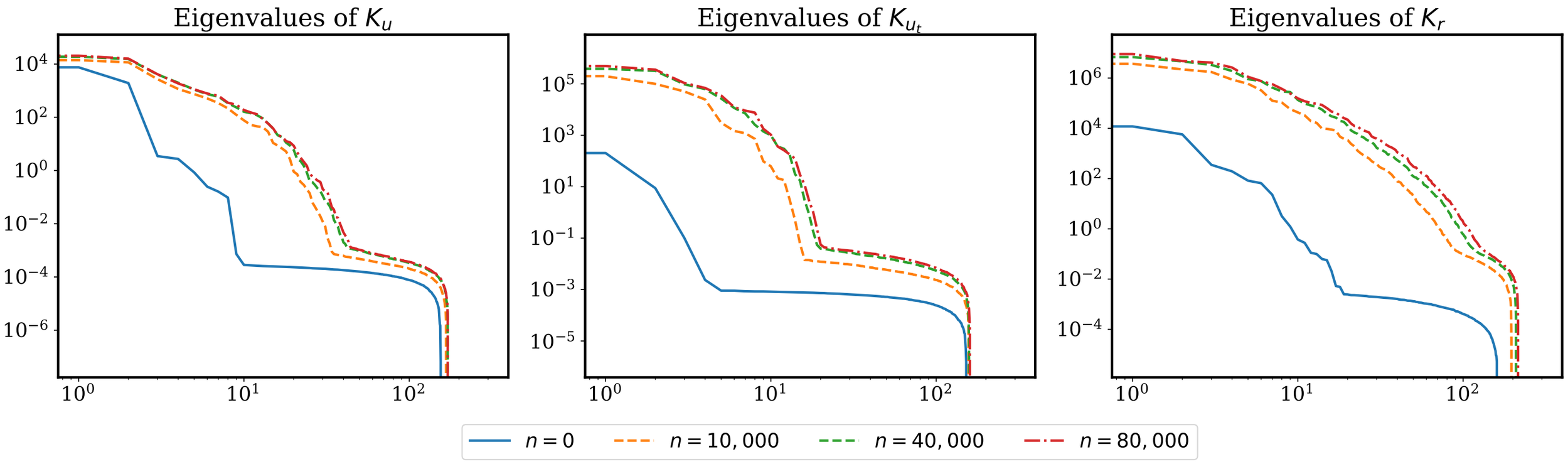
Numerical studies: Wave propagation

$$u_{tt}(x,t) - 4u_{xx}(x,t) = 0, \quad (x,t) \in (0,1) \times (0,1)$$
$$u(0,t) = u(1,t) = 0, \quad t \in [0,1]$$
$$u(x,0) = \sin(\pi x) + \frac{1}{2}\sin(4\pi x), \quad x \in [0,1]$$
$$u_t(x,0) = 0, \quad x \in [0,1].$$



$$\mathcal{L}(\boldsymbol{\theta}) = \frac{\lambda_u}{2N_u}\sum_{i=1}^{N_u}|u(\boldsymbol{x}_u^i,\boldsymbol{\theta}) - g(\boldsymbol{x}_u^i)|^2 + \frac{\lambda_{u_t}}{2N_{u_t}}\sum_{i=1}^{N_{u_t}}|u_t(\boldsymbol{x}_{u_t}^i,\boldsymbol{\theta})|^2 + \frac{\lambda_r}{2N_r}\sum_{i=1}^{N_r}|\mathcal{L}u(\boldsymbol{x}_r^i,\boldsymbol{\theta})|^2$$

To investigate the reason behind PINN's failure for this example, we compute its NTK and track it during training.



Numerical studies: Wave propagation

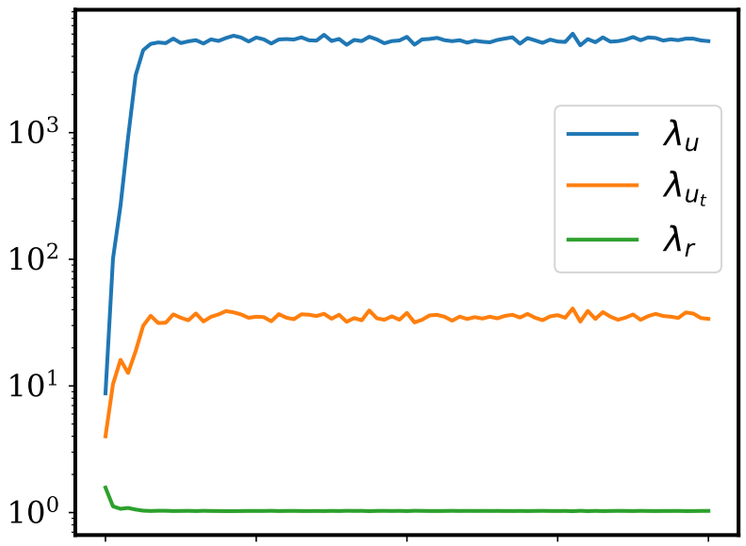
$$\mathcal{L}(\boldsymbol{\theta}) = \frac{\lambda_u}{2N_u}\sum_{i=1}^{N_u}|u(\boldsymbol{x}_u^i,\boldsymbol{\theta}) - g(\boldsymbol{x}_u^i)|^2 + \frac{\lambda_{u_t}}{2N_{u_t}}\sum_{i=1}^{N_{u_t}}|u_t(\boldsymbol{x}_{u_t}^i,\boldsymbol{\theta})|^2 + \frac{\lambda_r}{2N_r}\sum_{i=1}^{N_r}|\mathcal{L}u(\boldsymbol{x}_r^i,\boldsymbol{\theta})|^2$$

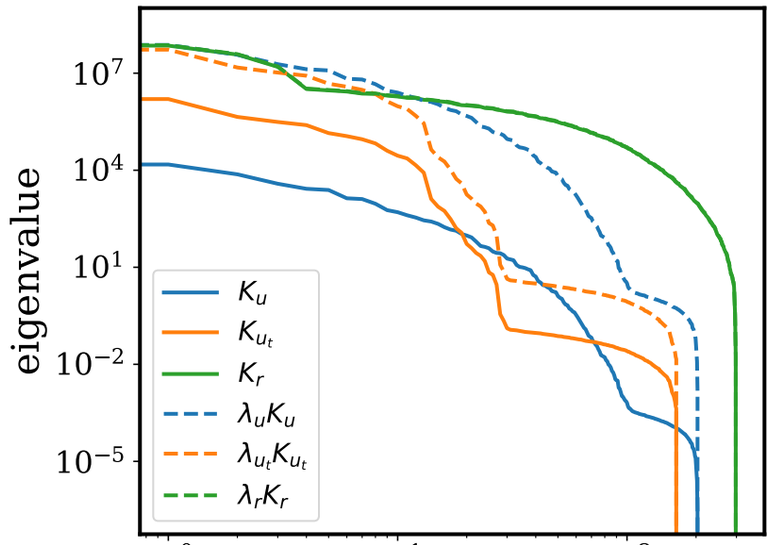
Adaptive weights:

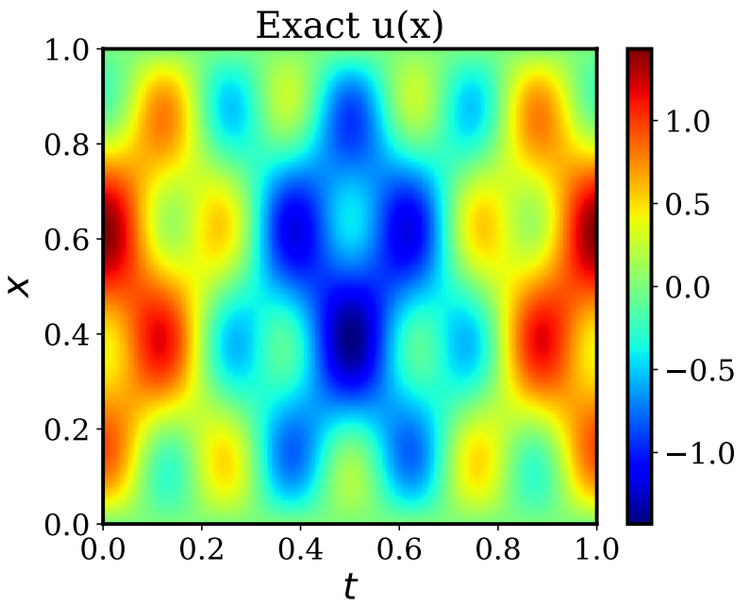
$$\lambda_u = \frac{Tr(\boldsymbol{K}_u) + Tr(\boldsymbol{K}_{u_t}) + Tr(\boldsymbol{K}_r)}{Tr(\boldsymbol{K}_u)}$$

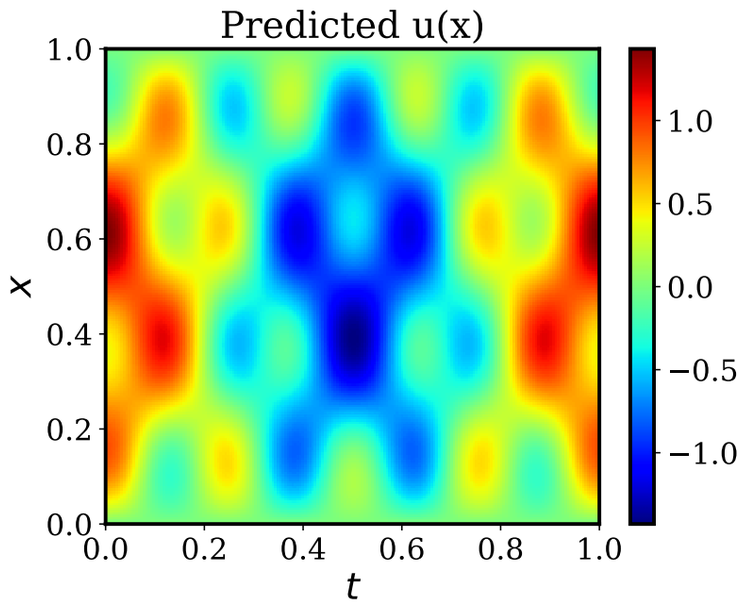
$$\lambda_{u_t} = \frac{Tr(\boldsymbol{K}_u) + Tr(\boldsymbol{K}_{u_t}) + Tr(\boldsymbol{K}_r)}{Tr(\boldsymbol{K}_{u_t})}$$

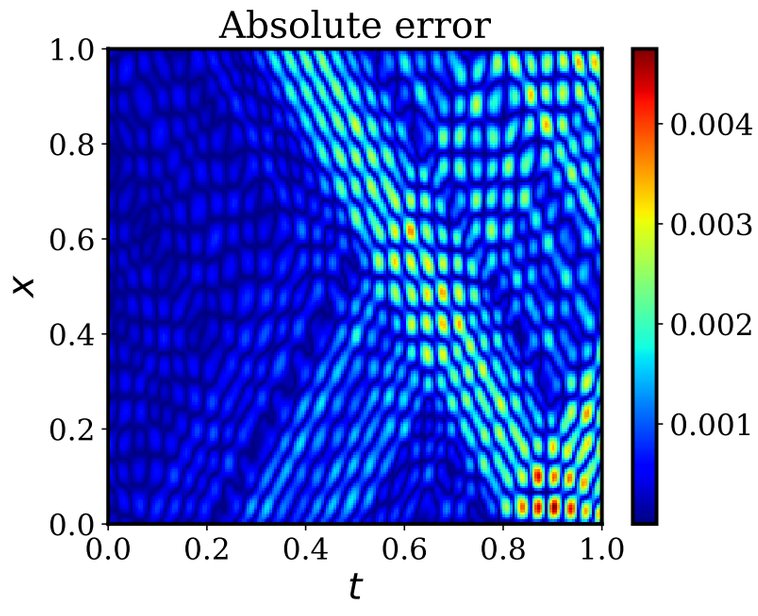
$$\lambda_r = \frac{Tr(\boldsymbol{K}_u) + Tr(\boldsymbol{K}_{u_t}) + Tr(\boldsymbol{K}_r)}{Tr(\boldsymbol{K}_r)}.$$











The fine prints…

- Fully-connected PINNs not only suffer from spectral bias, but also from a remarkable discrepancy of convergence rate in the different components of their loss function.

- This can be partly mitigated by adaptive training procedures.

- PINNs are currently far more effective for inverse (rather than forward) problems, indicating that the distribution of the observed data plays a profound role in their trainability.

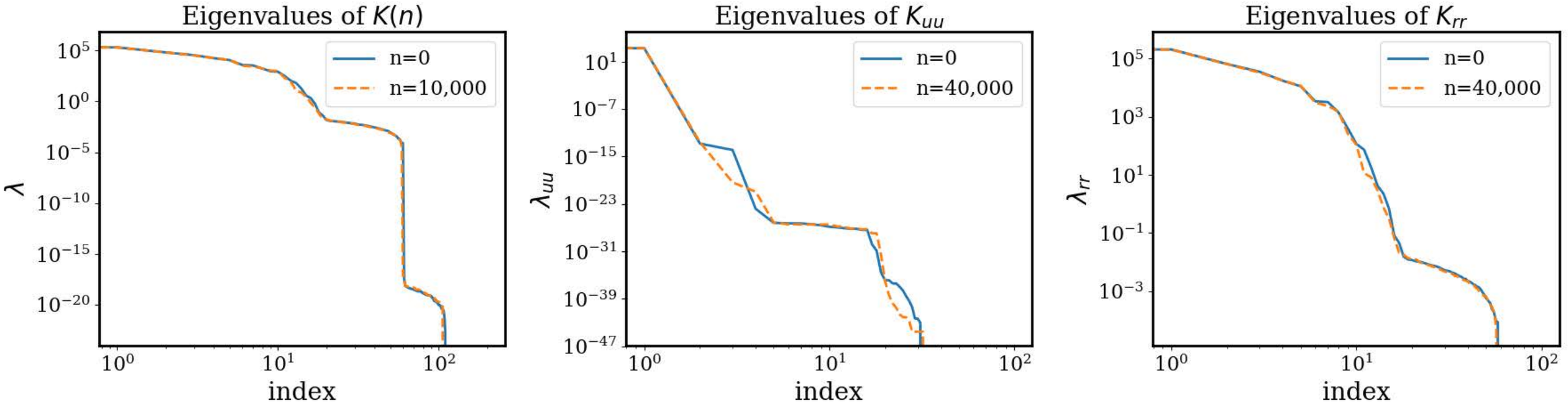
- The application of PINNs still remains challenging in problems involving coupled physics and multi-scale dynamics.

- Need for novel architectures that overcome spectral bias and training convergence pathologies.

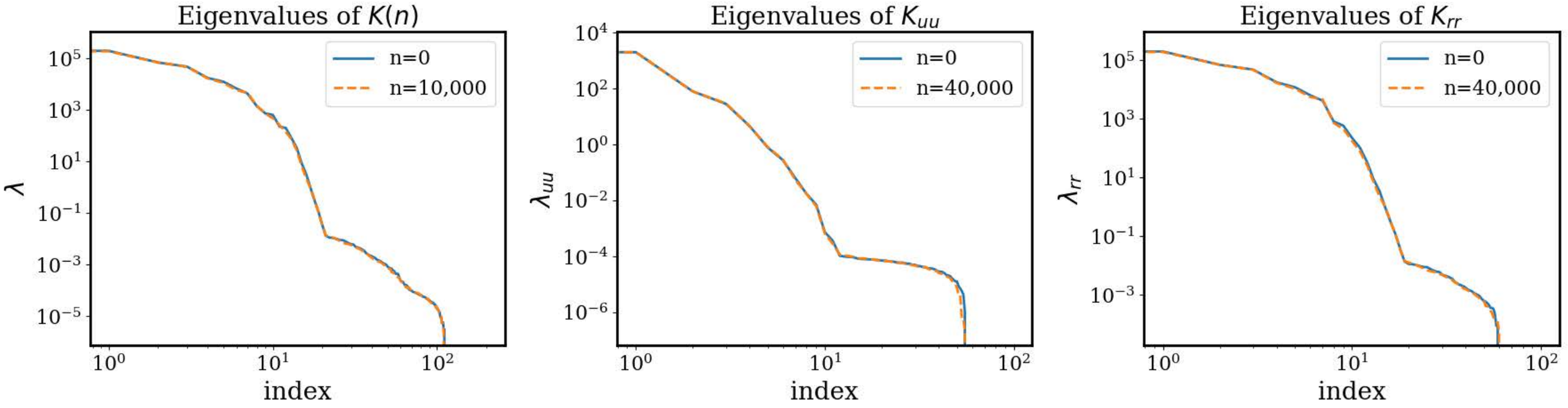
- Need for stable and effective optimization algorithms for PINNs, and multi-task learning in general.

- Predictive uncertainty estimates can be facilitated via an (expensive) Bayesian formulation, although averaging across an ensemble of neural nets still remains the most effective solution in practice.

Forward vs Inverse problems



Forward problem for 1D Poisson equation



Inverse problem for 1D Poisson equation

The data distribution plays a profound role in shaping the NTK spectrum, and, hence, the trainability of PINNs!

Basri, R., Galun, M., Geifman, A., Jacobs, D., Kasten, Y., & Kritchman, S. (2020). Frequency bias in neural networks for input of non-uniform density. arXiv preprint arXiv:2003.04560.

Stiffness in the gradient flow dynamics

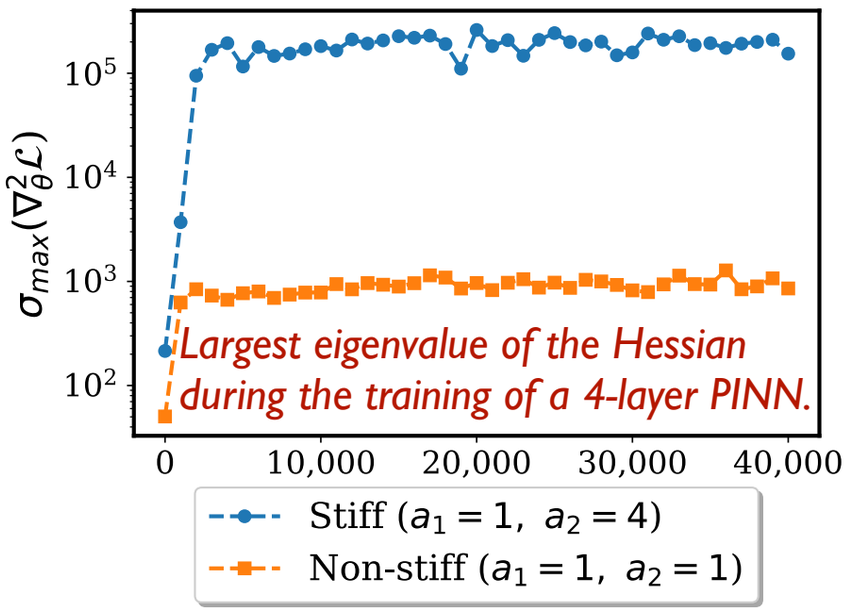
$$\frac{d\theta}{dt} = -\nabla_\theta \mathcal{L}_r(\theta) - \sum_{i=1}^{M} \nabla_\theta \mathcal{L}_i(\theta)$$

Stiffness:

- Stiffness of the gradient flow can be characterized by the largest eigenvalue of $\nabla^2_\theta \mathcal{L}(\theta)$.

Hypothesis:

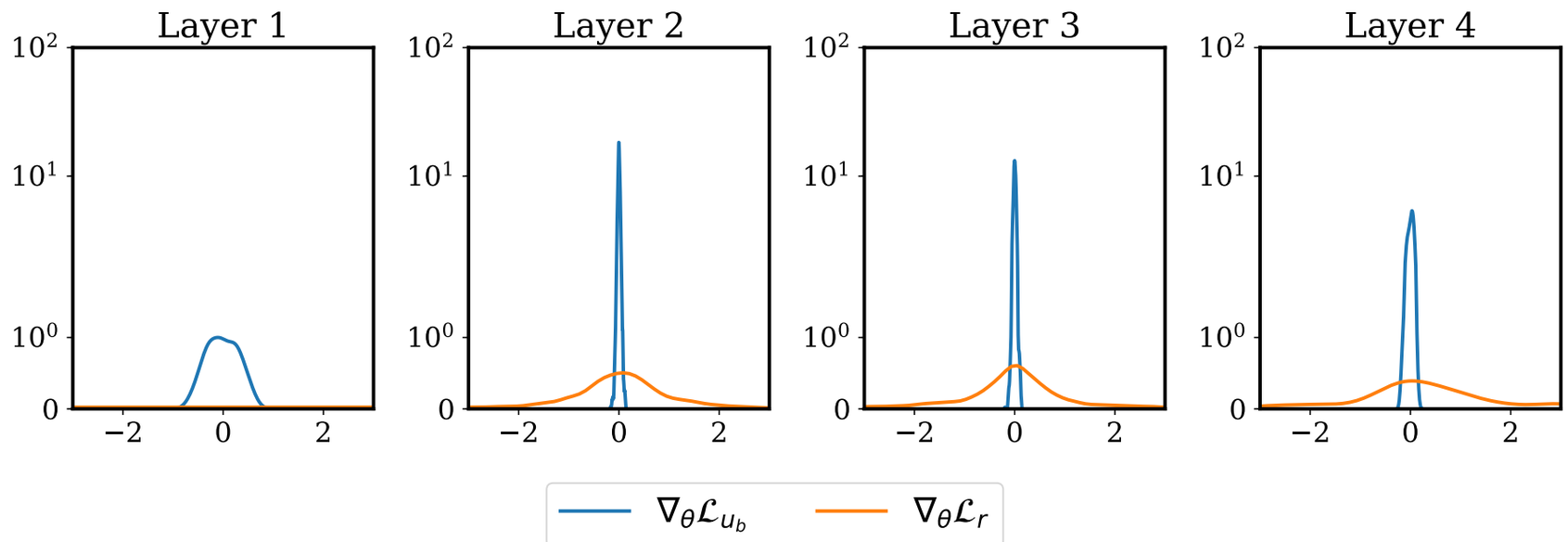
- The stiffness exists in the gradient flow dynamics of PINNs.
- The stiffness leads to imbalanced gradients during model training using gradient descent.
- The stiffness leads to difficulties in training PINNs via gradient descent $\theta_{n+1} = \theta_n - \eta \nabla_\theta \mathcal{L}(\theta_n)$.



Example:

Helmholtz equation in 2D

$$\Delta u(x,y) + k^2 u(x,y) = q(x,y),$$
$$u(x,y) = \sin(a_1 \pi x)\sin(a_2 \pi y)$$

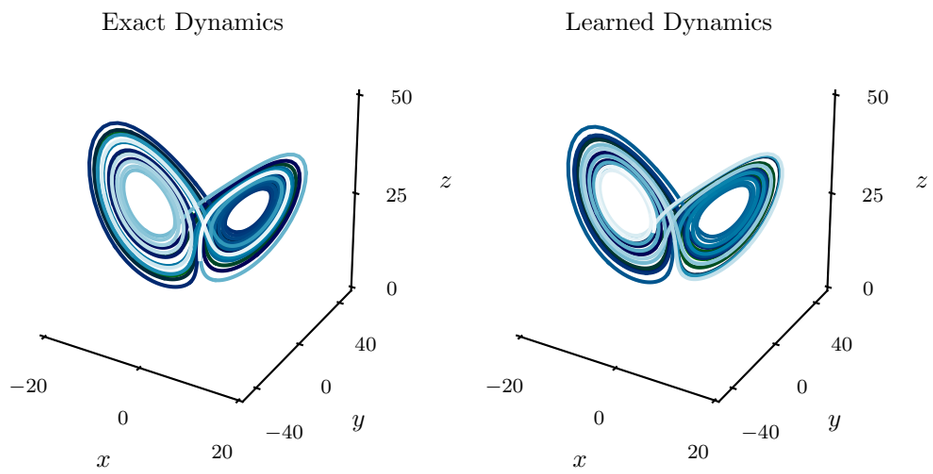


**training via gradient descent (aka Forward Euler) introduces severe restrictions on the learning rate ($\eta < 2/\lambda_{max}$).*

Wang, S., Teng, Y., & Perdikaris, P. (2020). Understanding and mitigating gradient pathologies in physics-informed neural networks.

Recent advances

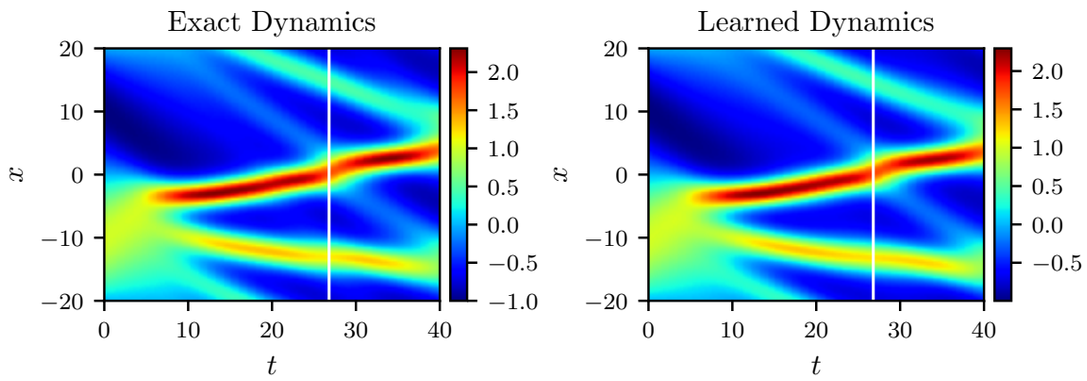
Discovery of ODEs



Raissi, M., Perdikaris, P., & Karniadakis, G. E. (2018). Multistep Neural Networks for Data-driven Discovery of Nonlinear Dynamical Systems. arXiv preprint

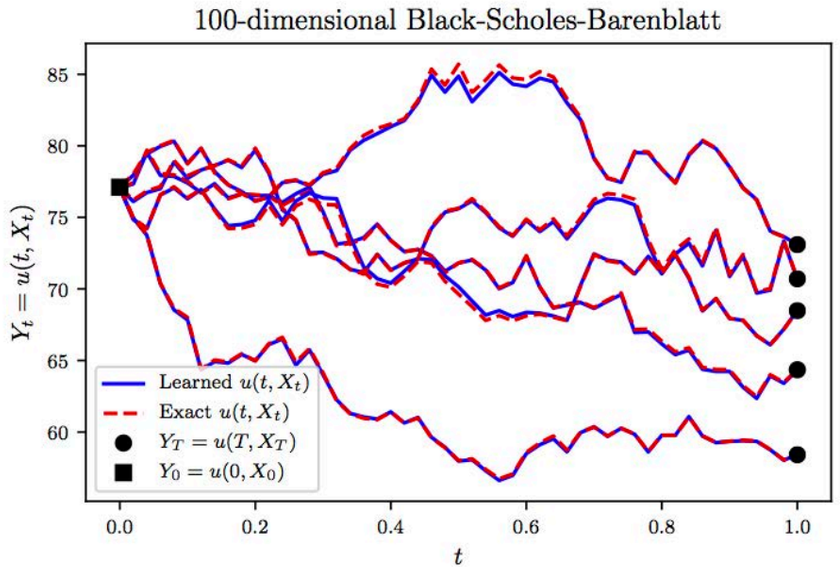
Discovery of PDEs

$$u_t = -uu_x - u_{xxx}$$



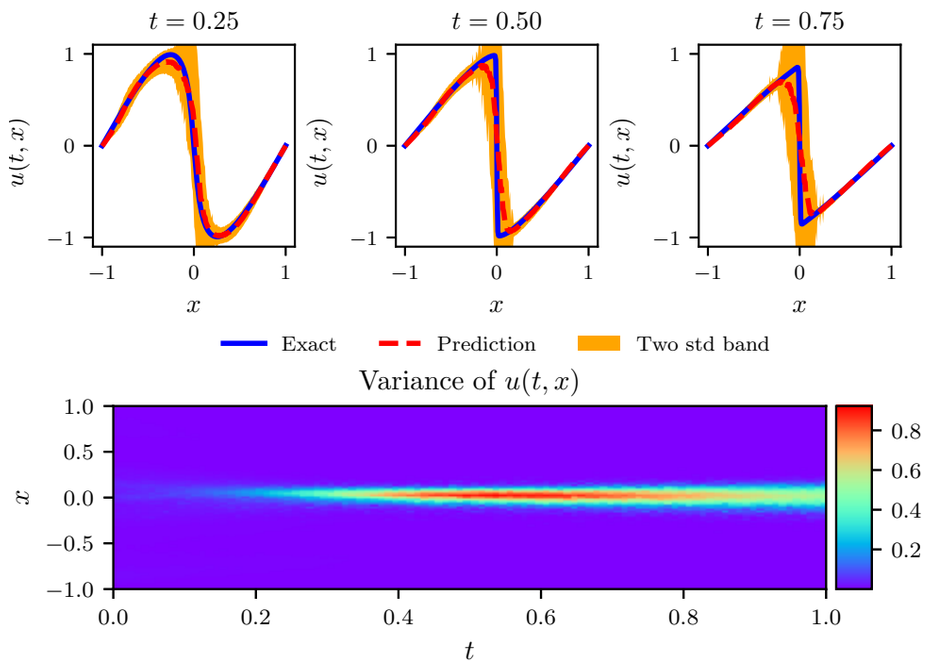
Raissi, M. (2018). Deep Hidden Physics Models: Deep Learning of Nonlinear Partial Differential Equations. arXiv preprint arXiv:1801.06637.

High-dimensional PDEs



Raissi, M. (2018). Forward-backward stochastic neural networks: Deep learning of high-dimensional partial differential equations. arXiv preprint arXiv: 1804.07010.

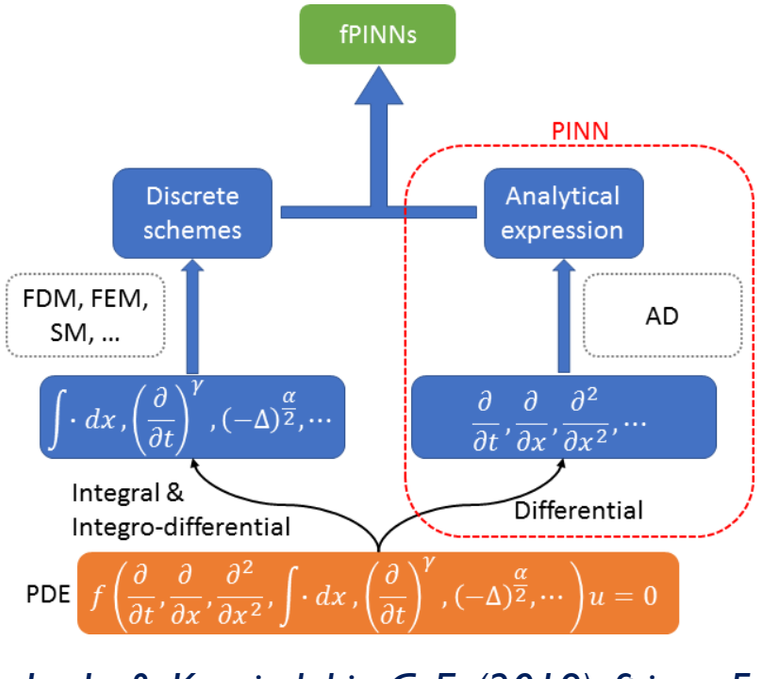
Stochastic PDEs



Yang, Y., & Perdikaris, P. (2019). Adversarial uncertainty quantification in physics-informed neural networks. Journal of Computational Physics.

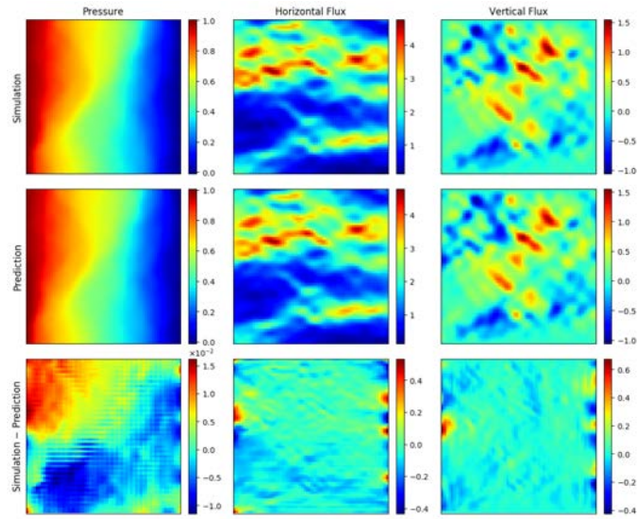
Recent advances

Fractional PDEs



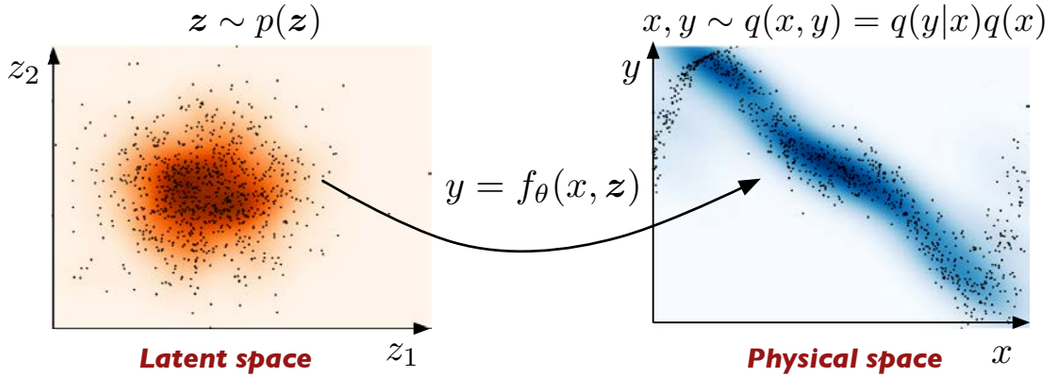
Pang, G., Lu, L., & Karniadakis, G. E. (2018). fpinns: Fractional physics-informed neural networks. arXiv preprint arXiv: 1811.08967.

Surrogate modeling & high-dimensional UQ



Zhu,Y., Zabaras, N., Koutsourelakis, P. S., & Perdikaris, P. (2019). Physics-constrained deep learning for high-dimensional surrogate modeling and uncertainty quantification without labeled data. Journal of Computational Physics, 394, 56-81.

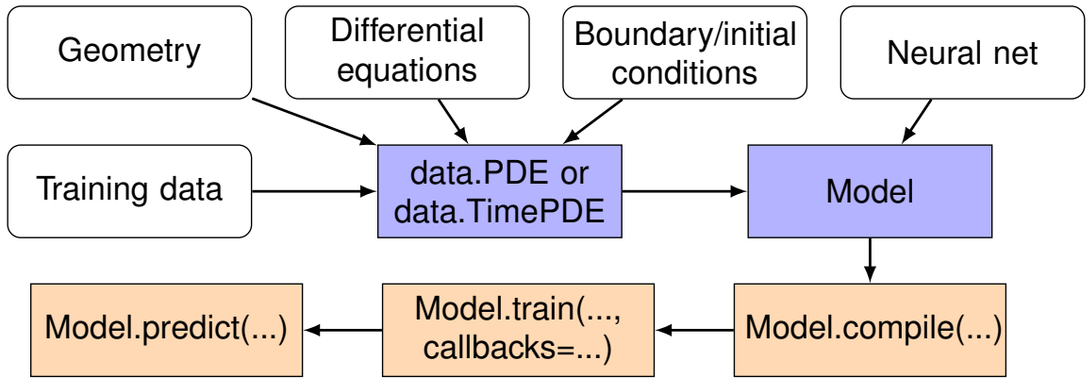
Multi-fidelity modeling for stochastic systems



$$y = f_\theta(x, \boldsymbol{z}),\ \boldsymbol{z} \sim p(\boldsymbol{z}) \ \Leftrightarrow\ y \sim p_\theta(y|x, \boldsymbol{z})$$

Conditional deep surrogate models for stochastic, high-dimensional, and multi-fidelity systems. Computational Mechanics, 1-18.

Integrated software



Lu, L., Meng, X., Mao, Z., & Karniadakis, G. E. (2019). DeepXDE: A deep learning library for solving differential equations. arXiv preprint arXiv:1907.04502.

Gradient analysis for PINNs

Example: Poisson equation in 1D

$$\Delta u(x) = g(x), \quad x \in [0,1]$$
$$u(x) = h(x), \quad x = 0 \text{ and } x = 1.$$

Let us consider exact solutions of the form $u(x) = \sin(Cx)$. Then we can use a deep neural network $f_\theta(x)$ to approximating $u(x)$.
The loss function is given by

$$\begin{aligned}\mathcal{L}(\theta) &= \mathcal{L}_r(\theta) + \mathcal{L}_{u_b}(\theta) \\ &= \frac{1}{N_b}\sum_{i=1}^{N_b}[f_\theta(x_b^i) - h(x_b^i)]^2 + \frac{1}{N_r}\sum_{i=1}^{N_r}[\frac{\partial^2}{\partial x^2} f_\theta(x_r^i) - g(x_r^i)]^2.\end{aligned}$$

Assumptions:

- $f_\theta(x) = u(x)\epsilon_\theta(x)$, where $\epsilon_\theta(x)$ is a smooth function defined in $[0,1]$.
- There exists $\epsilon > 0$ such that $|\epsilon_\theta(x) - 1| \leq \epsilon$ and $\|\frac{\partial^k \epsilon_\theta(x)}{\partial x^k}\|_{L^\infty} < \epsilon$, for all non-negative integer k.

We can show that

$$\|\nabla_\theta \mathcal{L}_{u_b}(\theta)\|_{L^\infty} \leq 2\epsilon \cdot \|\nabla_\theta \epsilon_\theta(x)\|_{L^\infty}$$
$$\|\nabla_\theta \mathcal{L}_r(\theta)\|_{L^\infty} \leq O(C^4) \cdot \epsilon \cdot \|\nabla_\theta \epsilon_\theta(x)\|_{L^\infty}$$

Wang, S., Teng, Y., & Perdikaris, P. (2020). Understanding and mitigating gradient pathologies in physics-informed neural networks. arXiv preprint arXiv:2001.04536.

Stiffness in the gradient flow dynamics

Example: Helmholtz equation in 2D, gradient descent update:

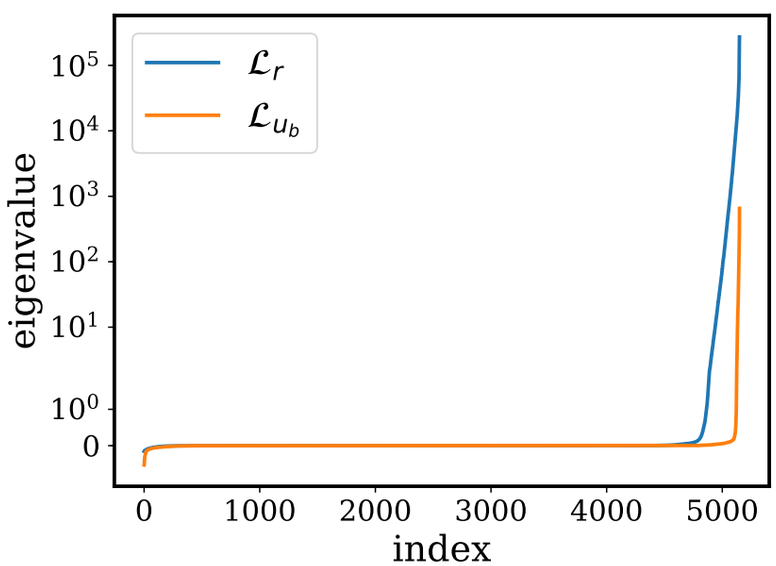
$$\theta_{n+1} = \theta_n - \eta \nabla_\theta \mathcal{L}(\theta_n) = \theta_n - \eta \left(\nabla_\theta \mathcal{L}_r(\theta_n) + \nabla_\theta \mathcal{L}_{u_b}(\theta_n) \right)$$

Applying second order Taylor expansion to the loss function $\mathcal{L}(\theta)$ at θ_n we can show that

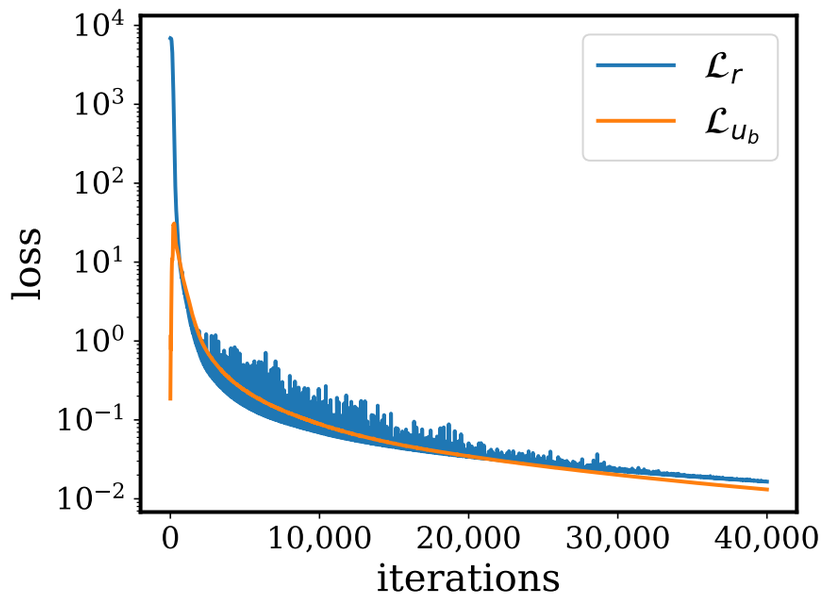
$$\mathcal{L}_r(\theta_{n+1}) - \mathcal{L}_r(\theta_n) = \eta \|\nabla_\theta \mathcal{L}(\theta_n)\|_2^2 (-1 + \frac{1}{2}\eta \sum_{i=1}^{N} \lambda_i^r y_i^2)$$

$$\mathcal{L}_{u_b}(\theta_{n+1}) - \mathcal{L}_{u_b}(\theta_n) = \eta \|\nabla_\theta \mathcal{L}(\theta_n)\|_2^2 (-1 + \frac{1}{2}\eta \sum_{i=1}^{N} \lambda_i^{u_b} y_i^2),$$

for some $\mathbf{y} = (y_1, \ldots, y_N)$ satisfying $\|\mathbf{y}\|_2^2 = \sum y_i^2 = 1$, where $\lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_N$ are eigenvalues of $\nabla_\theta^2 \mathcal{L}$.



Large Hessian eigenvalues indicate stiffness in the gradient flow dynamics



Even full-batch gradient descent can get trapped in limit cycles and does not guarantee a monotonic decrease of the loss.

Gradient pathologies in physics-informed neural networks

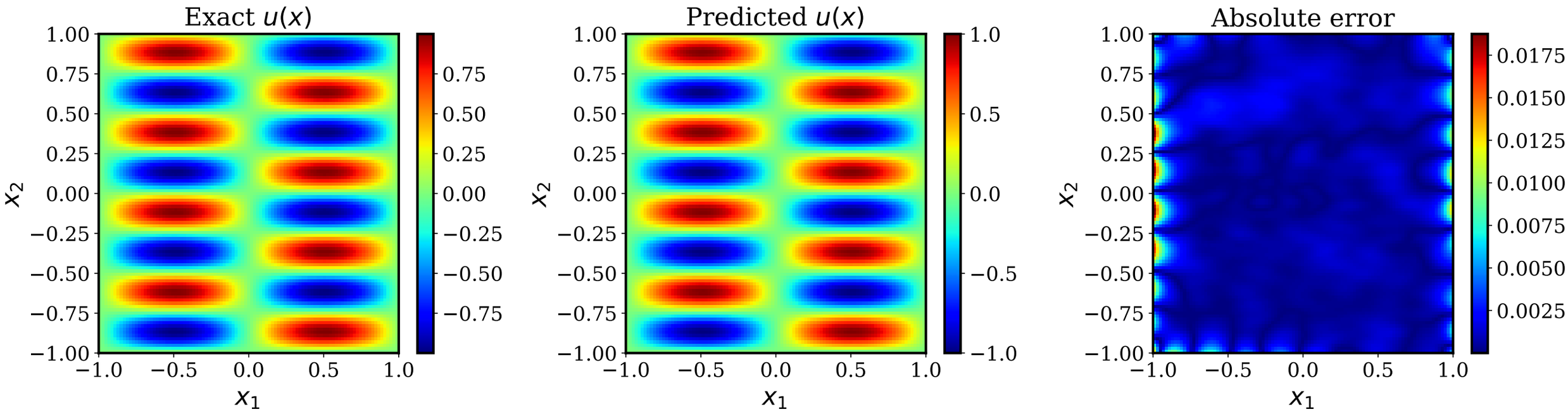
A simple benchmark (2D Helmholtz equation):

$$\Delta u + k^2 u = q(x,y) \;\; (x,y) \in [-1,1]$$

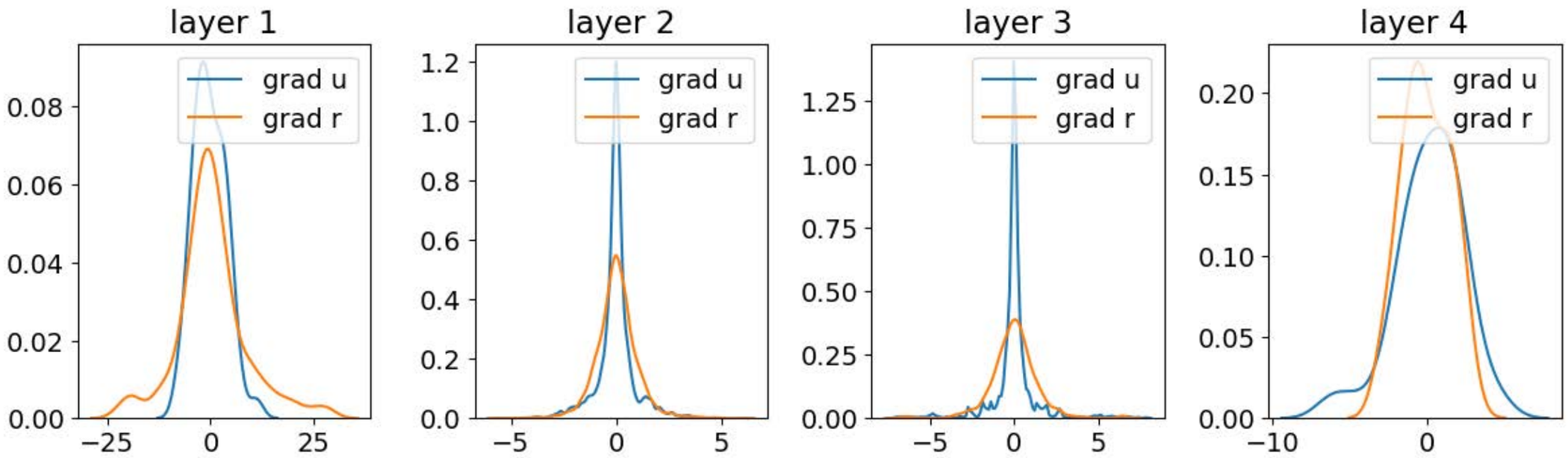
$$u(x,y) = (x+y)\sin(\pi x)\sin(6\pi y)$$

Loss function:

$$\mathcal{L}(\theta) := \lambda_1 \underbrace{\mathcal{L}_r(\theta)}_{\text{PDE residual}} + \lambda_2 \underbrace{\mathcal{L}_{u_b}(\theta)}_{\text{BCs fit}}$$



Prediction of a fully connected 4-layer deep physics-informed neural network (0.5% relative error)



Histograms of back-propagated gradients $\nabla_\theta \mathscr{L}_{u_b}(\theta), \nabla_\theta \mathscr{L}_r(\theta)$ *at each hidden layer*

…but how to choose the weights/learning rates?

$$\mathcal{L}(\theta) := \lambda_1 \underbrace{\mathcal{L}_u(\theta)}_{\text{Data fit}} + \lambda_2 \underbrace{\mathcal{L}_r(\theta)}_{\text{PDE residual}} + \lambda_3 \underbrace{\mathcal{L}_{u_0}(\theta)}_{\text{ICs fit}} + \lambda_4 \underbrace{\mathcal{L}_{u_b}(\theta)}_{\text{BCs fit}}$$

Adaptive moment estimation

Algorithm 1: *Adam*, our proposed algorithm for stochastic optimization. See section 2 for details, and for a slightly more efficient (but less clear) order of computation. g_t^2 indicates the elementwise square $g_t \odot g_t$. Good default settings for the tested machine learning problems are $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$. All operations on vectors are element-wise. With β_1^t and β_2^t we denote β_1 and β_2 to the power t.

Require: α: Stepsize
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
Require: $f(\theta)$: Stochastic objective function with parameters θ
Require: θ_0: Initial parameter vector
$m_0 \leftarrow 0$ (Initialize 1$^{\text{st}}$ moment vector)
$v_0 \leftarrow 0$ (Initialize 2$^{\text{nd}}$ moment vector)
$t \leftarrow 0$ (Initialize timestep)
while θ_t not converged **do**
$t \leftarrow t + 1$
$g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep t)
$m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
$v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
$\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
$\widehat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
$\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$ (Update parameters)
end while
return θ_t (Resulting parameters)

…i.e. use the gradient statistics during training to adaptively adjust the learning rate.

A learning rate annealing algorithm for PINNs

Algorithm 1: Learning rate annealing for physics-informed neural networks

Consider a physics-informed neural network $f_\theta(\boldsymbol{x})$ with parameters θ and a loss function

$$\mathcal{L}(\theta) := \mathcal{L}_r(\theta) + \sum_{i=1}^{M} \lambda_i \mathcal{L}_i(\theta),$$

where $\mathcal{L}_r(\theta)$ denotes the PDE residual loss, the $\mathcal{L}_i(\theta)$ correspond to data-fit terms (e.g., measurements, initial or boundary conditions, etc.), and $\lambda_i = 1, i = 1, \ldots, M$ are free parameters used to balance the interplay between the different loss terms. Then use S steps of a gradient descent algorithm to update the parameters θ as:

for $n = 1, \ldots, S$ **do**

(a) Compute $\hat{\lambda}_i$ by

$$\hat{\lambda}_i = \frac{\max_\theta\{|\nabla_\theta \mathcal{L}_r(\theta_n)|\}}{\overline{|\nabla_\theta \lambda_i \mathcal{L}_i(\theta_n)|}}, \quad i = 1, \ldots, M, \tag{40}$$

where $\overline{|\nabla_\theta \lambda_i \mathcal{L}_i(\theta_n)|}$ denotes the mean of $|\nabla_\theta \lambda_i \mathcal{L}_i(\theta_n)|$ with respect to parameters θ.

(b) Update the weights λ_i using a moving average of the form

$$\lambda_i = (1 - \alpha)\lambda_i + \alpha \hat{\lambda}_i, \quad i = 1, \ldots, M. \tag{41}$$

(c) Update the parameters θ via gradient descent

$$\theta_{n+1} = \theta_n - \eta \nabla_\theta \mathcal{L}_r(\theta_n) - \eta \sum_{i=1}^{M} \lambda_i \nabla_\theta \mathcal{L}_i(\theta_n) \tag{42}$$

end

The recommended hyper-parameter values are: $\eta = 10^{-3}$ and $\alpha = 0.1$.

Wang, S., Teng, Y., & Perdikaris, P. (2020). Understanding and mitigating gradient pathologies in physics-informed neural networks. arXiv preprint arXiv:2001.04536.

Soft physics-informed learning, a recap

$$\mathcal{L}(\theta) := \underbrace{\frac{1}{N_u}\sum_{i=1}^{N_u}[\boldsymbol{u}_i - f_\theta(\boldsymbol{x}_i)]^2}_{\text{Data fit}} + \underbrace{\frac{1}{\lambda}\mathcal{R}[f_\theta(\boldsymbol{x})]}_{\text{Physics regularization}}$$

An "unconventional" regularizer/prior that requires us to revisit standard deep learning practices:

- loss functions (e.g., square residual, *variational principle, Hamiltonian, etc.?*)
- network initialization (e.g., Glorot, *adaptive?*)
- normalization (e.g., zero-mean/unit-variance, *PDE solution bounds?*)
- optimization (e.g., Adam, *adaptive learning rates, proximal algorithms, meta-learning?*)
- network architecture (e.g., fully connected, *residual/recurrent/convolutional layers, attention?*)

$$\frac{d\theta}{dt} = -\nabla_\theta \mathcal{L}_r(\theta) - \sum_{i=1}^{M} \nabla_\theta \mathcal{L}_i(\theta)$$

$$\theta_{n+1} = \theta_n - \eta \nabla_\theta \mathcal{L}_r(\theta_n) - \eta \sum_{i=1}^{M} \nabla_\theta \mathcal{L}_i(\theta_n)$$

Stiffness in the gradient flow dynamics.

An improved neural architecture

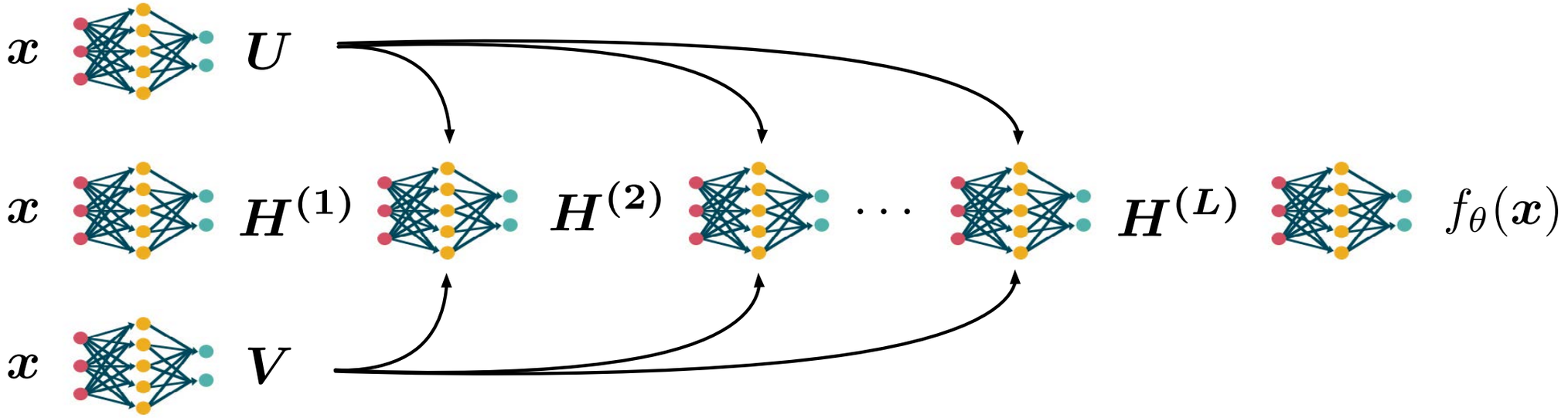
$$U = \phi(W^1 \vec{x} + b^1), \;\; V = \phi(W^2 \vec{x} + b^2)$$

$$H^{(1)} = \phi(W^{z,1} \vec{x} + b^{z,1})$$

$$Z^{(k)} = \phi(W^{z,k} H^{(k)} + b^{z,k}), \;\; k = 1, \ldots, L$$

$$H^{(k+1)} = (1 - Z^{(k)}) \odot U + Z^{(k)} \odot V, \;\; k = 1, \ldots, L$$

$$f(x; \theta) = W H^{(L+1)} + b$$



$$\theta = \{W^1, b^1, W^2, b^2, (W^{z,l}, b^{z,l})_{l=1}^{L}, W, b\}$$

Key points:

- Account for multiplicative interactions of the inputs, similar to attention mechanisms.
- Residual connections improve resilience against vanishing gradient pathologies.

Systematic comparison

M1: Baseline PINN model (Raissi et. al., 2019)
M2: PINN with the proposed learning rate annealing
M3: PINN with the proposed neural architecture
M4: PINN with the proposed learning rate annealing and improved neural architecture

Architecture	M1	M2	M3	M4
30 units / 3 hidden layers	2.44E-01	3.98E-02	5.31E-02	**2.56E-03**
50 units / 3 hidden layers	1.06E-01	1.58E-02	2.46E-02	**1.81E-03**
100 units / 3 hidden layers	9.07E-02	2.39E-03	1.17E-02	**1.28E-03**
30 units / 5 hidden layers	2.47E-01	8.91E-03	4.12E-02	**1.96E-03**
50 units / 5 hidden layers	1.40E-01	8.08E-03	1.97E-02	**1.86E-03**
100 units / 5 hidden layers	1.15E-01	3.25E-03	1.08E-02	**1.22E-03**
30 units / 7 hidden layers	3.10E-01	7.86E-03	3.17E-02	**1.98E-03**
50 units / 7 hidden layers	1.98E-01	3.66E-03	2.37E-02	**1.54E-03**
100 units / 7 hidden layers	8.14E-02	2.57E-03	9.36E-03	**1.40E-03**

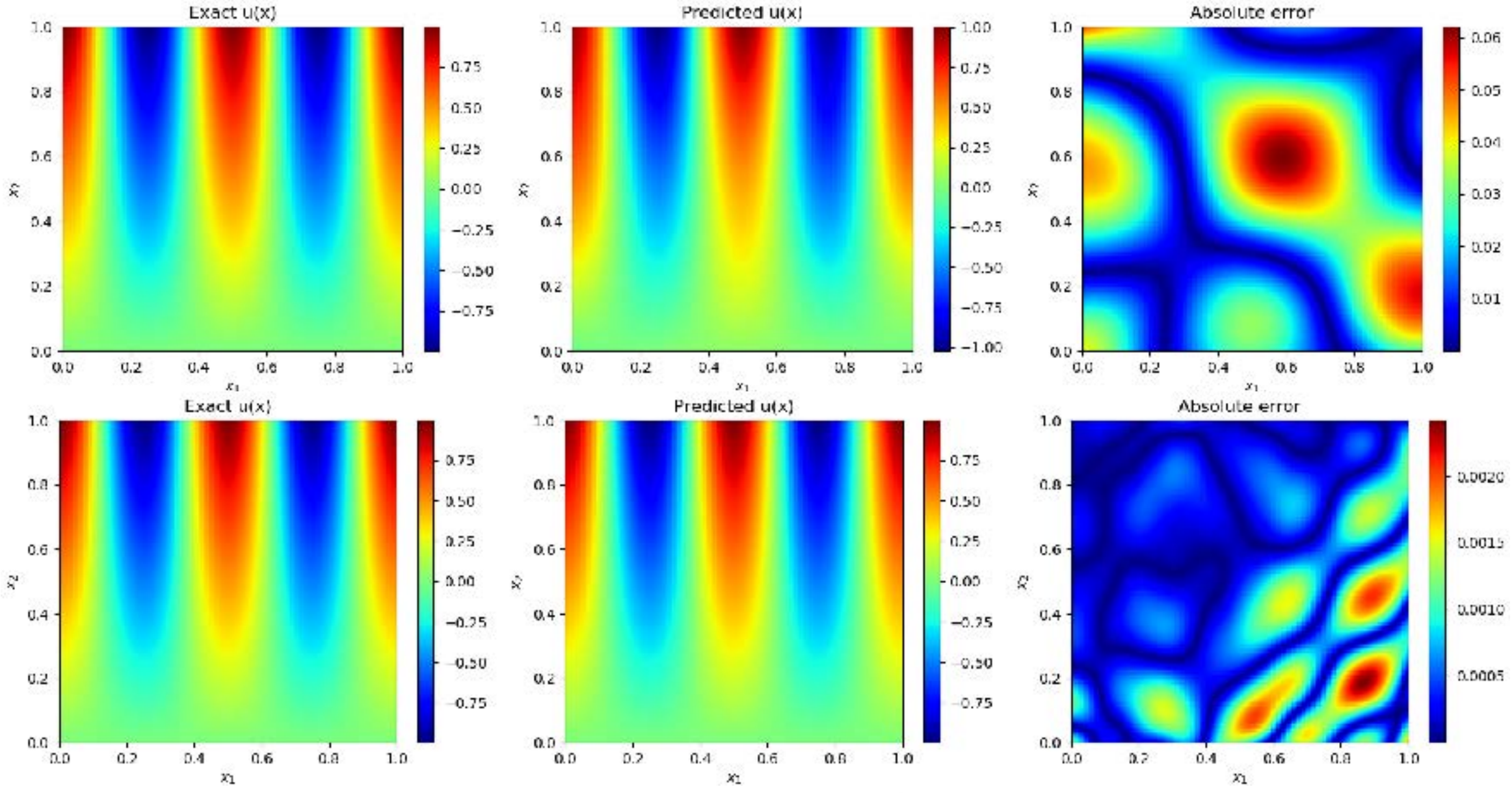
Relative prediction error (L2 norm) averaged over 10 independent trials for the 2D Helmholtz benchmark.

Klein Gordon equation

$$u_{tt} + \alpha u_{xx} + \beta u + \gamma u^k = f(x,t), \quad (x,t) \in \Omega \times [0,T]$$

$$u(x,0) = g_1(x), \quad x \in \Omega$$
$$u_t(x,0) = g_2(x), \quad x \in \Omega$$

$$u_(x,t) = h(x,t), \quad (x,t) \in \partial\Omega \times [0,T]$$



Top: Imbalanced gradients in a dense, 5-layer deep physics-informed neural network lead to considerable prediction errors (6.7%).

Bottom: Accurate predictions can be obtained using the proposed learning rate annealing and improved neural architecture strategy (relative prediction error: 0.1%).

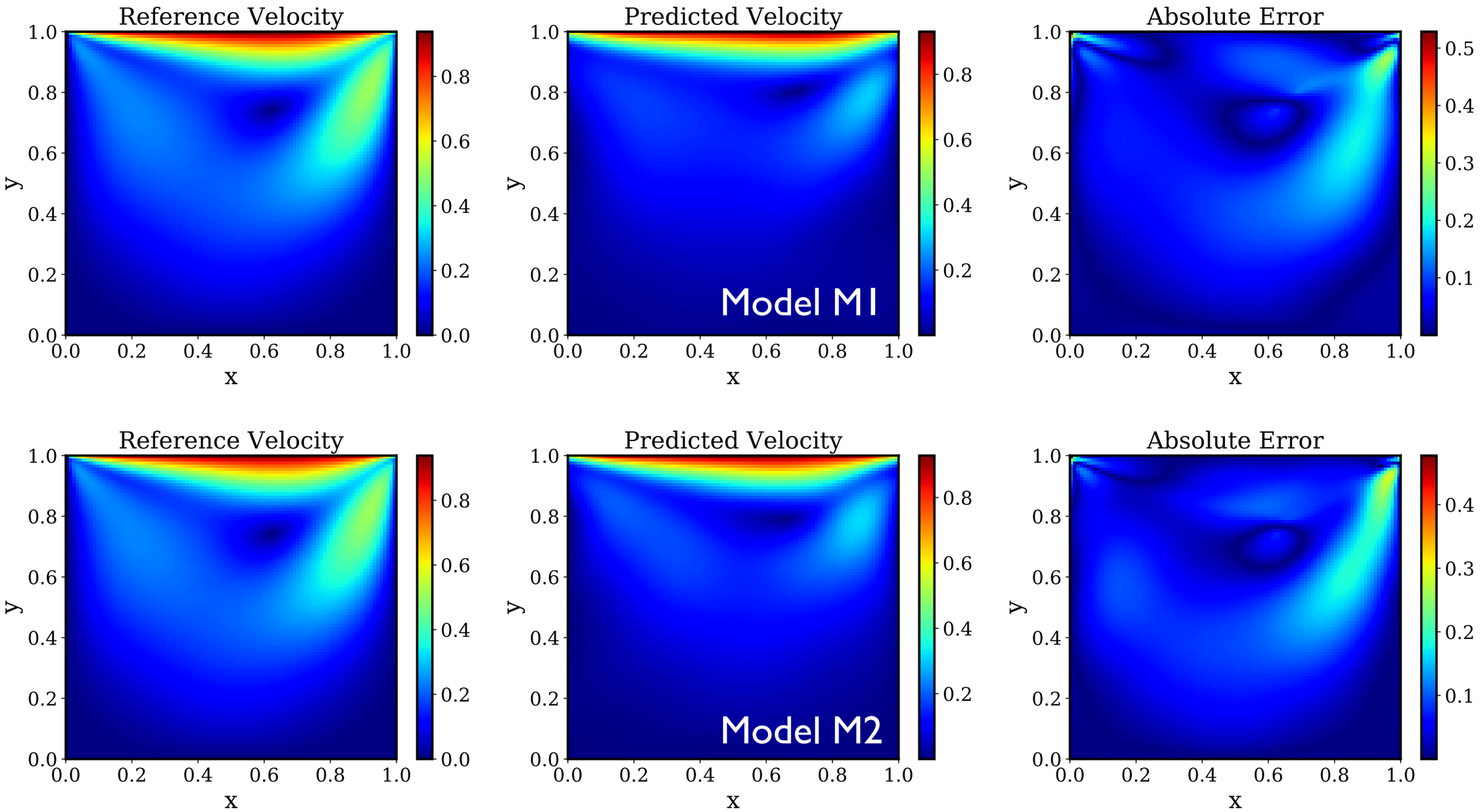
Flow in a lid-driven cavity

$$\boldsymbol{u} \cdot \nabla \boldsymbol{u} + \nabla p - \frac{1}{Re} \Delta \boldsymbol{u} = 0 \qquad \text{in } \Omega$$

$$\nabla \cdot \boldsymbol{u} = 0 \qquad \text{in } \Omega$$

$$\boldsymbol{u}(\boldsymbol{x}) = (1, 0) \qquad \text{on } \Gamma_1$$

$$\boldsymbol{u}(\boldsymbol{x}) = (0, 0) \qquad \text{on } \Gamma_0$$



Flow in a lid-driven cavity

$$\boldsymbol{u} \cdot \nabla \boldsymbol{u} + \nabla p - \frac{1}{Re} \Delta \boldsymbol{u} = 0 \qquad \text{in } \Omega$$

$$\nabla \cdot \boldsymbol{u} = 0 \qquad \text{in } \Omega$$

$$\boldsymbol{u}(\boldsymbol{x}) = (1, 0) \qquad \text{on } \Gamma_1$$

$$\boldsymbol{u}(\boldsymbol{x}) = (0, 0) \qquad \text{on } \Gamma_0$$