Introduction to Machine Learning with Applications to Aeroheating Database Generation

Thermal & Fluids Analysis Workshop

August 22, 2023

James B. Scoggins

Aerothermodynamics Branch, NASA Langley Research Center

Goals of this short-course

1. **Provide a foundation in ML and resources for you to learn on your own**
    - Machine learning is a very broad field, impossible to teach everything here
    - Instead, introduce core principles and vocabulary
    - Resources for self-learning

2. **Demonstrate recent examples of ML in my daily work at NASA**
    - How to approach typical problems
    - Combine physical intuition and knowledge with ML principles
    - Gaussian Process regression

What do we mean by “learning”?

A formal definition: A computer program is said to ***learn*** from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience [1].

1. Tom M. Mitchell. *Machine Learning*. McGraw-Hill, 1997.

What do we mean by “learning”?

A formal definition: A computer program is said to ***learn*** from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience [1].

- Well-posed learning problems require experience (data), a task (prediction), and a measure of performance

1. Tom M. Mitchell. *Machine Learning*. McGraw-Hill, 1997.

What do we mean by "learning"?

A formal definition: A computer program is said to ***learn*** from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience [1].

- Well-posed learning problems require experience (data), a task (prediction), and a measure of performance
- Machine learning is the science of designing computer algorithms that can <u>automatically</u> improve their performance at a given task, as additional data are provided to them

1. Tom M. Mitchell. *Machine Learning*. McGraw-Hill, 1997.

What do we mean by “learning”?

A formal definition: A computer program is said to ***learn*** from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience [1].

- Well-posed learning problems require experience (data), a task (prediction), and a measure of performance
- Machine learning is the science of designing computer algorithms that can <u>automatically</u> improve their performance at a given task, as additional data are provided to them

Stockfish

Task:
Win chess match.

Performance Measure:
Number of wins

Experience:
Human logic and intuition.

AlphaZero

Task:
Win chess match.

Performance Measure:
Number of wins

Experience:
Playing chess matches agains opponents and self.

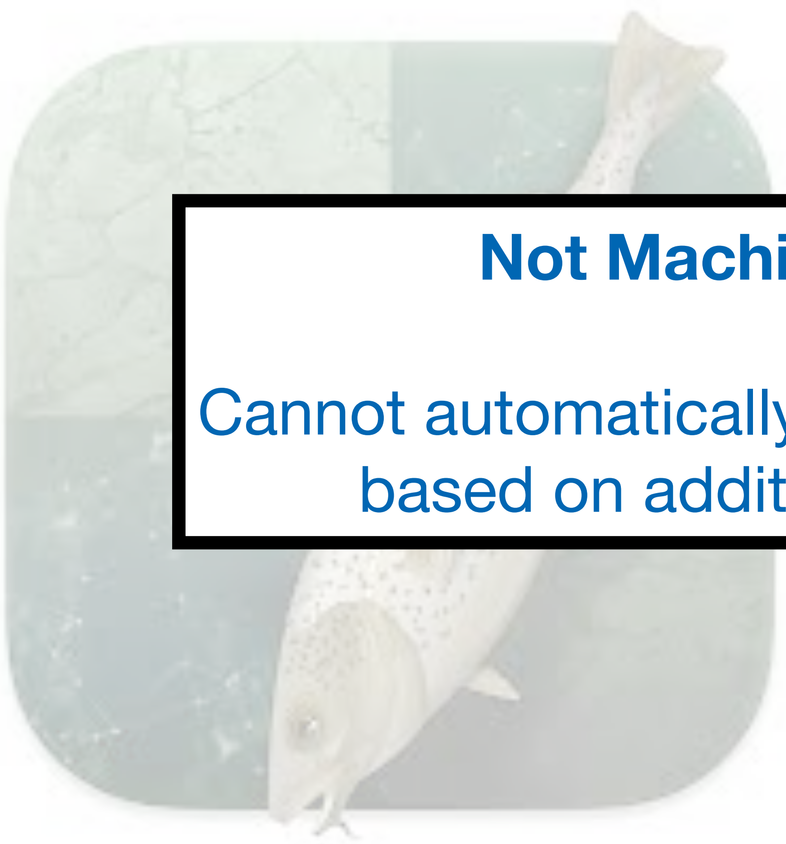
1. Tom M. Mitchell. *Machine Learning*. McGraw-Hill, 1997.

What do we mean by “learning”?

A formal definition: A computer program is said to ***learn*** from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience [1].

- Well-posed learning problems require experience (data), a task (prediction), and a measure of performance
- Machine learning is the science of designing computer algorithms that can <u>automatically</u> improve their performance at a given task, as additional data are provided to them

Stockfish



Task:
Win chess match.

...asure:

Human logic and intuition.

Not Machine Learning

Cannot automatically improve performance based on additional experience.

AlphaZero

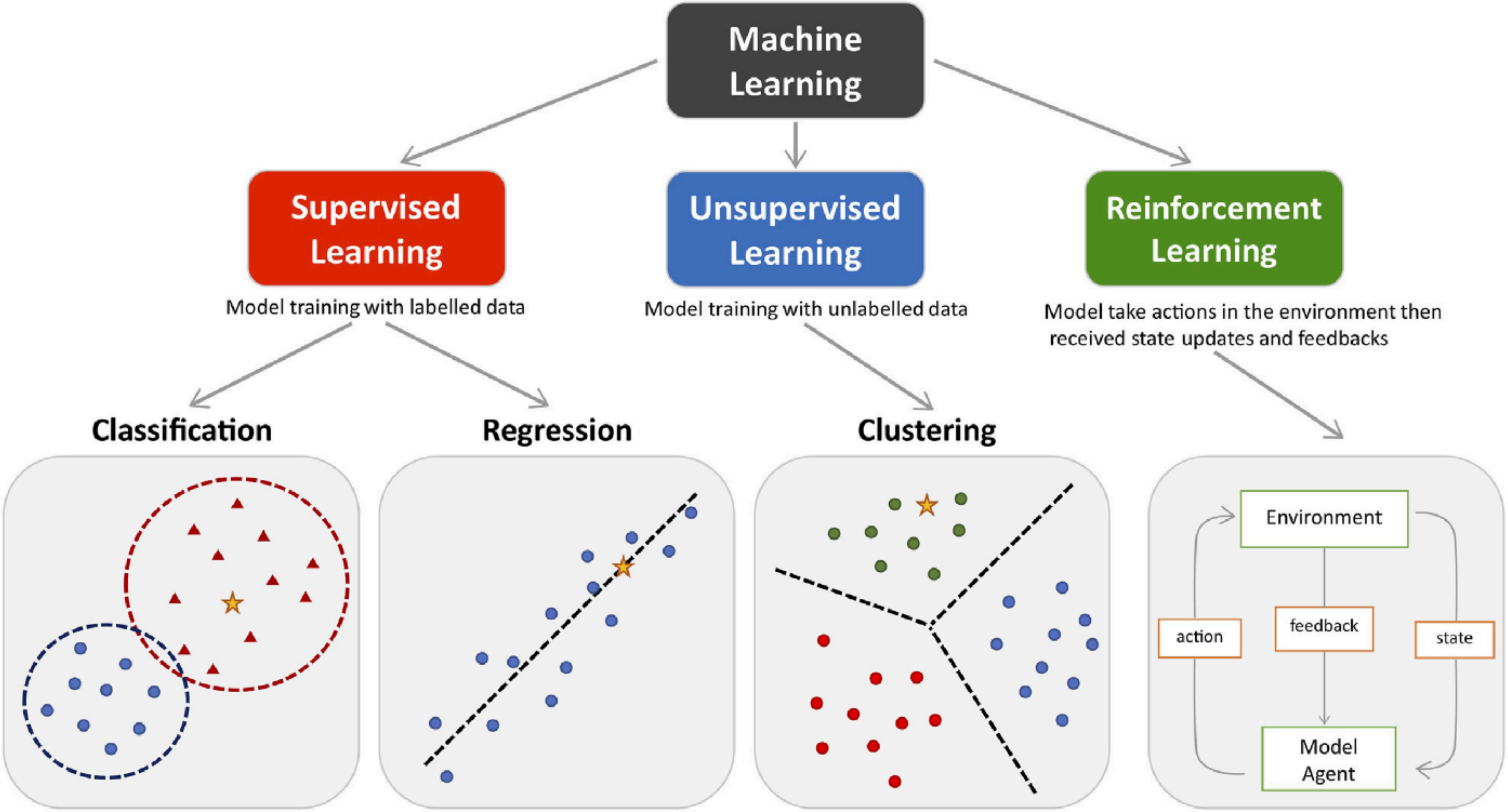
Task:
Win chess match.

Performance Measure:
Number of wins

Experience:
Playing chess matches agains opponents and self.

1. Tom M. Mitchell. *Machine Learning*. McGraw-Hill, 1997.

Types of learning problems



Peng, Jury, Donnes, Ciurtin. *Frontiers in Pharmacology* 12:720694, 2021.

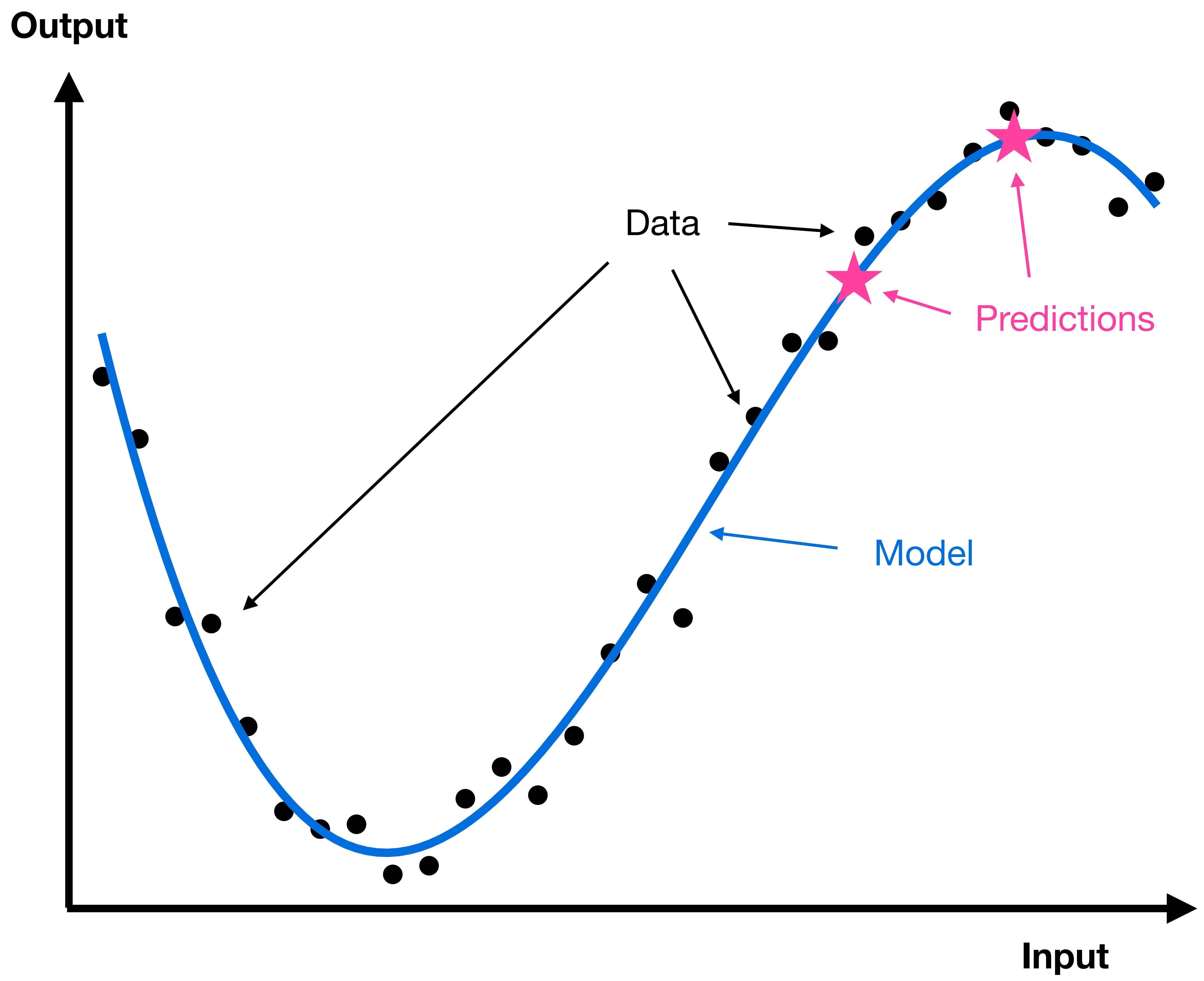
Regression in a nutshell

Learn a functional relationship (*model*) between data *inputs* and *outputs* to make *predictions* for unseen inputs

Supervised Learning Framework

- Given a *dataset*: $\mathscr{D} = \{(x_i, y_i) : x_i \in \Omega, \; y_i = f(x_i)\}$
- Given a (possibly *parametric*) model: $y = \hat{f}(x; \theta)$
- Find a model that best approximates the underlying relationship between inputs and outputs

$$\hat{f}(x; \theta^*) \approx f(x)$$



Regression in a nutshell

Learn a functional relationship (*model*) between data *inputs* and *outputs* to make *predictions* for unseen inputs

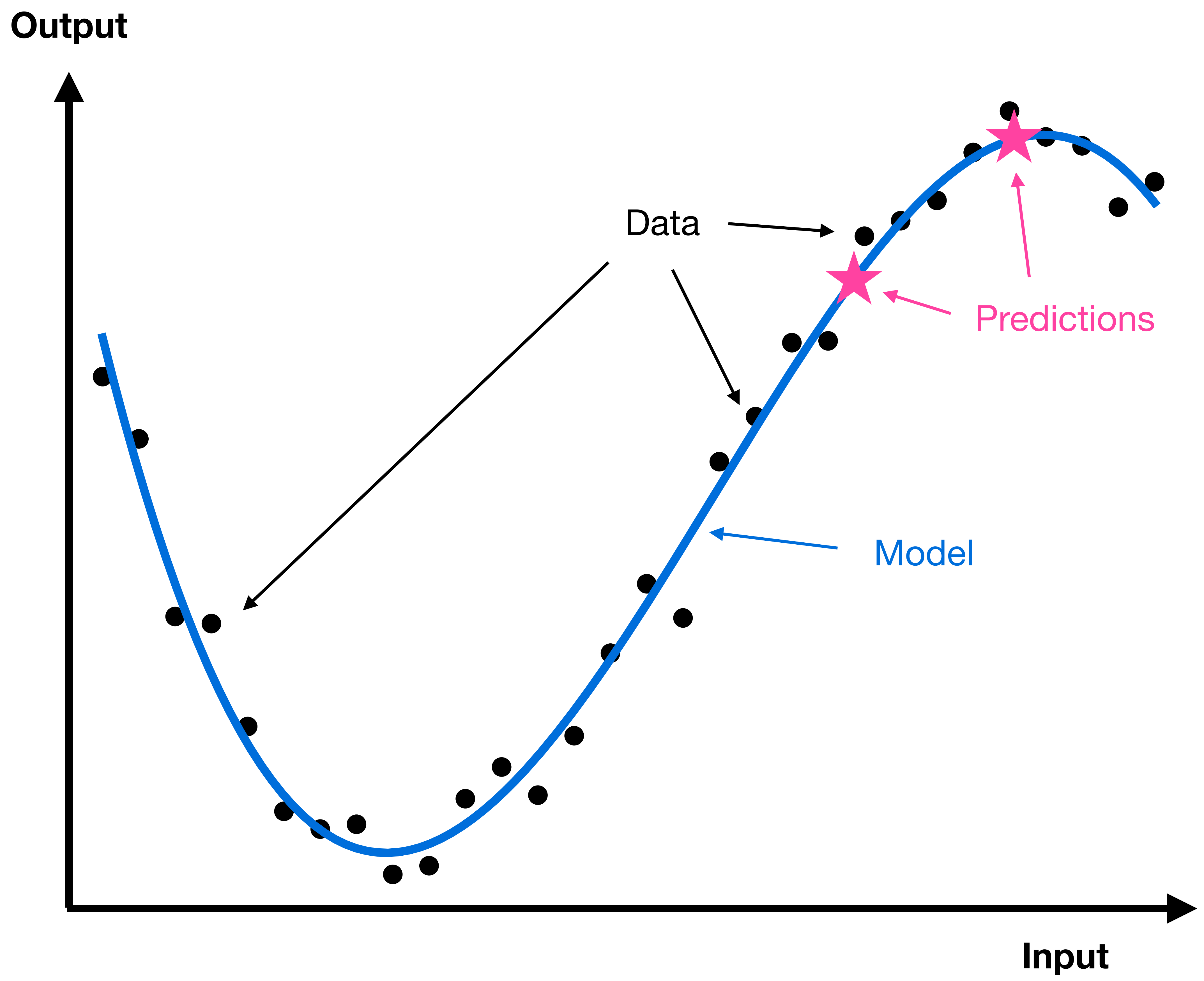
Supervised Learning Framework

- Given a *dataset*: $\mathscr{D} = \{(x_i, y_i) : x_i \in \Omega,\, y_i = f(x_i)\}$
- Given a (possibly *parametric*) model: $y = \hat{f}(x; \theta)$
- Find a model that best approximates the underlying relationship between inputs and outputs

$$\hat{f}(x; \theta^*) \approx f(x)$$

Key Questions

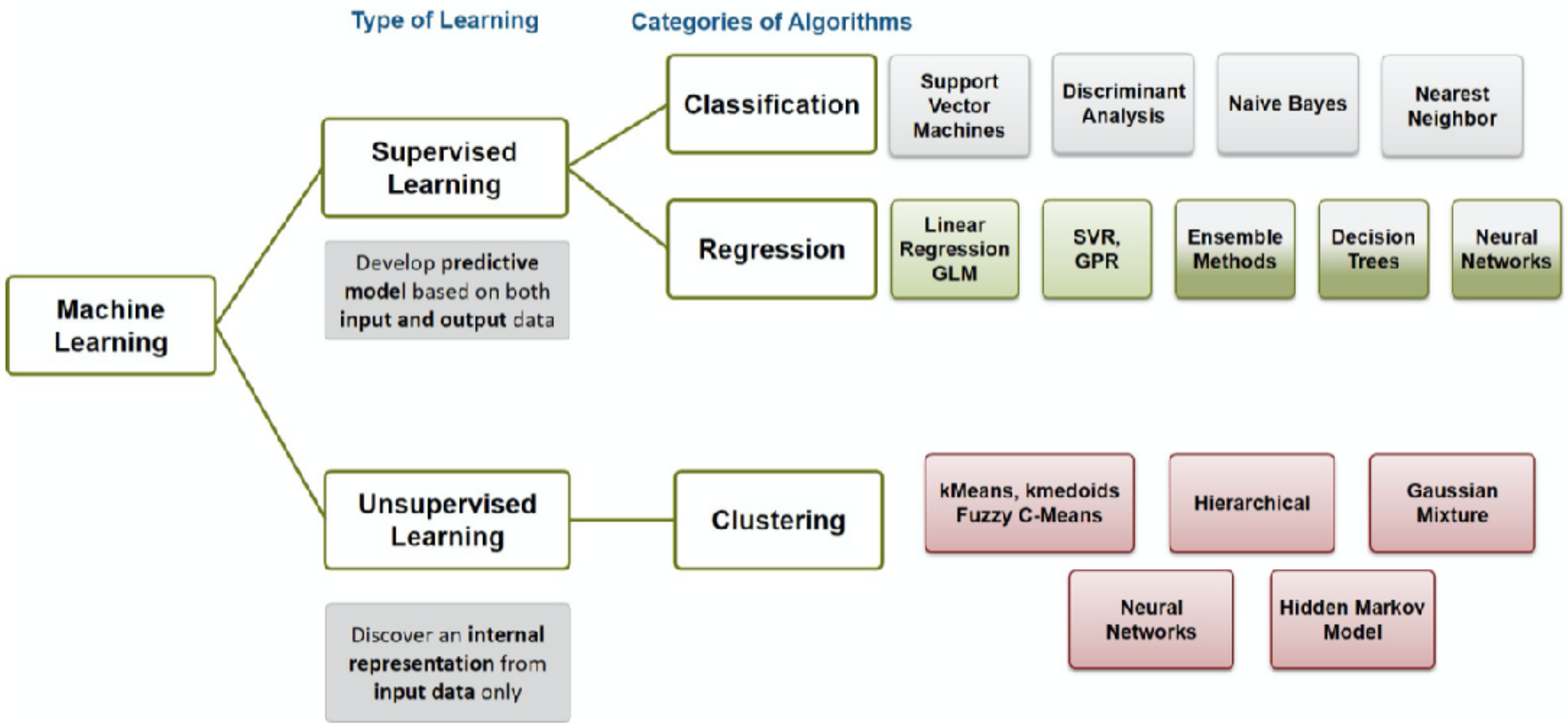
1. How do we know if a model is "good" (much less "best")?
2. What about noisy data?



Many models out there!

Examples:

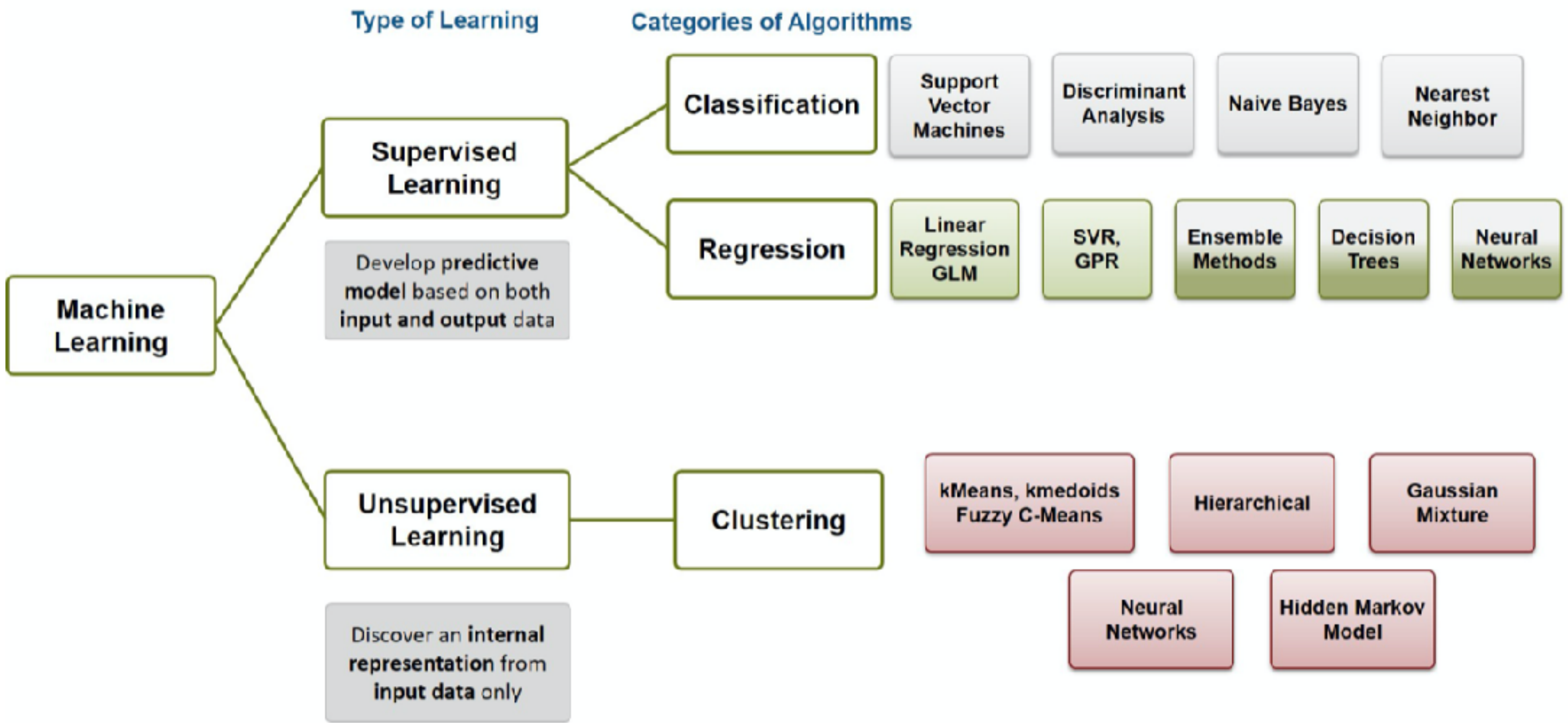
- Linear Models
- Support Vector Machines
- Gaussian Processes
- Neural Networks
- Decision Trees



Many models out there!

Examples:

- Linear Models
- Support Vector Machines
- Gaussian Processes
- Neural Networks
- Decision Trees



Choice depends on multiple factors:

- Training and evaluation cost
- Implementation and deployment
- Scalability
- Treatment of uncertainty

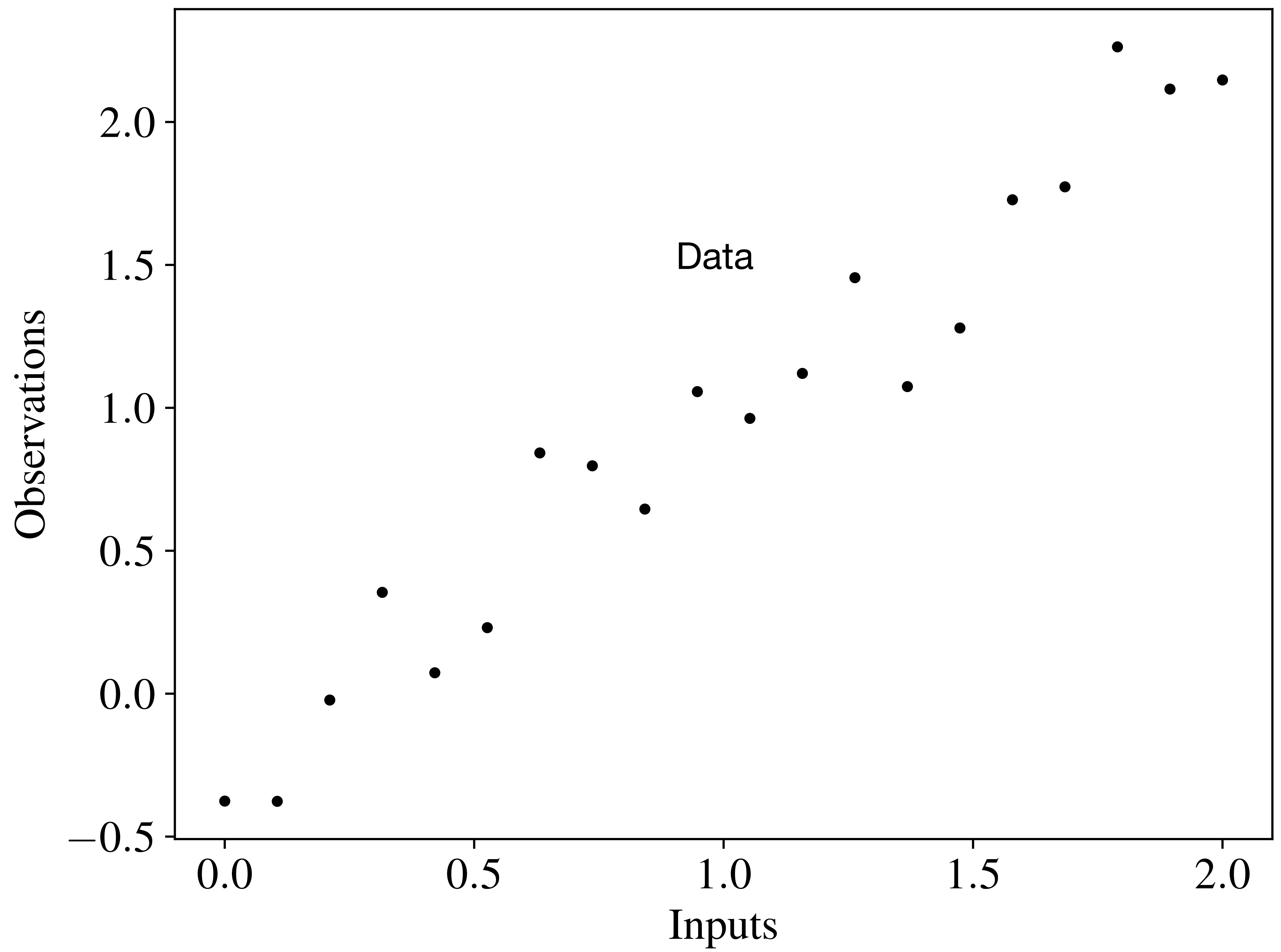
Linear regression

Given the data on the right, what are our initial thoughts?

- Observations generally increase with increasing input values
- Trend appears linear with a positive slope and negative intercept
- The trend is not perfect, noise or other unknown feature

Model assumption: response is linear with nonzero intercept

$$y = \hat{f}(x; w_0, w_1) = w_0 + w_1 x$$



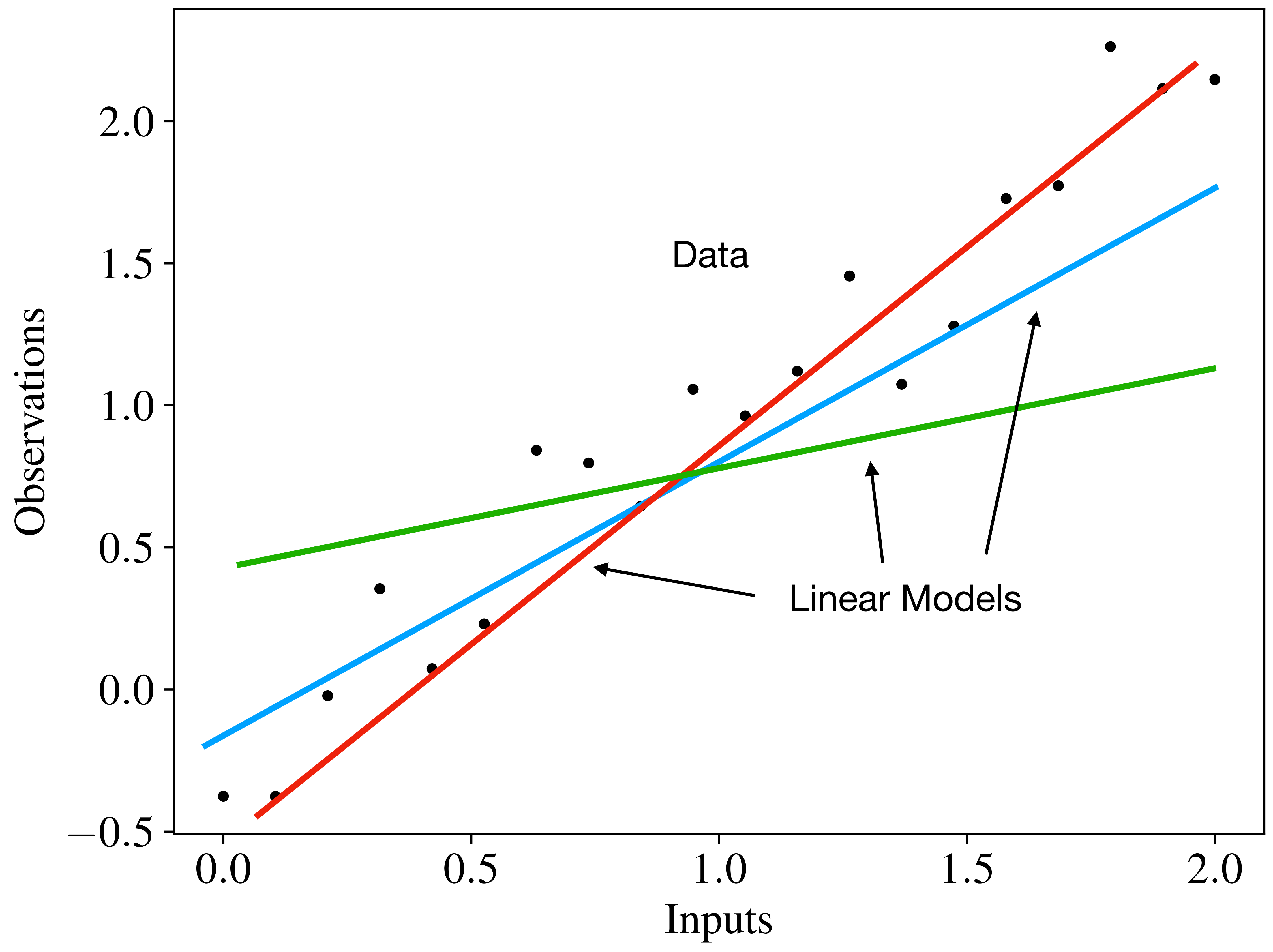
Linear regression

Given the data on the right, what are our initial thoughts?

- Observations generally increase with increasing input values
- Trend appears linear with a positive slope and negative intercept
- The trend is not perfect, noise or other unknown feature

Model assumption: response is linear with nonzero intercept

$$y = \hat{f}(x; w_0, w_1) = w_0 + w_1 x$$



Linear regression

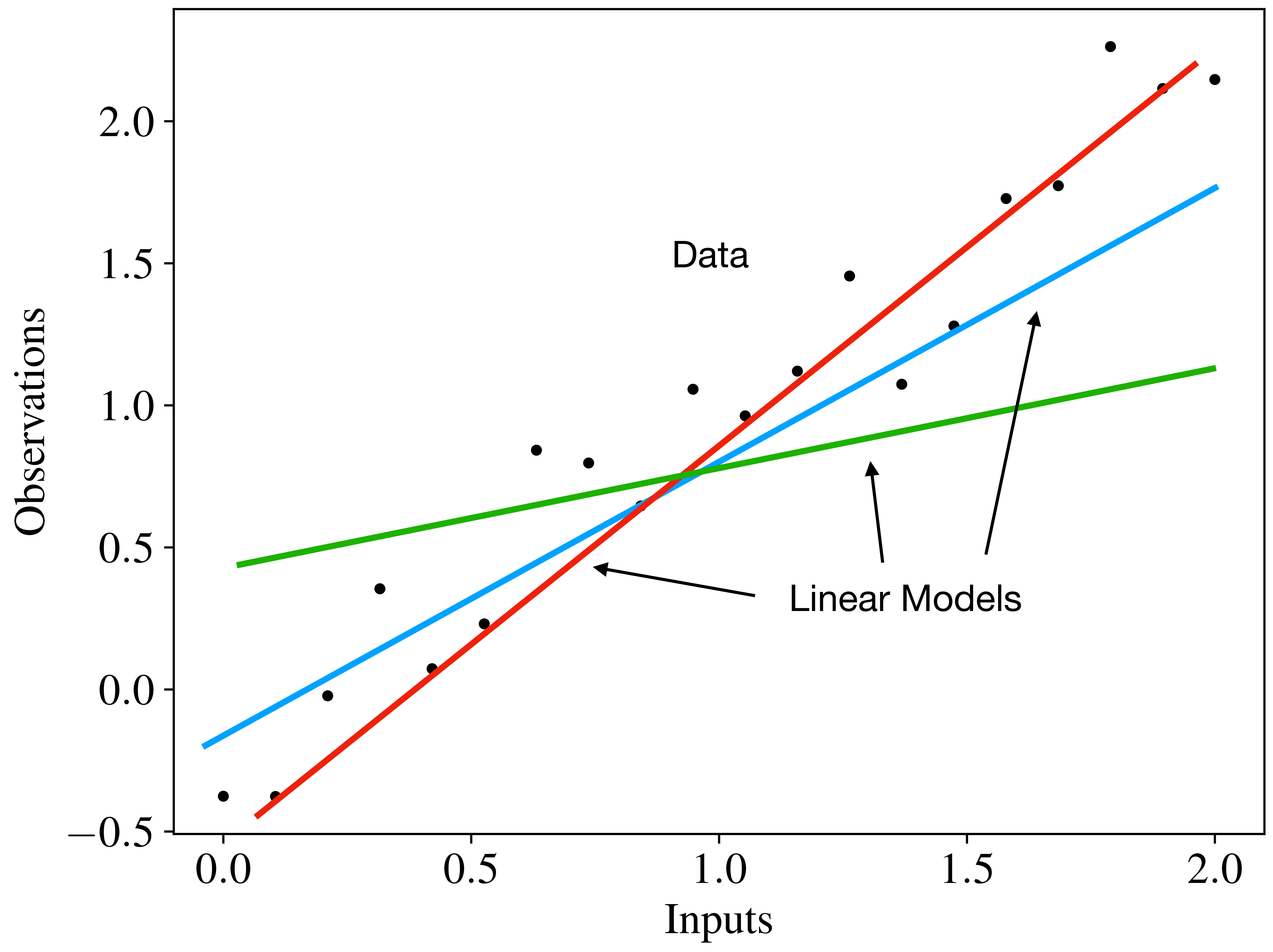
Given the data on the right, what are our initial thoughts?

- Observations generally increase with increasing input values
- Trend appears linear with a positive slope and negative intercept
- The trend is not perfect, noise or other unknown feature

Model assumption: response is linear with nonzero intercept

$$y = \hat{f}(x; w_0, w_1) = w_0 + w_1 x$$

How do we find the "best" model?



Loss functions

Loss functions

In general, we can think of data as samples of an underlying joint probability density function $p(x, y) = p(y \mid x)\, p(x)$, where

$$p(y \mid x) = \begin{cases} 1, & y = f(x) \\ 0, & \text{otherwise} \end{cases}$$

, for noiseless data.

Loss functions

In general, we can think of data as samples of an underlying joint probability density function $p(x, y) = p(y \,|\, x)\; p(x)$, where

$$p(y \,|\, x) = \begin{cases} 1, & y = f(x) \\ 0, & \text{otherwise} \end{cases}$$, for noiseless data.

Loss functions are a measure of our model performance on supervised learning tasks (how well we approximate $p(x, y)$)

- General rule is to make them positive and invariant to dataset size
- Decreasing loss means better model performance

Loss functions

In general, we can think of data as samples of an underlying joint probability density function $p(x, y) = p(y \mid x)\, p(x)$, where $p(y \mid x) = \begin{cases} 1, & y = f(x) \\ 0, & \text{otherwise} \end{cases}$, for noiseless data.

Loss functions are a measure of our model performance on supervised learning tasks (how well we approximate $p(x, y)$)

- General rule is to make them positive and invariant to dataset size
- Decreasing loss means better model performance

$$\mathscr{L}(\theta)[\hat{f}] \;=\; \mathbb{E}_{p(x,y)} l(y, \hat{f}(x;\theta)) \;\equiv\; \int_\Omega l(y, \hat{f}(x;\theta))\, p(x, y)\, dx\, dy$$

Loss as function of model parameters θ for given model $\hat{f}(x;\theta)$

Expected model error over the input probability distribution $p(x)$ for given error model l

Definition of the expectation of l on $p(x)$

Loss functions

In general, we can think of data as samples of an underlying joint probability density function $p(x, y) = p(y \,|\, x)\; p(x)$, where

$$p(y \,|\, x) = \begin{cases} 1, & y = f(x) \\ 0, & \text{otherwise} \end{cases}$$, for noiseless data.

Loss functions are a measure of our model performance on supervised learning tasks (how well we approximate $p(x, y)$)

- General rule is to make them positive and invariant to dataset size
- Decreasing loss means better model performance

$$\mathscr{L}(\theta)[\hat{f}] \;=\; \mathbb{E}_{p(x,y)} l(y, \hat{f}(x;\theta)) \;\equiv\; \int_{\Omega} l(y, \hat{f}(x;\theta))\; p(x, y)\; dx\; dy$$

Loss as function of model parameters θ for given model $\hat{f}(x;\theta)$

Expected model error over the input probability distribution $p(x)$ for given error model l

Definition of the expectation of l on $p(x)$

We generally don't know $p(x, y)$ because that's what we want to model!

Loss functions

In general, we can think of data as samples of an underlying joint probability density function $p(x, y) = p(y\,|\,x)\ p(x)$, where

$p(y\,|\,x) = \begin{cases} 1, & y = f(x) \\ 0, & \text{otherwise} \end{cases}$, for noiseless data.

Loss functions are a measure of our model performance on supervised learning tasks (how well we approximate $p(x, y)$)

- General rule is to make them positive and invariant to dataset size
- Decreasing loss means better model performance

$$\mathscr{L}(\theta)[\hat{f}] \;=\; \mathbb{E}_{p(x,y)} l(y, \hat{f}(x;\theta)) \;\equiv\; \int_\Omega l(y, \hat{f}(x;\theta))\; p(x,y)\; dx\; dy \;\approx\; \frac{1}{N} \sum_{(x_i,y_i)\in\mathscr{D}} l(y_i, \hat{f}(x_i;\theta))$$

Loss as function of model parameters θ for given model $\hat{f}(x;\theta)$

Expected model error over the input probability distribution $p(x)$ for given error model l

Definition of the expectation of l on $p(x)$

"Empirical" loss, evaluated on available dataset

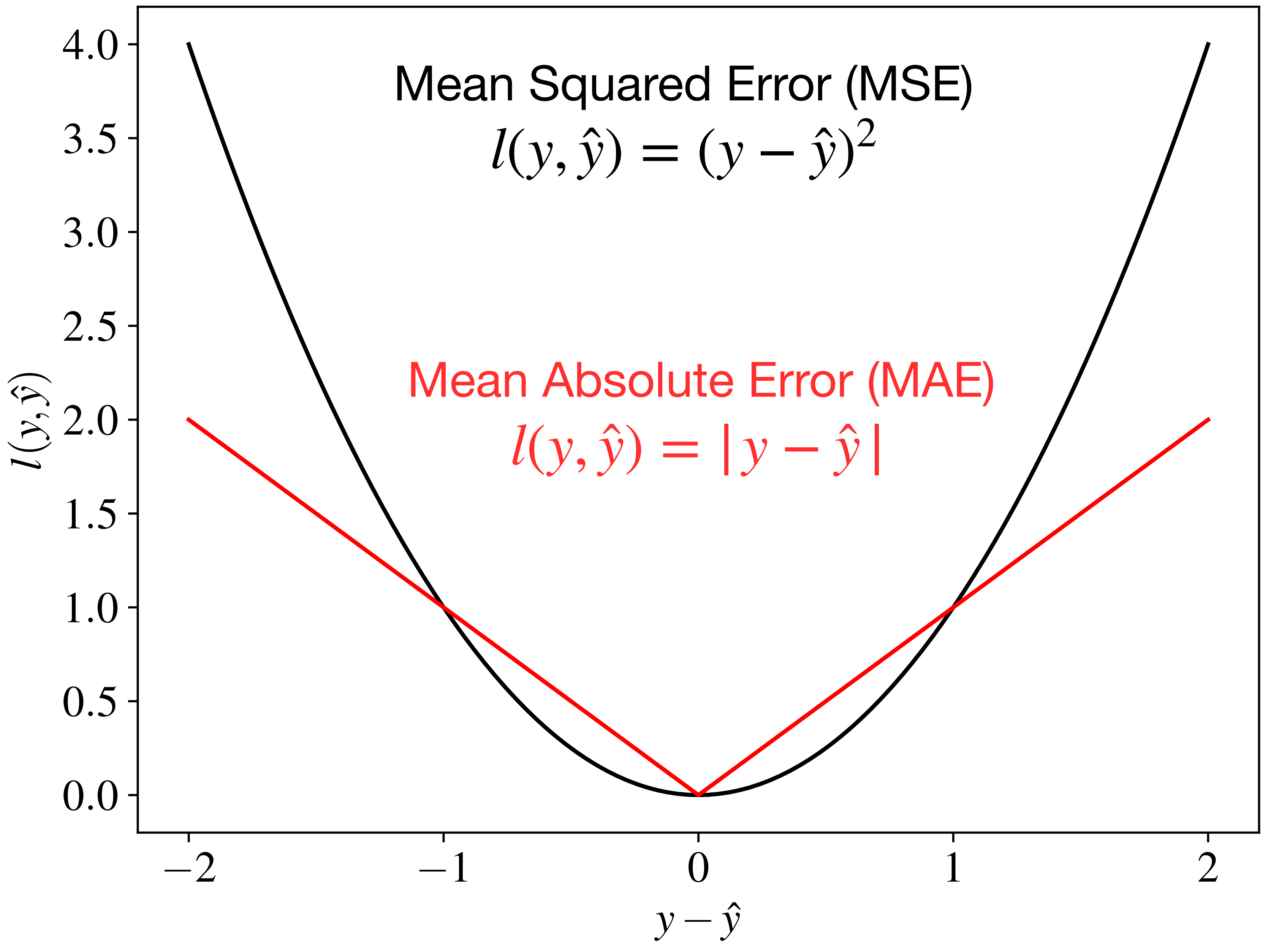
rely on approximate loss

$\boxed{\text{We generally don't know } p(x, y) \text{ because that's what we want to model!}}$

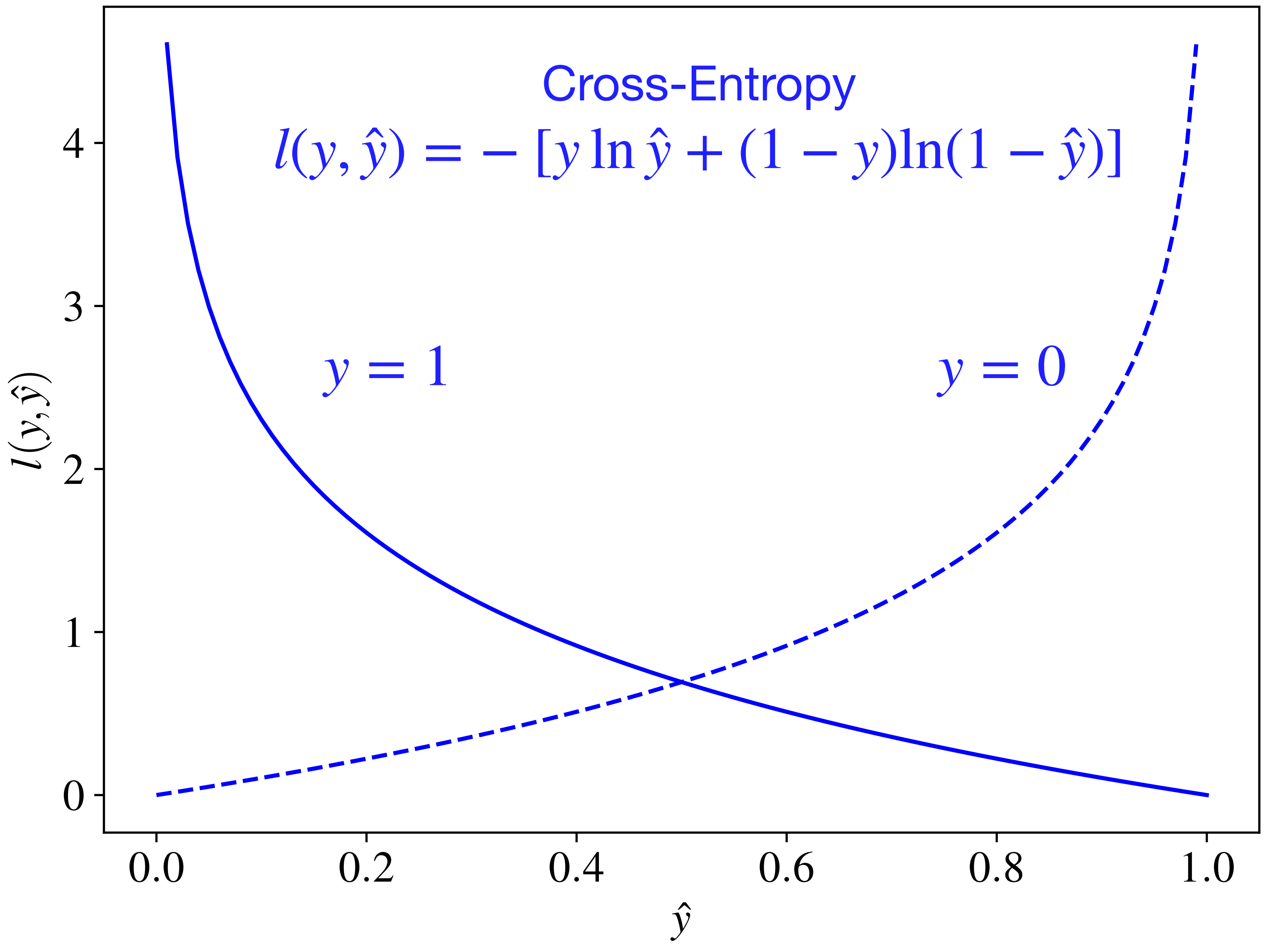
Example loss functions

Recall $\mathscr{L}(\theta)[\hat{f}] = \dfrac{1}{N} \sum_{(x_i, y_i) \in \mathscr{D}} l(y_i, \hat{f}(x_i; \theta))$

Model outputs are predicted values.



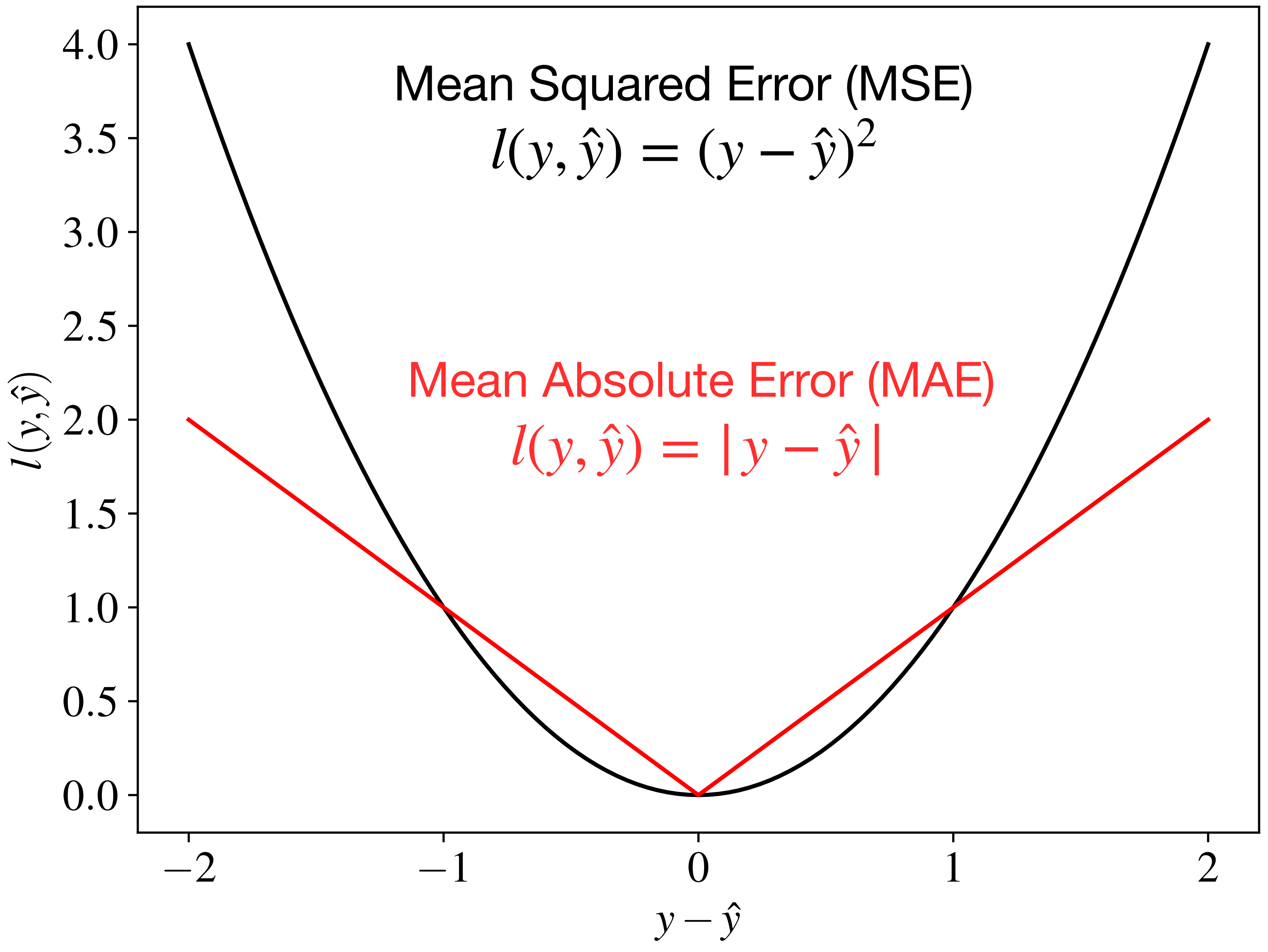
Model outputs are predicted probabilities.



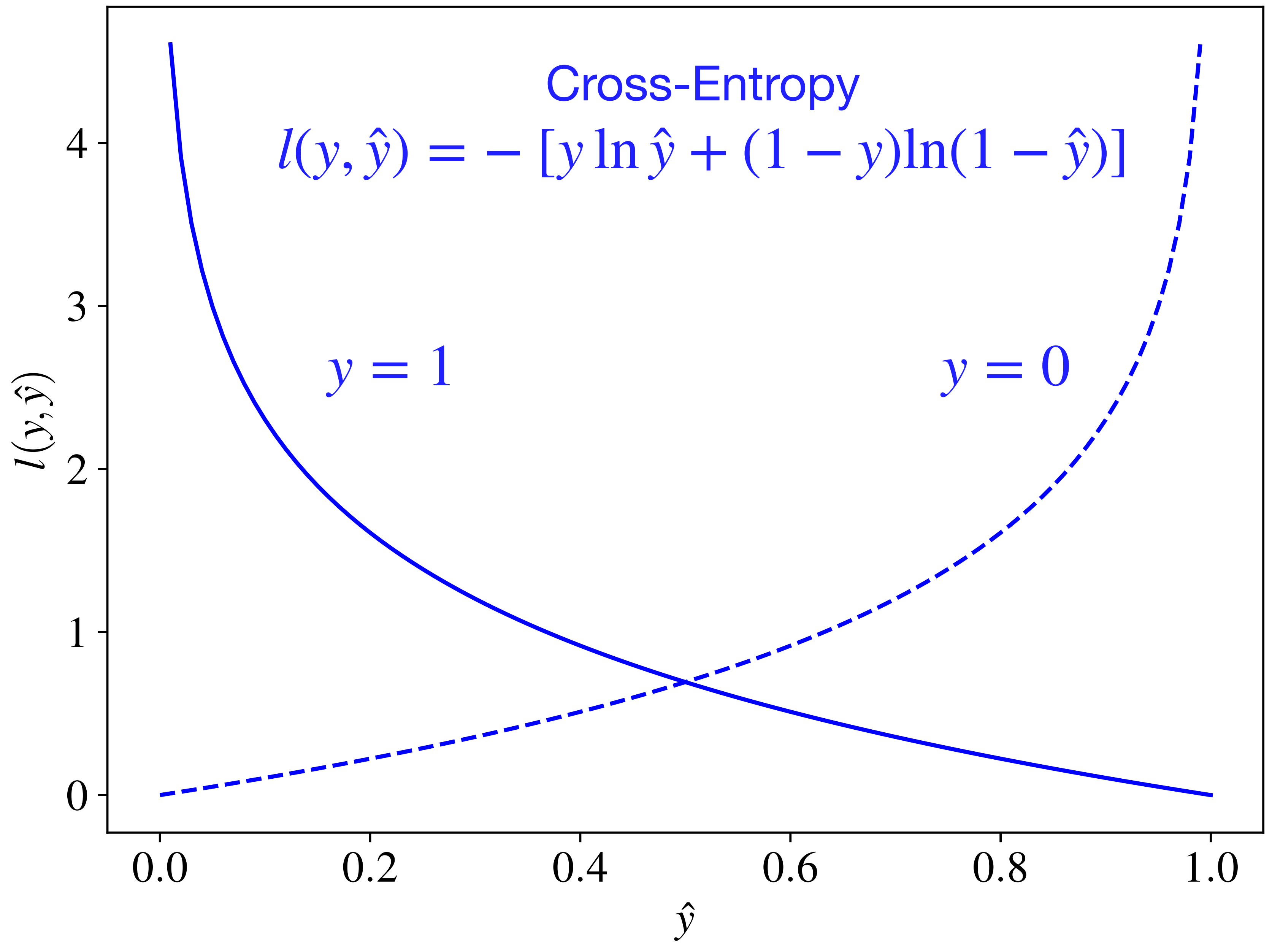
Example loss functions

Recall $\mathscr{L}(\theta)[\hat{f}] \;=\; \frac{1}{N}\sum_{(x_i,y_i)\in\mathscr{D}} l(y_i,\hat{f}(x_i;\theta))$

Model outputs are predicted values.



Model outputs are predicted probabilities.



Choice depends on type of data and model.

Linear least squares regression

- Revisiting our linear model, the MSE loss is given as

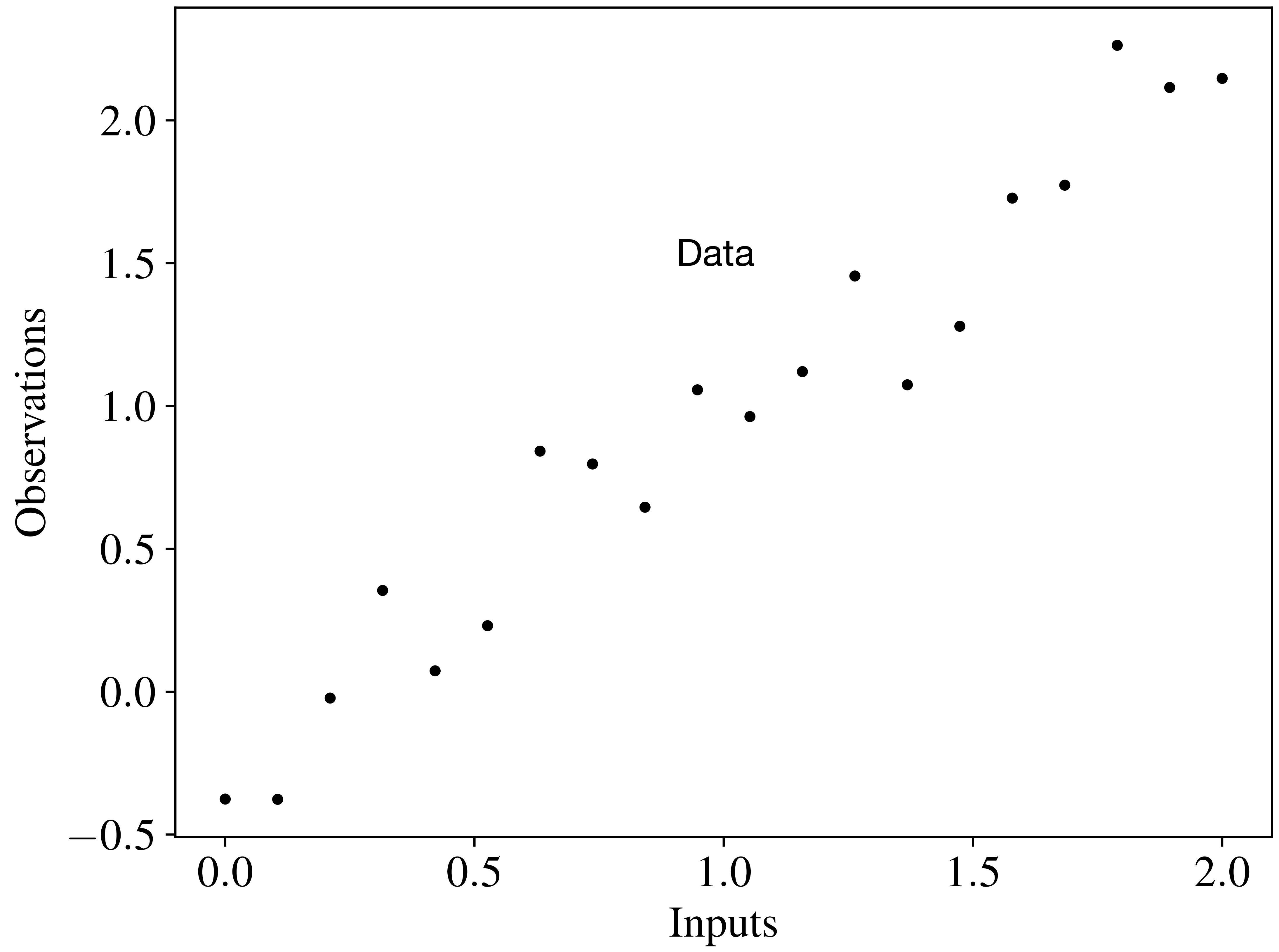
$$\mathscr{L}(w_0, w_1) = \frac{1}{N}\sum_{i=1}^{N}[y_i - (w_0 + w_1 x_i)]^2$$

- Best model is one that minimizes the loss, can derive this analytically for linear least squares loss

$$\frac{\partial \mathscr{L}}{\partial w_0} = 0 \implies w_0 = \bar{y} - w_1 \bar{x}$$

$$\frac{\partial \mathscr{L}}{\partial w_1} = 0 \implies w_1 = \frac{\overline{xy} - \bar{x}\bar{y}}{\overline{x^2} - \bar{x}^2}$$

How do we find the "best" model?



Linear least squares regression

- Revisiting our linear model, the MSE loss is given as

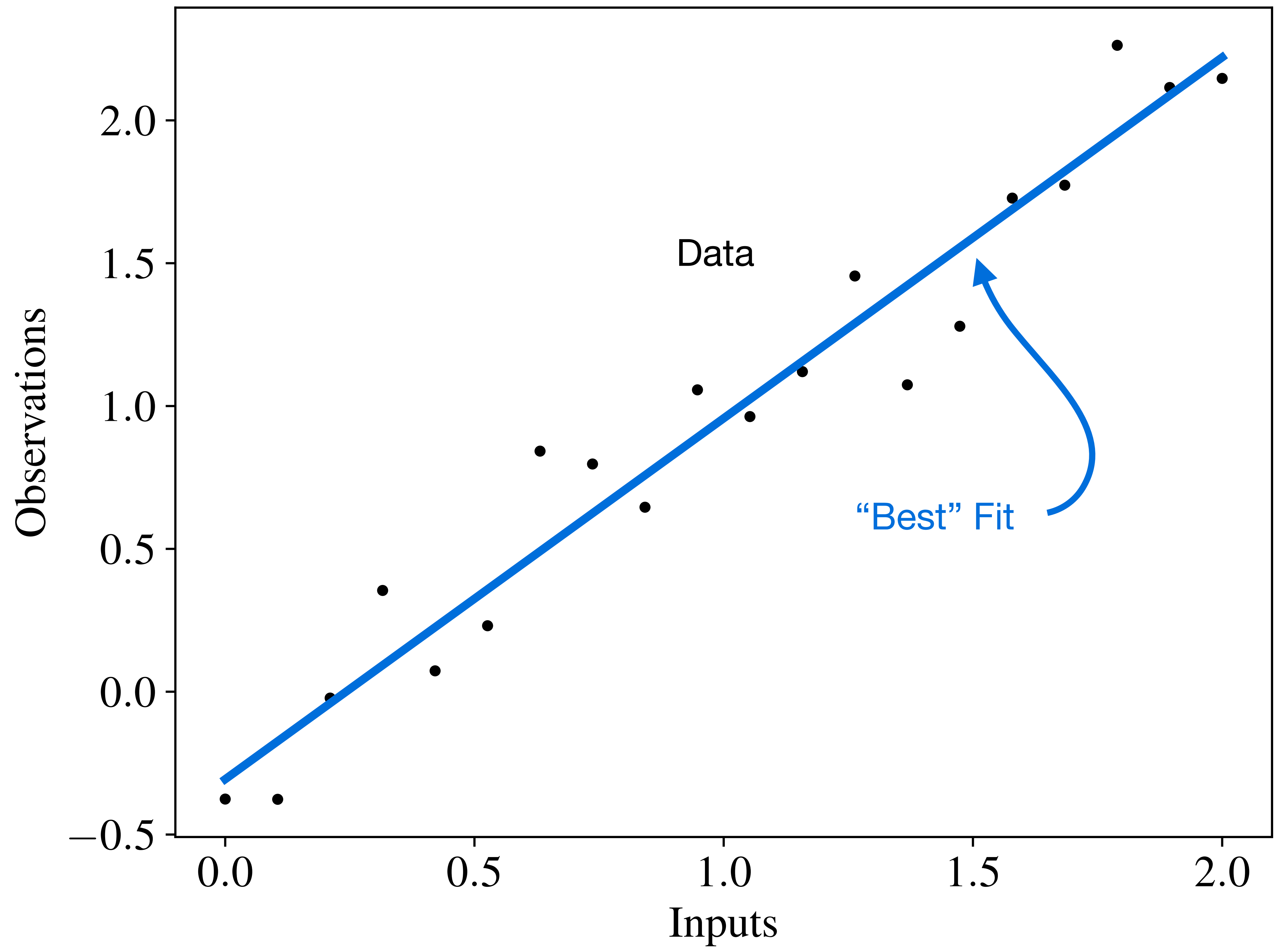
$$\mathscr{L}(w_0, w_1) = \frac{1}{N}\sum_{i=1}^{N}[y_i - (w_0 + w_1 x_i)]^2$$

- Best model is one that minimizes the loss, can derive this analytically for linear least squares loss

$$\frac{\partial \mathscr{L}}{\partial w_0} = 0 \implies w_0 = \bar{y} - w_1 \bar{x}$$

$$\frac{\partial \mathscr{L}}{\partial w_1} = 0 \implies w_1 = \frac{\overline{xy} - \bar{x}\bar{y}}{\overline{x^2} - \bar{x}^2}$$

How do we find the "best" model?



Learning nonlinear responses with linear model

- In general, linear model only needs to be linear in the parameters

$$\hat{f}(x) = w_0 h_0(x) + w_1 h_1(x) + w_2 h_2(x) + \dots$$

- We can write this compactly as

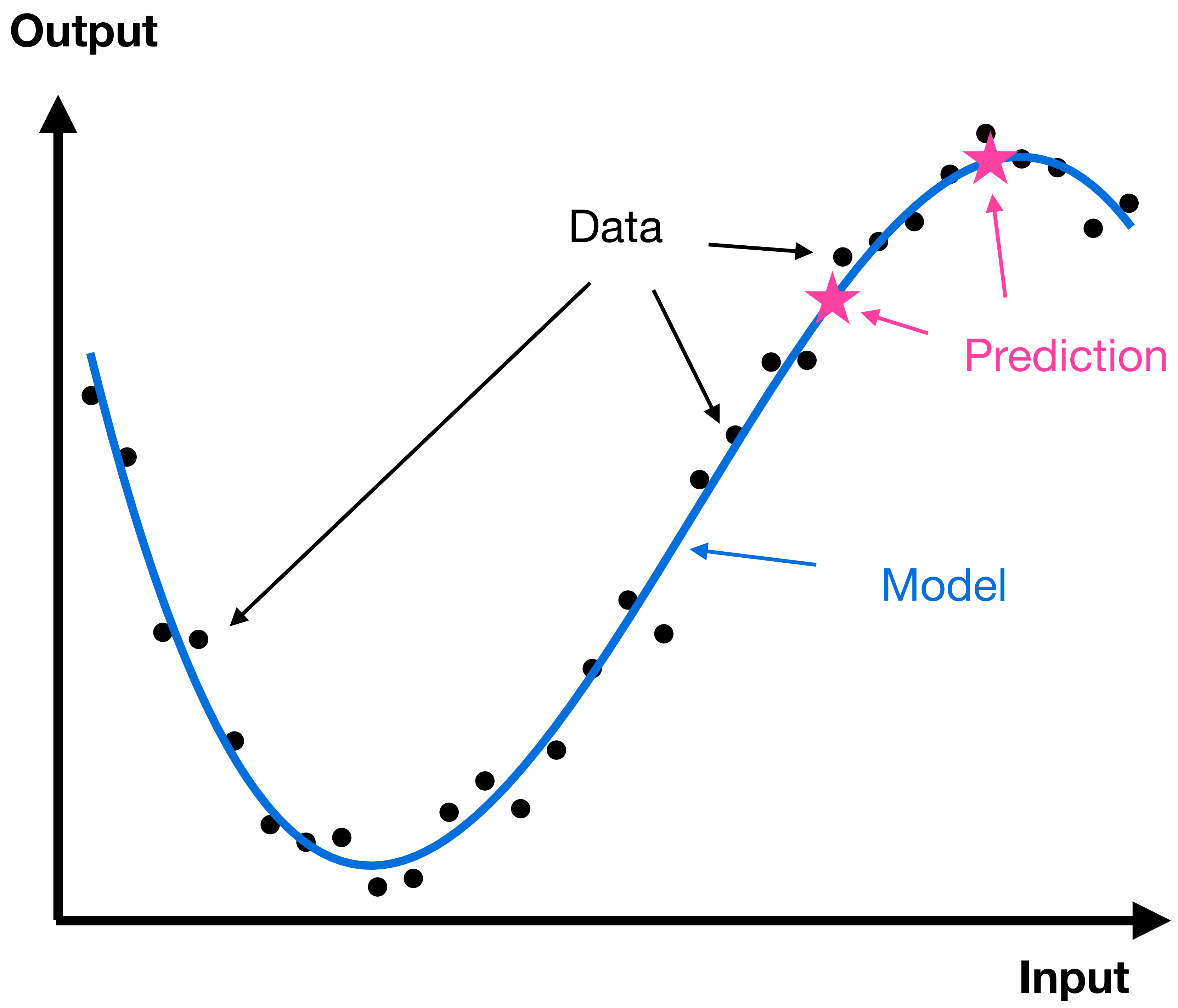
$$\hat{f}(x) = \mathbf{w} \cdot \mathbf{h}(x), \quad \mathbf{w} = [w_0, w_1, w_2, \dots]^T, \quad \mathbf{h}(x) = [h_0(x), h_1(x), h_2(x), \dots]^T$$

- This leads to a least-squares loss

$$\mathscr{L}(\mathbf{w}) = \frac{1}{N} \|\mathbf{y} - \mathbf{H}\mathbf{w}\|_2^2, \quad \mathbf{y} = [y_0, \dots, y_N]^T, \quad \mathbf{H} = [\mathbf{h}(x_0), \dots, \mathbf{h}(x_N)]$$

- Minimizing the loss leads to model of best fit

$$\mathbf{w} = (\mathbf{H}^T \mathbf{H})^{-1} \mathbf{H}^T \mathbf{y}$$



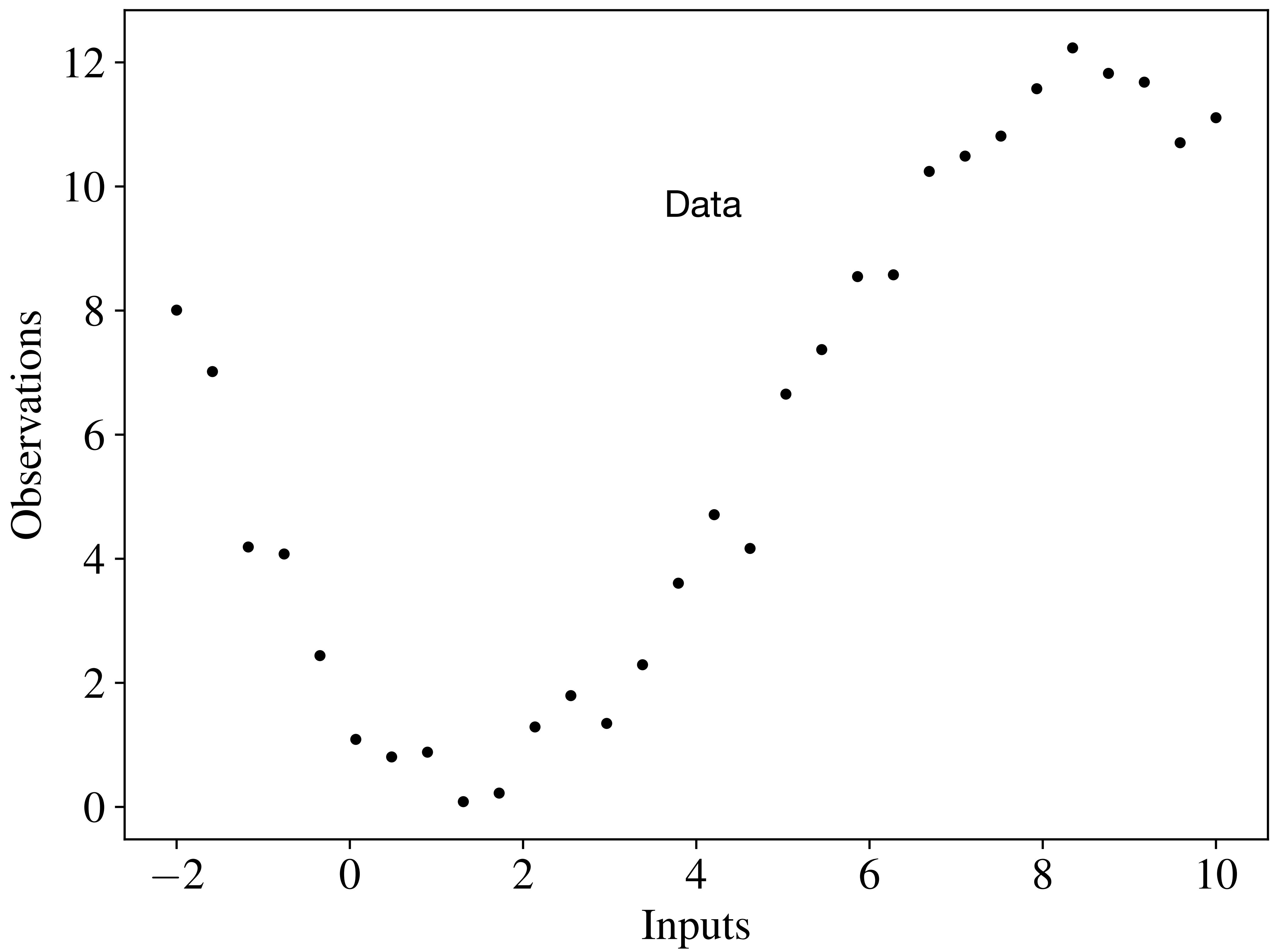
Polynomial regression

- *Polynomial regression* is a linear problem!
  (think in terms of the weights)

  $$\hat{f}(\mathbf{w}) = \mathbf{w} \cdot \mathbf{h}(x), \quad h_k(x) = x^k$$

- Using a least-squares loss function, we obtain

  $$\mathbf{w} = (\mathbf{H}^T\mathbf{H})\mathbf{H}^T\mathbf{y}$$



Polynomial regression

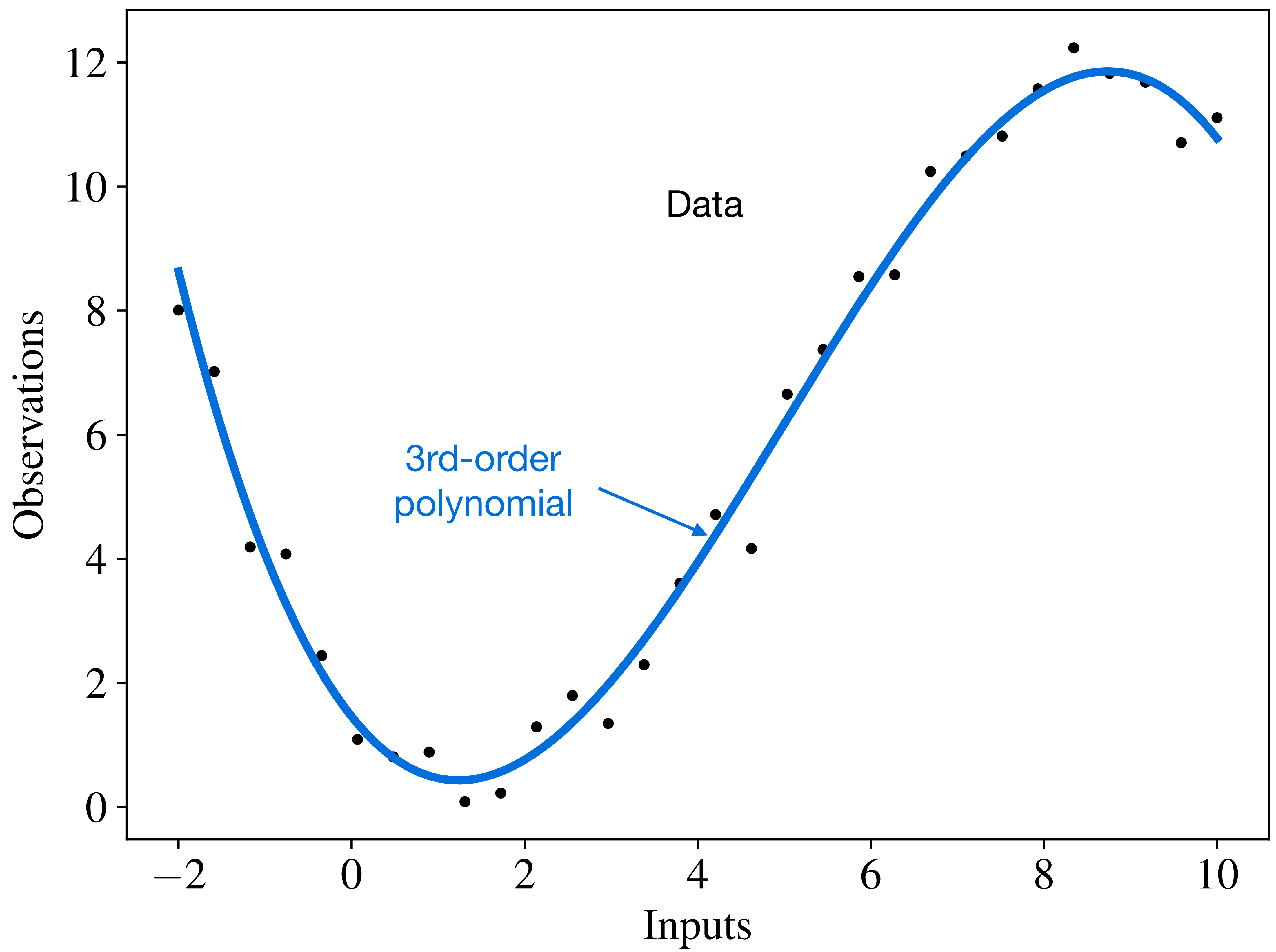
- *Polynomial regression* is a linear problem!
  (think in terms of the weights)

$$\hat{f}(\mathbf{w}) = \mathbf{w} \cdot \mathbf{h}(x), \quad h_k(x) = x^k$$

- Using a least-squares loss function, we obtain

$$\mathbf{w} = (\mathbf{H}^T\mathbf{H})\mathbf{H}^T\mathbf{y}$$

- **Model assumption**: response follows a third-order polynomial



Polynomial regression

- *Polynomial regression* is a linear problem!
  (think in terms of the weights)

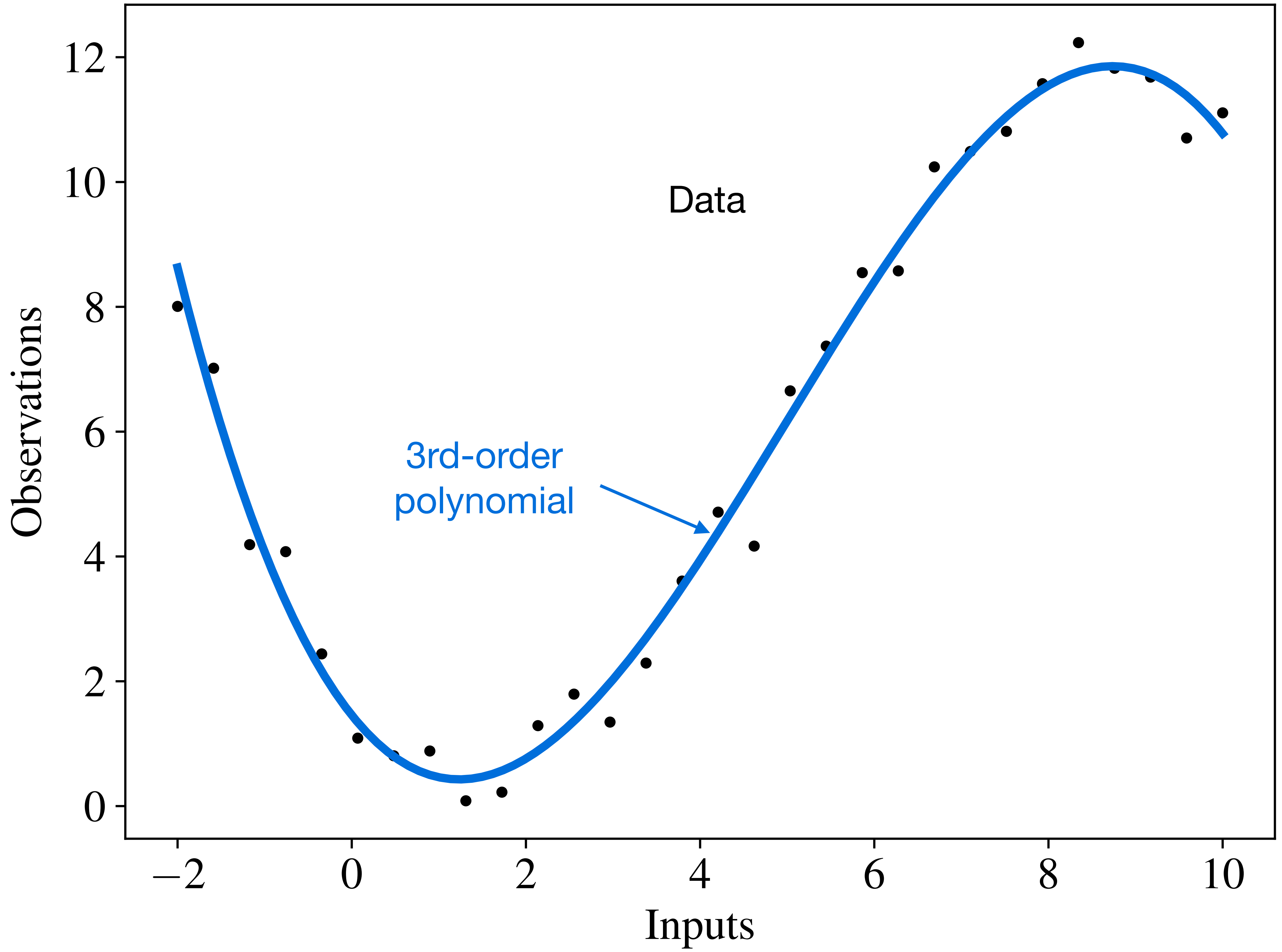
$$\hat{f}(\mathbf{w}) = \mathbf{w} \cdot \mathbf{h}(x), \quad h_k(x) = x^k$$

- Using a least-squares loss function, we obtain

$$\mathbf{w} = (\mathbf{H}^T\mathbf{H})\mathbf{H}^T\mathbf{y}$$

- **Model assumption**: response follows a third-order polynomial

- Looks great! But if a 3rd-order is good, why not 12th-order?



Polynomial regression

- *Polynomial regression* is a linear problem!
  (think in terms of the weights)

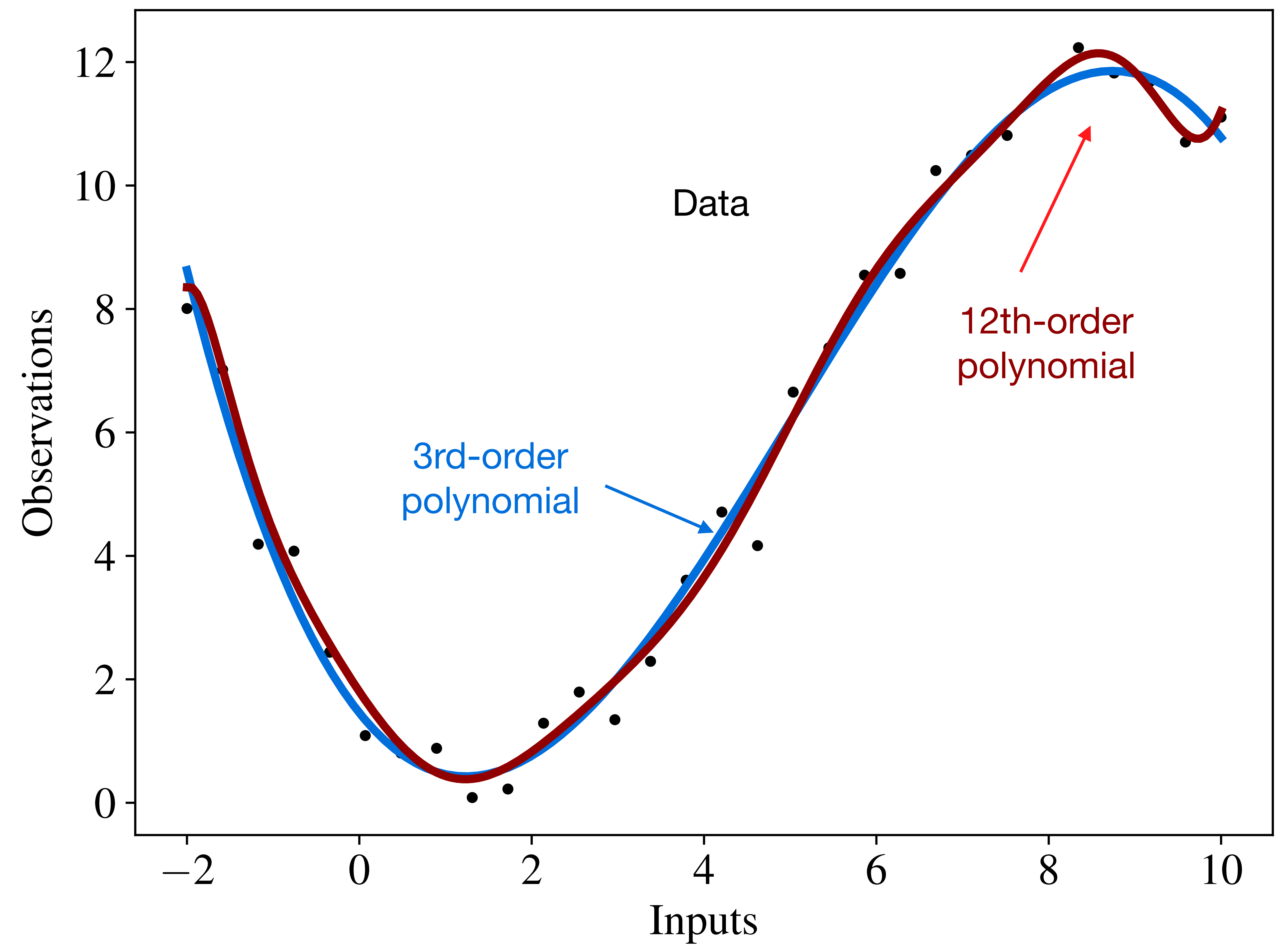
$$\hat{f}(\mathbf{w}) = \mathbf{w} \cdot \mathbf{h}(x), \quad h_k(x) = x^k$$

- Using a least-squares loss function, we obtain

$$\mathbf{w} = (\mathbf{H}^T\mathbf{H})\mathbf{H}^T\mathbf{y}$$

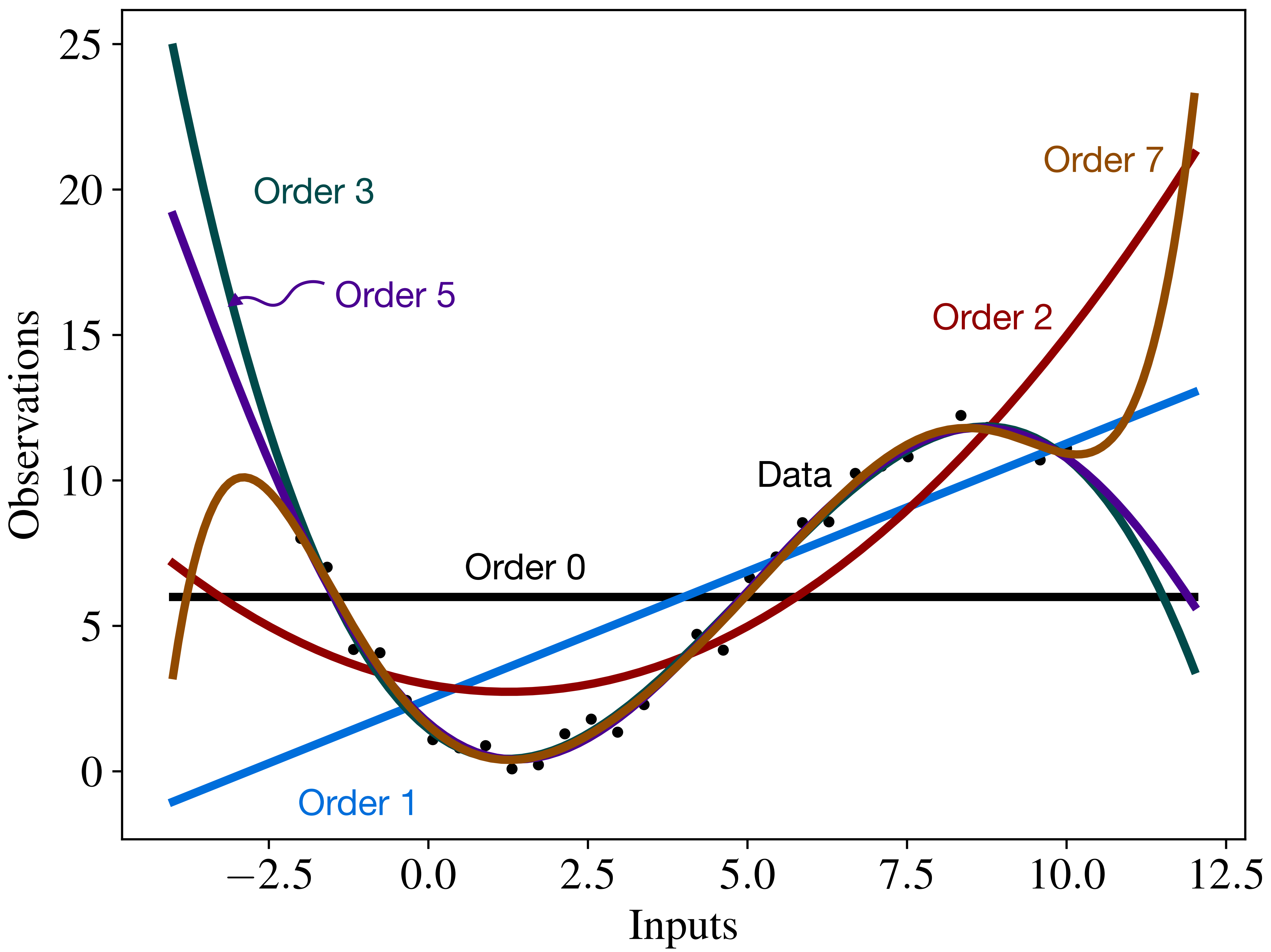
- **Model assumption**: response follows a third-order polynomial

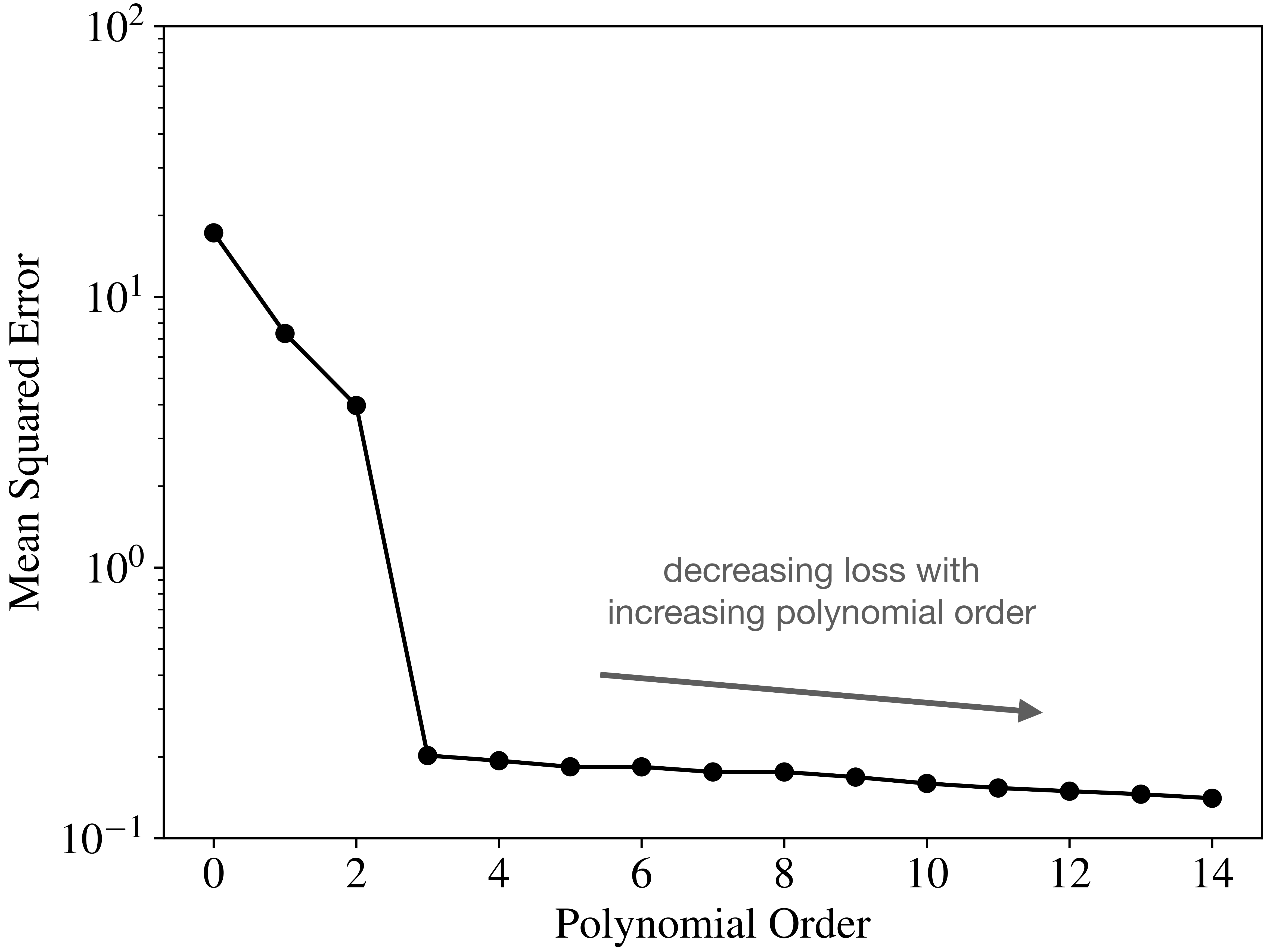
- Looks great! But if a 3rd-order is good, why not 12th-order?



Generalization and overfitting

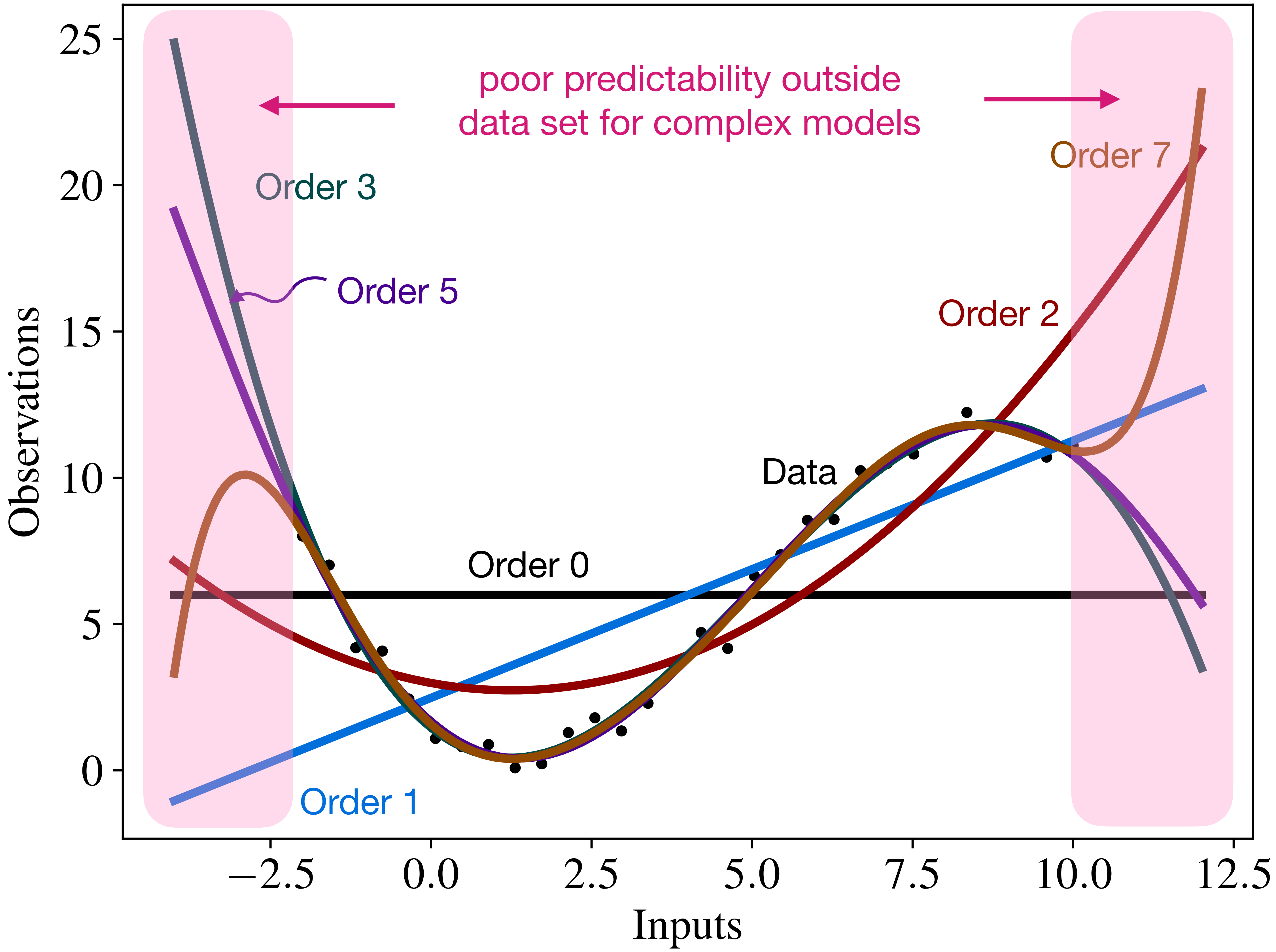
Least-squares loss favors model complexity over predictability!

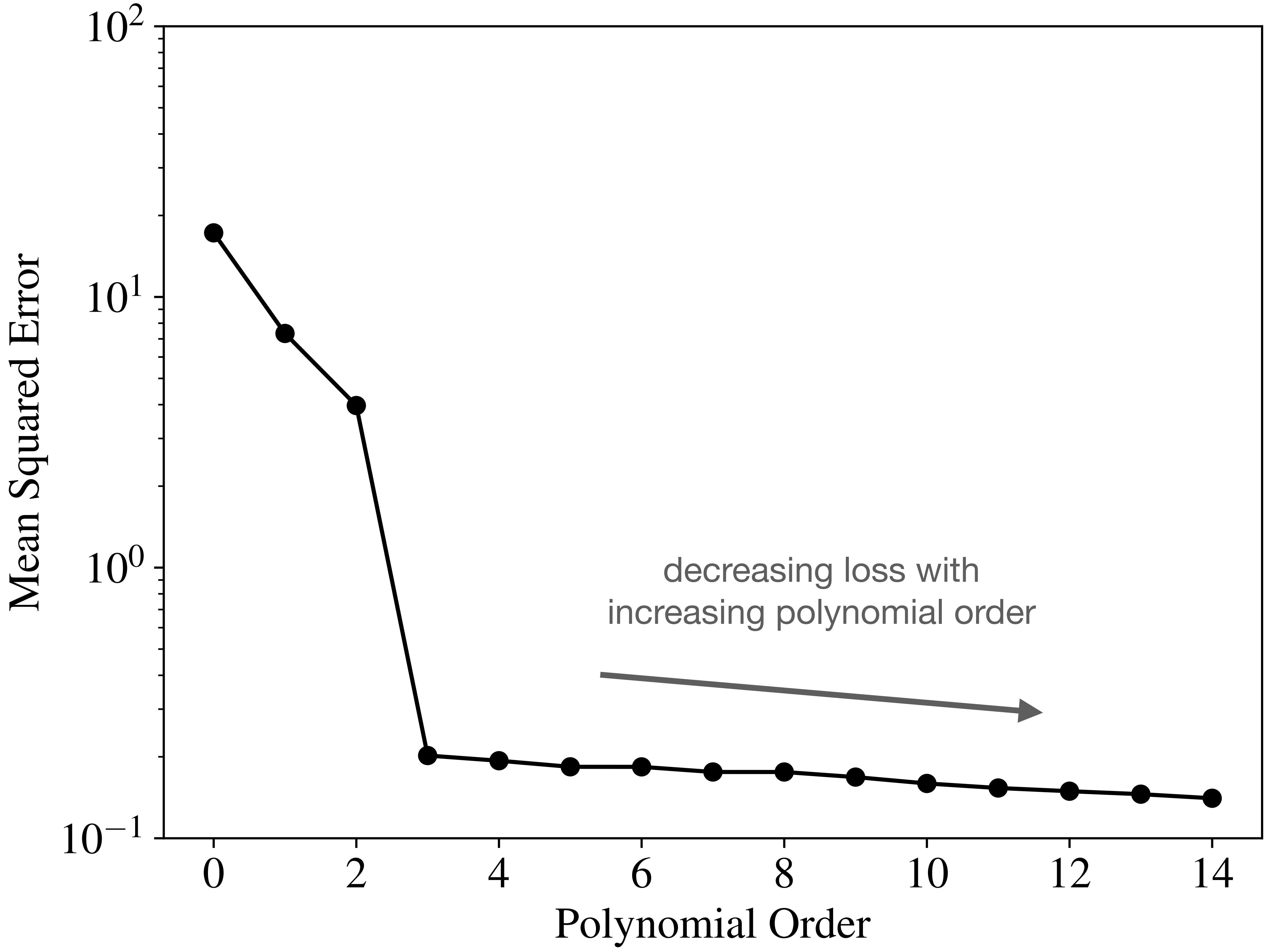




Generalization and overfitting

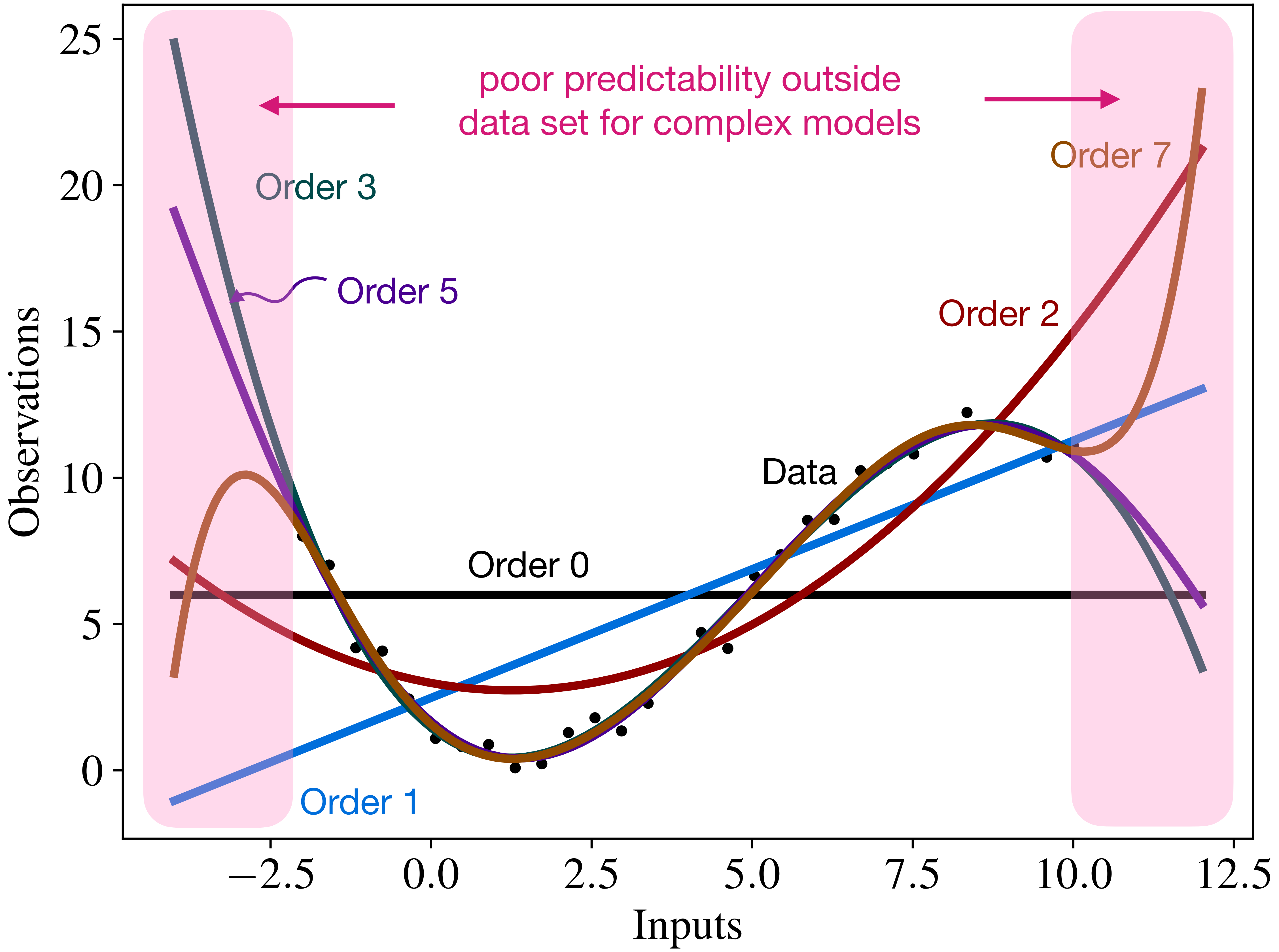
Least-squares loss favors model complexity over predictability!

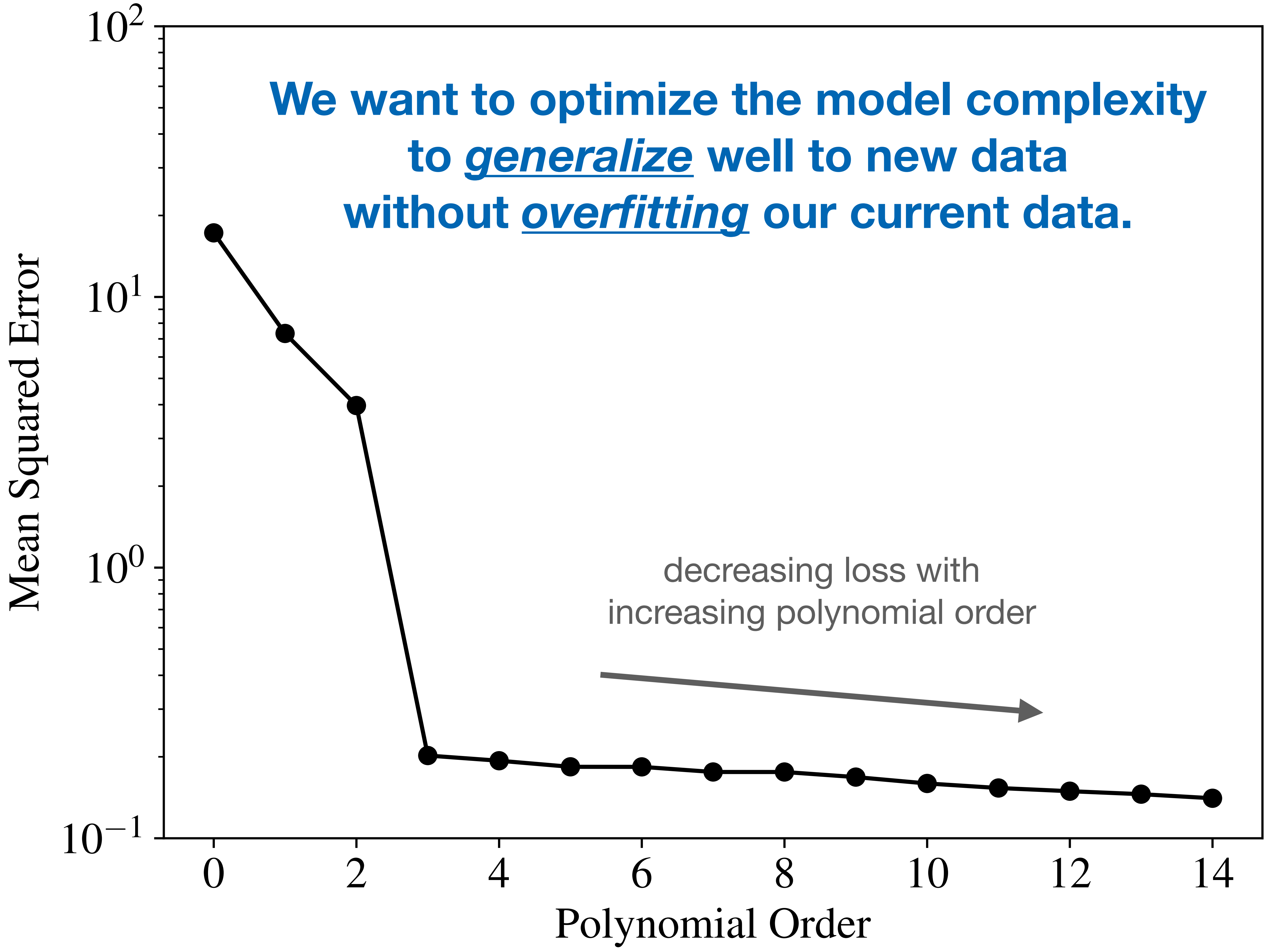




Generalization and overfitting

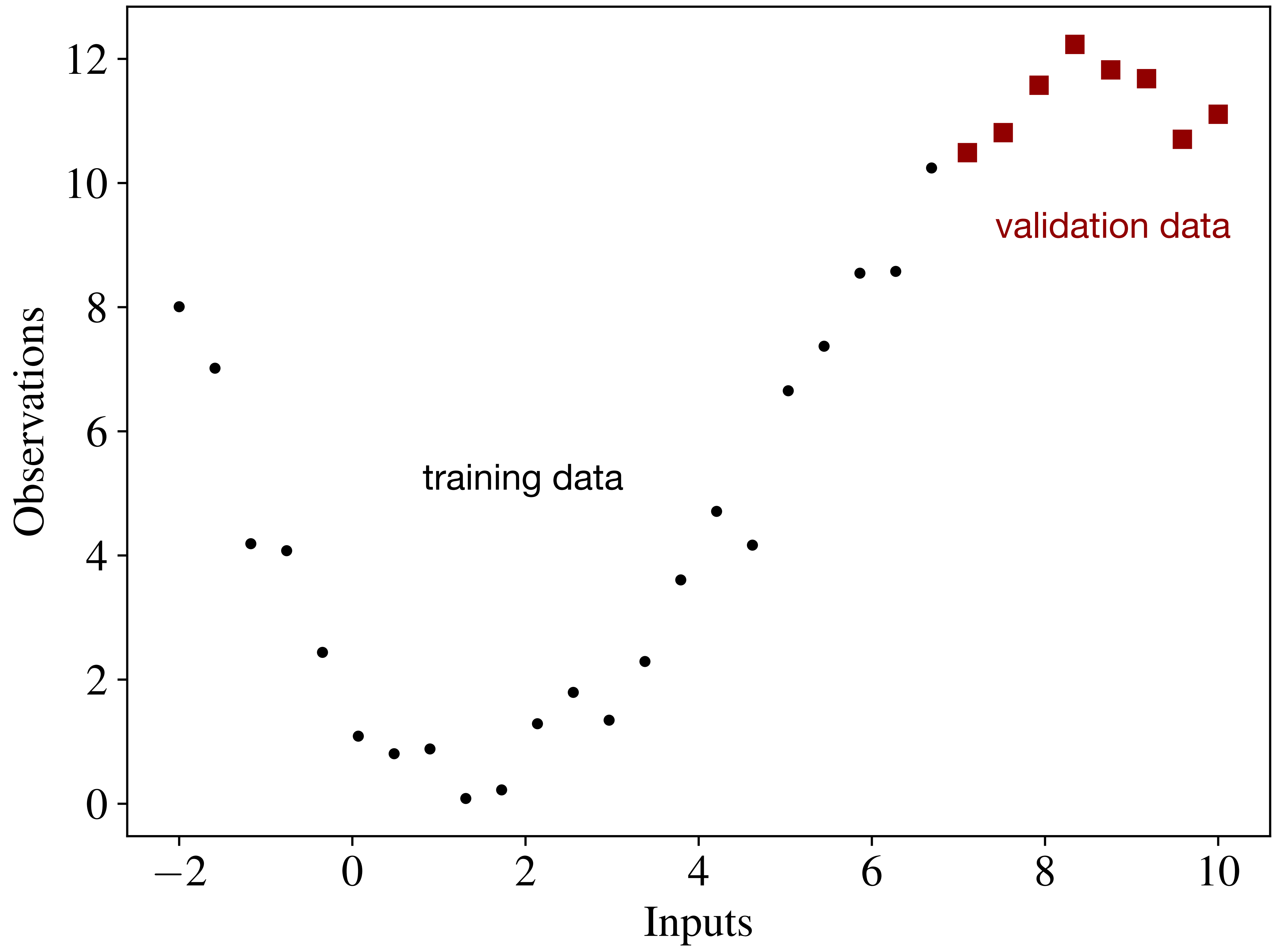
Least-squares loss favors model complexity over predictability!

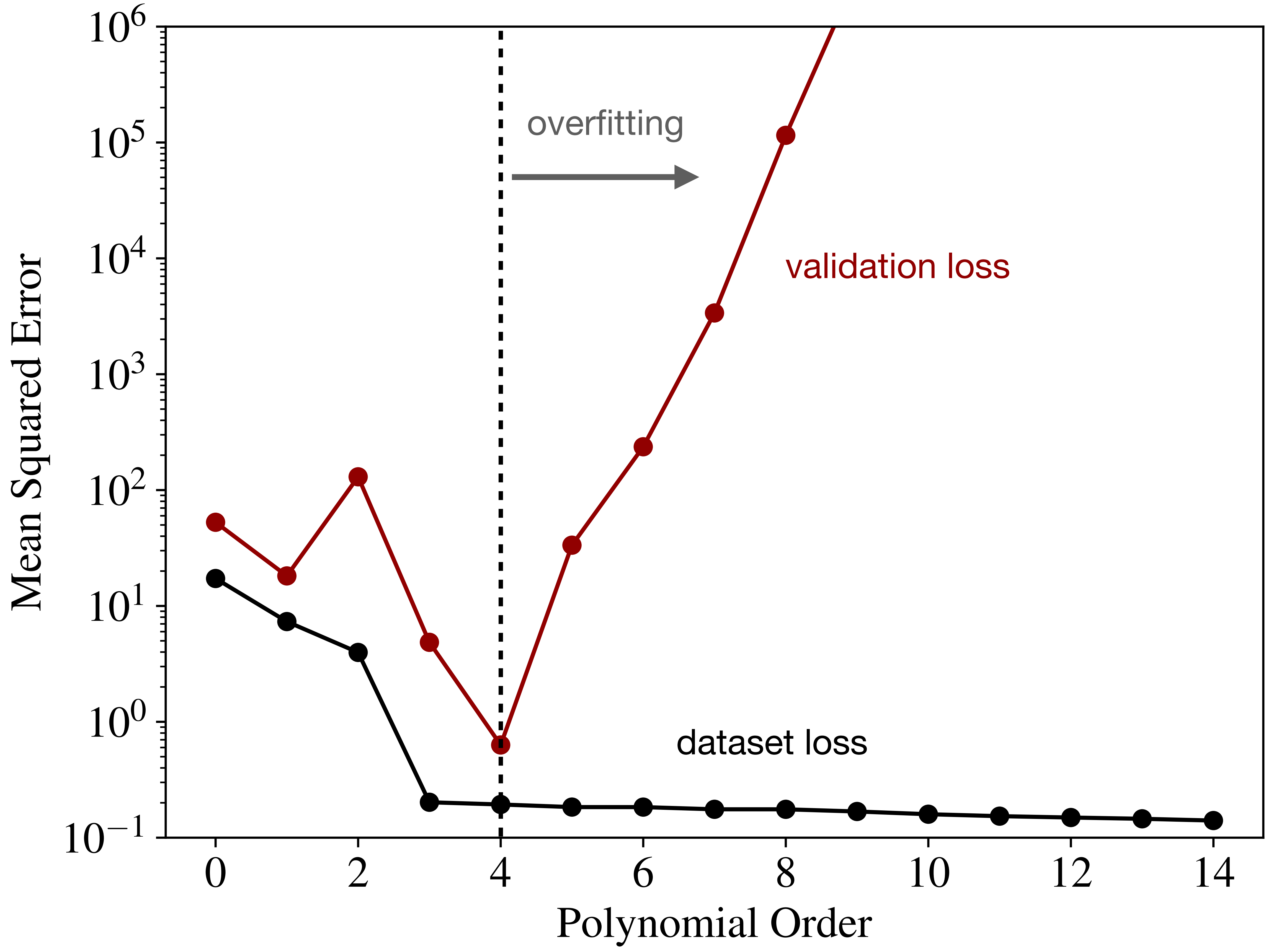




Validation data

Idea: hold back some *validation data* as a surrogate for unseen data to check model's generalizability





Cross-validation

Cross-validation

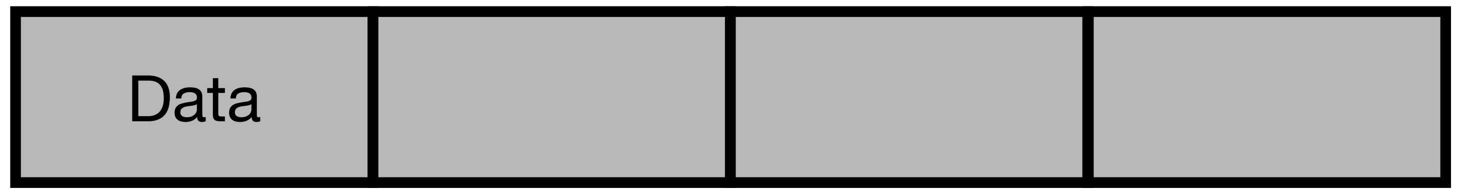
- Validation loss is sensitive to which data we choose to hold back
- Can improve on this idea by taking the average validation loss over multiple choices of train/validation sets

Cross-validation

- Validation loss is sensitive to which data we choose to hold back
- Can improve on this idea by taking the average validation loss over multiple choices of train/validation sets
- **K-Fold Cross-Validation (CV)**
  1. Split dataset into K equal parts
  2. For each part, train model on remaining K-1 parts and compute validation loss w.r.t. part K
  3. Average validation loss over all K parts

Cross-validation

- Validation loss is sensitive to which data we choose to hold back
- Can improve on this idea by taking the average validation loss over multiple choices of train/validation sets
- **K-Fold Cross-Validation (CV)**
    1. Split dataset into K equal parts
    2. For each part, train model on remaining K-1 parts and compute validation loss w.r.t. part K
    3. Average validation loss over all K parts



$K = 4$

Cross-validation

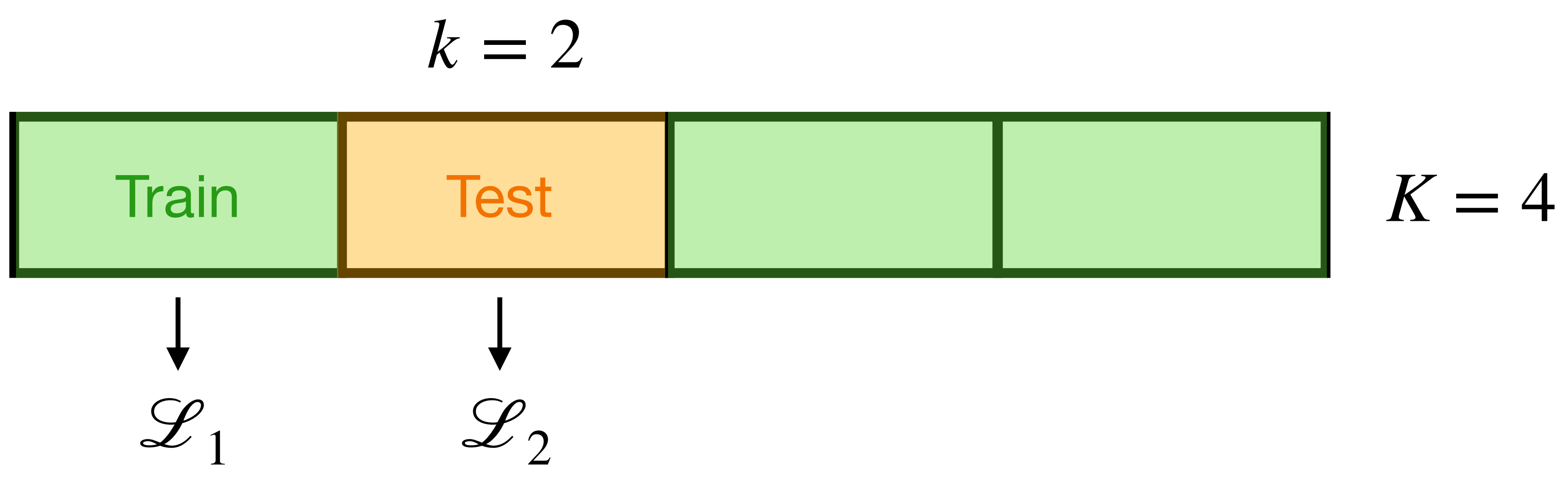
- Validation loss is sensitive to which data we choose to hold back
- Can improve on this idea by taking the average validation loss over multiple choices of train/validation sets
- **K-Fold Cross-Validation (CV)**

  1. Split dataset into K equal parts
  2. For each part, train model on remaining K-1 parts and compute validation loss w.r.t. part K
  3. Average validation loss over all K parts

Cross-validation

- Validation loss is sensitive to which data we choose to hold back
- Can improve on this idea by taking the average validation loss over multiple choices of train/validation sets
- **K-Fold Cross-Validation (CV)**

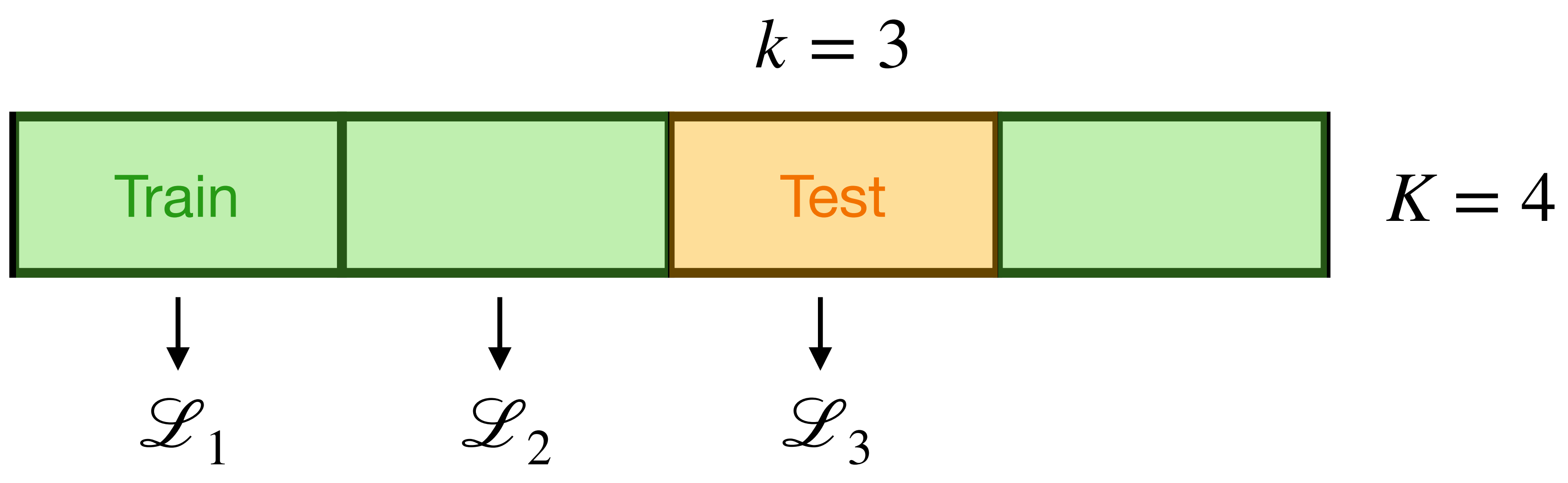
1. Split dataset into K equal parts
2. For each part, train model on remaining K-1 parts and compute validation loss w.r.t. part K
3. Average validation loss over all K parts



Cross-validation

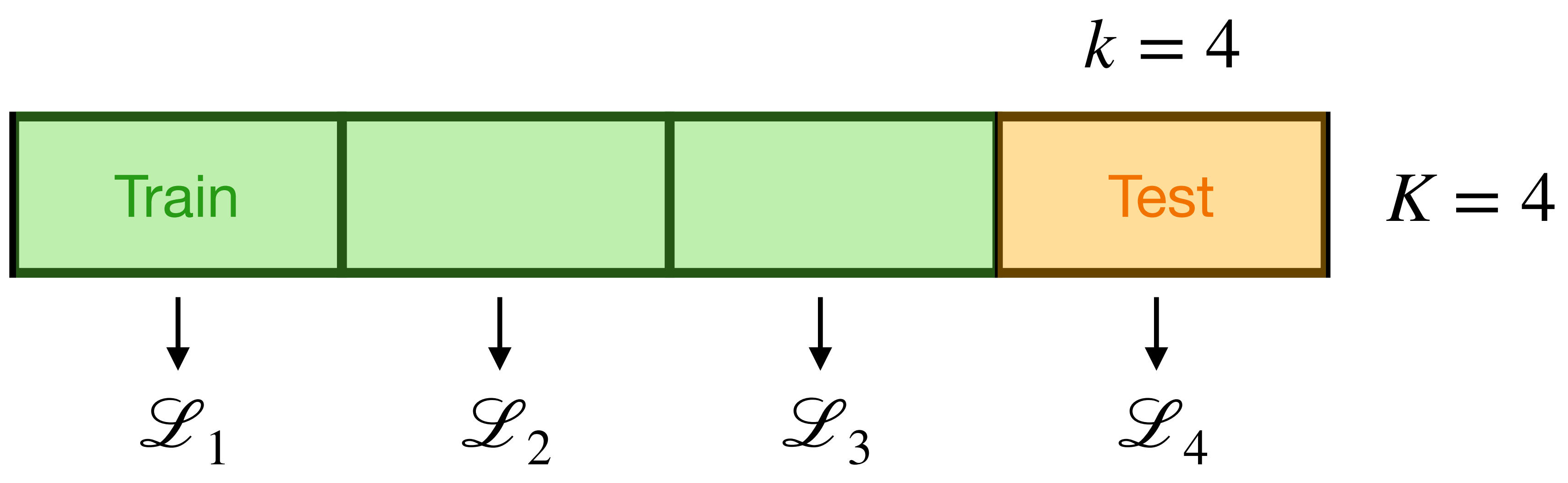
- Validation loss is sensitive to which data we choose to hold back
- Can improve on this idea by taking the average validation loss over multiple choices of train/validation sets
- **K-Fold Cross-Validation (CV)**

1. Split dataset into K equal parts
2. For each part, train model on remaining K-1 parts and compute validation loss w.r.t. part K
3. Average validation loss over all K parts



Cross-validation

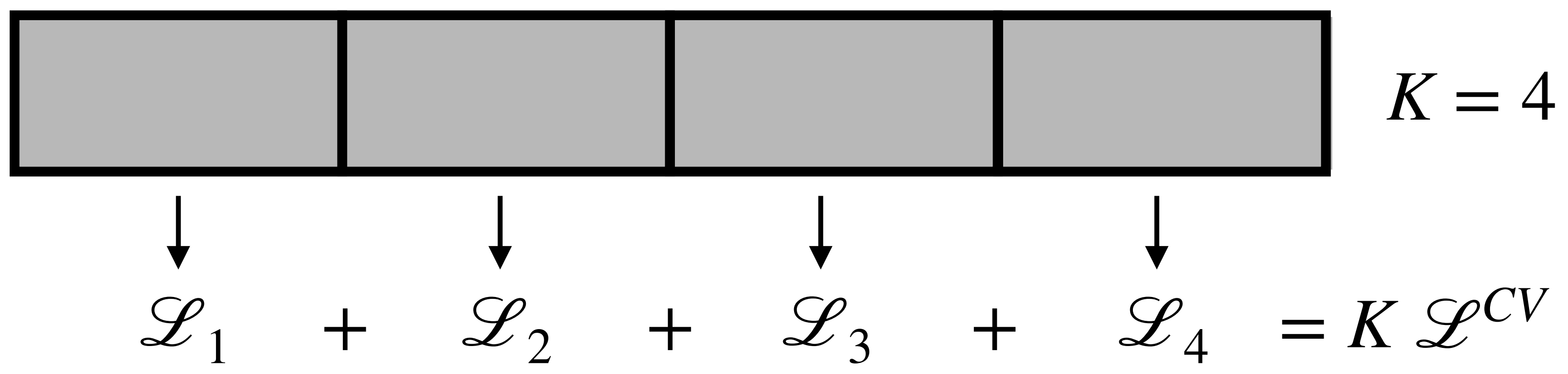
- Validation loss is sensitive to which data we choose to hold back
- Can improve on this idea by taking the average validation loss over multiple choices of train/validation sets
- **K-Fold Cross-Validation (CV)**
  1. Split dataset into K equal parts
  2. For each part, train model on remaining K-1 parts and compute validation loss w.r.t. part K
  3. Average validation loss over all K parts



Cross-validation

- Validation loss is sensitive to which data we choose to hold back
- Can improve on this idea by taking the average validation loss over multiple choices of train/validation sets
- **K-Fold Cross-Validation (CV)**

1. Split dataset into K equal parts
2. For each part, train model on remaining K-1 parts and compute validation loss w.r.t. part K
3. Average validation loss over all K parts



Cross-validation

- Validation loss is sensitive to which data we choose to hold back
- Can improve on this idea by taking the average validation loss over multiple choices of train/validation sets
- **K-Fold Cross-Validation (CV)**
  1. Split dataset into K equal parts
  2. For each part, train model on remaining K-1 parts and compute validation loss w.r.t. part K
  3. Average validation loss over all K parts
- **Leave-One-Out Cross-Validation (LOOCV)**
  - Special case of K-Fold CV where K is number of data points

$$\mathscr{L}_1 + \mathscr{L}_2 + \mathscr{L}_3 + \mathscr{L}_4 = K\,\mathscr{L}^{CV}$$

$$\mathscr{L}^{CV} = \frac{1}{N}\sum_{i=1}^{N} l(y_i, \hat{f}(x_i; \theta^*_{-i}))$$

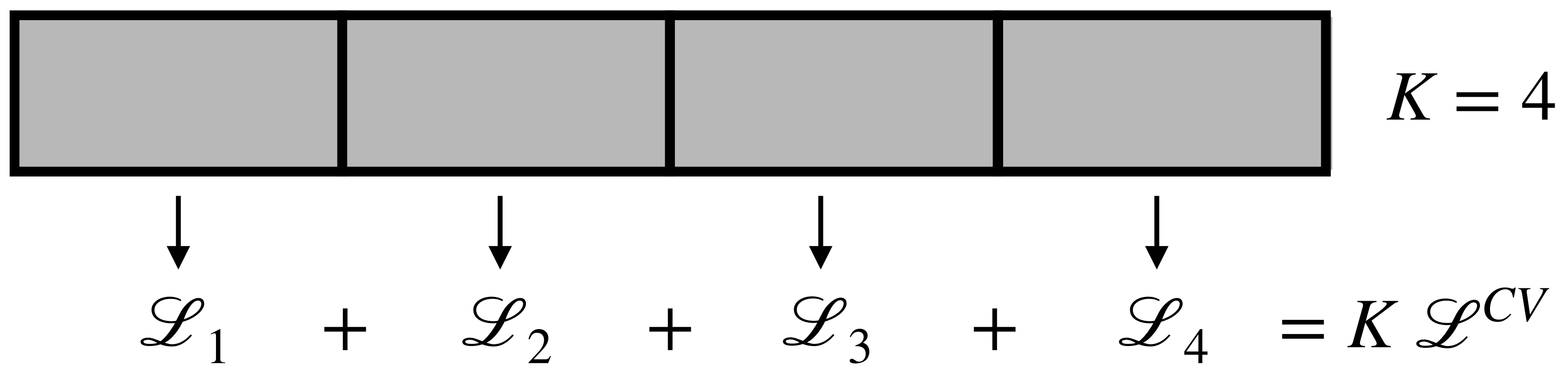
trained parameters on data without i^{th} point

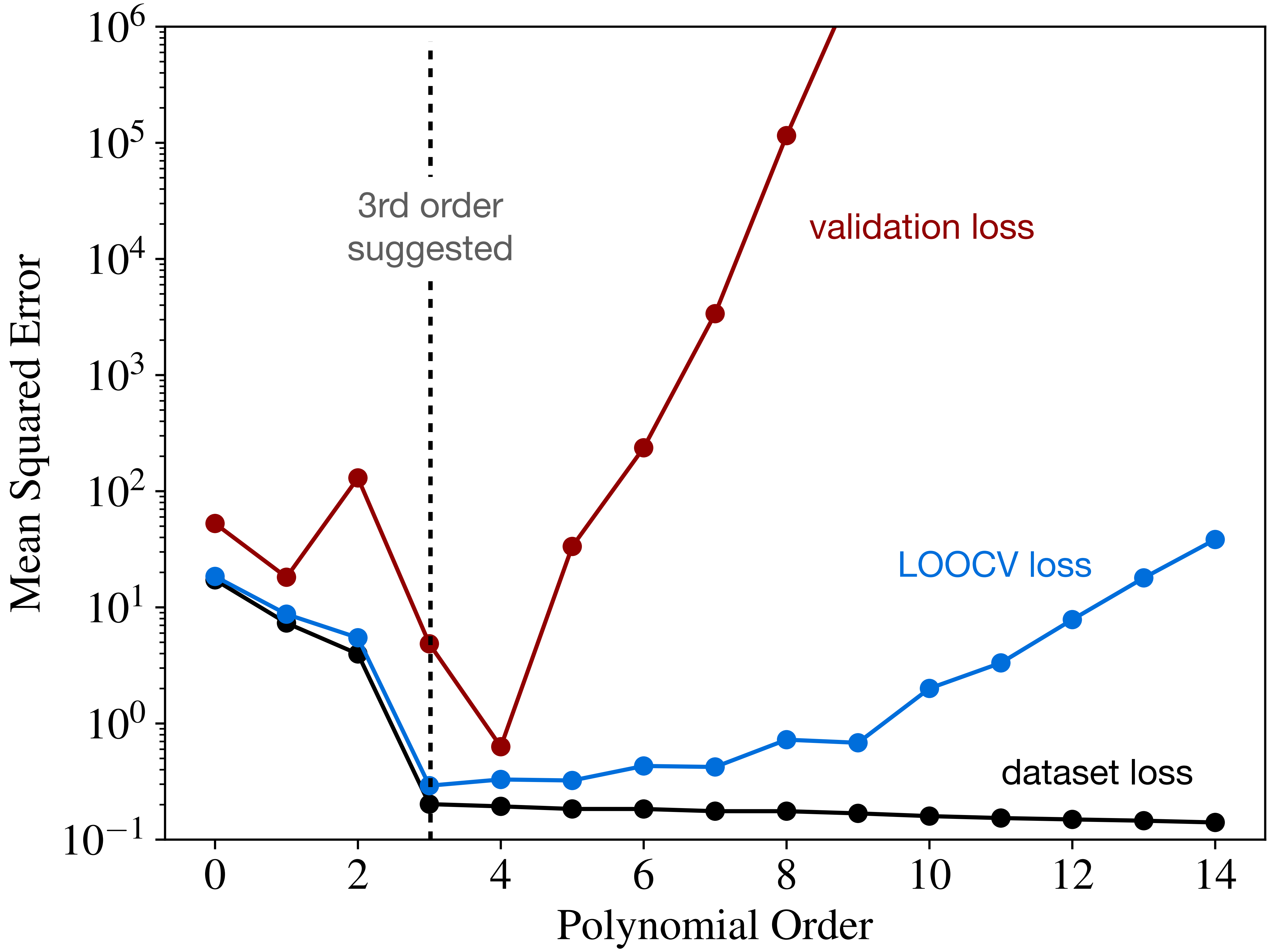
Cross-validation

- Validation loss is sensitive to which data we choose to hold back
- Can improve on this idea by taking the average validation loss over multiple choices of train/validation sets
- **K-Fold Cross-Validation (CV)**
  1. Split dataset into K equal parts
  2. For each part, train model on remaining K-1 parts and compute validation loss w.r.t. part K
  3. Average validation loss over all K parts
- **Leave-One-Out Cross-Validation (LOOCV)**
  - Special case of K-Fold CV where K is number of data points

$$\mathscr{L}^{CV} = \frac{1}{N}\sum_{i=1}^{N} l(y_i, \hat{f}(x_i; \theta^*_{-i}))$$

trained parameters on data without i^{th} point





Regularization

Regularization **improves generalizability by penalizing model complexity in the loss function**

Regularized Linear Least-Squares

- Least complex model with $\mathbf{w} = \mathbf{0}$
- "Complexity" increases as parameters become more nonzero
- **Idea:** Add sum of parameters squared to loss

$$\mathscr{L}(\mathbf{w}) = \frac{1}{N}\|\mathbf{y} - \mathbf{H}\mathbf{w}\|_2^2 + \lambda \mathbf{w}^T\mathbf{w}, \quad \lambda \geq 0$$

least-squares loss

regularization

- Minimizing regularized loss leads to

$$\mathbf{w} = (\mathbf{X}^T\mathbf{X} + N\lambda\mathbf{I})^{-1}\mathbf{X}^T\mathbf{y}$$

Regularization

Regularization **improves generalizability by penalizing model complexity in the loss function**

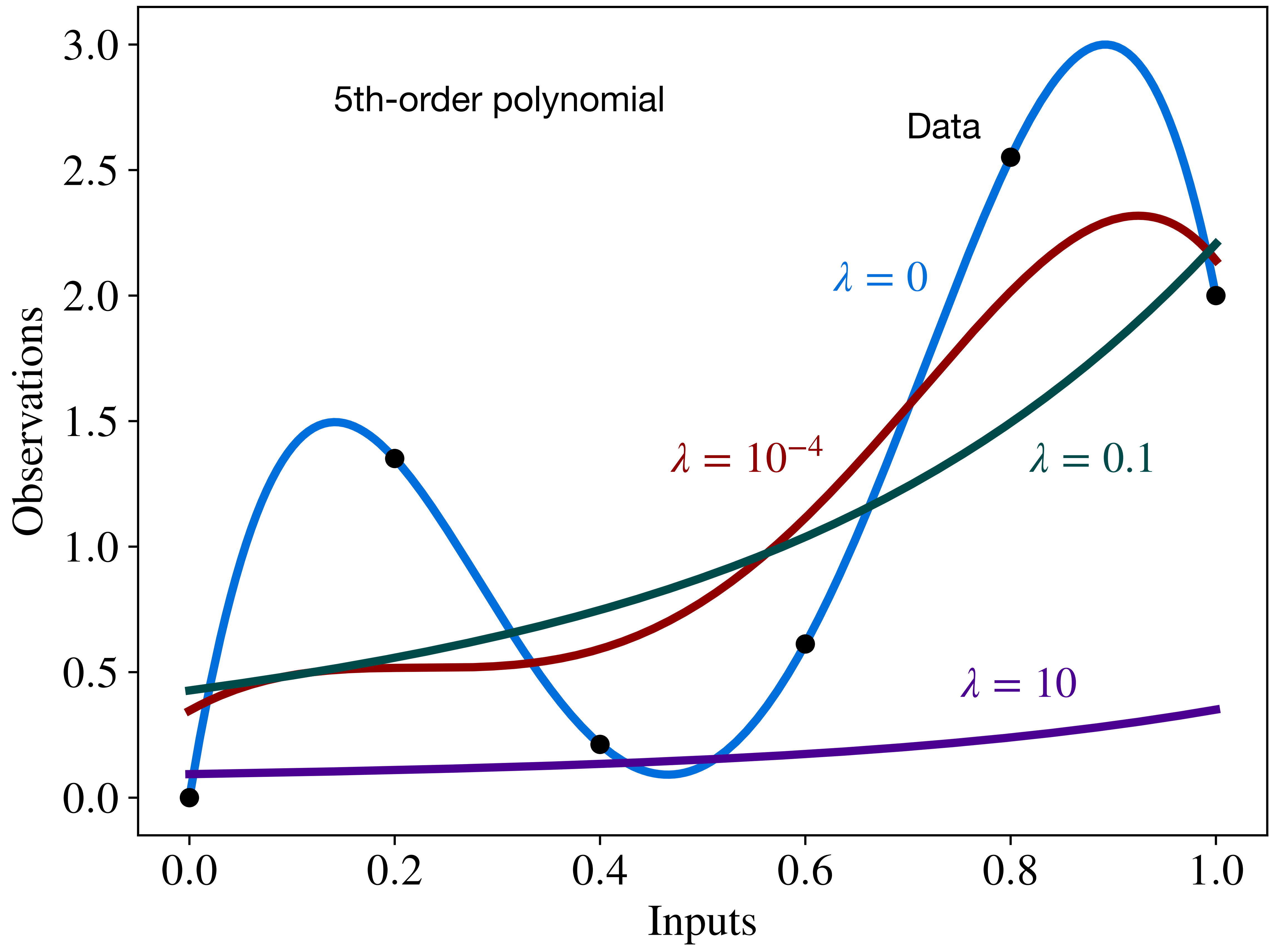
Regularized Linear Least-Squares

- Least complex model with $\mathbf{w} = \mathbf{0}$
- "Complexity" increases as parameters become more nonzero
- **Idea:** Add sum of parameters squared to loss

$$\mathscr{L}(\mathbf{w}) = \underbrace{\frac{1}{N}\|\mathbf{y} - \mathbf{H}\mathbf{w}\|_2^2}_{\text{least-squares loss}} + \underbrace{\lambda \mathbf{w}^T\mathbf{w}}_{\text{regularization}}, \quad \lambda \geq 0$$

- Minimizing regularized loss leads to

$$\mathbf{w} = (\mathbf{X}^T\mathbf{X} + N\lambda\mathbf{I})^{-1}\mathbf{X}^T\mathbf{y}$$



Normalization

Generally a good idea to *normalize* dataset prior to model training!

Examples:

- "Standard" normalization centers and scales to unit variance

$$\hat{x} = \frac{x - \bar{x}}{\sigma_x}$$

- "Min-Max" transforms data into range of [0,1]

$$\hat{x} = \frac{x - \min x}{\max x - \min x}$$

Notes:

- Choice depends on model, algorithm, data
- Perform on inputs and outputs
- Remember to denormalize predictions!

Normalization

Generally a good idea to *normalize* dataset prior to model training!

Examples:

- "Standard" normalization centers and scales to unit variance

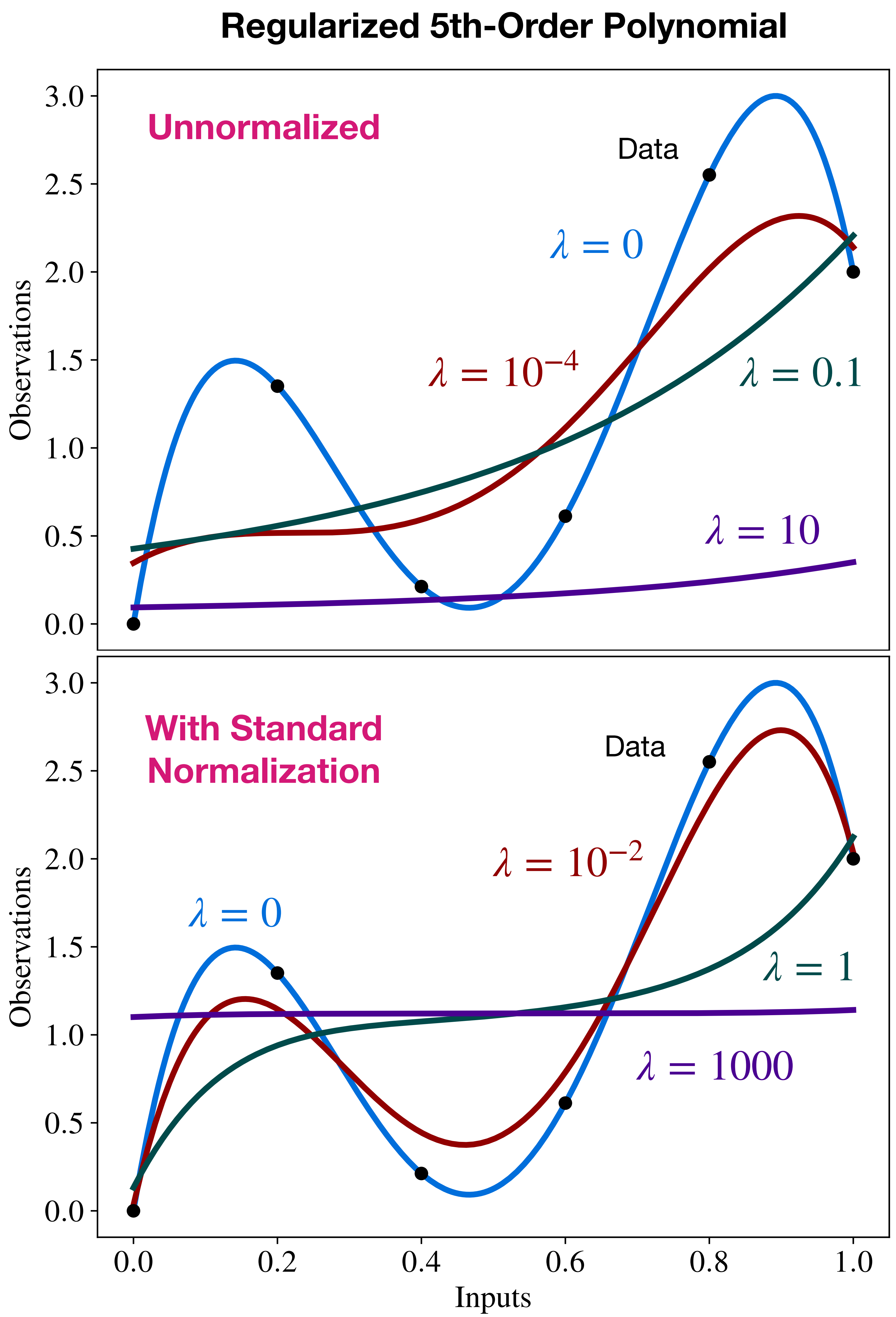
$$\hat{x} = \frac{x - \bar{x}}{\sigma_x}$$

- "Min-Max" transforms data into range of [0,1]

$$\hat{x} = \frac{x - \min x}{\max x - \min x}$$

Notes:

- Choice depends on model, algorithm, data
- Perform on inputs and outputs
- Remember to denormalize predictions!

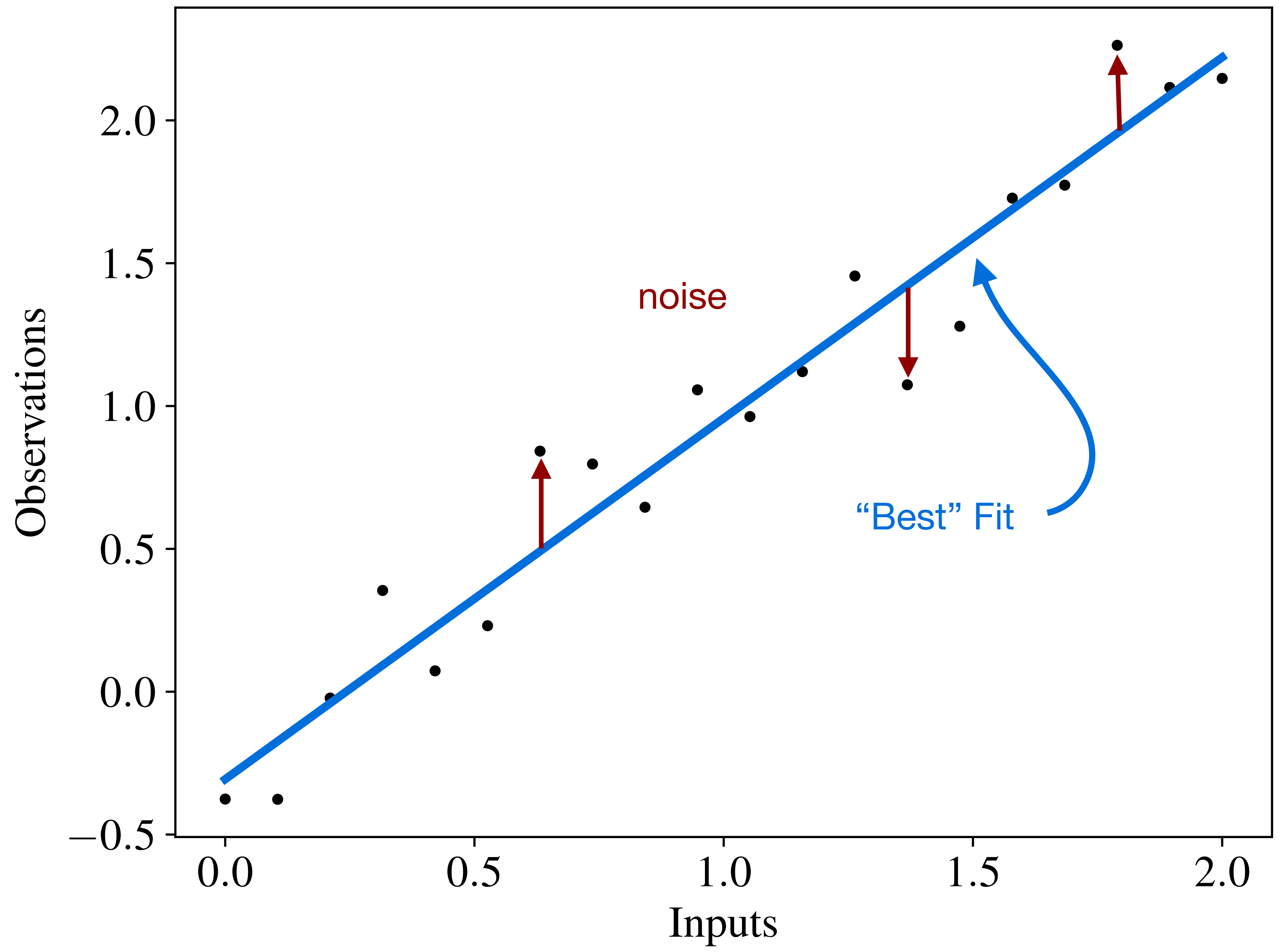


Summary of the basic ML pipeline

1. Gather data
    - Often the most challenging part of ML!
    - Study, plot, reason about, clean up, etc.
    - Ensure dataset covers all potential outcomes and is evenly weighted
2. Normalize dataset
3. Split into train, validation, and test datasets (shuffle)
4. Perform model selection / hyper parameter tuning
    - Look to maximize generalizability and prevent overfitting
    - Cross-validation
5. Train chosen model(s)
6. Deploy model
    - Monitor performance and go back to (1) if needed

What about noise?

- So far, we have neglected the *noise* in our data
- Noise represents *uncertainty* or *randomness* in the *generating process* used to create the data
  - *Latent* (hidden) variables
  - Measurement uncertainties
  - Model uncertainties (for derived data)
- From a modeling perspective, noise represents potential error in our model, because we are using imperfect data
- Interested in knowing the uncertainty in our model predictions
- **Not a course on *Uncertainty Quantification (UQ)*:** instead we will try to get a flavor of the ideas involved



Thinking generatively

Data generation is an inherently complex process!

- We can try to model this process by approaching the supervised learning task in a new way
    - Instead of looking for model that best fits the data,
    - Look for model that is most likely to generate that data
    - In general, these types of models are called _generative models_

How can build a model that can generate data that “looks” like ours?

- Obviously, we accept that this isn’t the real generating process
- However, this will be a useful strategy
- **Key Idea: Add randomness to our model that mimics the randomness present in the data**

A generative linear model

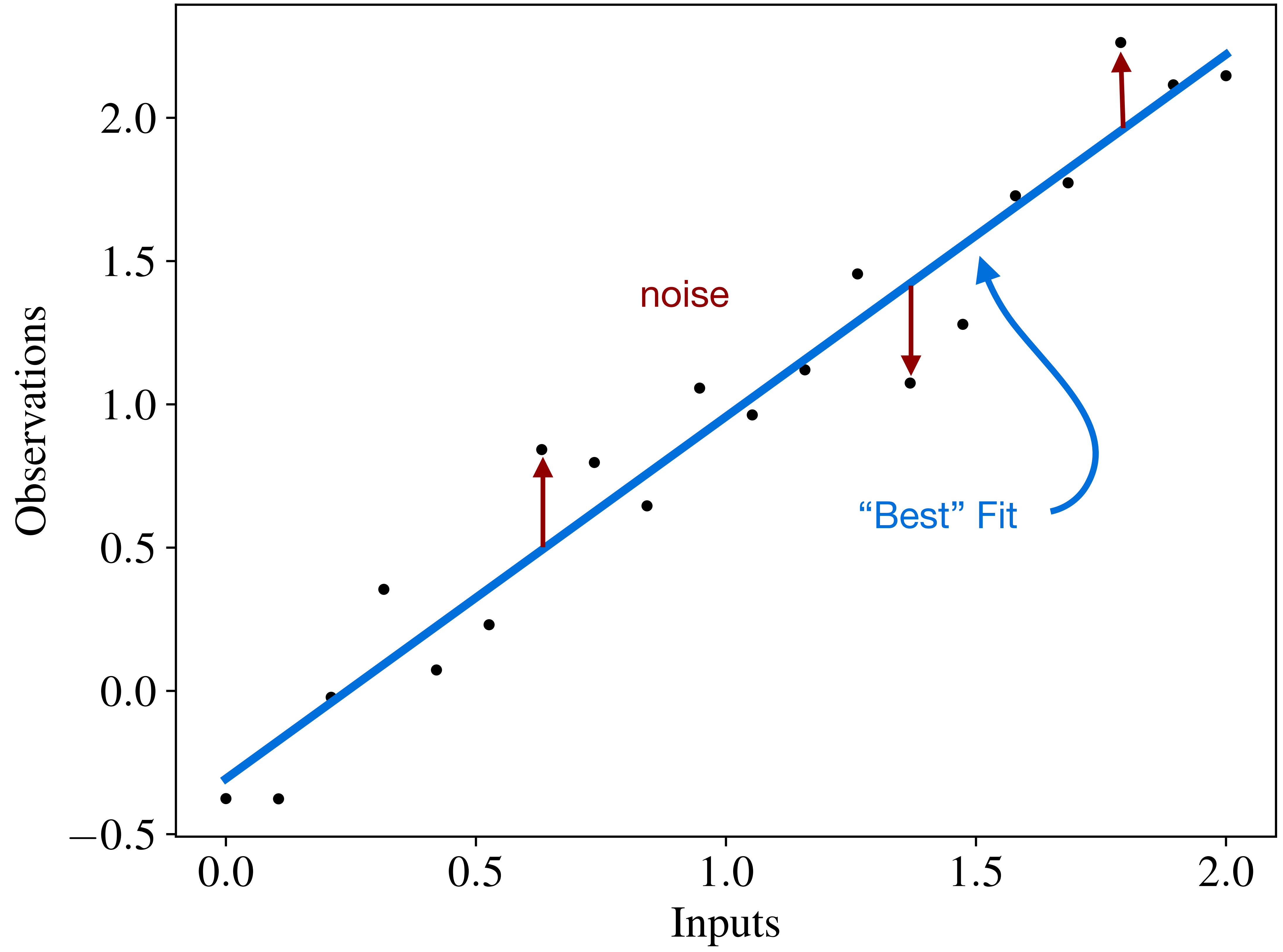
- Recall that our generalized linear model takes the form

$$\hat{f}(x) = \mathbf{w} \cdot \mathbf{h}(x)$$

- We can modify this by incorporating a random variable ε which represents the noise in our generative model

$$\hat{f}(x) = \underbrace{\mathbf{w} \cdot \mathbf{h}(x)}_{\text{deterministic}} + \underbrace{\varepsilon}_{\text{stochastic}}$$

- Note that the addition of ε into our linear model makes our model output random as well!
- Subtle point: we are implicitly assuming that the noise is independent of input location (not always true)
- **Left with 2 key problems:**
  1. What is the probability density of the stochastic component?
  2. How can we fit a random model to our data?



Choice of probability distribution $p(\varepsilon)$

In general, this will depend on your data and any knowledge you may have about the generating mechanism

- **For now, let's think of the key characteristics of our noise**
  - As written, it represents a deviation from the deterministic trend
  - Can be positive or negative
  - Likely to be closer to the nominal than far away
- These characteristics suggest that a Gaussian (normal) distribution with zero mean is a reasonable choice

$$p(\varepsilon) = \mathcal{N}(0, \sigma^2)$$

Likelihood

- Recall that our generative linear model is random, therefore, it has a probability density

$$\hat{y} = \hat{f}(x) = \mathbf{w} \cdot \mathbf{h}(x) + \varepsilon, \quad p(\varepsilon) = \mathcal{N}(0,\sigma^2)$$

- The probability density of the sum of a normally distributed random variable and a scalar shifts the mean

$$p(\hat{y} \,|\, \mathbf{w}, x, \sigma^2) = \mathcal{N}(\mathbf{w} \cdot \mathbf{h}(x), \sigma^2)$$

- The value of this distribution for a given set of parameters, input, and noise variance, is often called the *likelihood* because it represents how "likely" the model will output that particular value

- We can therefore define a *dataset likelihood* as the likelihood that our model will generate our particular dataset as

$$L = p(\mathbf{y} \,|\, \mathbf{x}, \mathbf{w}, \sigma^2) = \prod_{i=1}^{N} p(y_i \,|\, x_i, \mathbf{w}, \sigma^2) = \prod_{i=1}^{N} \mathcal{N}(\mathbf{w} \cdot \mathbf{h}(x_i), \sigma^2)$$

Maximum likelihood estimate

The *Maximum likelihood estimate* (MLE) maximizes the likelihood of generating the dataset with the model

- Specifically, we minimize the negative log dataset likelihood (NLL) for $\mathbf{w}$ and σ^2

$$\mathscr{L} = -\ln p(\mathbf{y} \,|\, \mathbf{x}, \mathbf{w}, \sigma^2) = -\frac{N}{2}\ln 2\pi - N\ln\sigma - \frac{1}{2\sigma^2}\sum_{i=1}^{N}(y_i - \mathbf{w}\cdot\mathbf{h}(x))^2$$

$$\frac{\partial \mathscr{L}}{\partial \mathbf{w}} = 0 \implies \mathbf{w} = (\mathbf{H}^T\mathbf{H})^{-1}\mathbf{H}^T\mathbf{y}$$

identical to our least-squares solution!

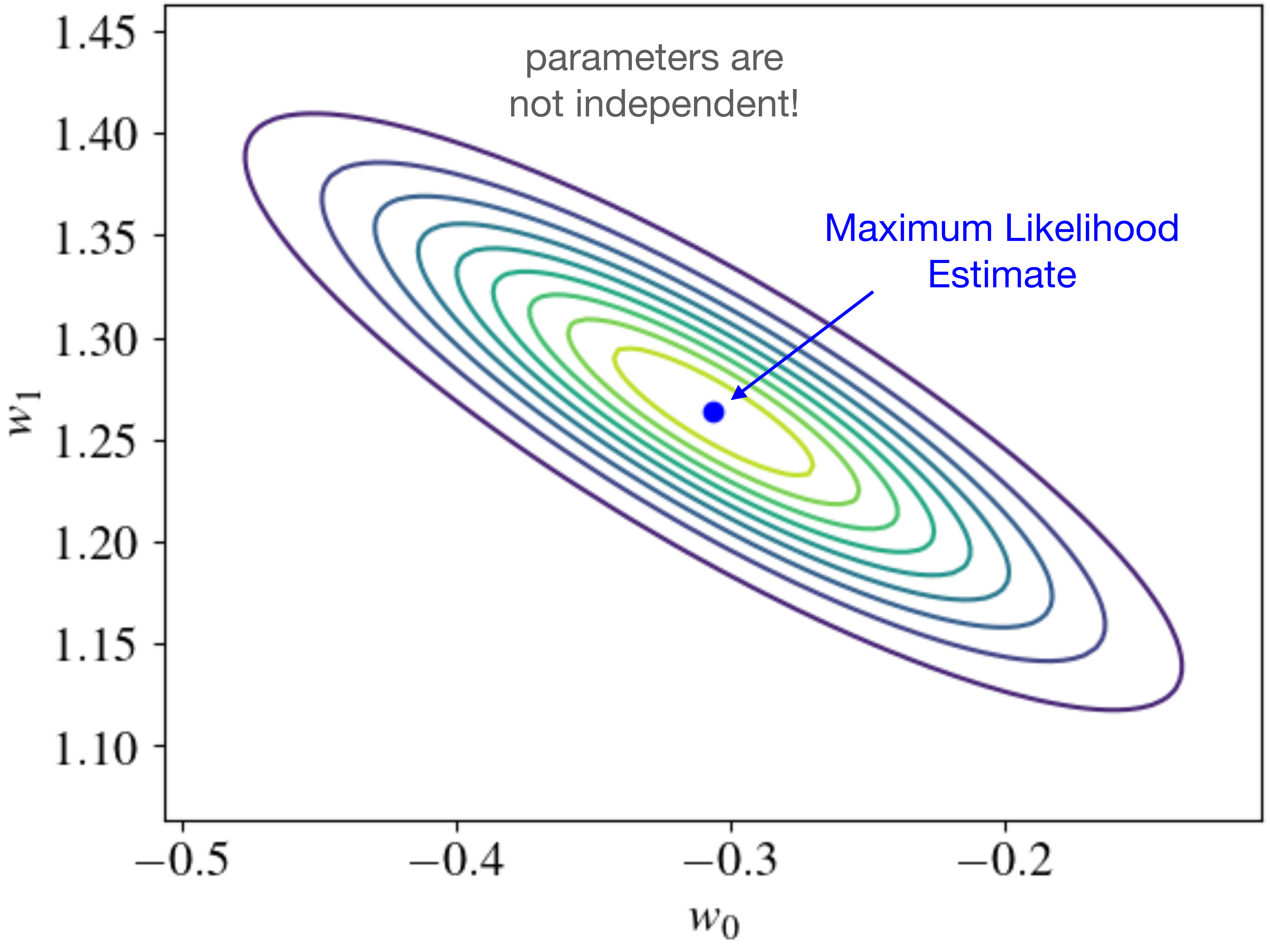
$$\frac{\partial \mathscr{L}}{\partial \mathbf{w}} = 0 \implies \sigma^2 = \frac{1}{N}\sum_{i=1}^{N}(y_i - \mathbf{w}\cdot\mathbf{h}(x_i))^2$$

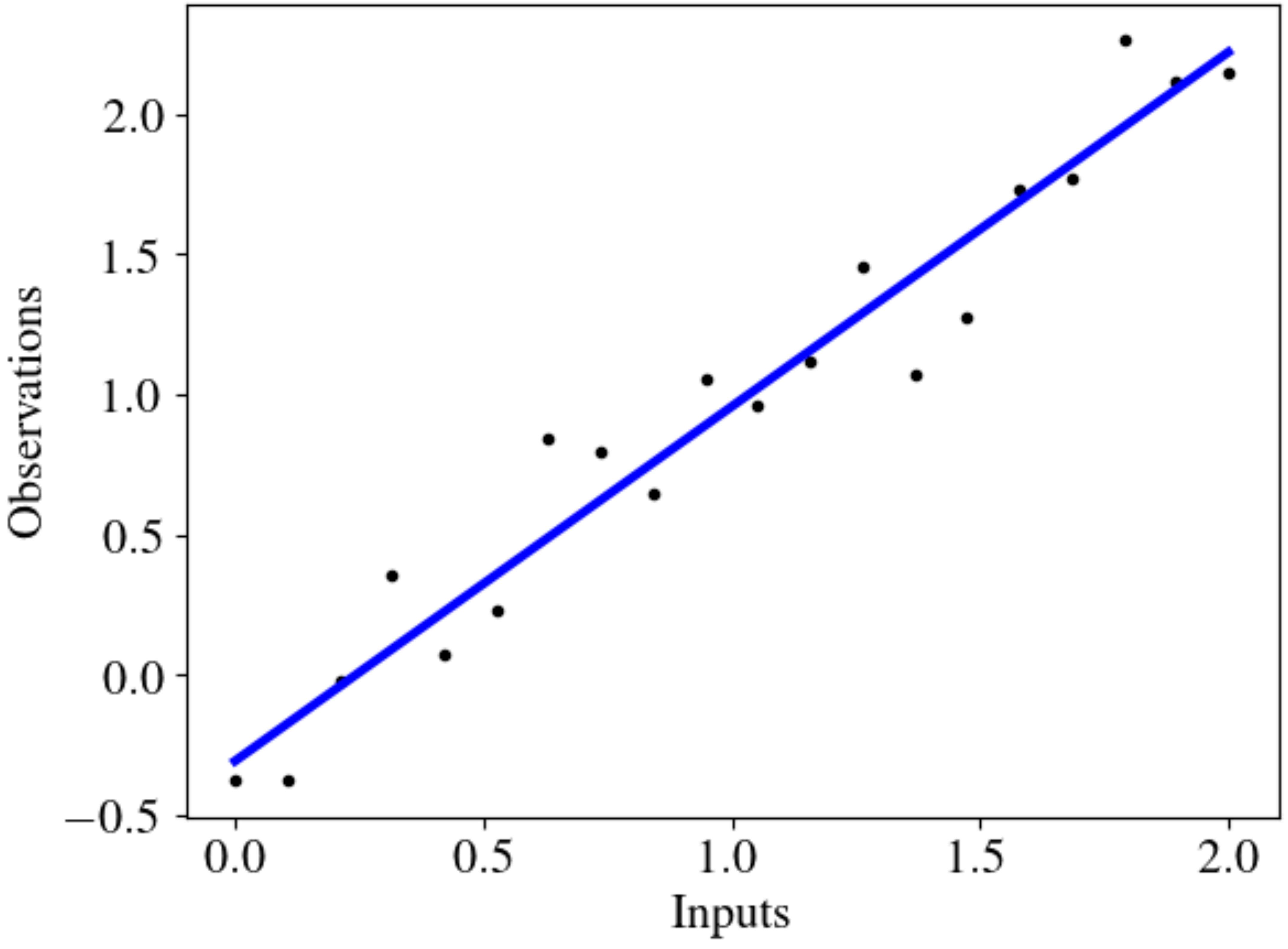
mean squared-error

Effect of noise on parameter estimates

Using our generative model, we can create fake datasets and see how our model parameters would be effected.

- For linear models, can derive analytical probability density of parameters, taking noise into account (give this a try!)
- Sampling from this distribution, provides a notion of predictive model uncertainty

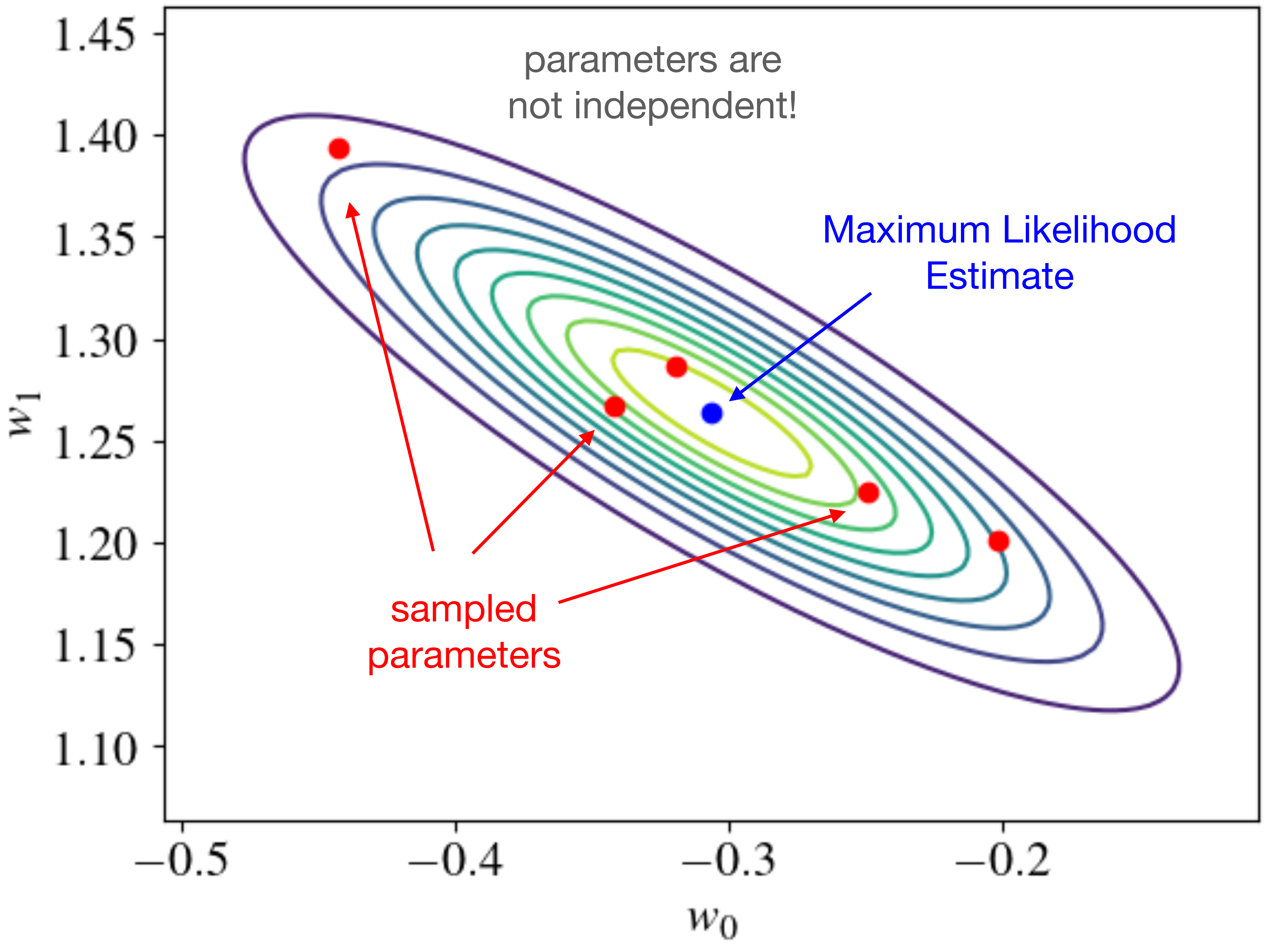


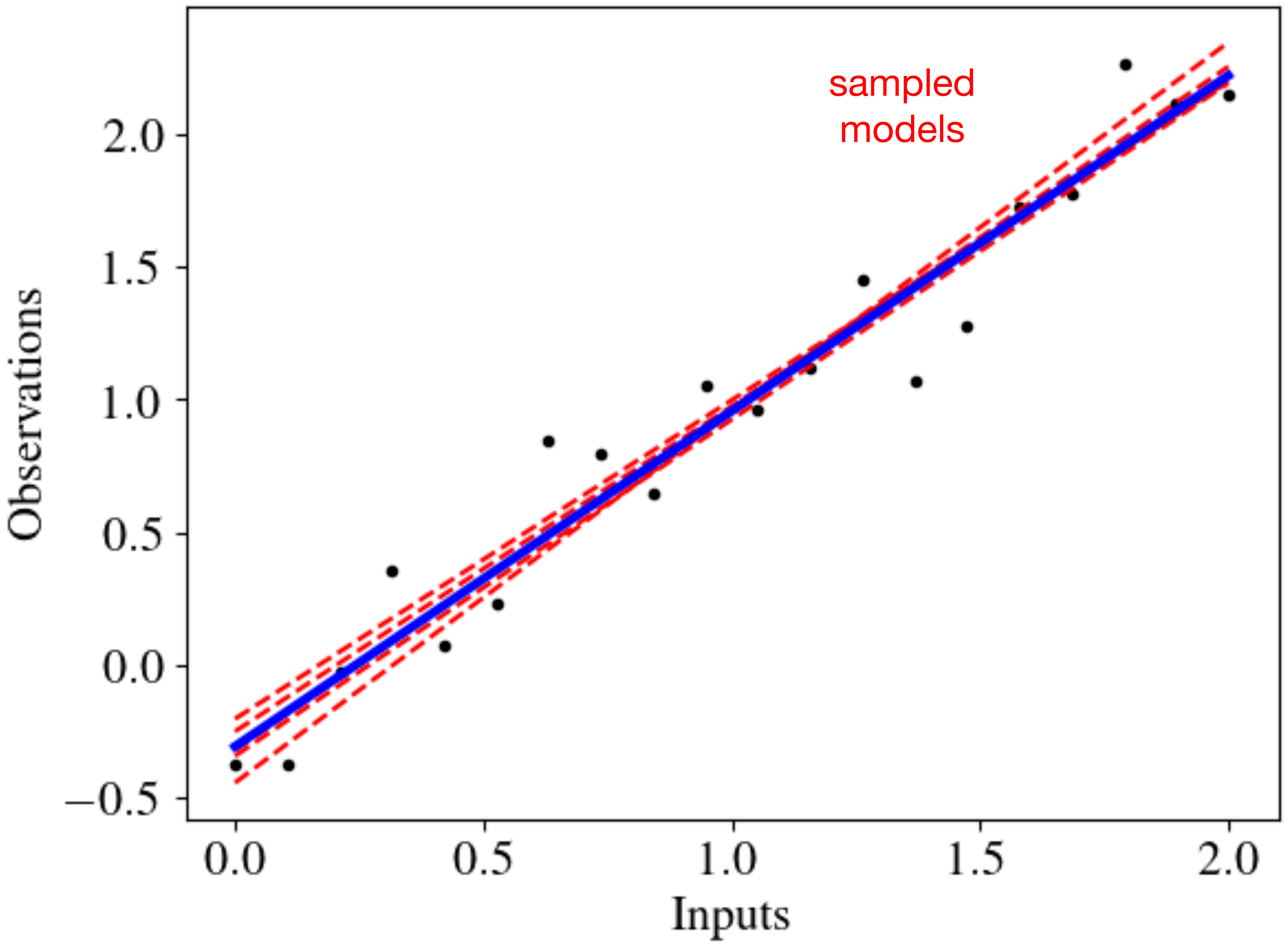


Effect of noise on parameter estimates

Using our generative model, we can create fake datasets and see how our model parameters would be effected.

- For linear models, can derive analytical probability density of parameters, taking noise into account (give this a try!)
- Sampling from this distribution, provides a notion of predictive model uncertainty



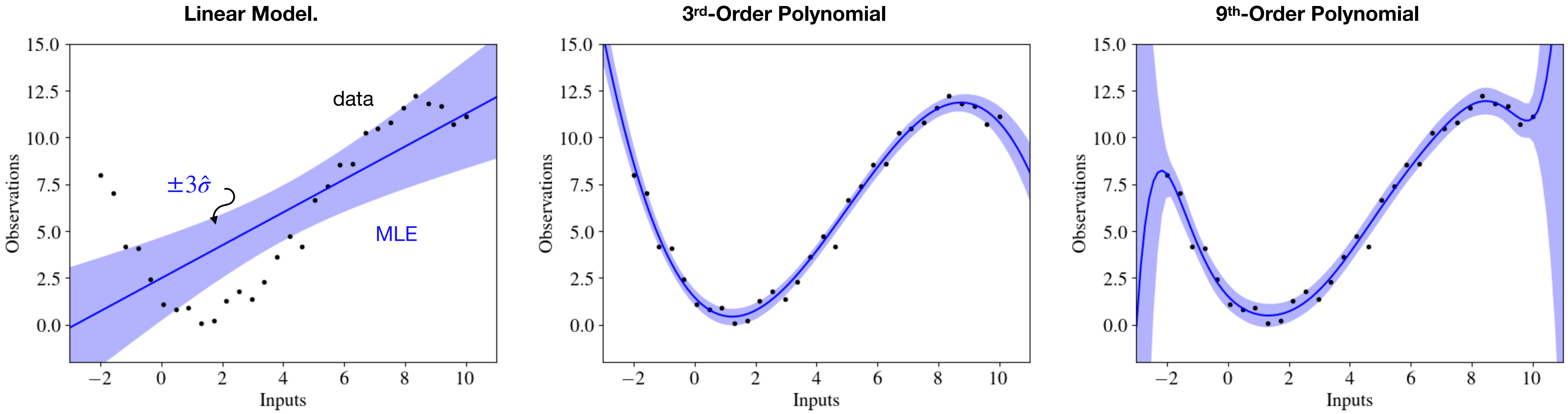


Predictive variance

- Summarizing prediction and variance for linear MLE model (skipping the details)

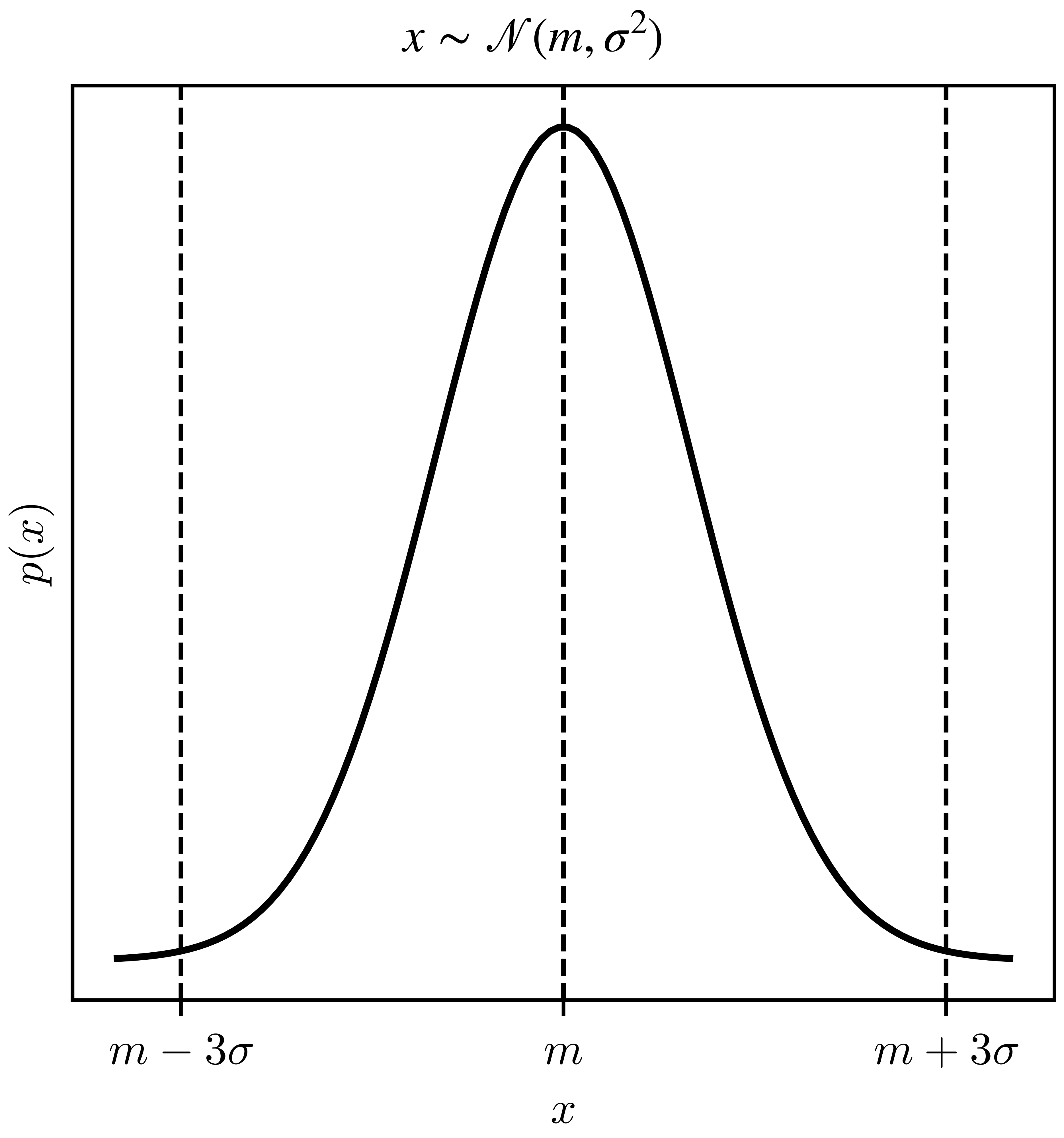
$$\hat{y} = \hat{f}(x) = \mathbf{h}(x)^T \mathbf{w} = \mathbf{h}(x)^T (\mathbf{H}^T \mathbf{H})^{-1} \mathbf{H}^T \mathbf{y}$$

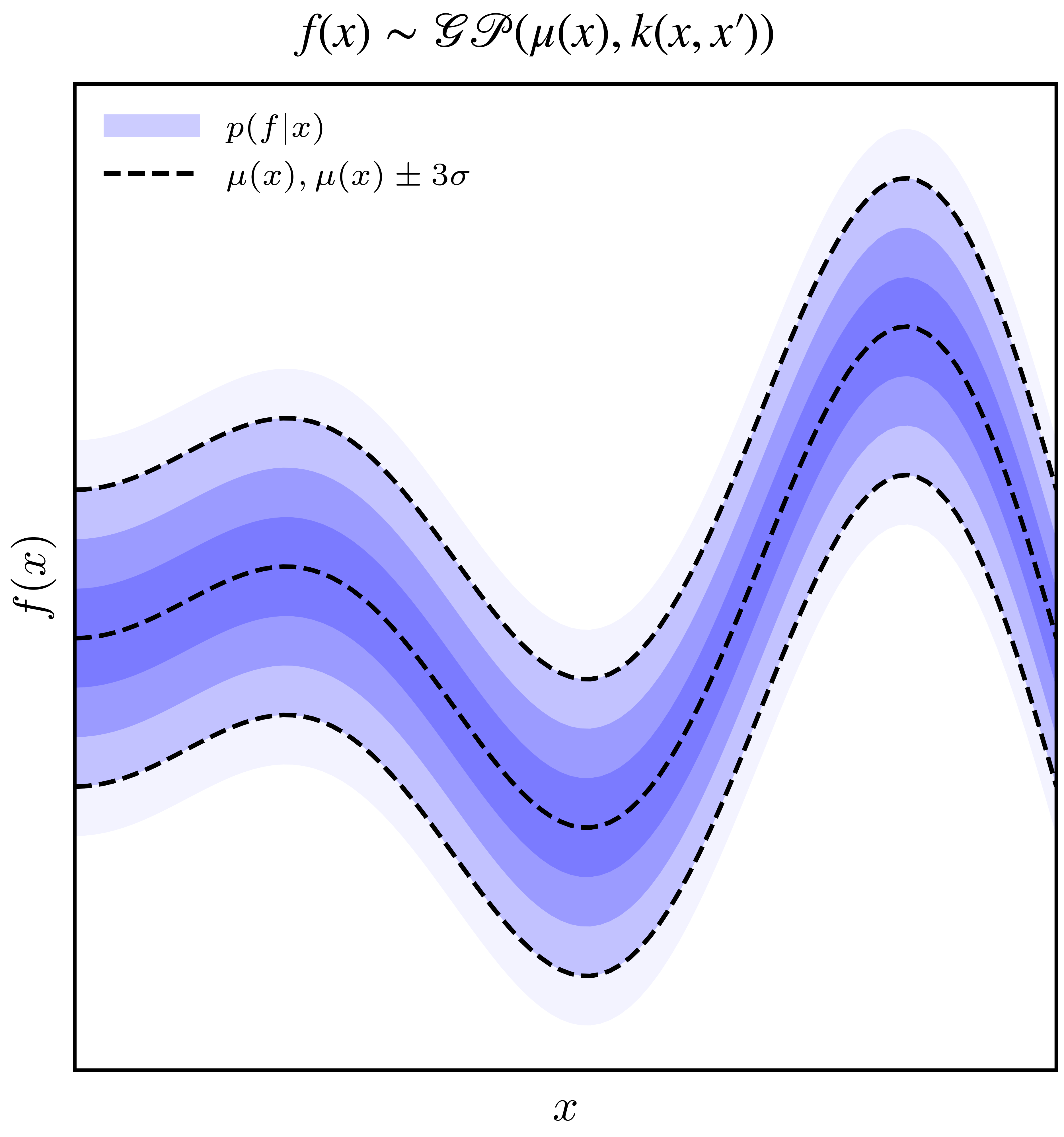
$$\hat{\sigma}^2(x) = \mathbf{h}^T(x) \, \text{cov}\{\mathbf{w}\} \, \mathbf{h}(x) = \sigma^2 \, \mathbf{h}^T(x) (\mathbf{H}^T \mathbf{H})^{-1} \mathbf{h}(x)$$



Gaussian Processes (GPs)

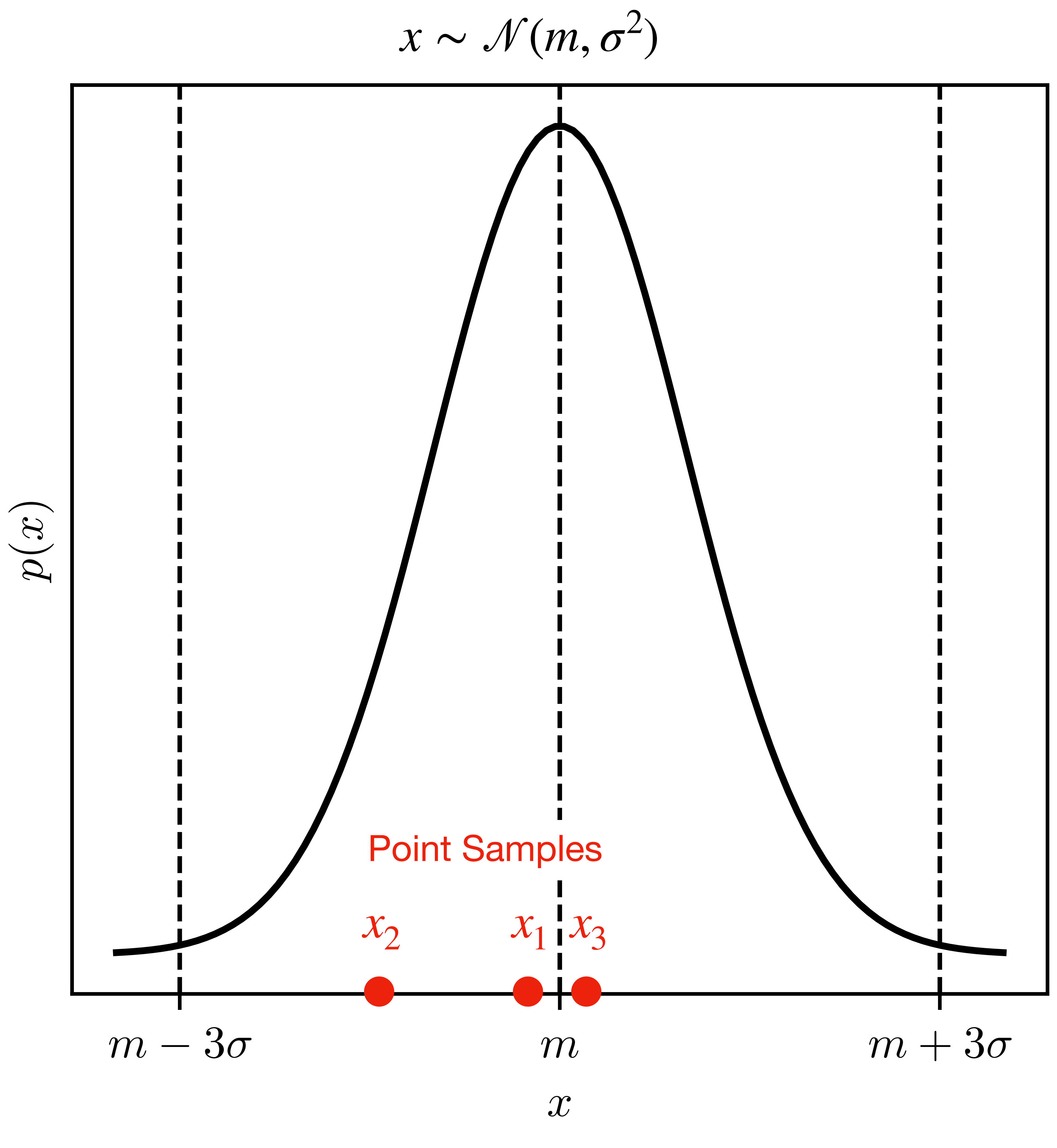
Generalization of the Gaussian distribution for random scalars to random functions

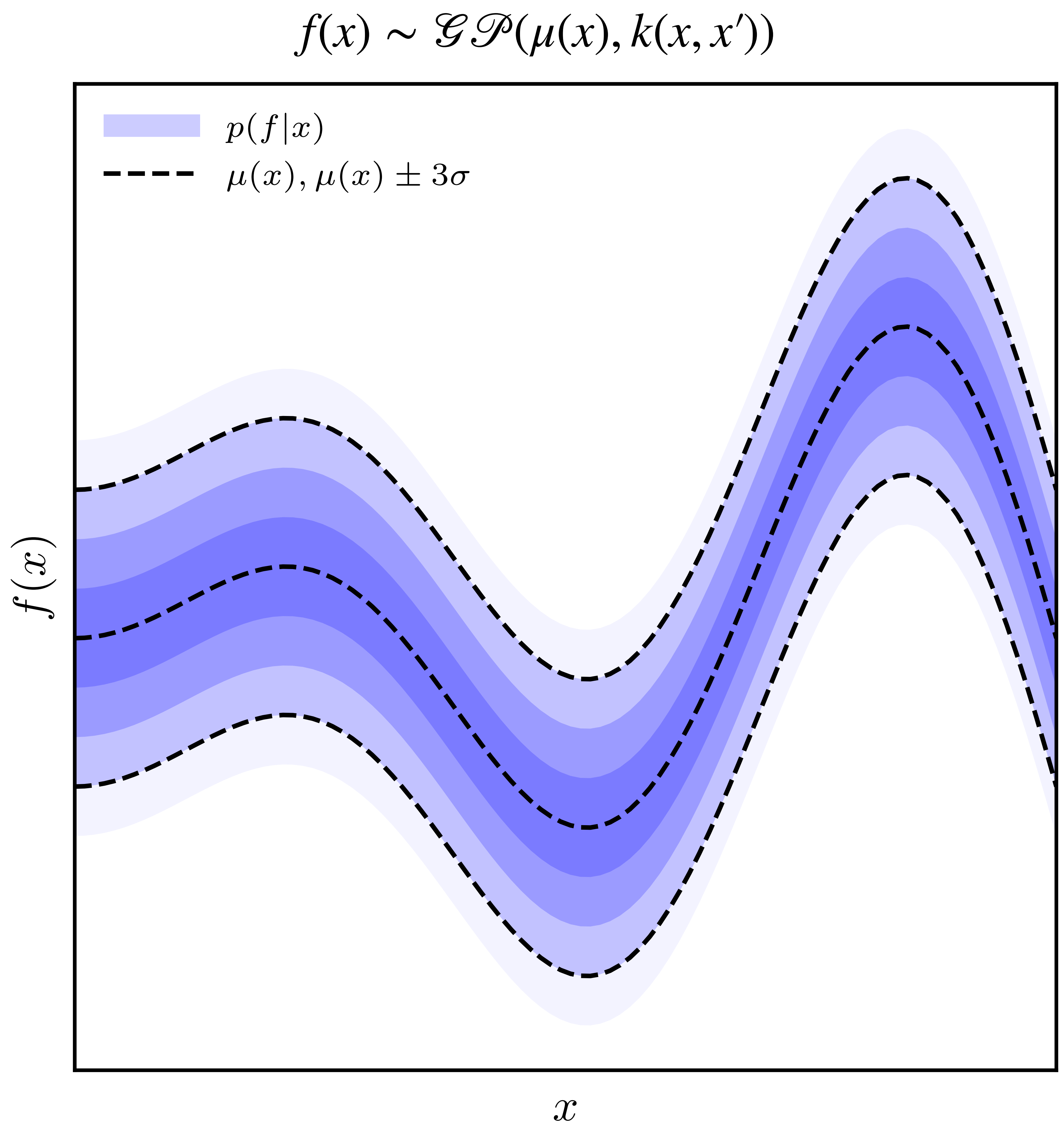




Gaussian Processes (GPs)

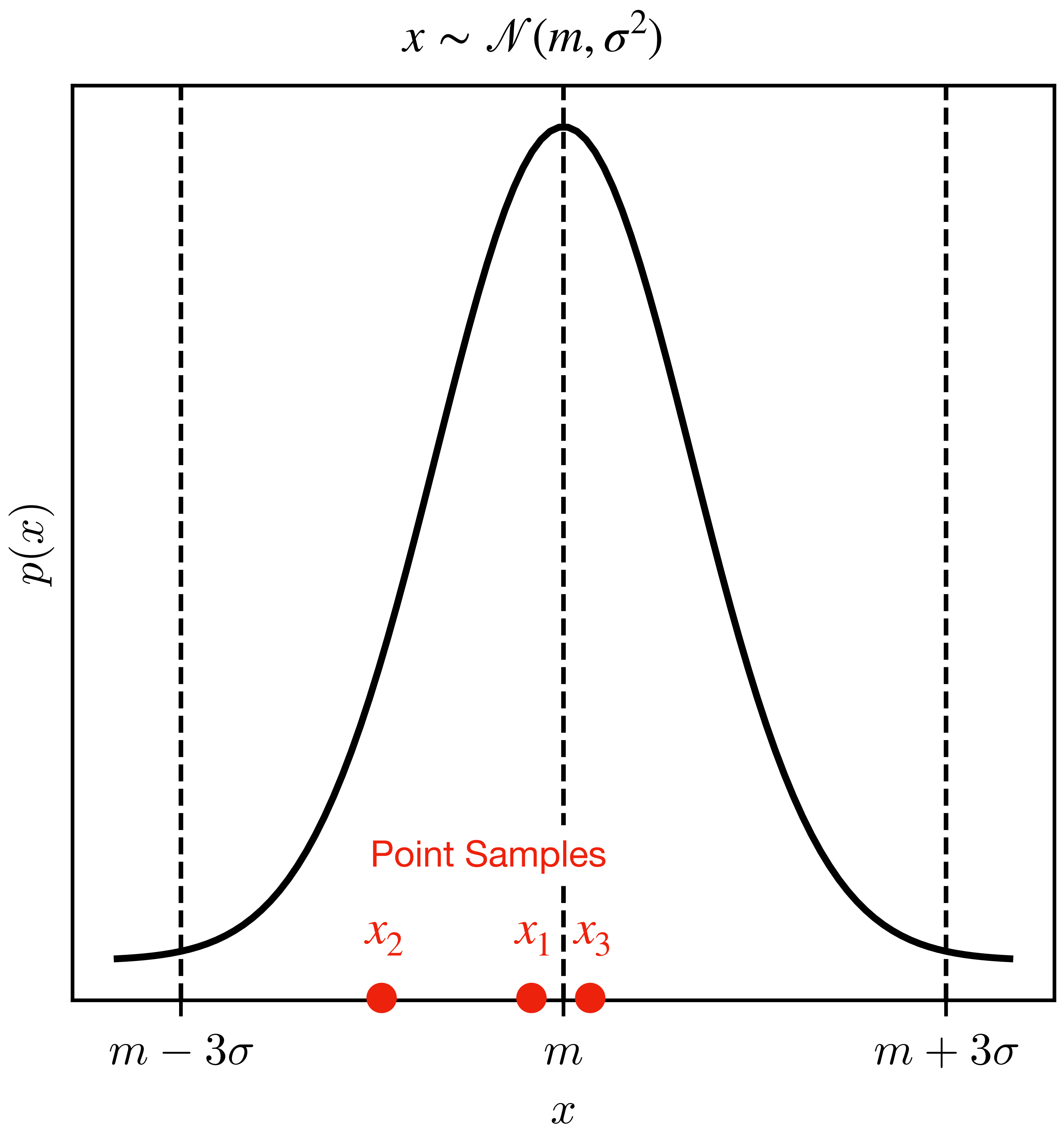
Generalization of the Gaussian distribution for random scalars to random functions

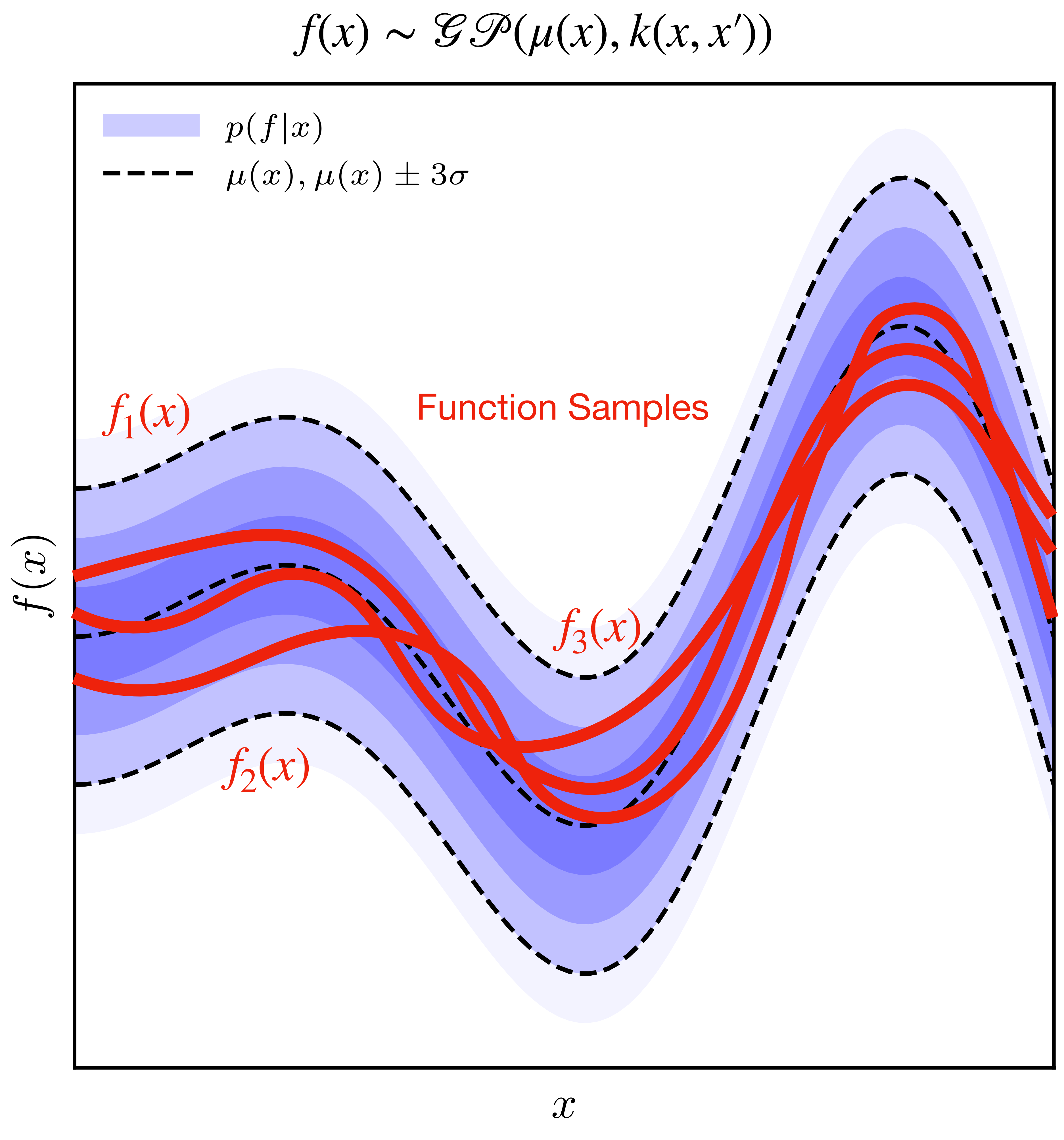




Gaussian Processes (GPs)

Generalization of the Gaussian distribution for random scalars to random functions





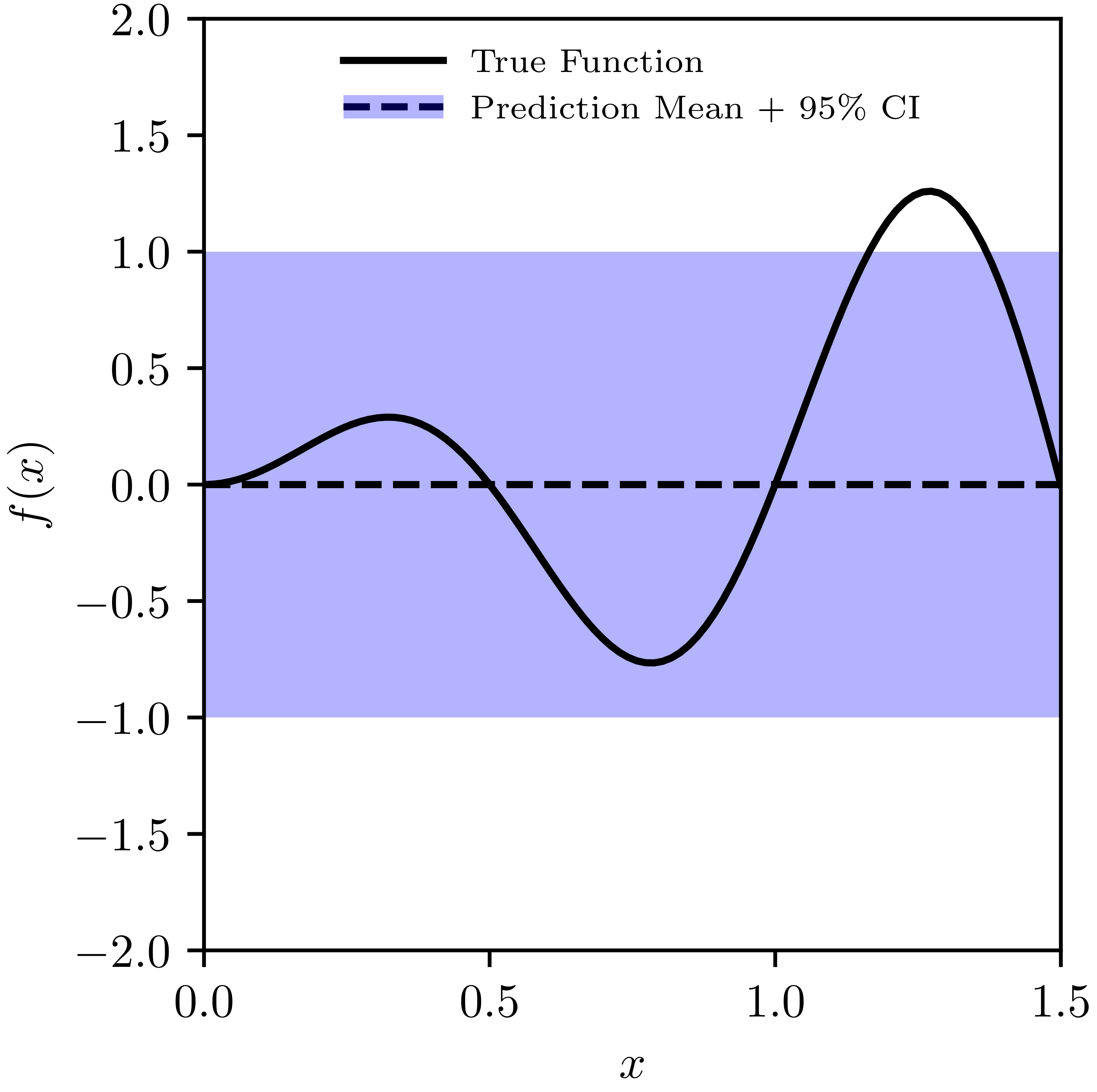
Gaussian Process Regression

Any finite set of index points $\{x_1, \ldots, x_n\}$ represents a multivariate Gaussian distribution for function values.

prior

$$f(x) \sim p(f \,|\, x) = \mathcal{GP}(\mu, k)$$

$$\implies \begin{bmatrix} f(x_1) \\ f(x_2) \\ f(x_3) \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \mu(x_1) \\ \mu(x_2) \\ \mu(x_3) \end{bmatrix}, \begin{bmatrix} k(x_1, x_1) & k(x_1, x_2) & k(x_1, x_3) \\ k(x_2, x_1) & k(x_2, x_2) & k(x_2, x_3) \\ k(x_3, x_1) & k(x_3, x_2) & k(x_3, x_3) \end{bmatrix} \right)$$



Gaussian Process Regression

Any finite set of index points $\{x_1, \ldots, x_n\}$ represents a multivariate Gaussian distribution for function values.

prior

$$f(x) \sim p(f\,|\,x) = \mathcal{GP}(\mu, k)$$

$$\Longrightarrow \begin{bmatrix} f(x_1) \\ f(x_2) \\ f(x_3) \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \mu(x_1) \\ \mu(x_2) \\ \mu(x_3) \end{bmatrix}, \begin{bmatrix} k(x_1, x_1) & k(x_1, x_2) & k(x_1, x_3) \\ k(x_2, x_1) & k(x_2, x_2) & k(x_2, x_3) \\ k(x_3, x_1) & k(x_3, x_2) & k(x_3, x_3) \end{bmatrix} \right)$$

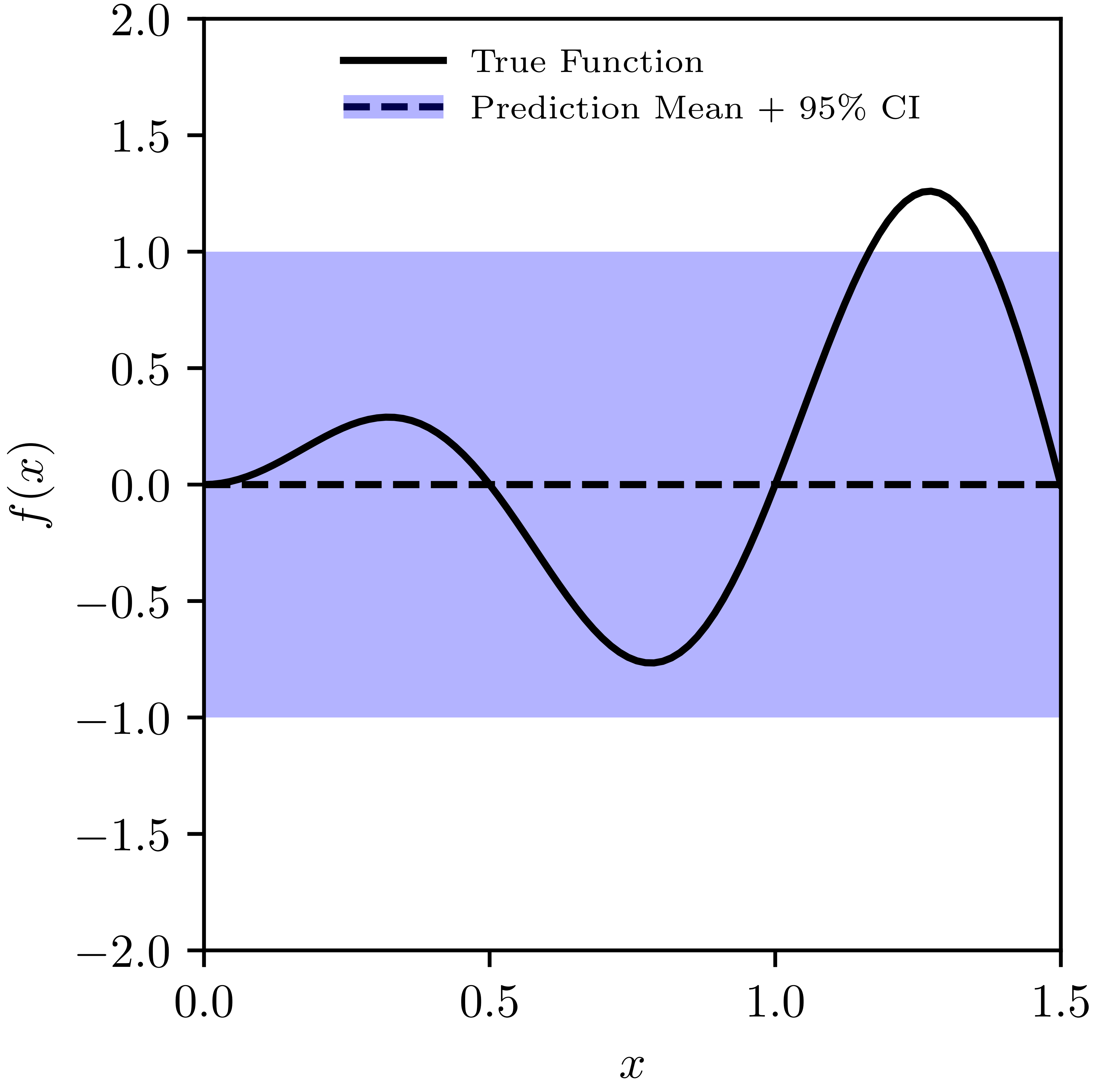
Regression performed by conditioning the distribution on data
$\mathcal{D} = (X, f(X) = y)$

posterior

$$\hat{f}(x) \sim p(f\,|\,\mathcal{D}, x) = \mathcal{GP}(\hat{\mu}, \hat{k})$$

$$\hat{\mu}(x) = \mu(x) + k(x, X)\; k(X, X)^{-1}\; (y - \mu(X))$$

$$\hat{k}(x, x') = k(x, x') + k(x, X)\; k(X, X)^{-1}\; k(X, x')$$



Gaussian Process Regression

Any finite set of index points $\{x_1, \ldots, x_n\}$ represents a multivariate Gaussian distribution for function values.

prior

$$f(x) \sim p(f\,|\,x) = \mathcal{GP}(\mu, k)$$

$$\implies \begin{bmatrix} f(x_1) \\ f(x_2) \\ f(x_3) \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \mu(x_1) \\ \mu(x_2) \\ \mu(x_3) \end{bmatrix}, \begin{bmatrix} k(x_1, x_1) & k(x_1, x_2) & k(x_1, x_3) \\ k(x_2, x_1) & k(x_2, x_2) & k(x_2, x_3) \\ k(x_3, x_1) & k(x_3, x_2) & k(x_3, x_3) \end{bmatrix} \right)$$

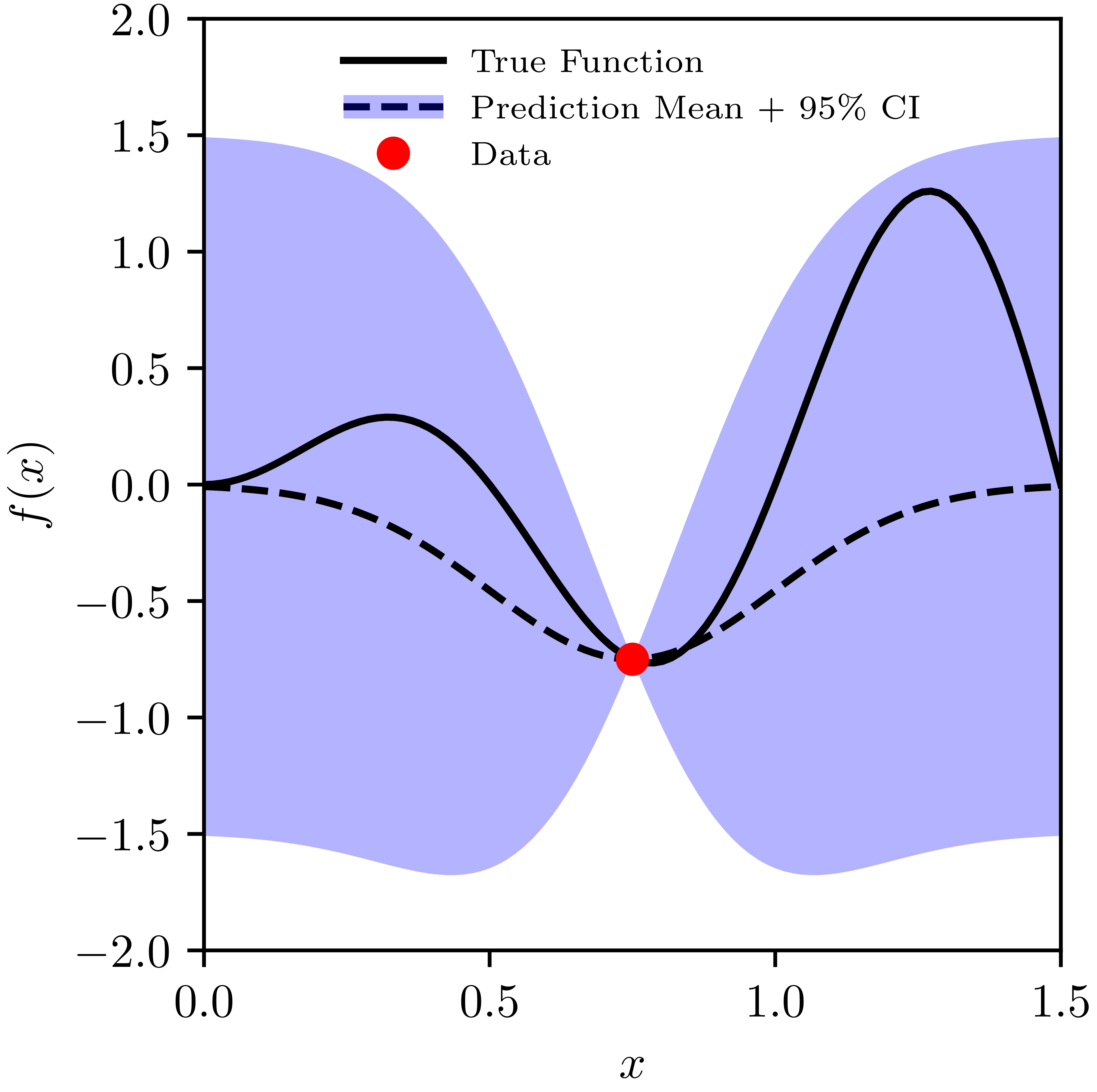
Regression performed by conditioning the distribution on data
$\mathcal{D} = (X, f(X) = y)$

posterior

$$\hat{f}(x) \sim p(f\,|\,\mathcal{D}, x) = \mathcal{GP}(\hat{\mu}, \hat{k})$$

$$\hat{\mu}(x) = \mu(x) + k(x, X)\; k(X, X)^{-1}\; (y - \mu(X))$$

$$\hat{k}(x, x') = k(x, x') + k(x, X)\; k(X, X)^{-1}\; k(X, x')$$



Gaussian Process Regression

Any finite set of index points $\{x_1, \ldots, x_n\}$ represents a multivariate Gaussian distribution for function values.

prior

$$f(x) \sim p(f\,|\,x) = \mathscr{GP}(\mu, k)$$

$$\implies \begin{bmatrix} f(x_1) \\ f(x_2) \\ f(x_3) \end{bmatrix} \sim \mathscr{N}\left(\begin{bmatrix} \mu(x_1) \\ \mu(x_2) \\ \mu(x_3) \end{bmatrix}, \begin{bmatrix} k(x_1, x_1) & k(x_1, x_2) & k(x_1, x_3) \\ k(x_2, x_1) & k(x_2, x_2) & k(x_2, x_3) \\ k(x_3, x_1) & k(x_3, x_2) & k(x_3, x_3) \end{bmatrix} \right)$$

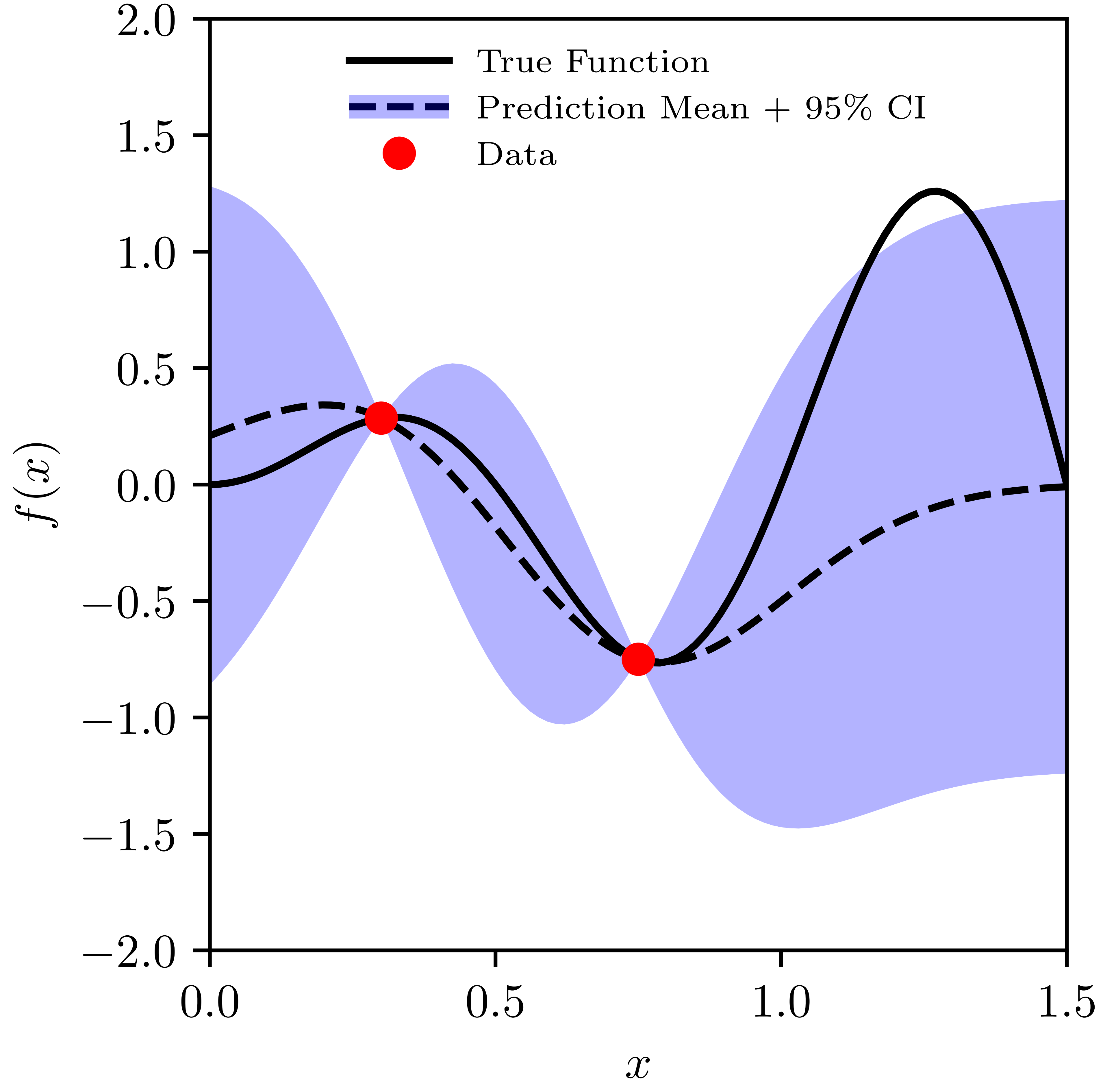
Regression performed by conditioning the distribution on data
$\mathscr{D} = (X, f(X) = y)$

posterior

$$\hat{f}(x) \sim p(f\,|\,\mathscr{D}, x) = \mathscr{GP}(\hat{\mu}, \hat{k})$$

$$\hat{\mu}(x) = \mu(x) + k(x, X)\; k(X, X)^{-1}\; (y - \mu(X))$$

$$\hat{k}(x, x') = k(x, x') + k(x, X)\; k(X, X)^{-1}\; k(X, x')$$



Gaussian Process Regression

Any finite set of index points $\{x_1, \ldots, x_n\}$ represents a multivariate Gaussian distribution for function values.

prior

$$f(x) \sim p(f\,|\,x) = \mathcal{GP}(\mu, k)$$

$$\Longrightarrow \begin{bmatrix} f(x_1) \\ f(x_2) \\ f(x_3) \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \mu(x_1) \\ \mu(x_2) \\ \mu(x_3) \end{bmatrix}, \begin{bmatrix} k(x_1, x_1) & k(x_1, x_2) & k(x_1, x_3) \\ k(x_2, x_1) & k(x_2, x_2) & k(x_2, x_3) \\ k(x_3, x_1) & k(x_3, x_2) & k(x_3, x_3) \end{bmatrix} \right)$$

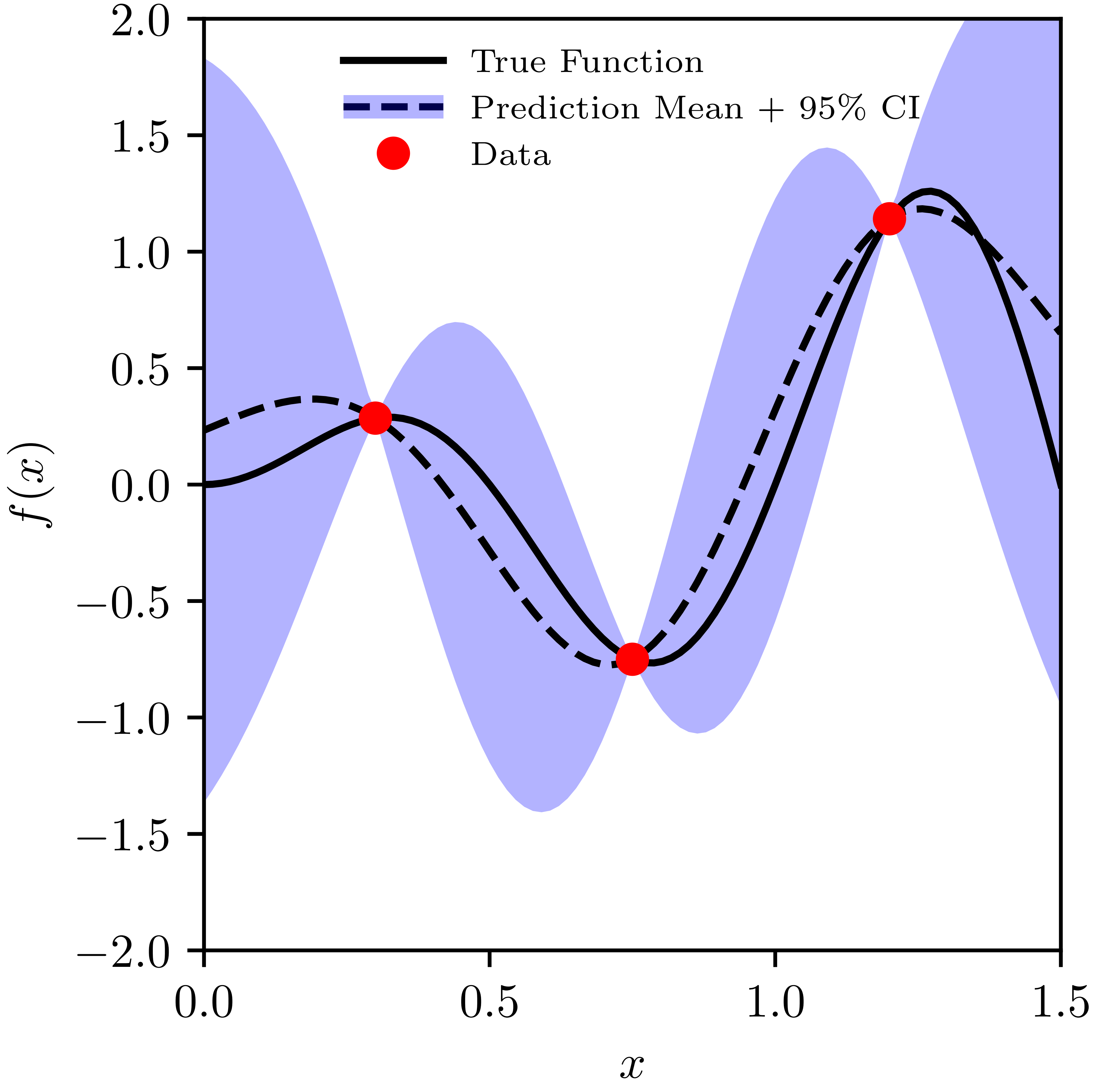
Regression performed by conditioning the distribution on data
$\mathcal{D} = (X, f(X) = y)$

posterior

$$\hat{f}(x) \sim p(f\,|\,\mathcal{D}, x) = \mathcal{GP}(\hat{\mu}, \hat{k})$$

$$\hat{\mu}(x) = \mu(x) + k(x, X)\; k(X, X)^{-1}\; (y - \mu(X))$$

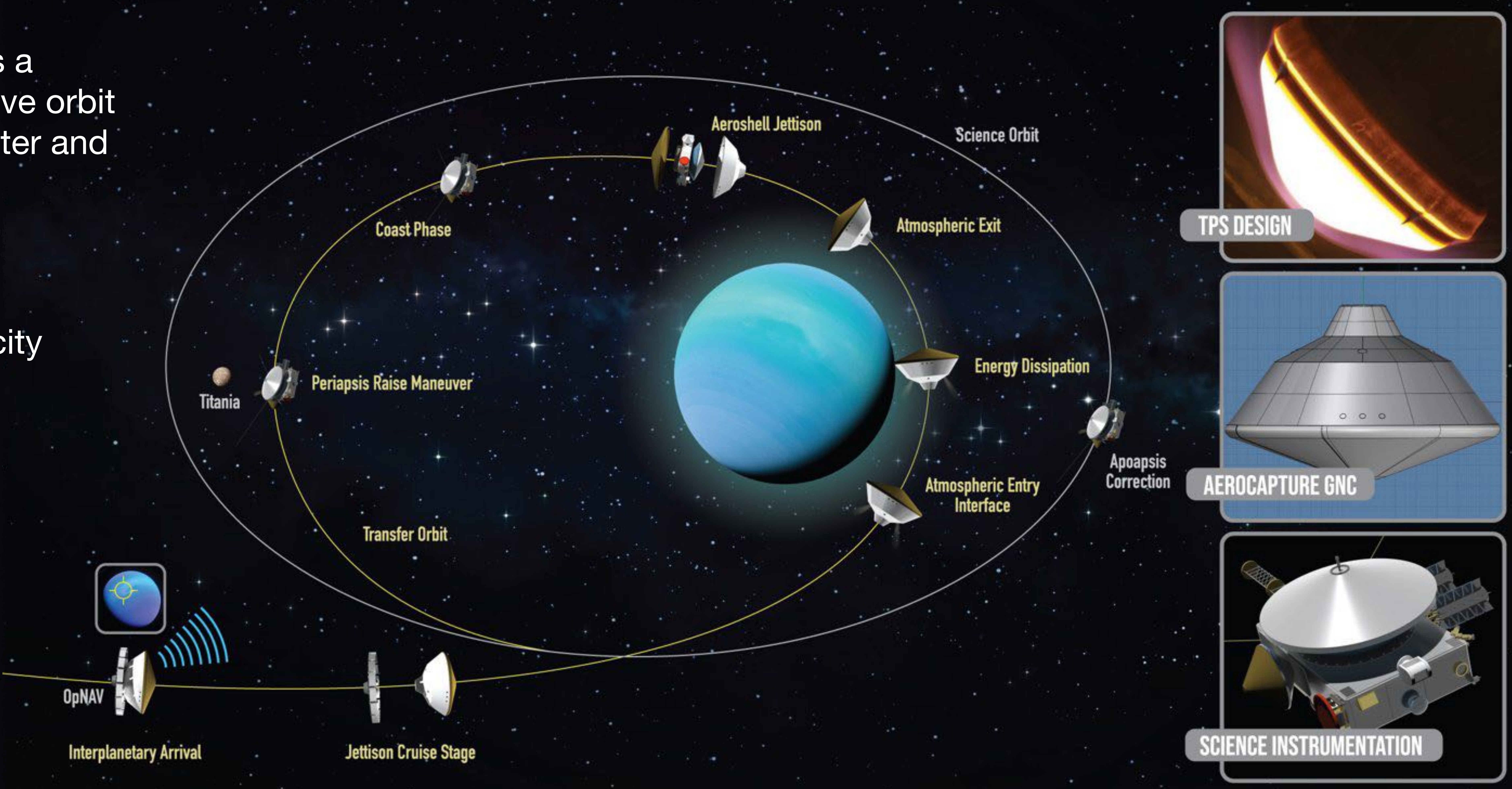
$$\hat{k}(x, x') = k(x, x') + k(x, X)\; k(X, X)^{-1}\; k(X, x')$$



Example 1: Uranus Aerocapture

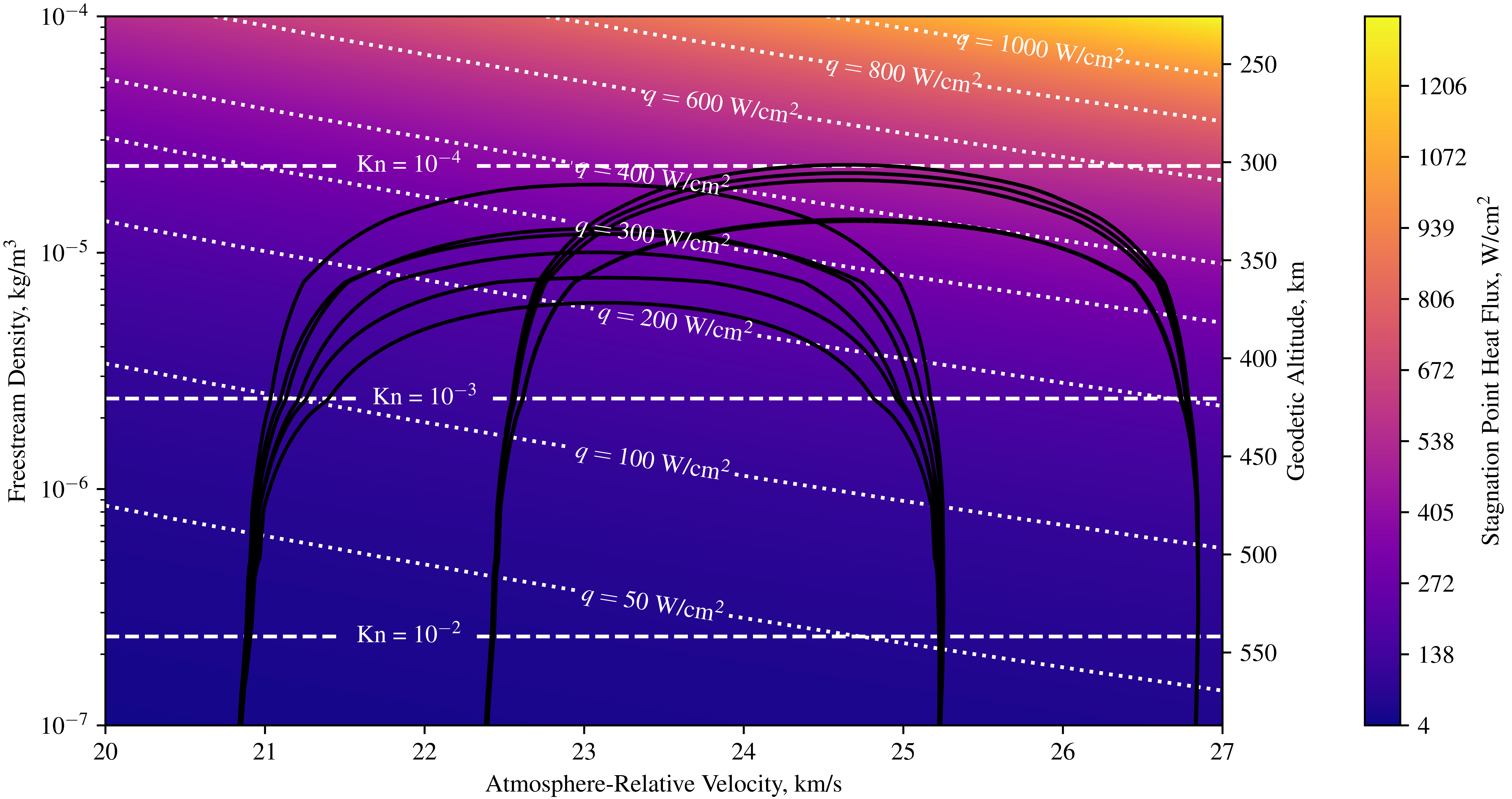
Early Career Initiative (ECI) Project

- Demonstrate *aerocapture* as a viable alternative to propulsive orbit insertions for Gas Giant orbiter and probe missions
- Benefits:
  - Increased payload capacity
  - Decrease cruise time

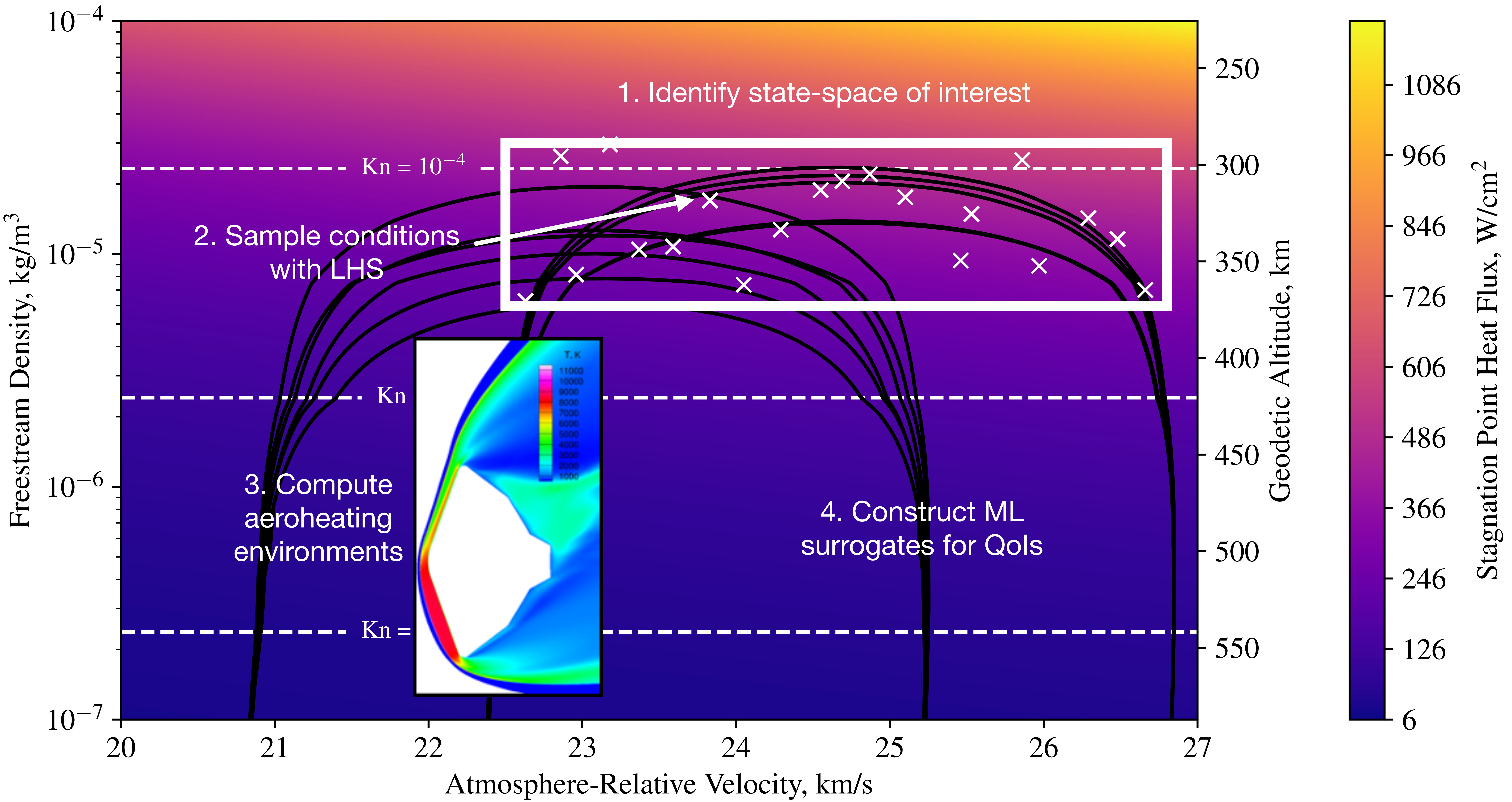


LaRC, ARC, JSC, JPL, Draper Laboratories, Booz Allen Hamilton, Intuitive Machines

Aeroheating database generation



Aeroheating database generation



Axisymmetric forebody dataset

Conditions
X
Surface Points

Density	Velocity	Body Coordinate	Convective Heat Flux	Wall Pressure	Shear Stress
$\rho_{\infty 1}$	$V_{\infty 1}$	s_1	q_1	p_{w1}	τ_{w1}
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$\rho_{\infty N}$	$V_{\infty N}$	s_N	q_N	p_{wN}	τ_{wN}

Axisymmetric forebody dataset

Conditions X # Surface Points

Input Data			Output Data		
Density	Velocity	Body Coordinate	Convective Heat Flux	Wall Pressure	Shear Stress
$\rho_{\infty 1}$	$V_{\infty 1}$	s_1	q_1	p_{w1}	τ_{w1}
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$\rho_{\infty N}$	$V_{\infty N}$	s_N	q_N	p_{wN}	τ_{wN}

Axisymmetric forebody dataset

Input Data | **Output Data**

Conditions X # Surface Points

Density	Velocity	Body Coordinate	Convective Heat Flux	Wall Pressure	Shear Stress
$\rho_{\infty 1}$	$V_{\infty 1}$	s_1	q_1	p_{w1}	τ_{w1}
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$\rho_{\infty N}$	$V_{\infty N}$	s_N	q_N	p_{wN}	τ_{wN}

Want to create surrogate models that fit the data and provide surface heat flux, pressure, and shear stress over entire state-space of interest in order to provide estimates over computed trajectories

- Maintain physical scaling when making predictions outside of the dataset range
- Estimate model uncertainties

Dataset normalization

- Good opportunity to ask “What do I know about my data?”
  - Dimensionality reduction, known scaling laws or engineering correlations, limits or bounds?
  - Sutton-Graves model for max convective heating:

$$q_{conv}^{max} = K\sqrt{\frac{\rho_\infty}{R_n}}V_\infty^3$$

  - Newtonian pressure theory:

$$C_p^{max} = \frac{p_{max} - p_\infty}{\frac{1}{2}\rho_\infty V_\infty^2} \approx 2 \implies p_{max} \approx A\rho_\infty V_\infty^2$$

  - Suggests that maximum value of QoIs for each freestream condition follow generalized Sutton-Graves relation

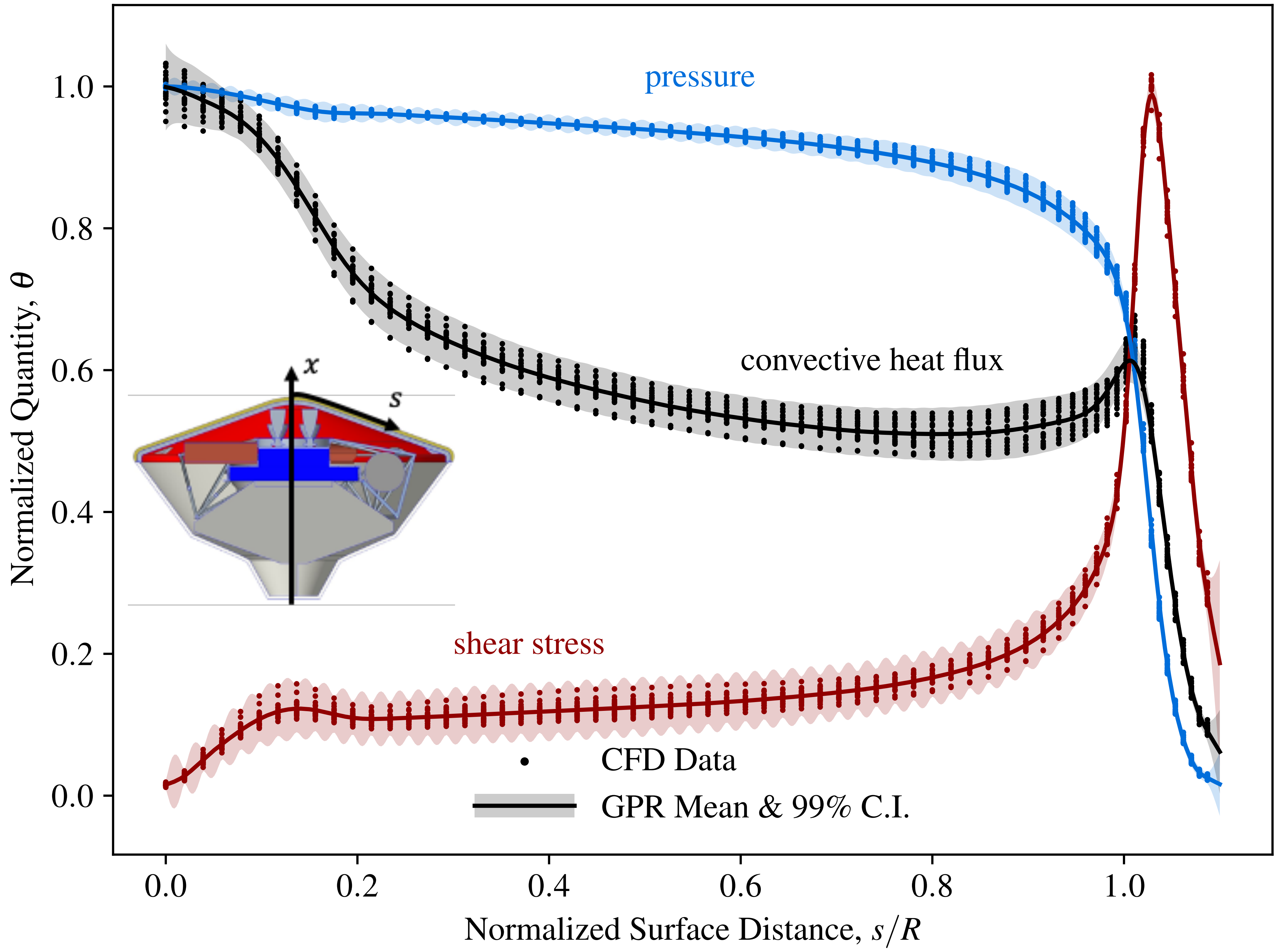
$$\theta_{max} = C_\theta\, \rho_\infty^{m_\theta}\, V_\infty^{n_\theta}$$

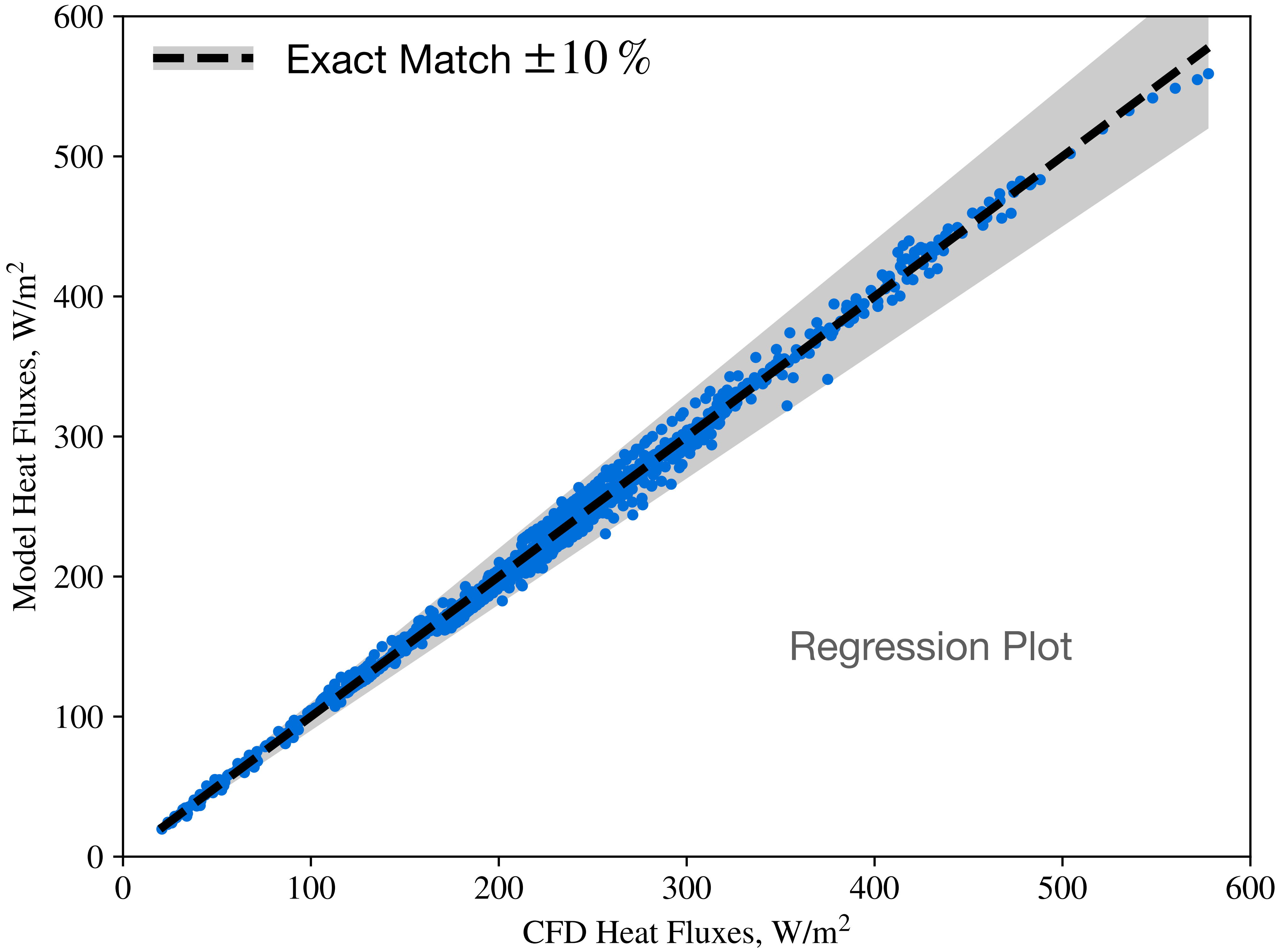
Linear models to rescue

- The generalized Sutton-Graves model is linear in it's parameters with appropriate transformation!

$$\theta_{max} = C_\theta\, \rho_\infty^{m_\theta}\, V_\infty^{n_\theta} \implies \ln \theta_{max} = \ln C_\theta + m_\theta \ln \rho_\infty + n_\theta \ln V_\infty$$

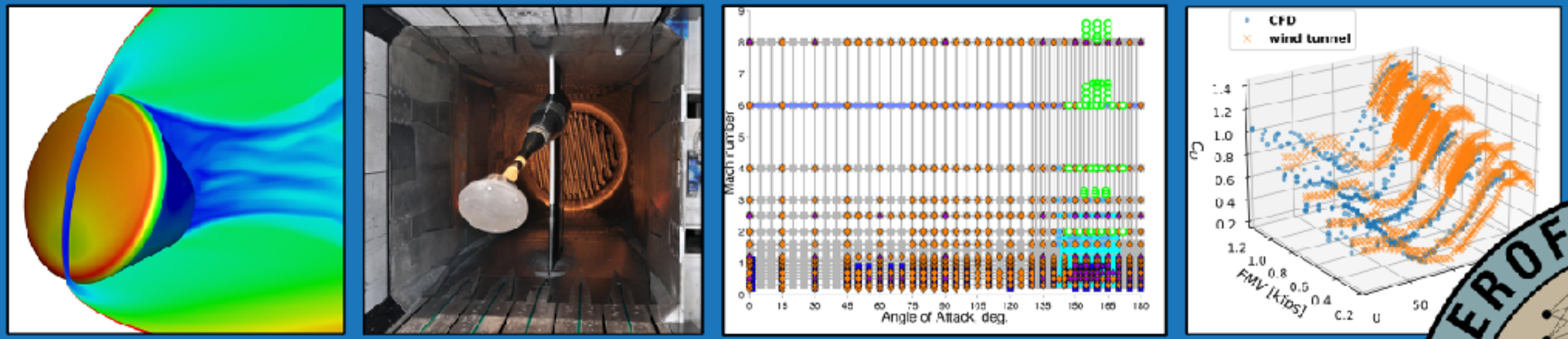
- Normalizing all the data by our new fits reduces the dimensionality of the problem to the body coordinate



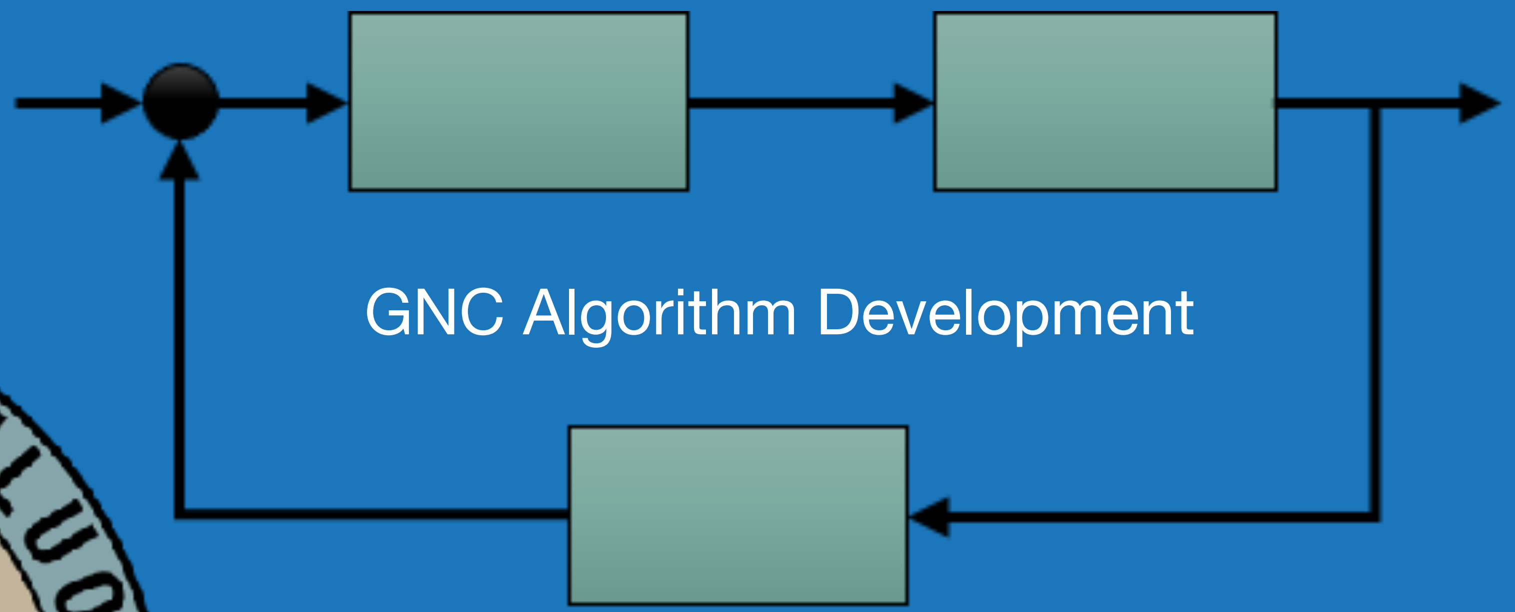


Example 2: Aerofusion Early Career Initiative

CFD Solutions and Wind Tunnel Tests

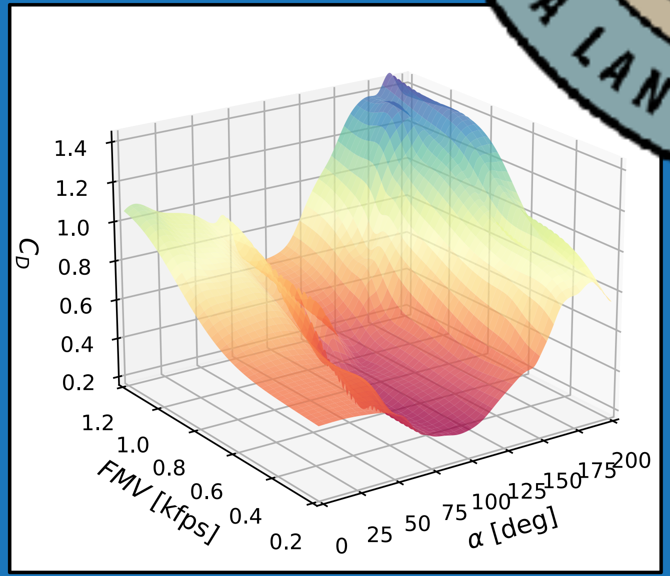


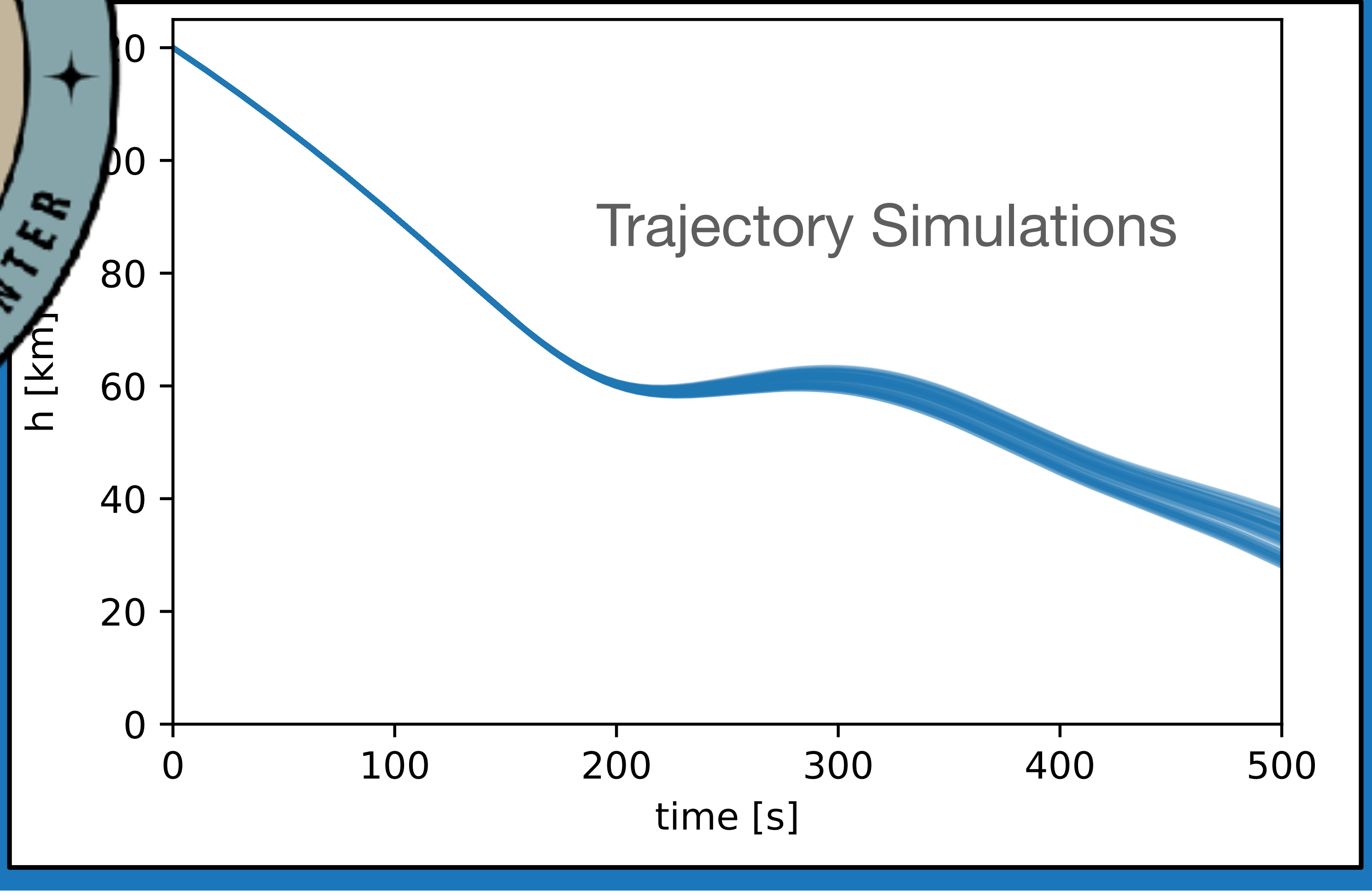
Engineering and Mission Design



Aerodynamic Database Construction

$$\begin{bmatrix} \text{FMV} & \text{Mach or Velocity} \\ \alpha & \text{Angle of Attack} \\ \beta & \text{Side-slip Angle} \\ \text{Re} & \text{Reynolds Number} \\ \vdots & \end{bmatrix} \Longrightarrow \begin{bmatrix} C_L & \text{Lift Coef.} \\ C_D & \text{Drag Coef.} \\ C_m & \text{Pitching Coef.} \\ \vdots & \end{bmatrix}$$





See recent publications for more details about the project:

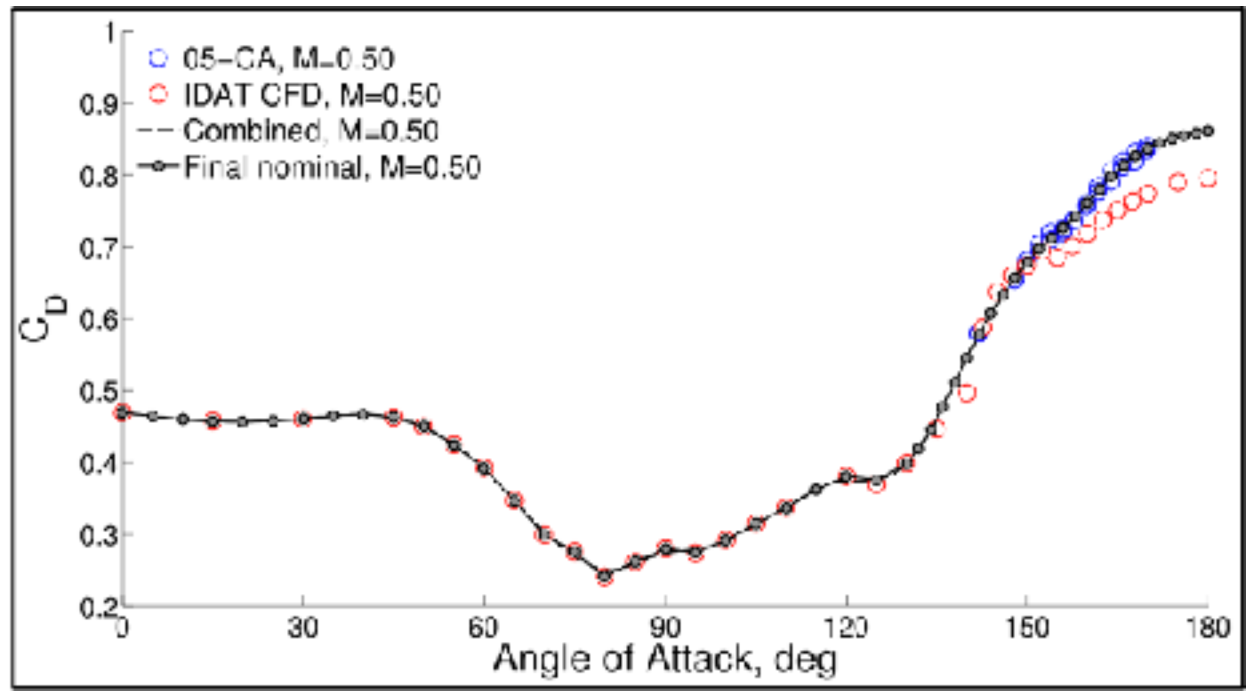
- Snyder et al. "AeroFusion: Data Fusion and Uncertainty Quantification for Lander Vehicles." *SciTech 2023.* AIAA 2023-1182.
- Scoggins et al. "Multi-hierarchy Gaussian Process Models for Probabilistic Aerodynamic Databases using Uncertain Nominal and Off- Nominal Configuration Data." *SciTech 2023*. AIAA 2023-1185.

Data fusion with uncertainties

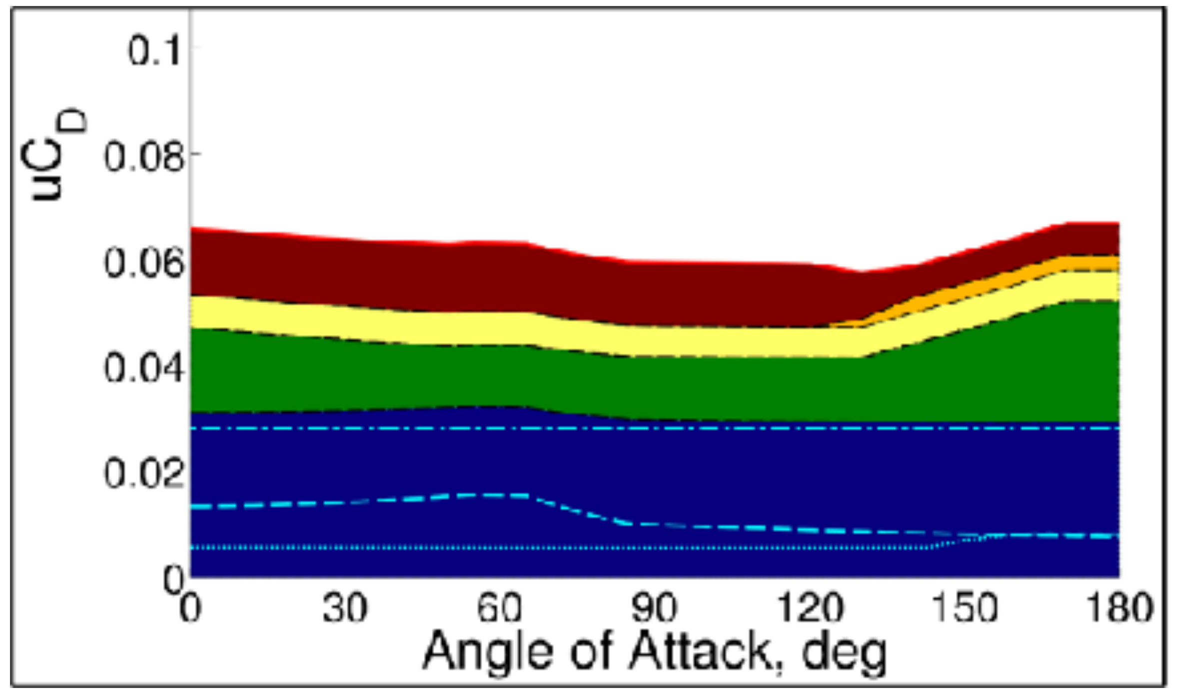
Typical data is noisy, with varying degrees of fidelity to flight vehicle

- Data continuously updated as design matures
- Different levels of fidelity in computational tools
- Wind tunnel models approximate vehicle geometry and roughness
- Wind tunnels cannot always reproduce flight conditions

Current state of the practice: “UQ by Inspection”



Nominal aerocoefficients constructed using expert judgment, given multiple sources of data.



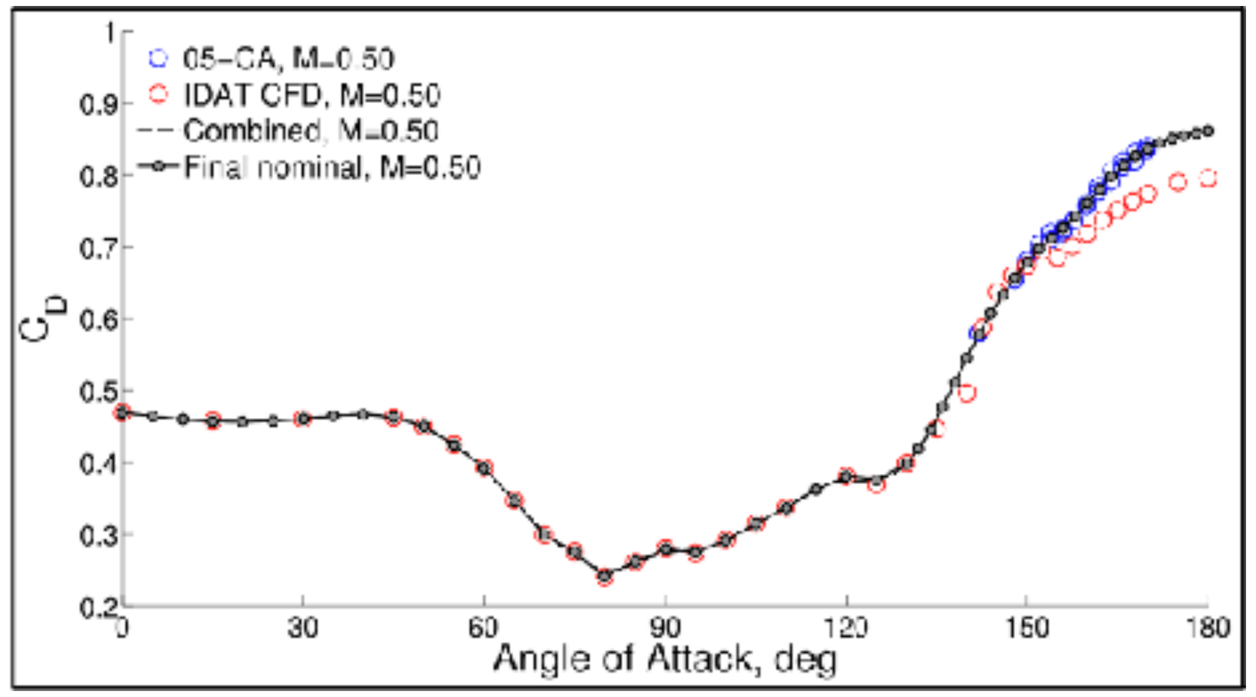
Uncertainty buildup based on dispersion factors, tuned to cover varying data sources.

Data fusion with uncertainties

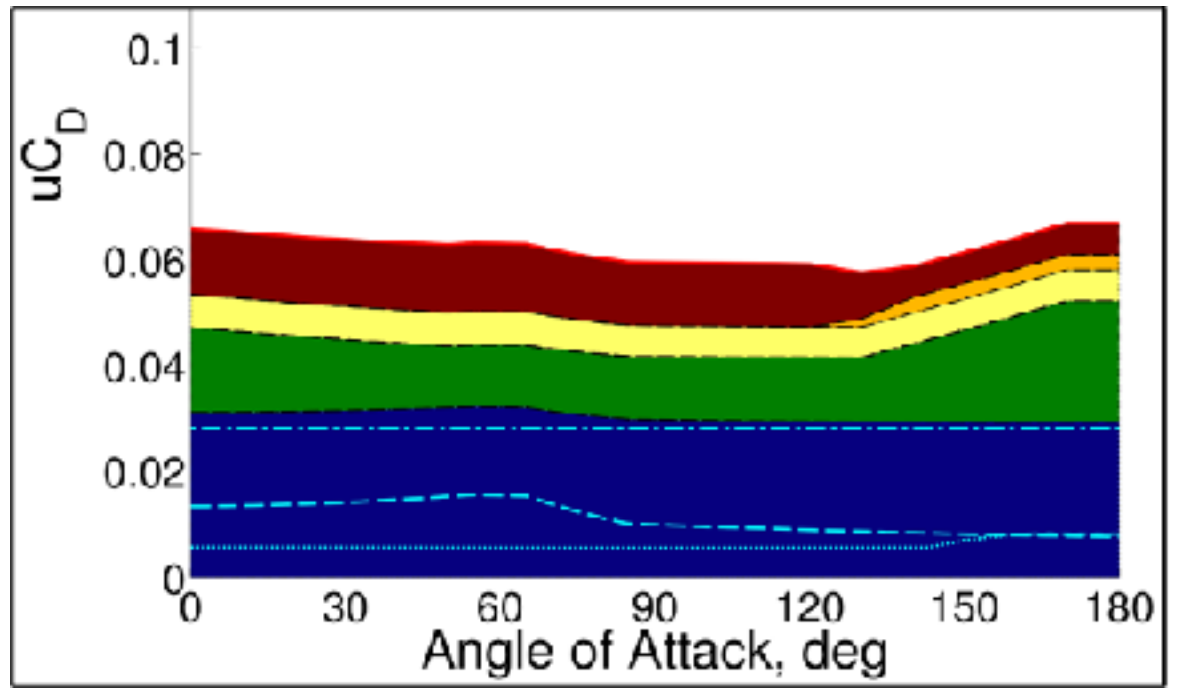
Typical data is noisy, with varying degrees of fidelity to flight vehicle

- Data continuously updated as design matures
- Different levels of fidelity in computational tools
- Wind tunnel models approximate vehicle geometry and roughness
- Wind tunnels cannot always reproduce flight conditions

Current state of the practice: "UQ by Inspection"

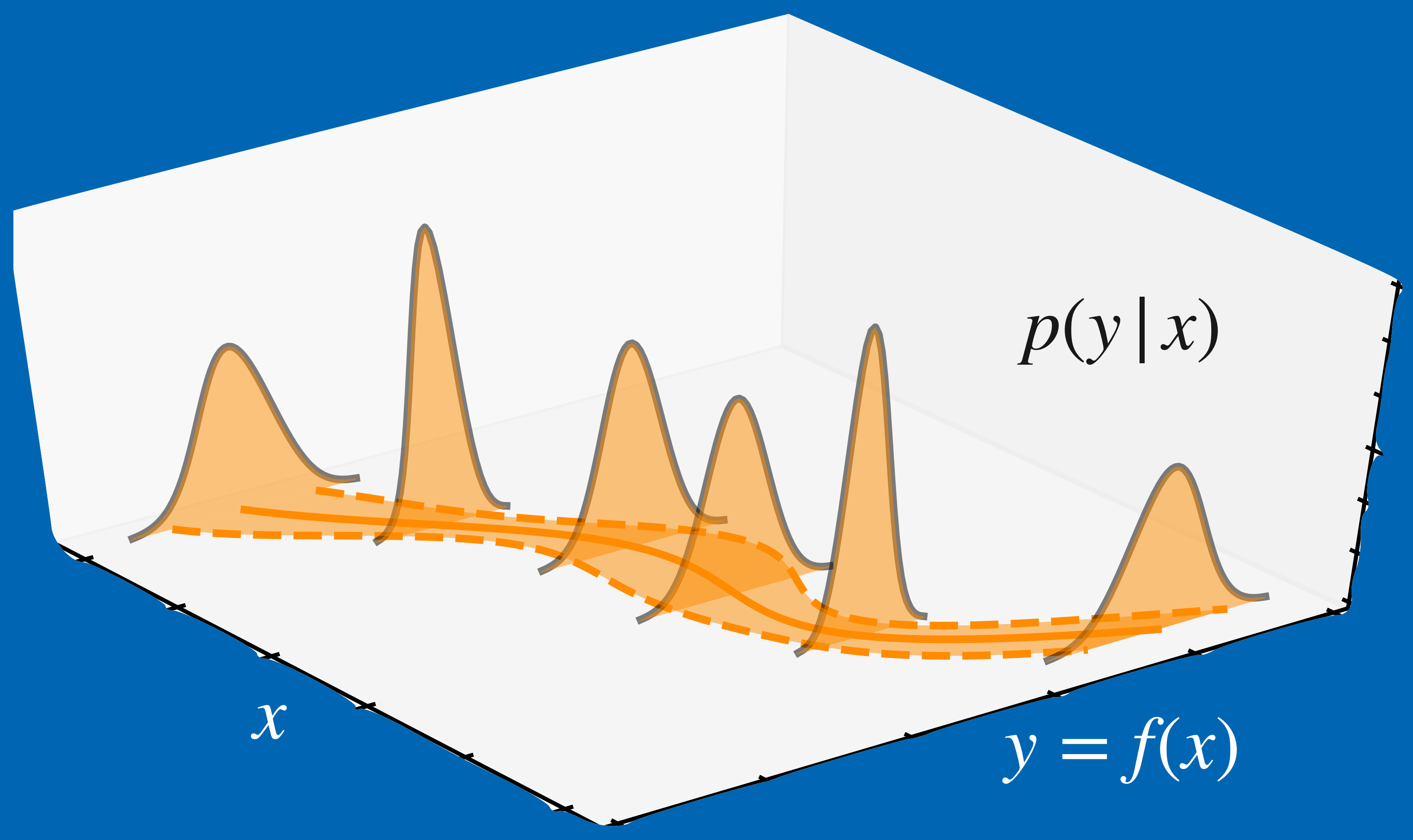


Nominal aerocoefficients constructed using expert judgment, given multiple sources of data.



Uncertainty buildup based on dispersion factors, tuned to cover varying data sources.

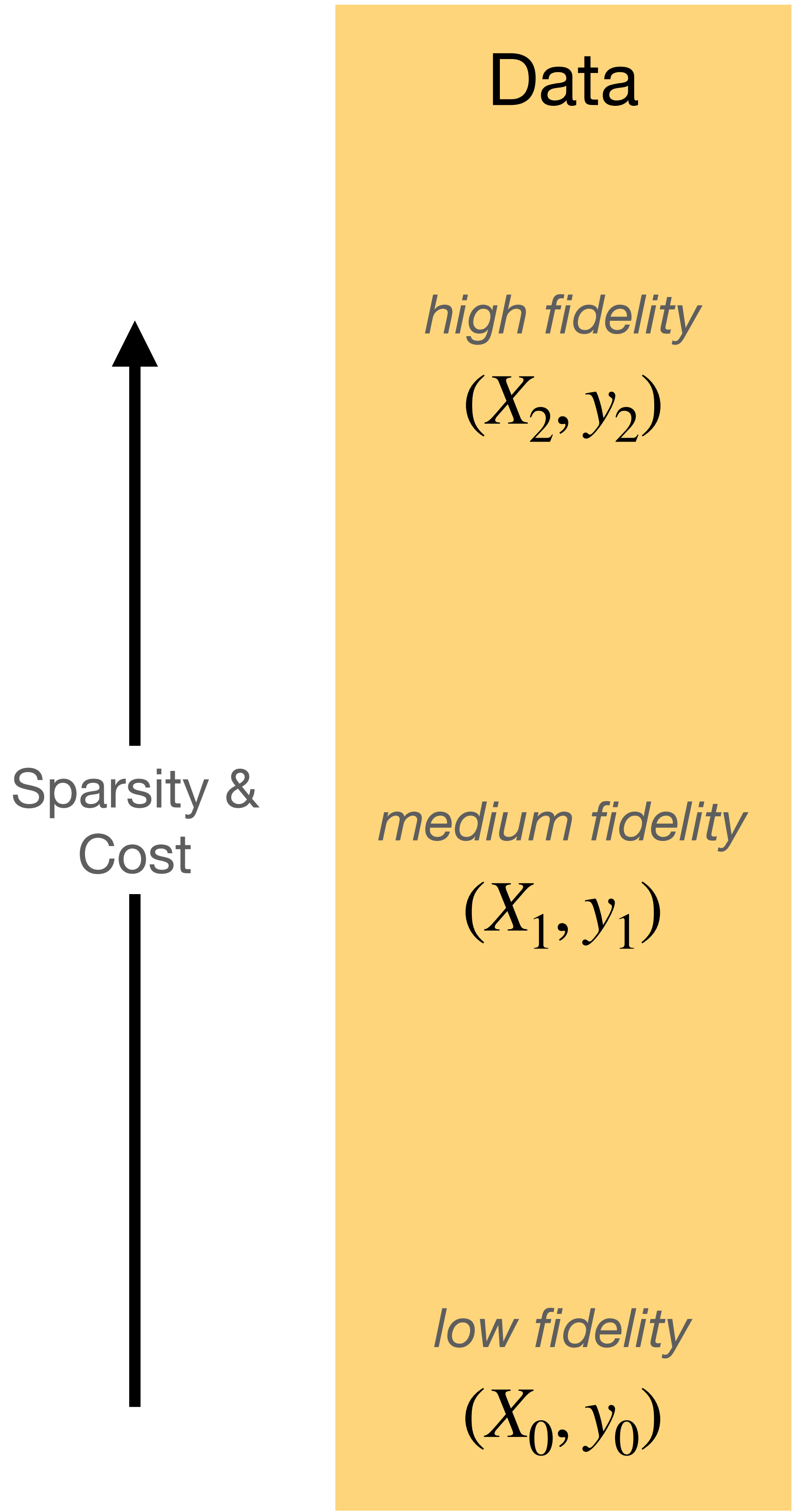
<u>Want to "learn" a surrogate conditional probability distribution, given all data sources</u>



- $p(y\,|\,x)$ defines the "probability of outcome y given x"
- surrogate model is "stochastic" but not "random"

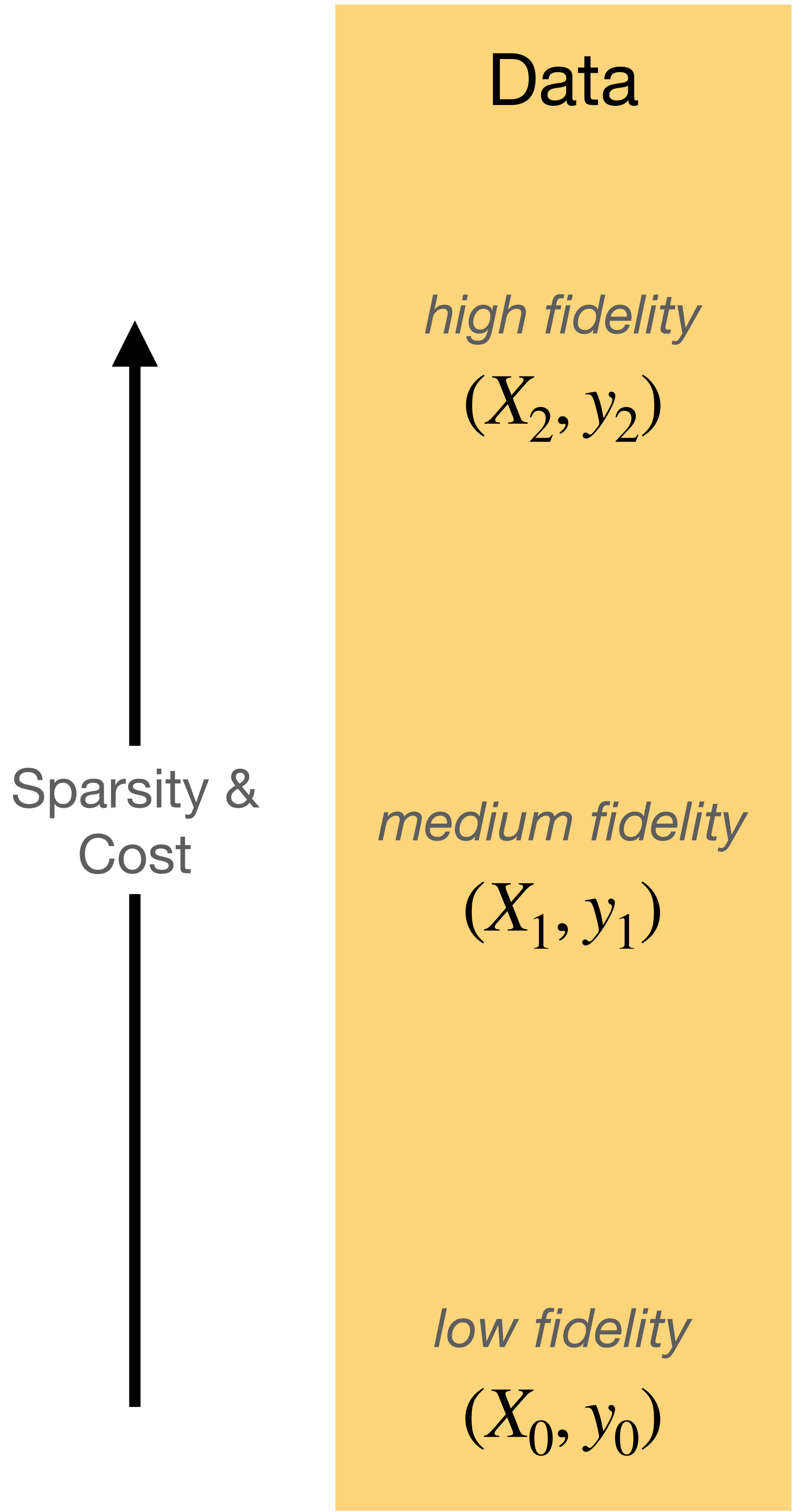
Multifidelity Gaussian Process Regression

- Multiple sources of data of increasing fidelity
  - Increasing CFD mesh resolutions
  - Heat flux correlations and 3D CFD solutions
  - CFD solutions and wind tunnel data



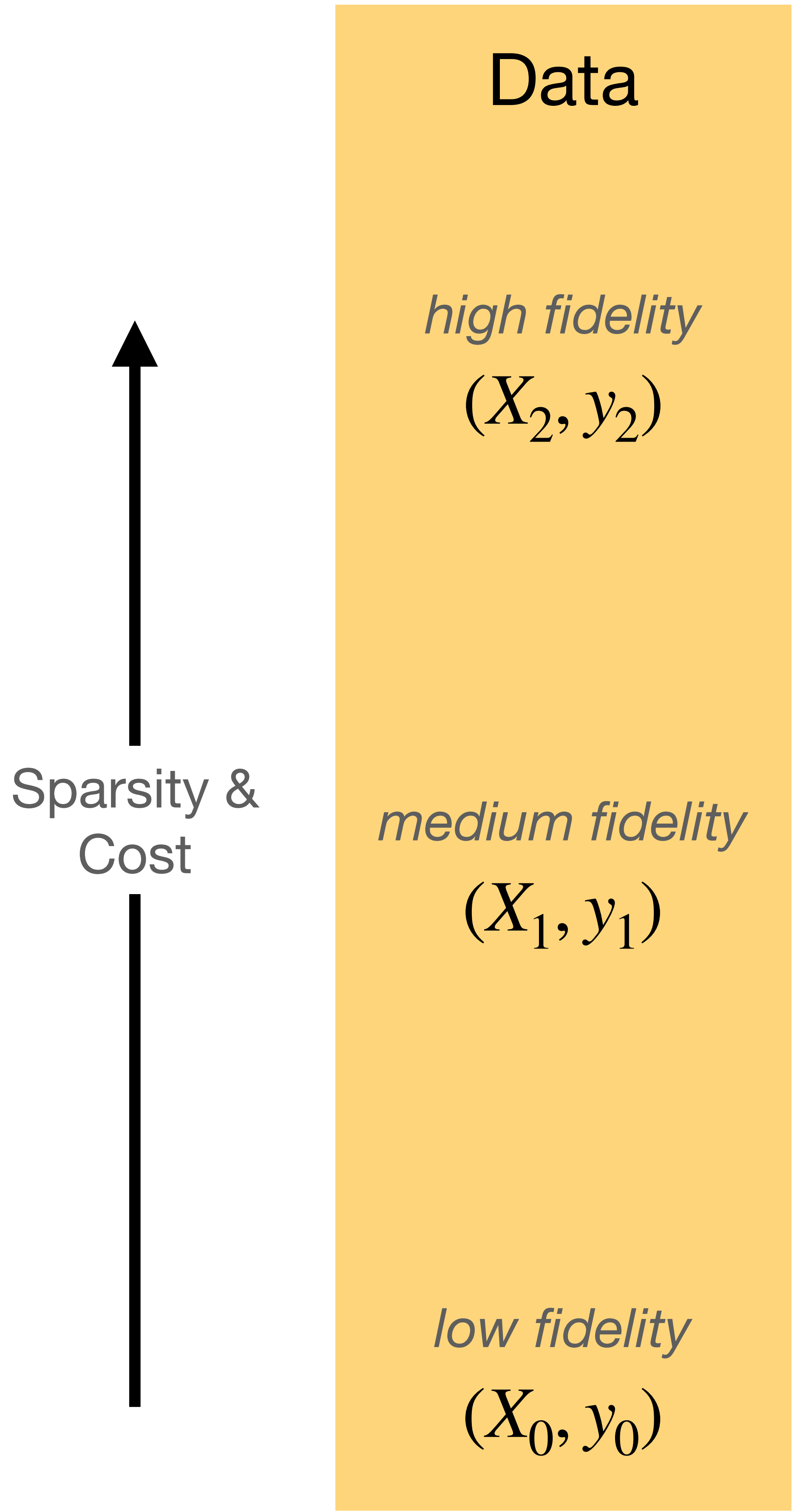
Multifidelity Gaussian Process Regression

- Multiple sources of data of increasing fidelity
  - Increasing CFD mesh resolutions
  - Heat flux correlations and 3D CFD solutions
  - CFD solutions and wind tunnel data
- Low fidelity is dense and cheap to obtain, high fidelity is sparse and expensive



Multifidelity Gaussian Process Regression

- Multiple sources of data of increasing fidelity
  - Increasing CFD mesh resolutions
  - Heat flux correlations and 3D CFD solutions
  - CFD solutions and wind tunnel data
- Low fidelity is dense and cheap to obtain, high fidelity is sparse and expensive
- **Goal:** use low fidelity data to inform high fidelity model (with uncertainties)



Multifidelity Gaussian Process Regression

- Multiple sources of data of increasing fidelity
  - Increasing CFD mesh resolutions
  - Heat flux correlations and 3D CFD solutions
  - CFD solutions and wind tunnel data
- Low fidelity is dense and cheap to obtain, high fidelity is sparse and expensive
- **Goal:** use low fidelity data to inform high fidelity model (with uncertainties)
- Autoregressive (AR1) model [1] linearly combines GP models for increasing fidelity levels

$$f_k(x) = \rho_k f_{k-1}(x) + \delta_k(x)$$

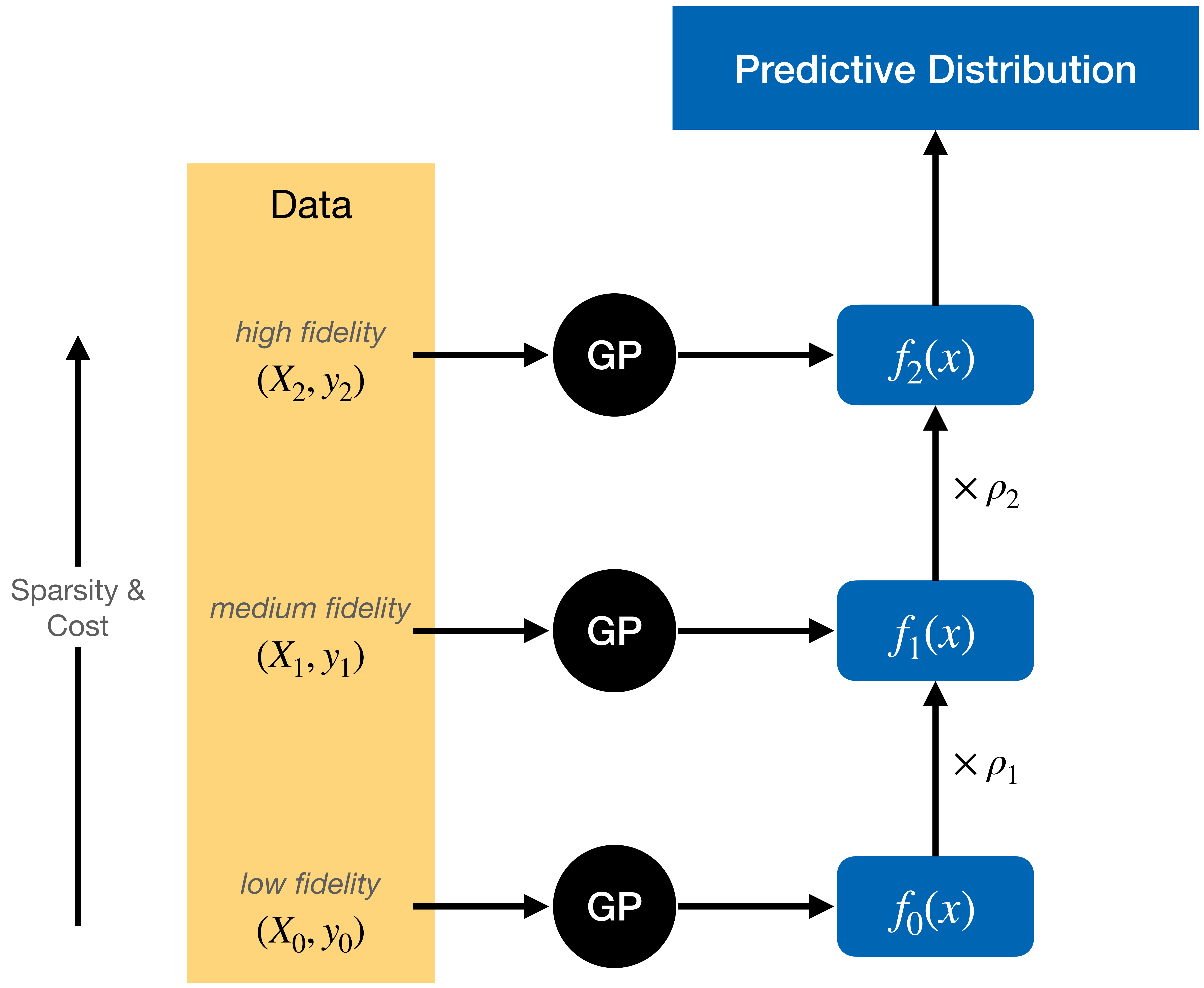


Diagram of the AR1 multifidelity GP model.

[1] Kennedy and O'Hagan. *Biometrika* 87:1-13, 2000.

Multifidelity Gaussian Process Regression

- Multiple sources of data of increasing fidelity
  - Increasing CFD mesh resolutions
  - Heat flux correlations and 3D CFD solutions
  - CFD solutions and wind tunnel data
- Low fidelity is dense and cheap to obtain, high fidelity is sparse and expensive
- **Goal:** use low fidelity data to inform high fidelity model (with uncertainties)
- Autoregressive (AR1) model [1] linearly combines GP models for increasing fidelity levels

$$f_k(x) = \rho_k f_{k-1}(x) + \delta_k(x)$$

- **Requires an obvious hierarchy of fidelity levels!**

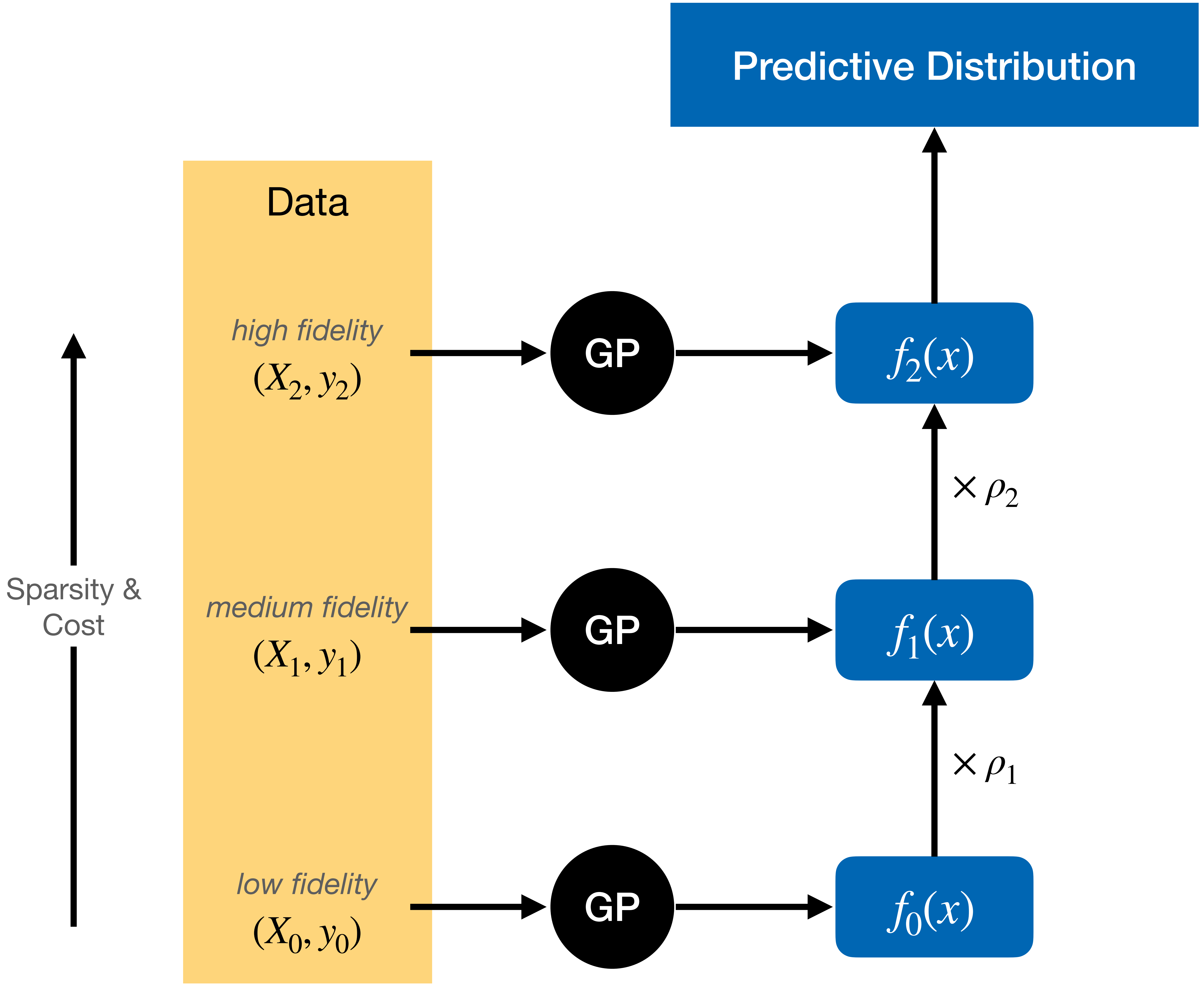


Diagram of the AR1 multifidelity GP model.

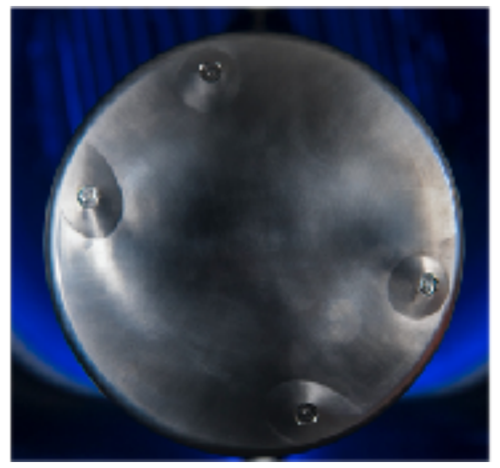
[1] Kennedy and O'Hagan. *Biometrika* 87:1-13, 2000.

Multihierarchy Gaussian Process Regression

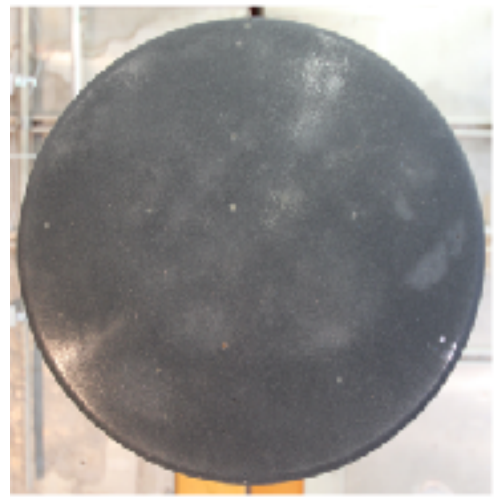
- Real world data typically cannot be organized into hierarchy of fidelity levels with single "truth"
- Easier to categorize "nominal" and "off-nominal" data

Orion heatshield models used in 133-CA test campaign in the National Transonic Facility

Asymmetric, Smooth



Symmetric, Rough



Symmetric, Smooth

Data

+ second effect

(X_2, y_2)

+ first effect

(X_1, y_1)

nominal

Multihierarchy Gaussian Process Regression

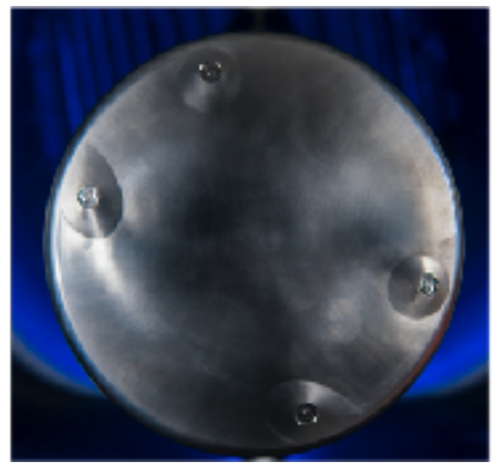
- Real world data typically cannot be organized into hierarchy of fidelity levels with single "truth"
- Easier to categorize "nominal" and "off-nominal" data

Predictive Distribution

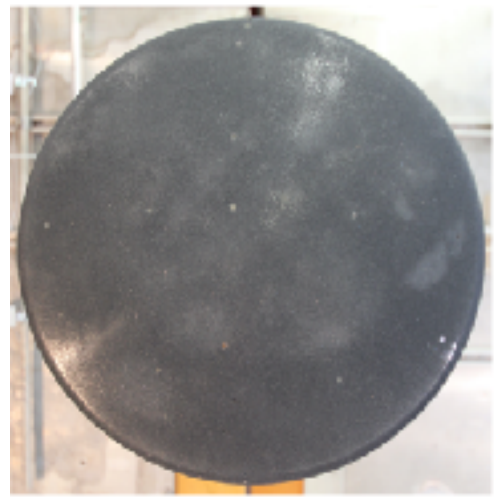
$$f(x) = f_0(x) + w_1 \Delta f_1(x) + w_2 \Delta f_2(x)$$

Orion heatshield models used in 133-CA test campaign in the National Transonic Facility

Asymmetric, Smooth



Symmetric, Rough



Symmetric, Smooth

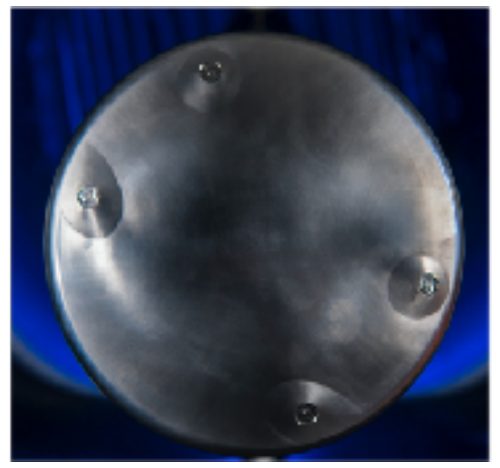
Data
+ *second effect* (X_2, y_2)
+ *first effect* (X_1, y_1)
nominal

Multihierarchy Gaussian Process Regression

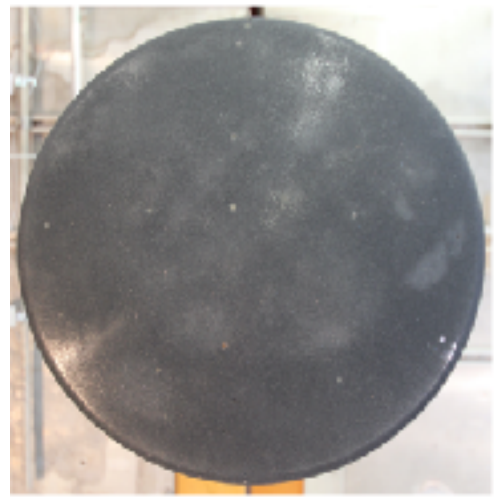
- Real world data typically cannot be organized into hierarchy of fidelity levels with single “truth”
- Easier to categorize “nominal” and “off-nominal” data

Orion heatshield models used in 133-CA test campaign in the National Transonic Facility

Asymmetric, Smooth



Symmetric, Rough



Symmetric, Smooth

Data

+ second effect

(X_2, y_2)

+ first effect

(X_1, y_1)

nominal

GP

$f_0(x)$

Predictive Distribution

$f(x) = f_0(x) + w_1 \Delta f_1(x) + w_2 \Delta f_2(x)$

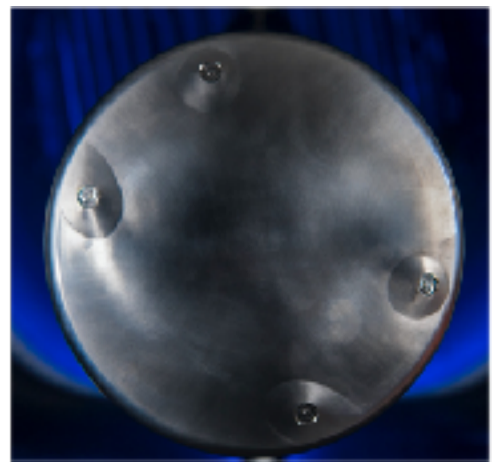
Multihierarchy Gaussian Process Regression

- Real world data typically cannot be organized into hierarchy of fidelity levels with single “truth”
- Easier to categorize “nominal” and “off-nominal” data

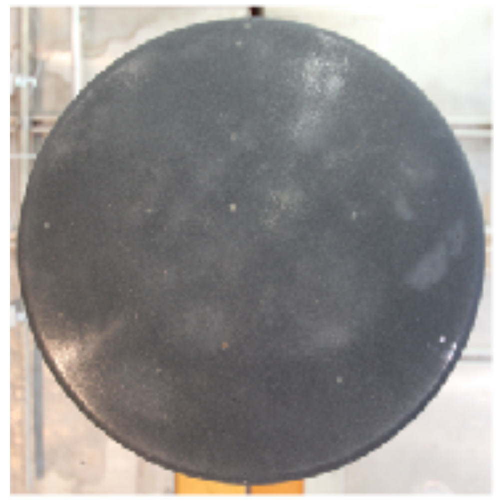
Predictive Distribution

$$f(x) = f_0(x) + w_1 \Delta f_1(x) + w_2 \Delta f_2(x)$$

Asymmetric, Smooth

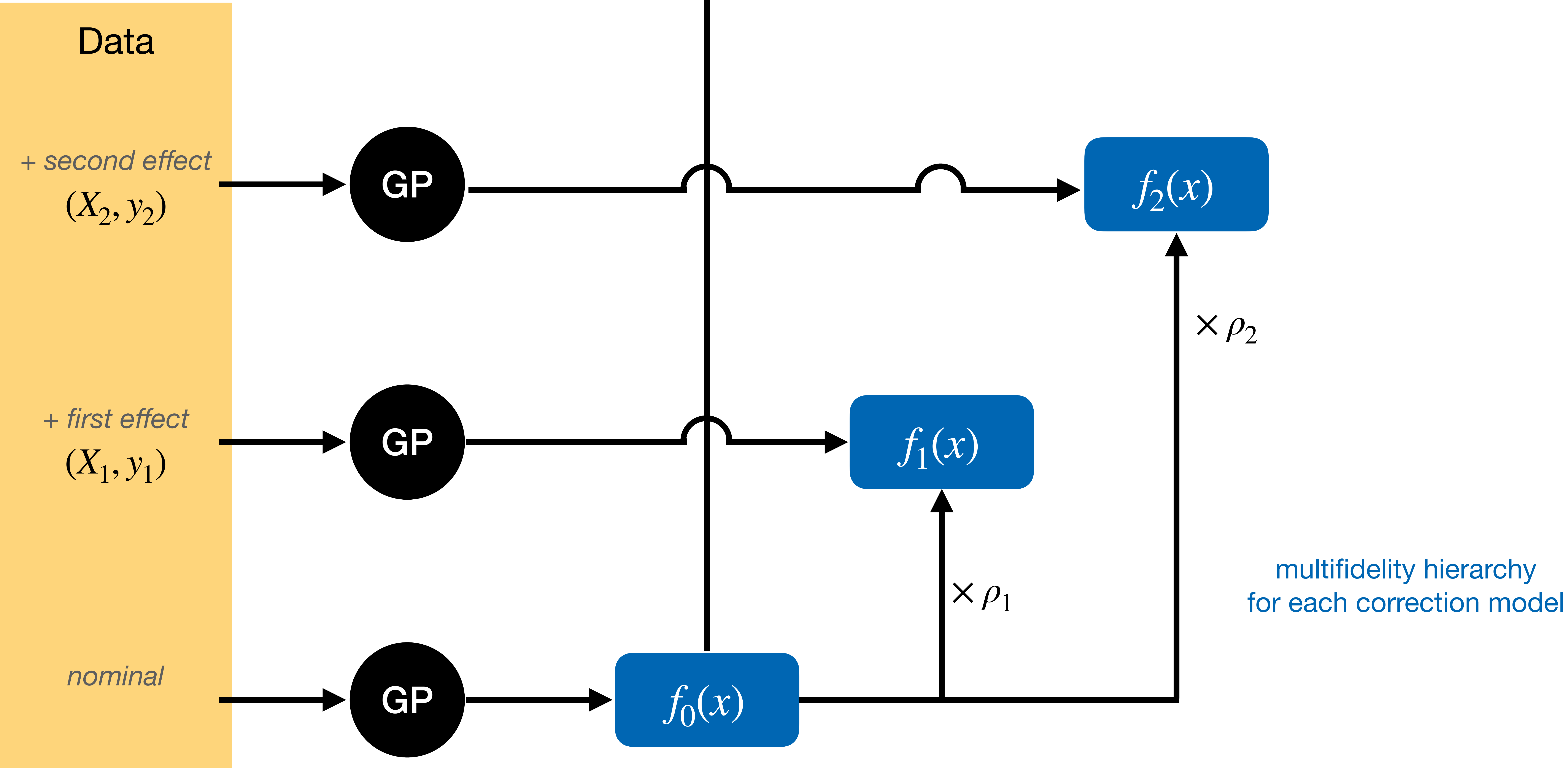


Symmetric, Rough



Symmetric, Smooth

Orion heatshield models used in 133-CA test campaign in the National Transonic Facility

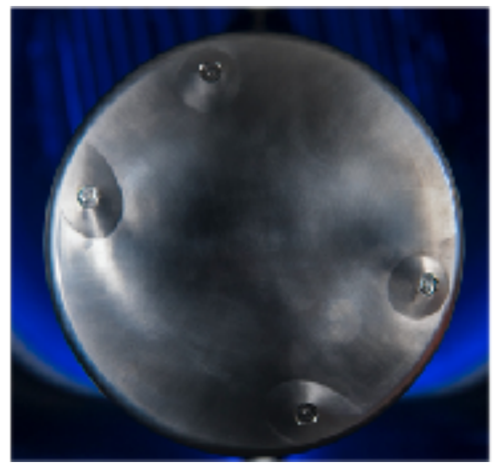


Multihierarchy Gaussian Process Regression

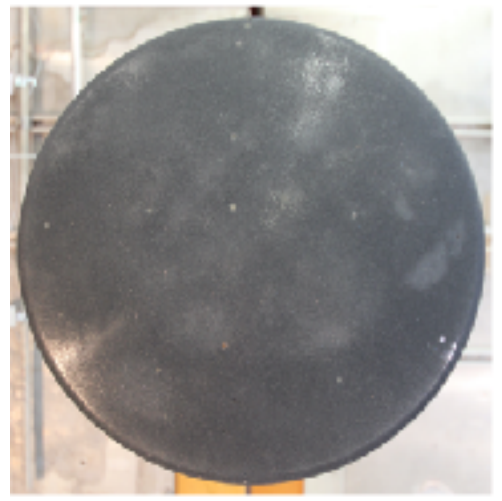
- Real world data typically cannot be organized into hierarchy of fidelity levels with single “truth”
- Easier to categorize “nominal” and “off-nominal” data

Orion heatshield models used in 133-CA test campaign in the National Transonic Facility

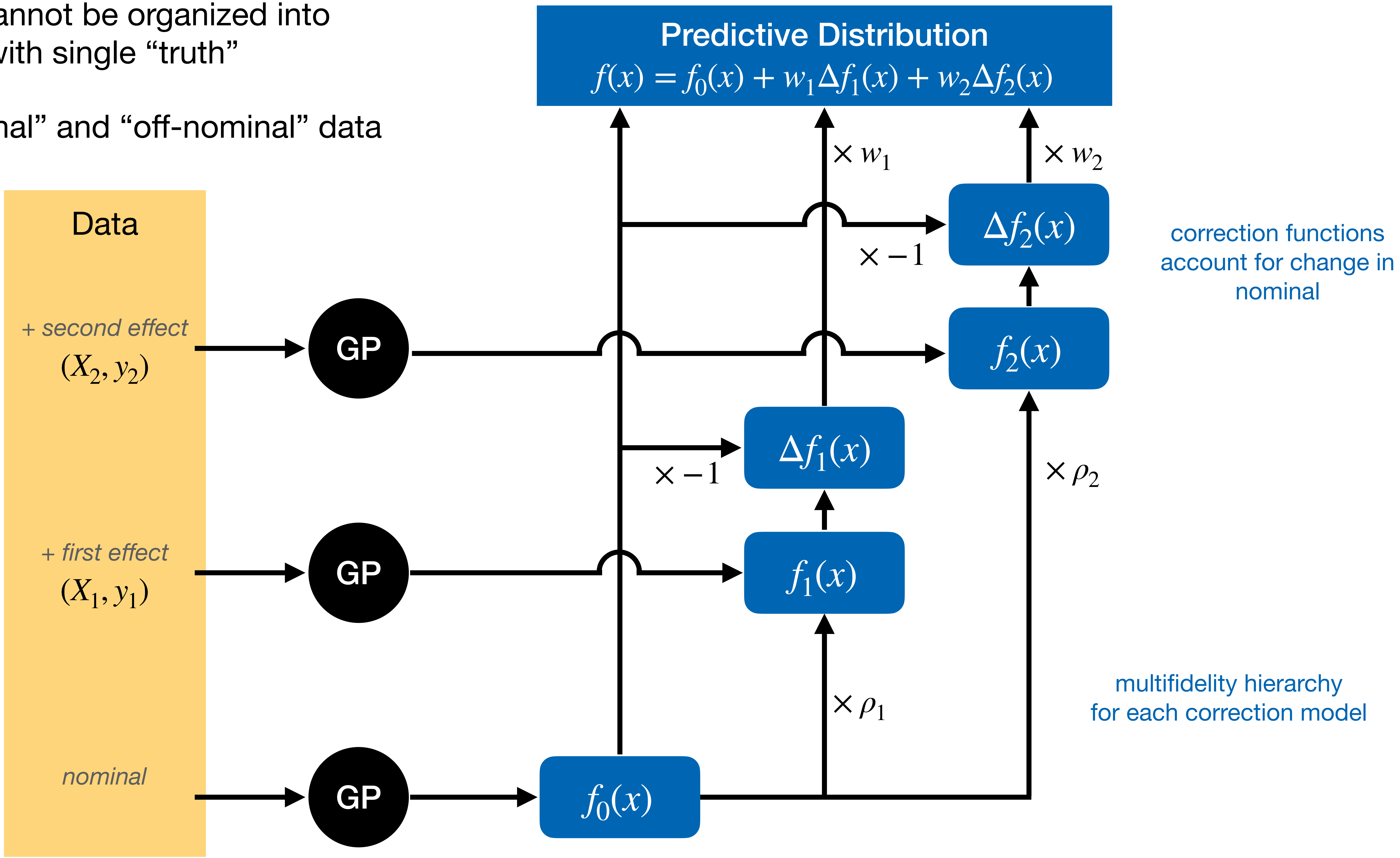
Asymmetric, Smooth



Symmetric, Rough



Symmetric, Smooth

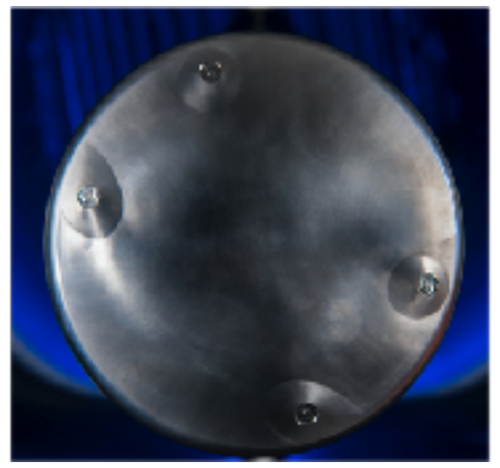


Multihierarchy Gaussian Process Regression

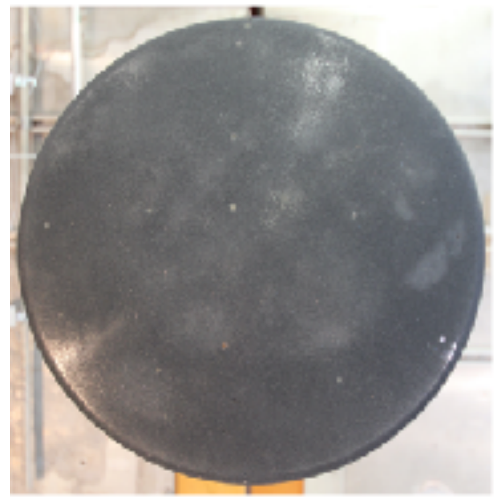
- Real world data typically cannot be organized into hierarchy of fidelity levels with single “truth”
- Easier to categorize “nominal” and “off-nominal” data

Orion heatshield models used in 133-CA test campaign in the National Transonic Facility

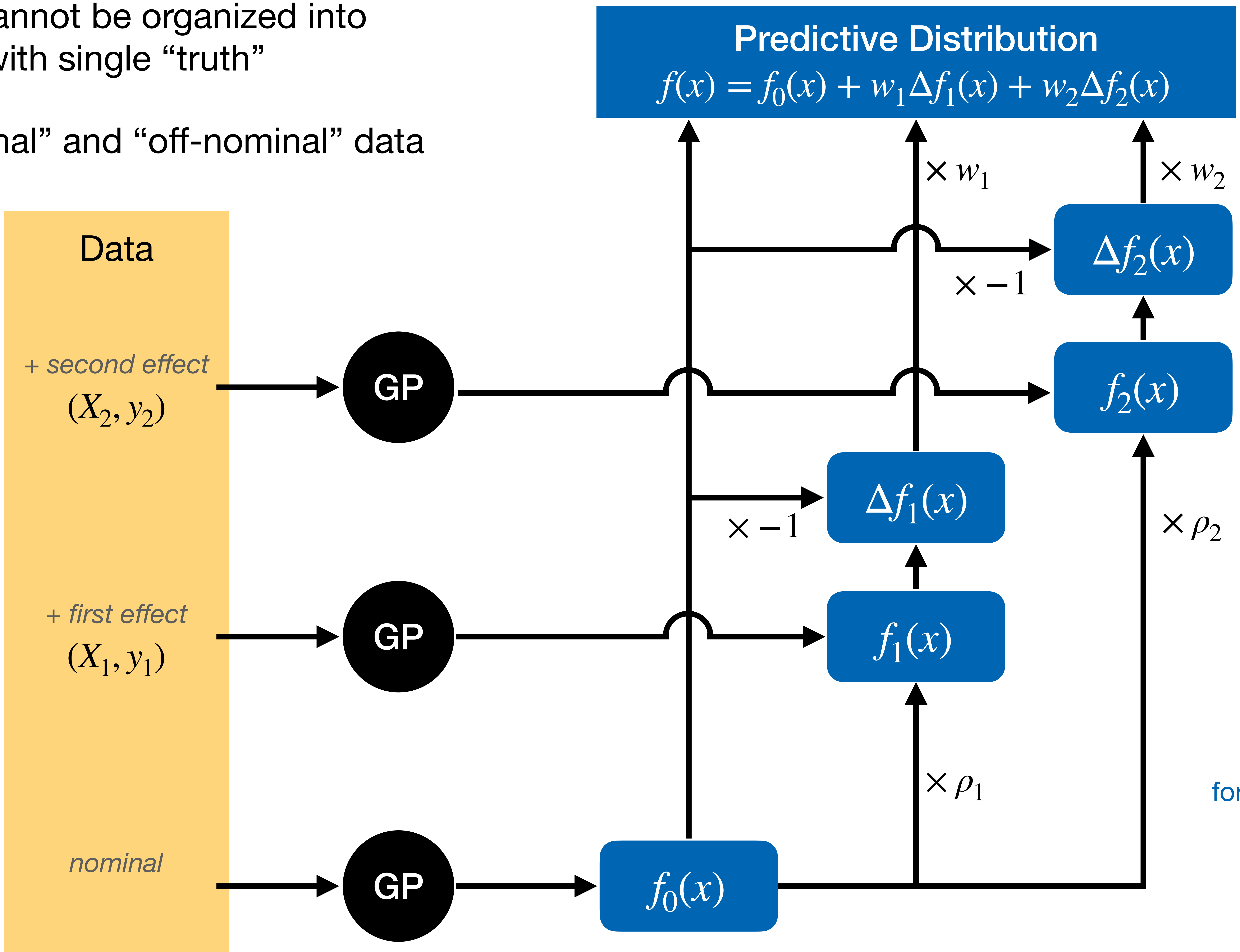
Asymmetric, Smooth



Symmetric, Rough



Symmetric, Smooth



no assumed truth!
weight parameters learned from flight data or estimated from previous vehicles

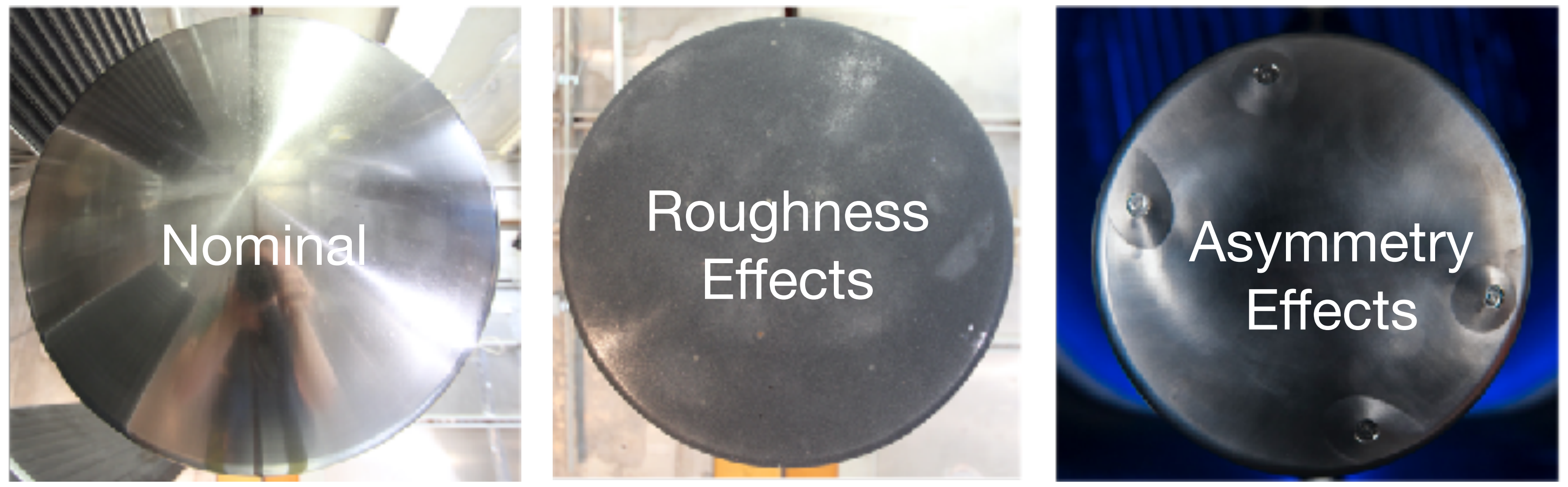
correction functions account for change in nominal

multifidelity hierarchy for each correction model

Example Orion NTF data

- All reported results based on Orion 133-CA test campaign performed in the National Transonic Facility at NASA LaRC [1]

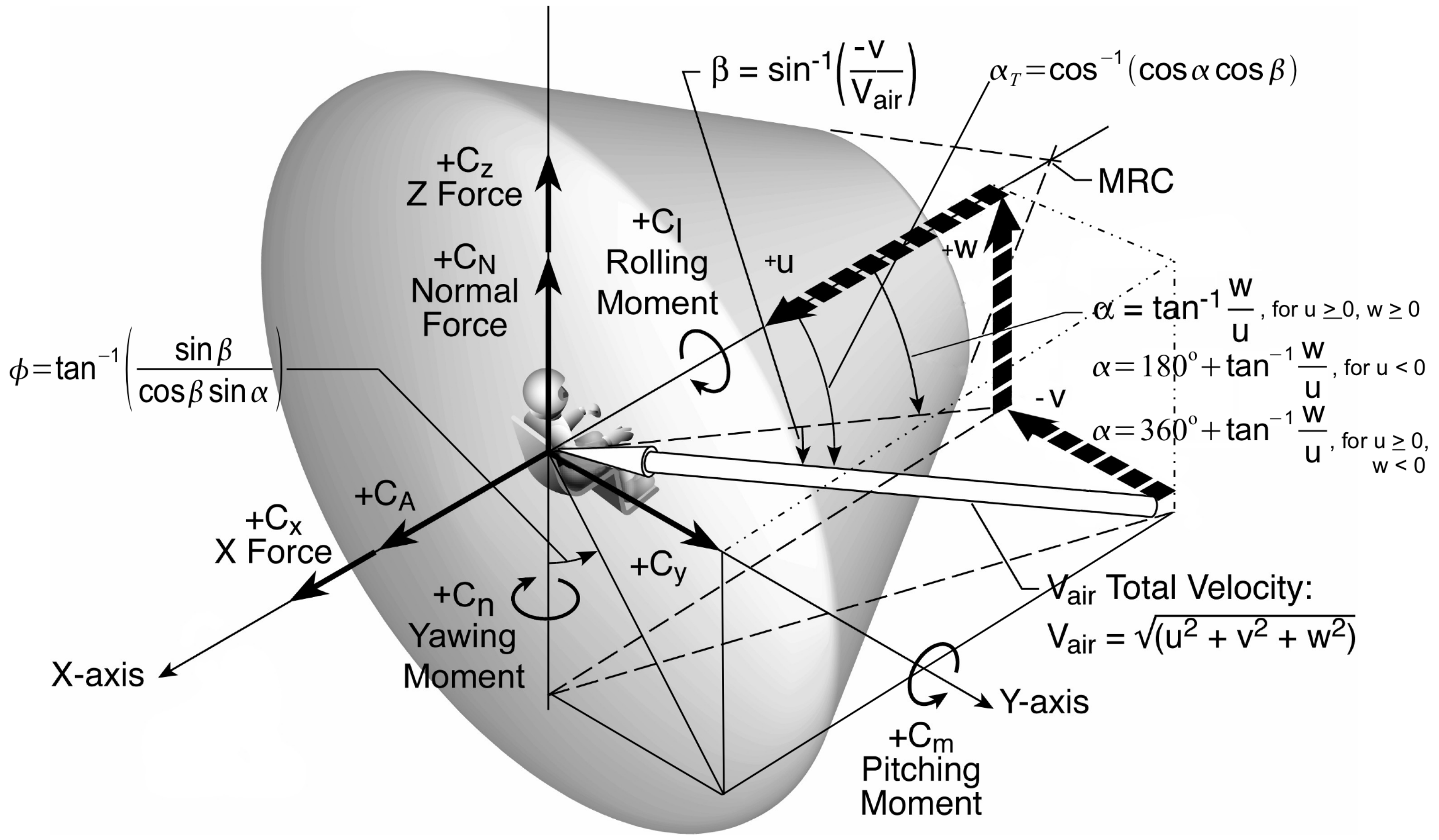
Train



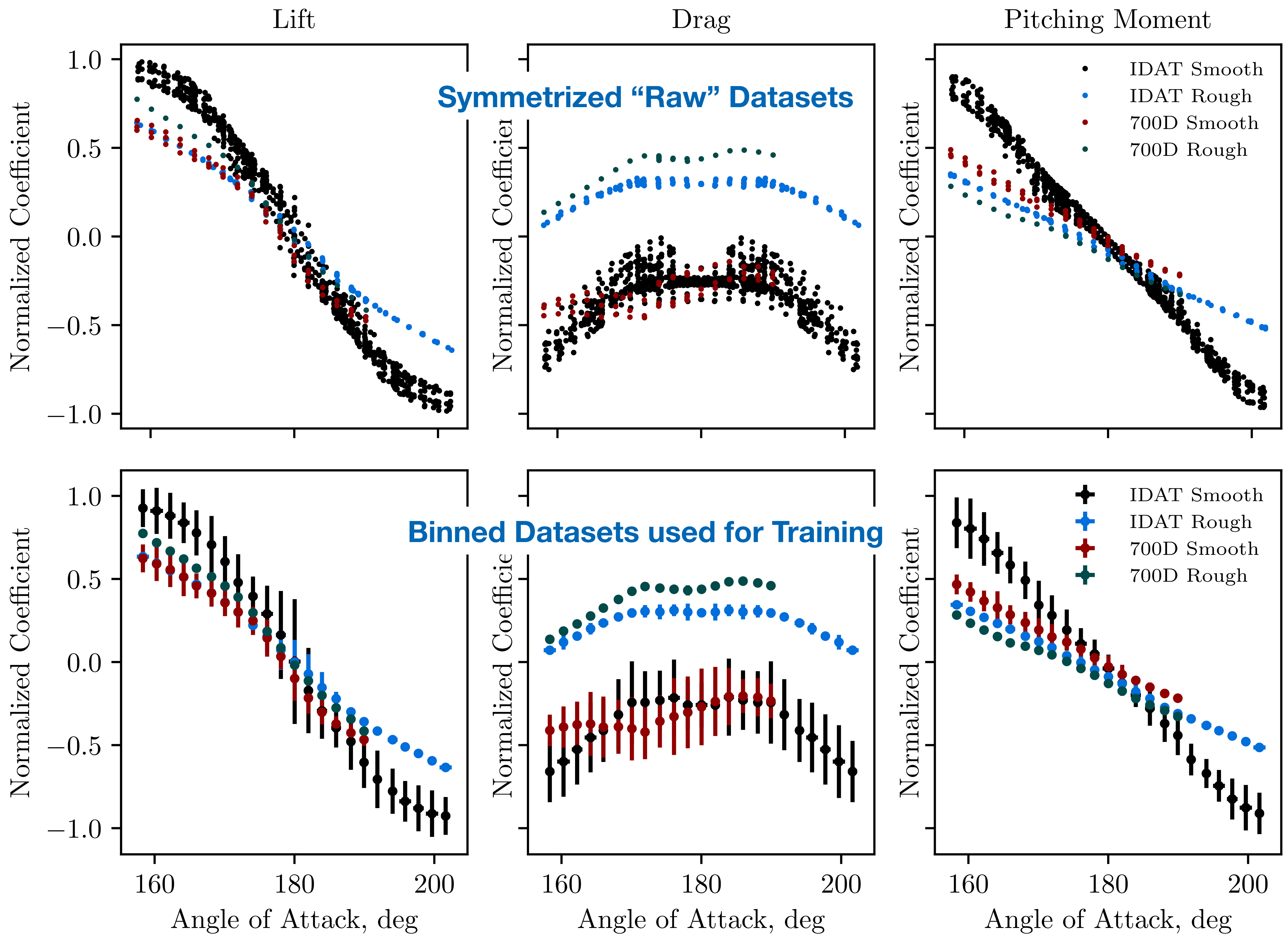
IDAT Smooth | IDAT 240 Grit | 700D Smooth

700D Fenced + 240 Grit

Test



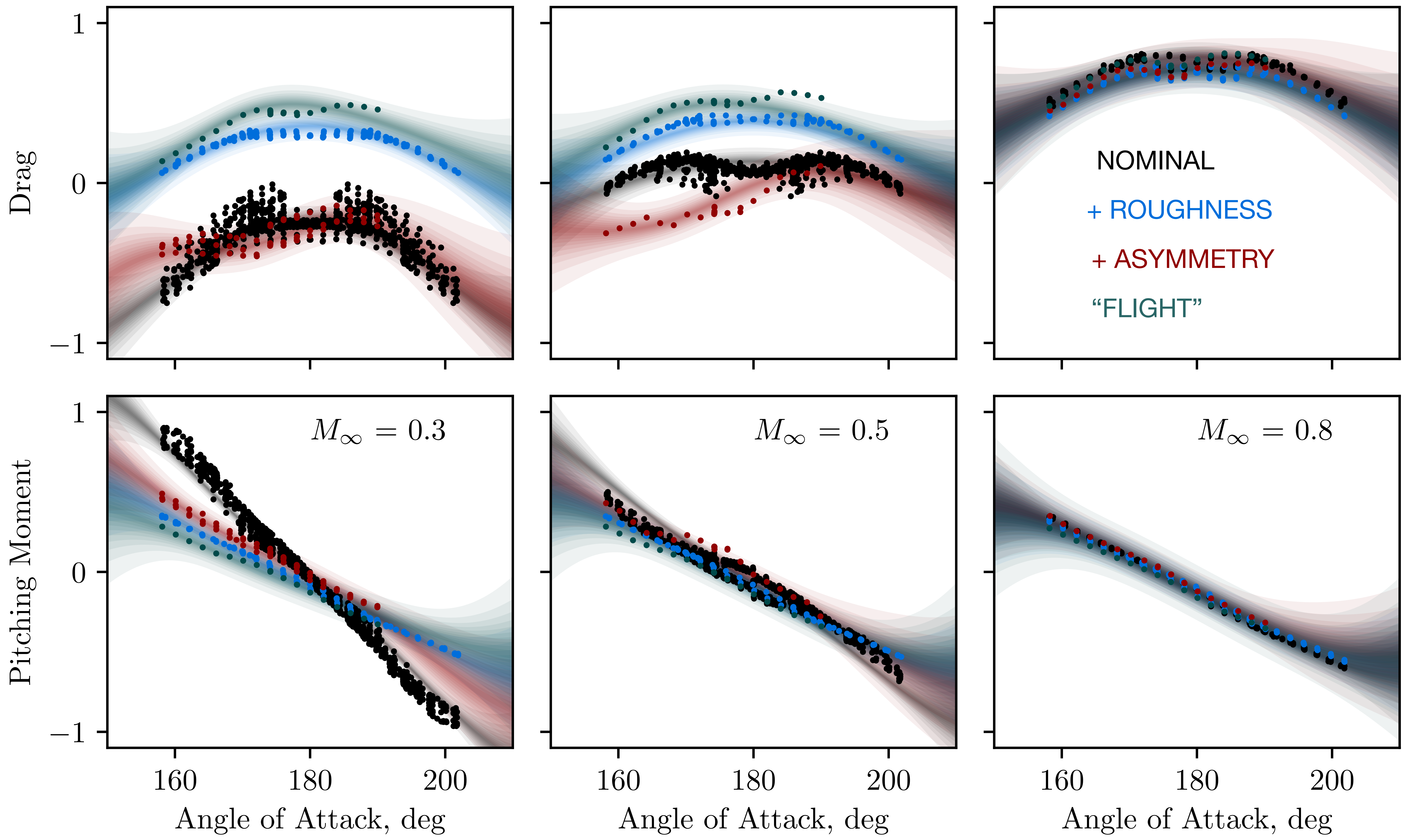
Orion "IDAT" Geometry with coordinates, forces, and moments.



Slices of data around Mach 0.3 and Reynolds 7.5x10⁶.

[1] Brauckmann. *CAP WTT Report EG-CAP-12-65*, NASA LaRC, 2022 (under preparation).

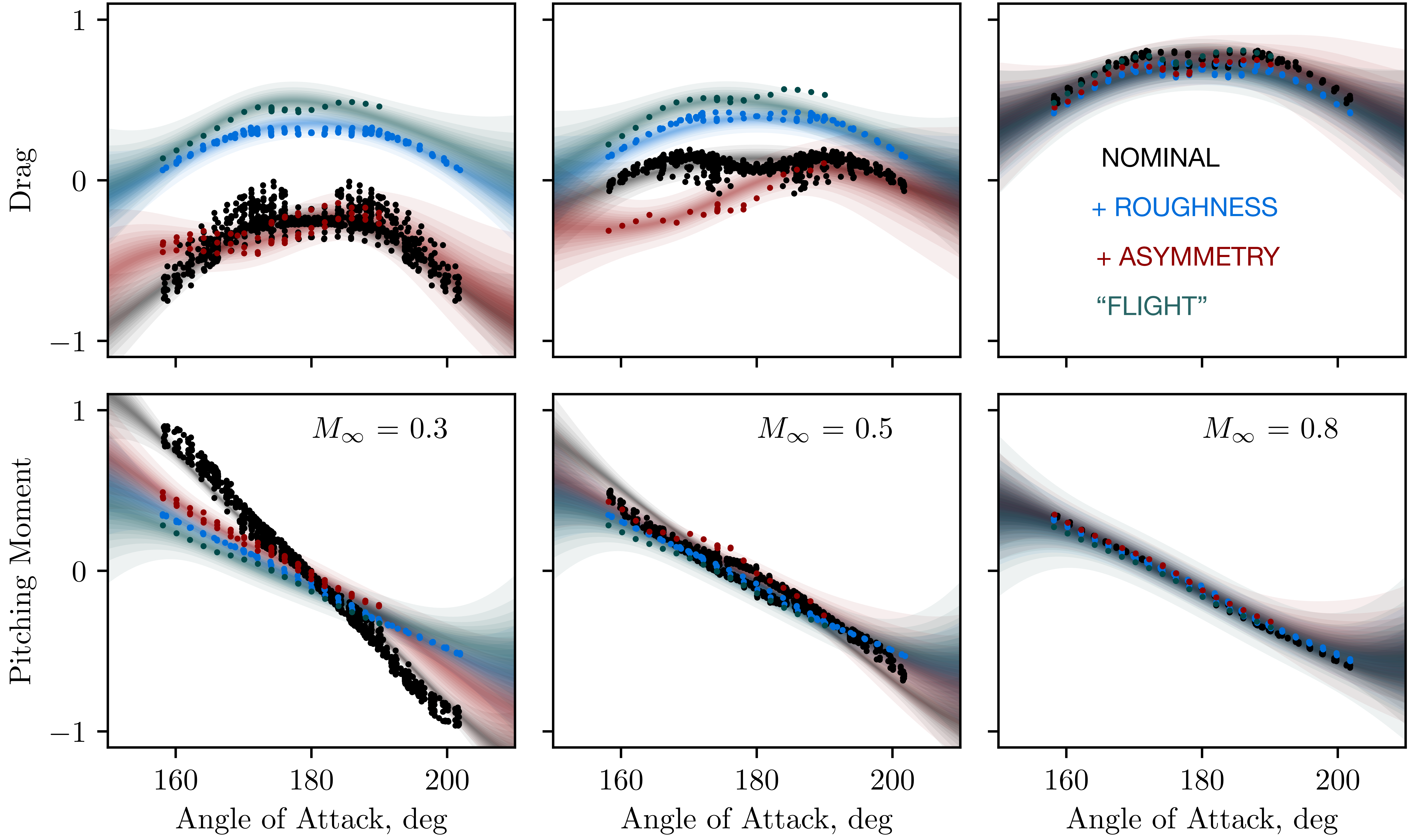
Model distributions compared to data



Normalized aerodynamic coefficient function distributions at 3 Mach numbers and Reynolds 7.5x10^6.

Model distributions compared to data

excellent agreement with “noisy” data

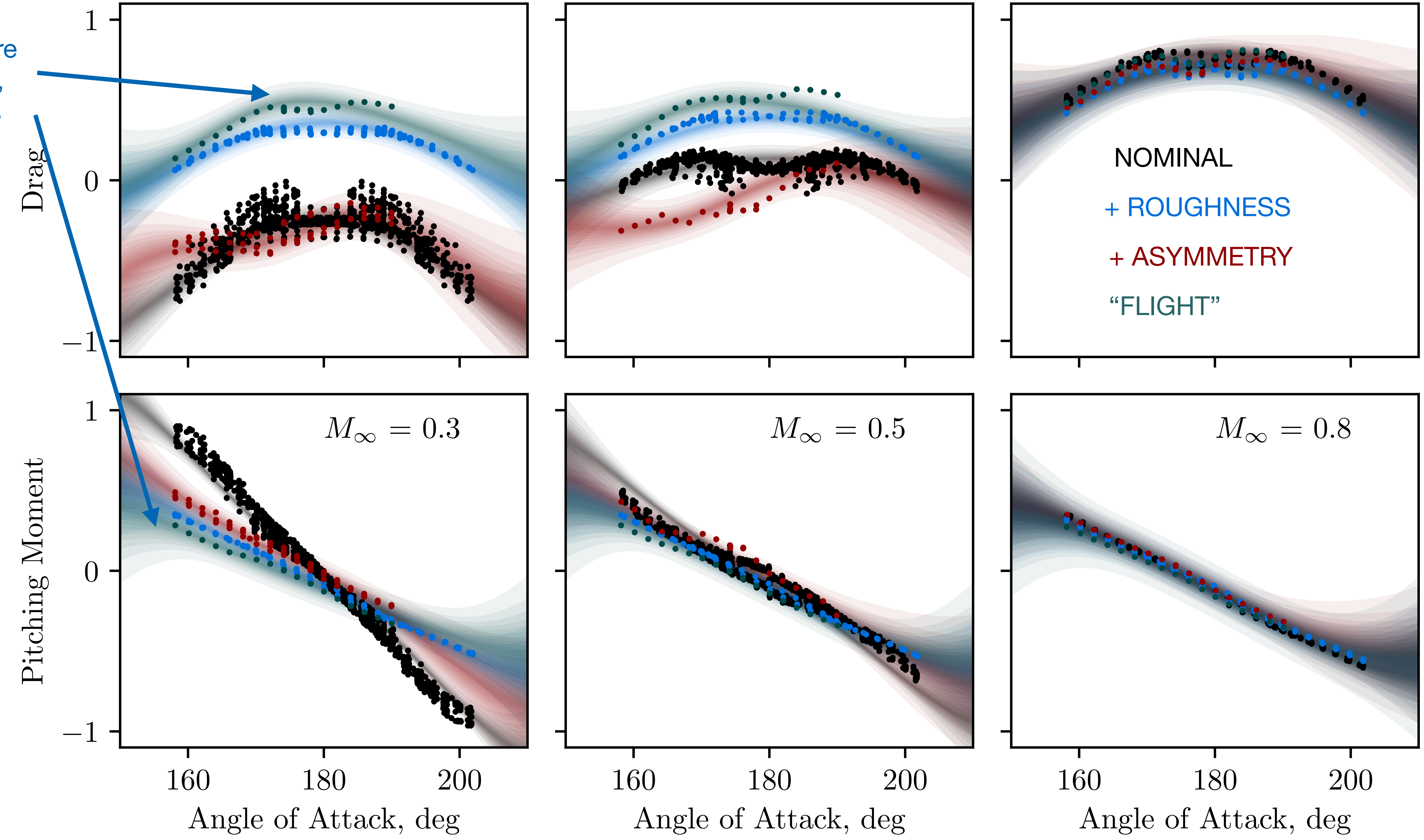


Normalized aerodynamic coefficient function distributions at 3 Mach numbers and Reynolds 7.5x10⁶.

Model distributions compared to data

simulated "flight" data are reasonably reproduced, only correction weights

excellent agreement with "noisy" data

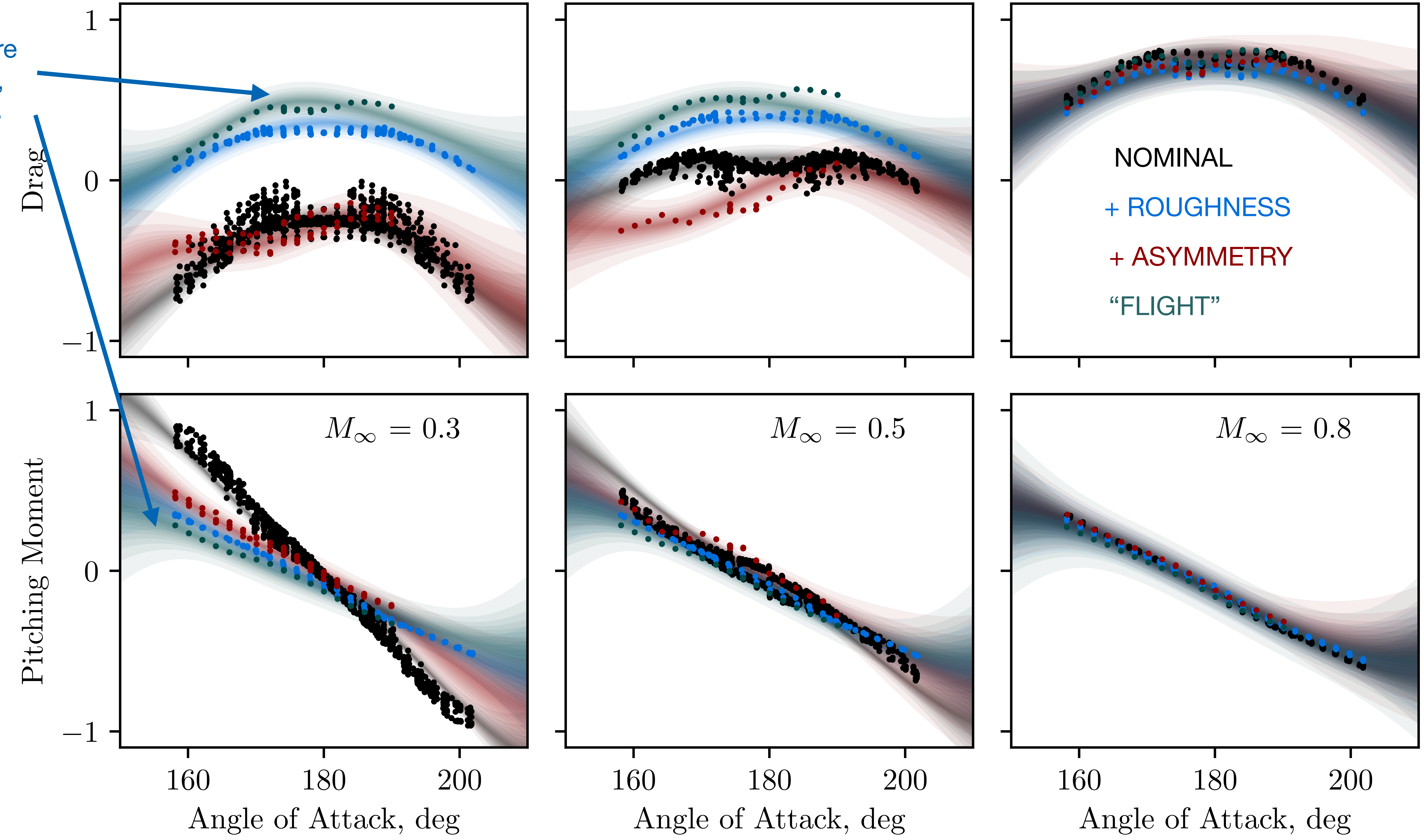


Normalized aerodynamic coefficient function distributions at 3 Mach numbers and Reynolds 7.5x10⁶.

Model distributions compared to data

simulated “flight" data are reasonably reproduced, only correction weights

excellent agreement with “noisy” data

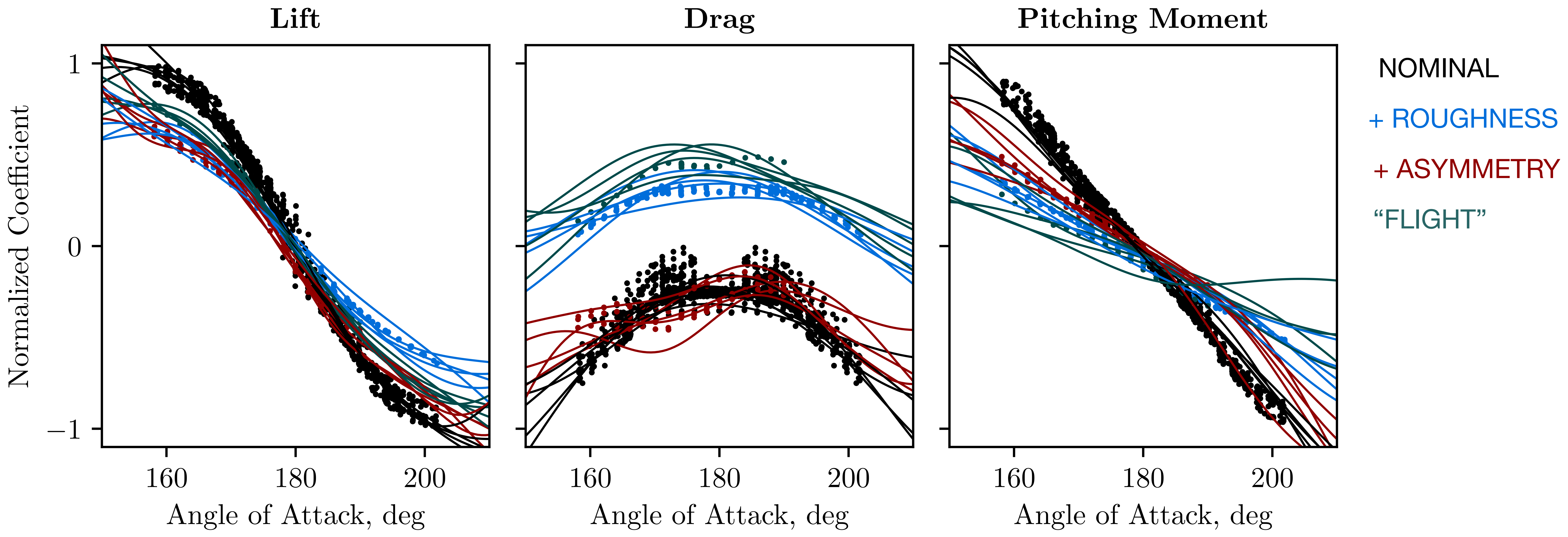


“out-of-distribution” uncertainty in regions of no data

Normalized aerodynamic coefficient function distributions at 3 Mach numbers and Reynolds 7.5x10⁶.

Function sampling from model distributions

- State-of-the-practice uses random dispersion offsets from the nominal for Monte Carlo trajectory simulations
- Proposed approach allows for function sampling that is more consistent with underlying conditional distribution
- **Each function is a plausible explanation of the data, reproduces conditional distribution in aggregate**



Aerodynamic coefficient function samples at Mach 0.5 and Reynolds 7.5x10^6.

Probability distributions for derived quantities

Model distributions can be used to obtain conditional distributions on derived quantities of interest

Probability distributions for derived quantities

Model distributions can be used to obtain conditional distributions on derived quantities of interest

- **Example:** trim angle of attack found using *Bayes' Theorem*

$$p(\alpha = \alpha_{\text{trim}}) = p(\alpha \,|\, C_{\text{m,cg}} = 0) \propto p(C_{\text{m,cg}} = 0 \,|\, \alpha)\; p(\alpha)$$

Probability distributions for derived quantities

Model distributions can be used to obtain conditional distributions on derived quantities of interest

- **Example:** trim angle of attack found using *Bayes' Theorem*

$$p(\alpha = \alpha_{\mathrm{trim}}) = p(\alpha \,|\, C_{\mathrm{m,cg}} = 0) \propto \boxed{p(C_{\mathrm{m,cg}} = 0 \,|\, \alpha)}\, p(\alpha)$$

- Probability of pitching moment being zero for given alpha is directly obtained from model distribution (Gaussian)

Probability distributions for derived quantities

Model distributions can be used to obtain conditional distributions on derived quantities of interest

- **Example:** trim angle of attack found using *Bayes' Theorem*

$$p(\alpha = \alpha_{\text{trim}}) = p(\alpha \,|\, C_{\text{m,cg}} = 0) \propto p(C_{\text{m,cg}} = 0 \,|\, \alpha)\, \boxed{p(\alpha)}$$

- Probability of pitching moment being zero for given alpha is directly obtained from model distribution (Gaussian)

- Using an "uninformative" prior for the probability of angle of attack yields the desired result

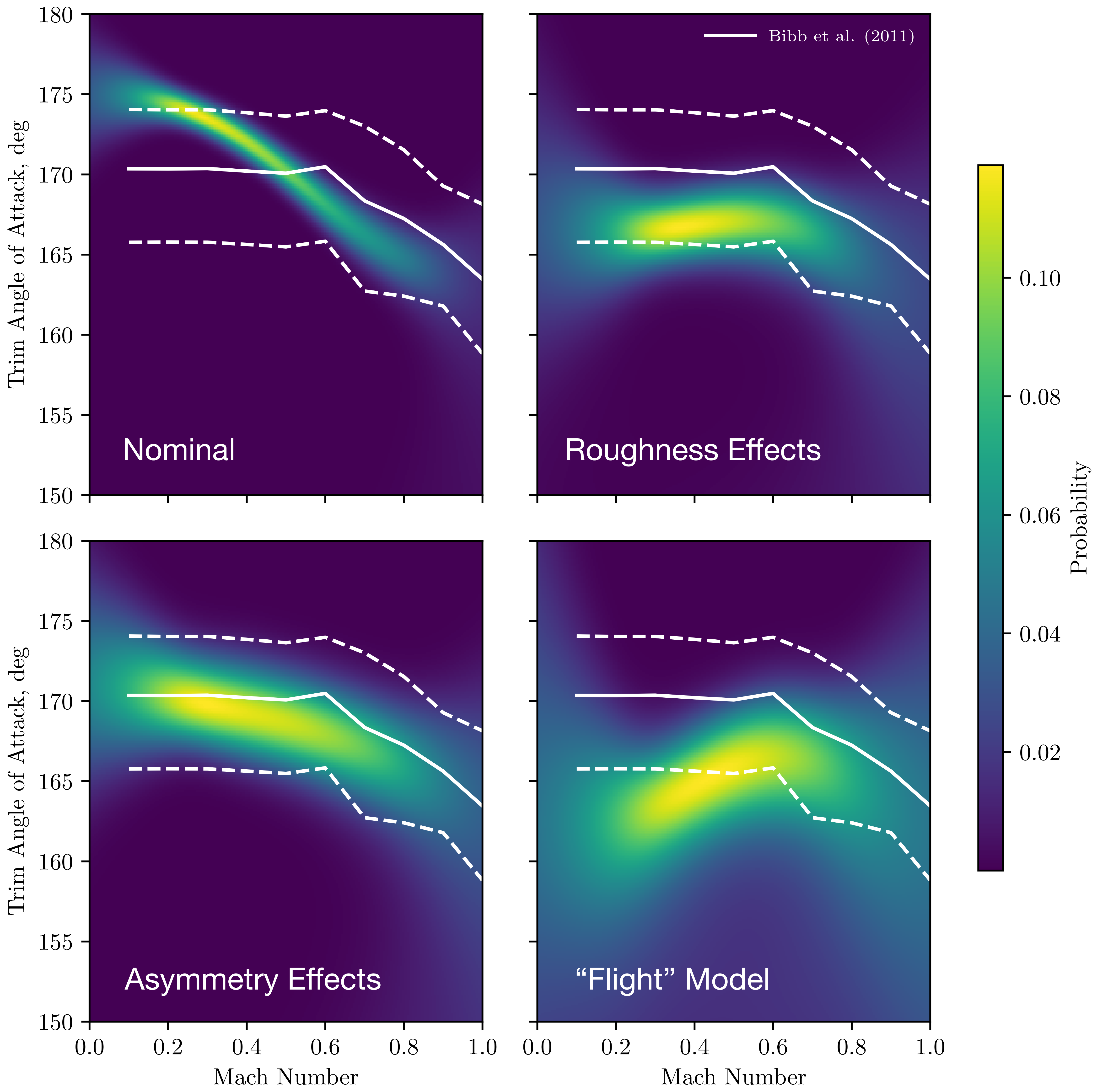
Probability distributions for derived quantities

Model distributions can be used to obtain conditional distributions on derived quantities of interest

- **Example:** trim angle of attack found using *Bayes' Theorem*

$$p(\alpha = \alpha_{\text{trim}}) = p(\alpha \,|\, C_{\text{m,cg}} = 0) \propto p(C_{\text{m,cg}} = 0 \,|\, \alpha)\; p(\alpha)$$

- Probability of pitching moment being zero for given alpha is directly obtained from model distribution (Gaussian)
- Using an "uninformative" prior for the probability of angle of attack yields the desired result



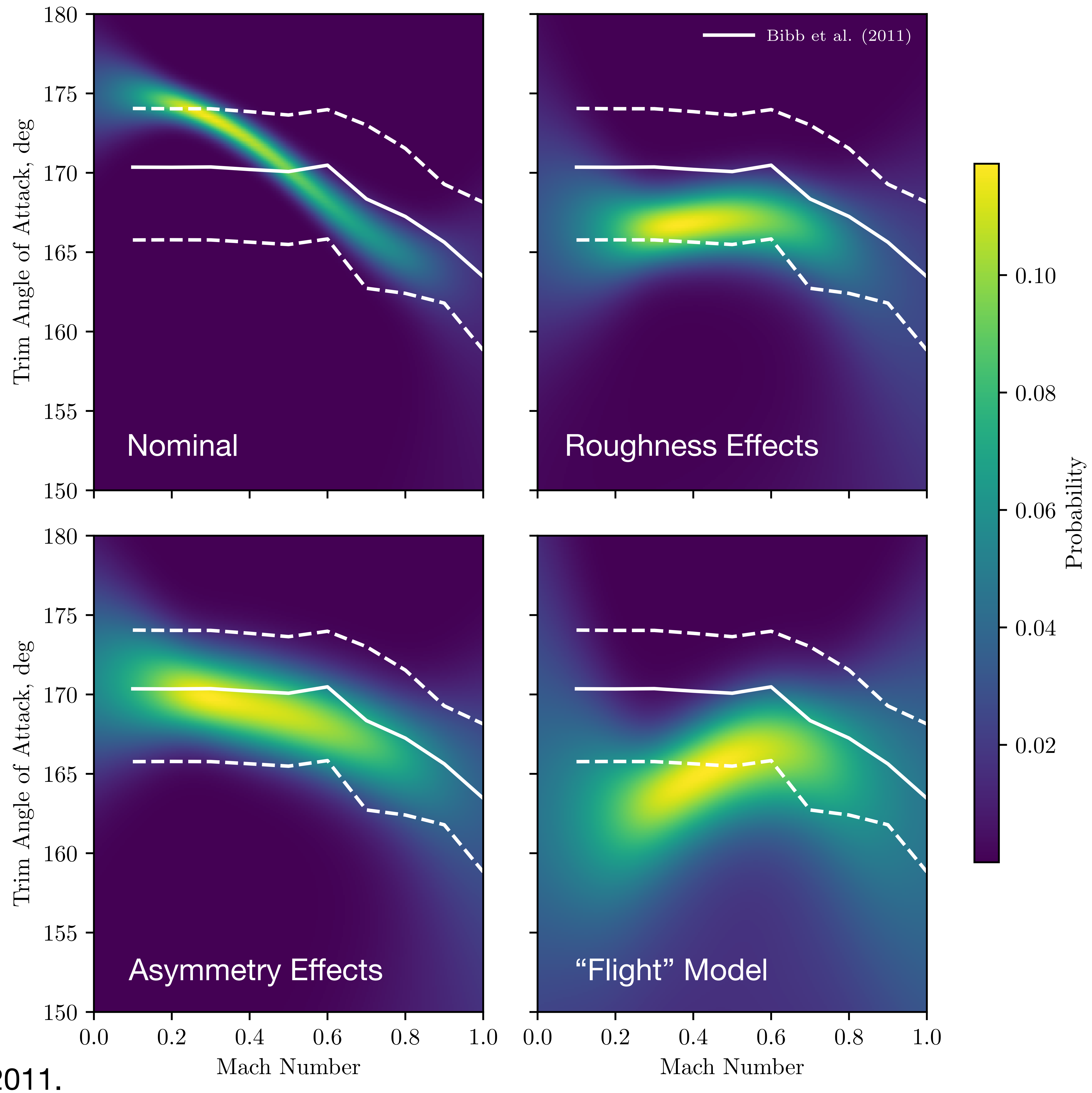
Probability distributions for derived quantities

Model distributions can be used to obtain conditional distributions on derived quantities of interest

- **Example:** trim angle of attack found using *Bayes' Theorem*

$$p(\alpha = \alpha_{\text{trim}}) = p(\alpha \,|\, C_{\text{m,cg}} = 0) \propto p(C_{\text{m,cg}} = 0 \,|\, \alpha)\; p(\alpha)$$

- Probability of pitching moment being zero for given alpha is directly obtained from model distribution (Gaussian)
- Using an "uninformative" prior for the probability of angle of attack yields the desired result
- Orion v0.60 DB nominal and 100% CI bounds [1] provided for comparison



[1] Bibb, Walker, Brauckmann, Robinson. *29th AIAA Applied Aero. Conf.,* No. 2011-3507, 2011.

Questions?

Where to find additional resources:

- **Books I recommend**
  - I. Goodfellow, Y. Bengio, A. Courville. *Deep Learning*. MIT Press, 2016. (www.deeplearningbook.org)
  - C.E. Rasmussen, C.K.I. Williams. *Gaussian Processes for Machine Learning*. MIT Press, 2006. (gaussianprocess.org/gpml)
  - S. Rogers, M. Girolami. *A First Course in Machine Learning, 2nd Ed.* CRC Press, 2017.
  - D.S. Sivia. *Data Analysis: A Bayesian Tutorial*. Oxford University Press, 2006.
  - R.B. Gramacy. *Surrogates: Gaussian Process Modeling, Design, and Optimization for the Applied Sciences*. CRC Press, 2020. (bobby.gramacy.com/surrogates)
- **Free online courses**
  - Stanford CS230: Deep Learning. Video lectures available at cs230.stanford.edu/lecture.
  - MIT 6.036: Introduction to Machine Learning. Course notes and lectures at openlearninglibrary.mit.edu/courses/course-v1:MITx+6.036+1T2019.
- **Python packages**: scikit-learn, Pytorch, Tensorflow, JAX, GPy, …

Backup

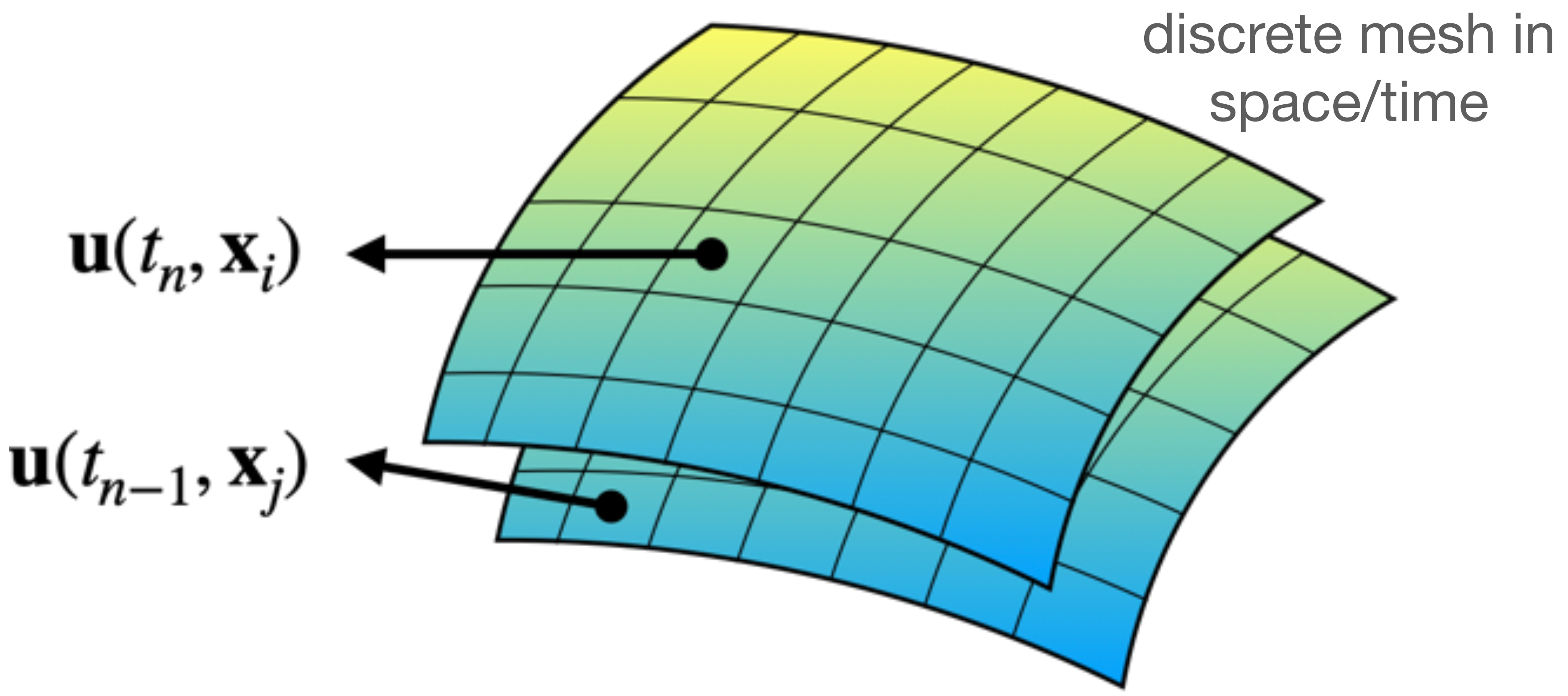
Physics Informed Neural Networks

- Networks are trained with/without data, but regularized using physical laws
- Loss function constructed from data term and residuals of governing equations
- Boundary conditions treated like data (constrained) or enforced by construction (unconstrained) of the neural network

Physics Informed Neural Networks

- Networks are trained with/without data, but regularized using physical laws
- Loss function constructed from data term and residuals of governing equations
- Boundary conditions treated like data (constrained) or enforced by construction (unconstrained) of the neural network

Conventional Discretization Approaches (CFD)

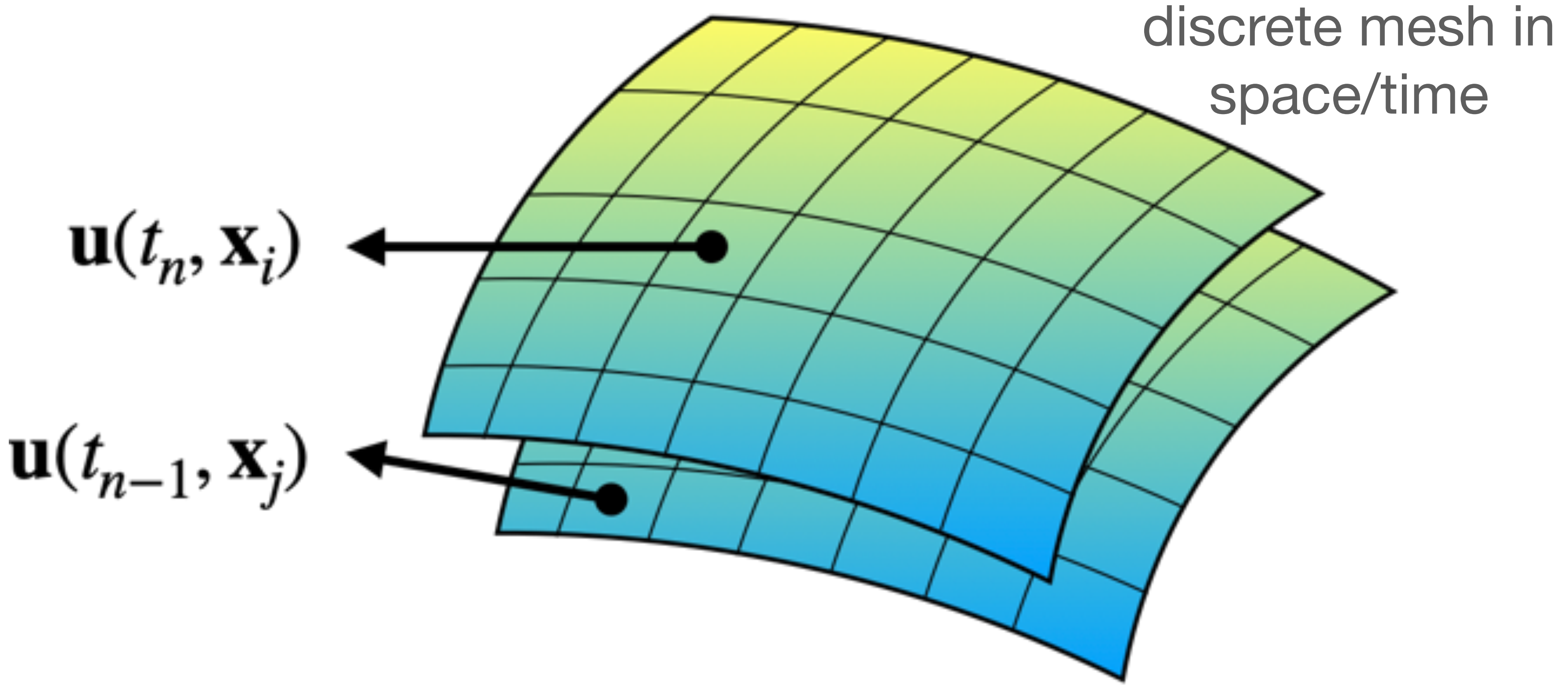


- Space discretization leads to large system of ODEs
- Solution defined and dependent on mesh discretization
- Solution satisfies system of PDEs in a weak sense
- Rigorous theory for convergence and stability

Physics Informed Neural Networks

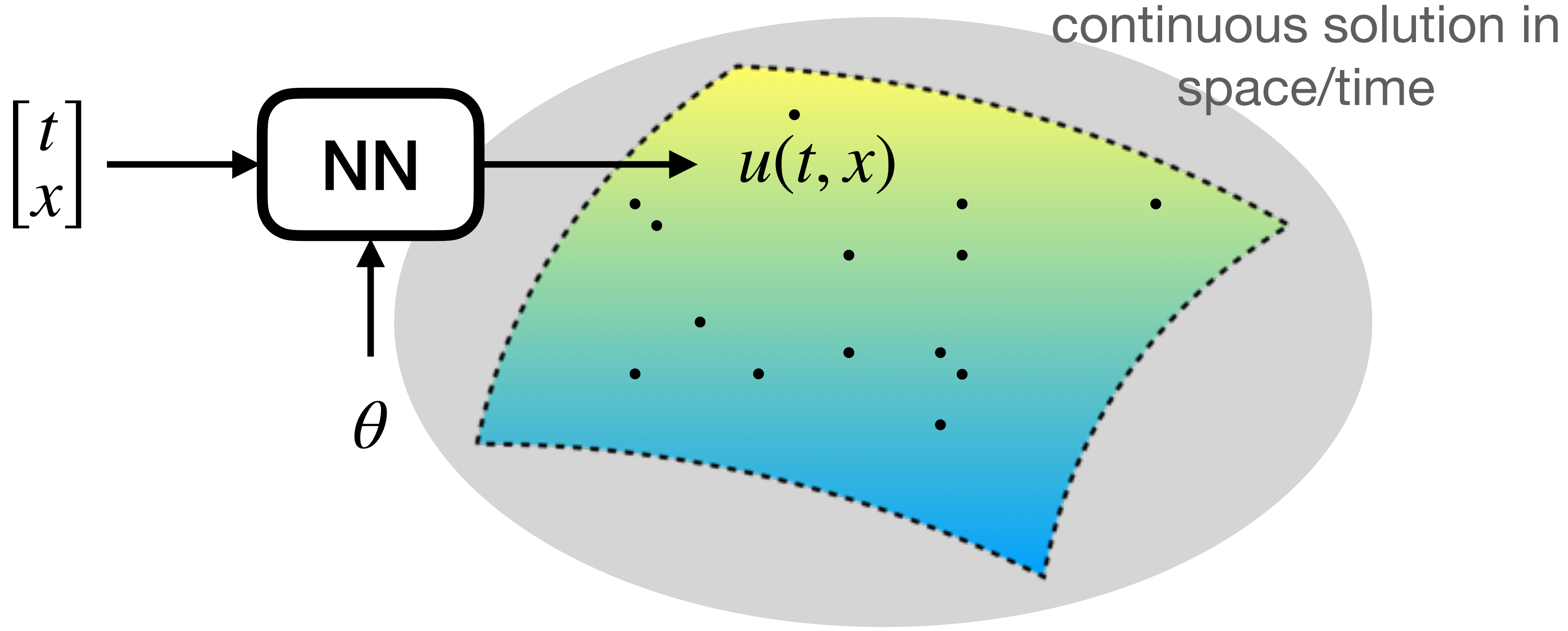
- Networks are trained with/without data, but regularized using physical laws
- Loss function constructed from data term and residuals of governing equations
- Boundary conditions treated like data (constrained) or enforced by construction (unconstrained) of the neural network

Conventional Discretization Approaches (CFD)



- Space discretization leads to large system of ODEs
- Solution defined and dependent on mesh discretization
- Solution satisfies system of PDEs in a weak sense
- Rigorous theory for convergence and stability

Deep-Learning Approach

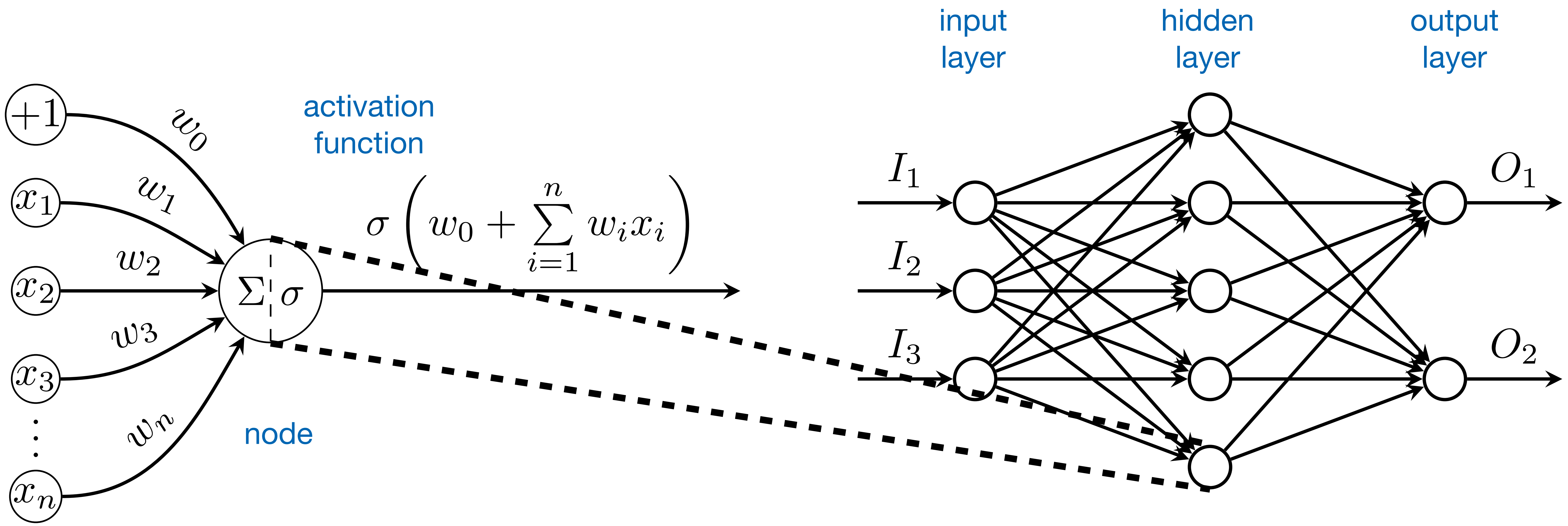


- PDEs converted into large optimization problem on params.
- Solution dependent on training points, defined everywhere
- Solution satisfies system of PDEs in a continuous sense
- Convergence and stability are active fields of research

What's a Neural Network?

- Nothing more than a function mapping an input space to an output space via a series of linear/nonlinear transformations

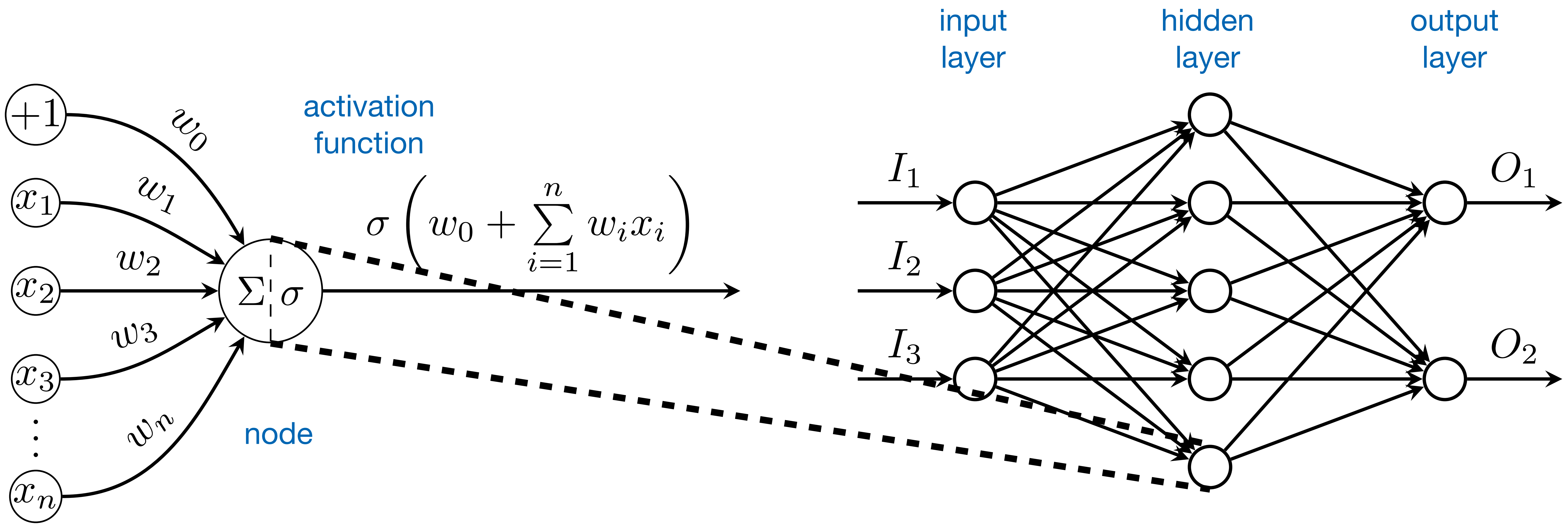
Conceptional View



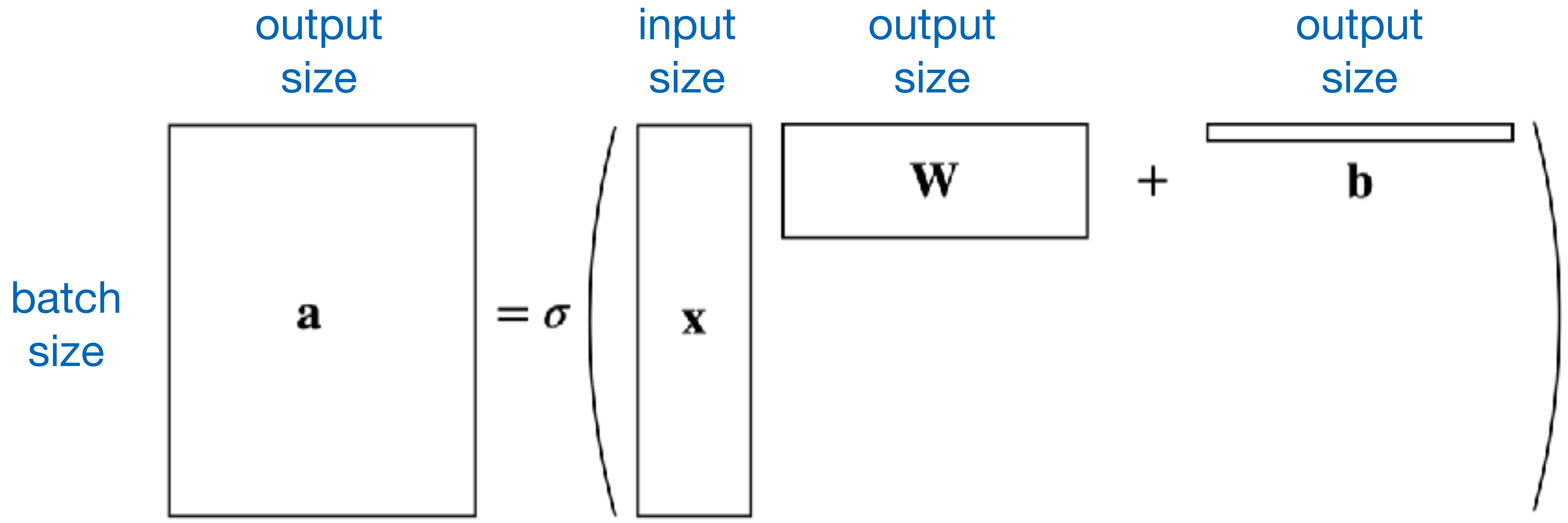
What's a Neural Network?

- Nothing more than a function mapping an input space to an output space via a series of linear/nonlinear transformations

Conceptional View



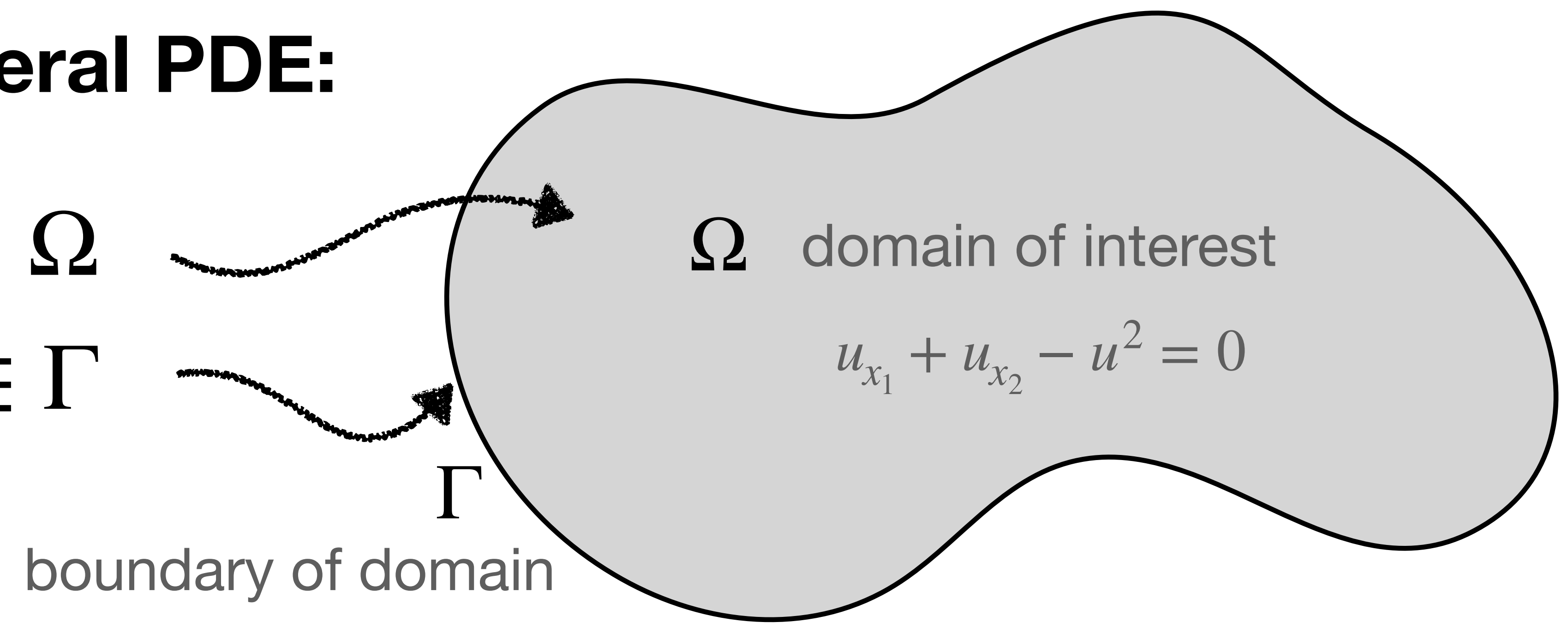
Practical Layerwise Implementation



But seriously, how do we learn equations?

1. **Consider the general PDE:**

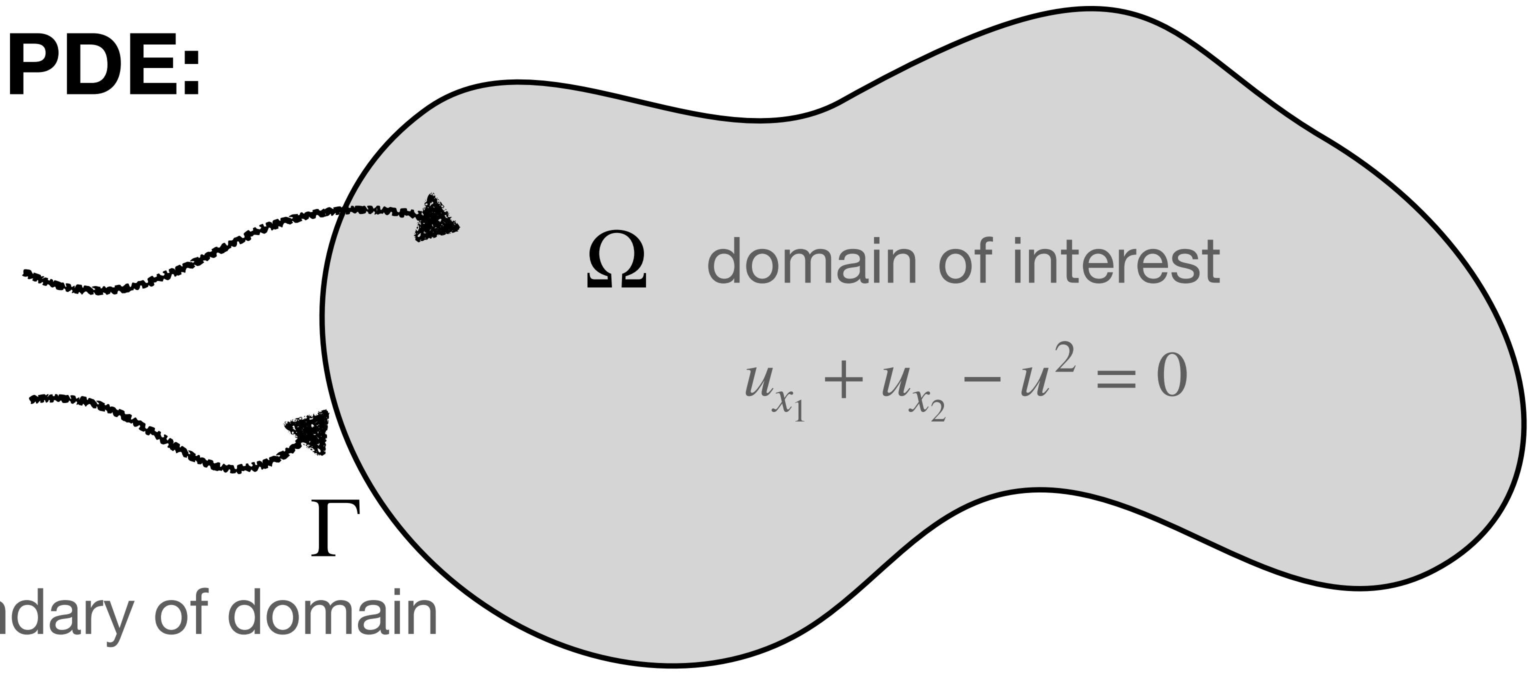
$$\mathfrak{F}[u](x) = 0, \quad x \in \Omega$$
$$\mathfrak{B}[u](x) = 0, \quad x \in \Gamma$$



But seriously, how do we learn equations?

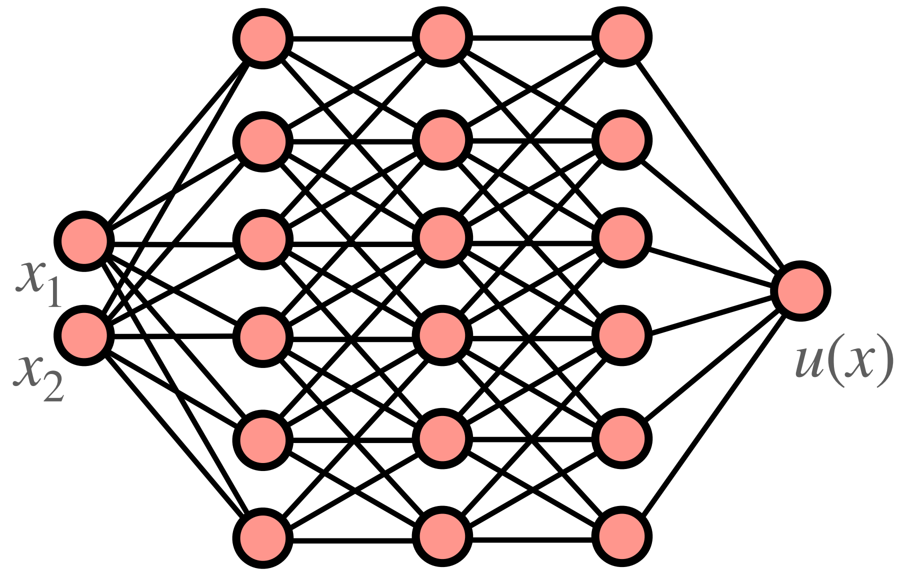
1. Consider the general PDE:

$$\mathfrak{F}[u](x) = 0, \quad x \in \Omega$$
$$\mathfrak{B}[u](x) = 0, \quad x \in \Gamma$$



2. Build a NN to approximate $u(x)$

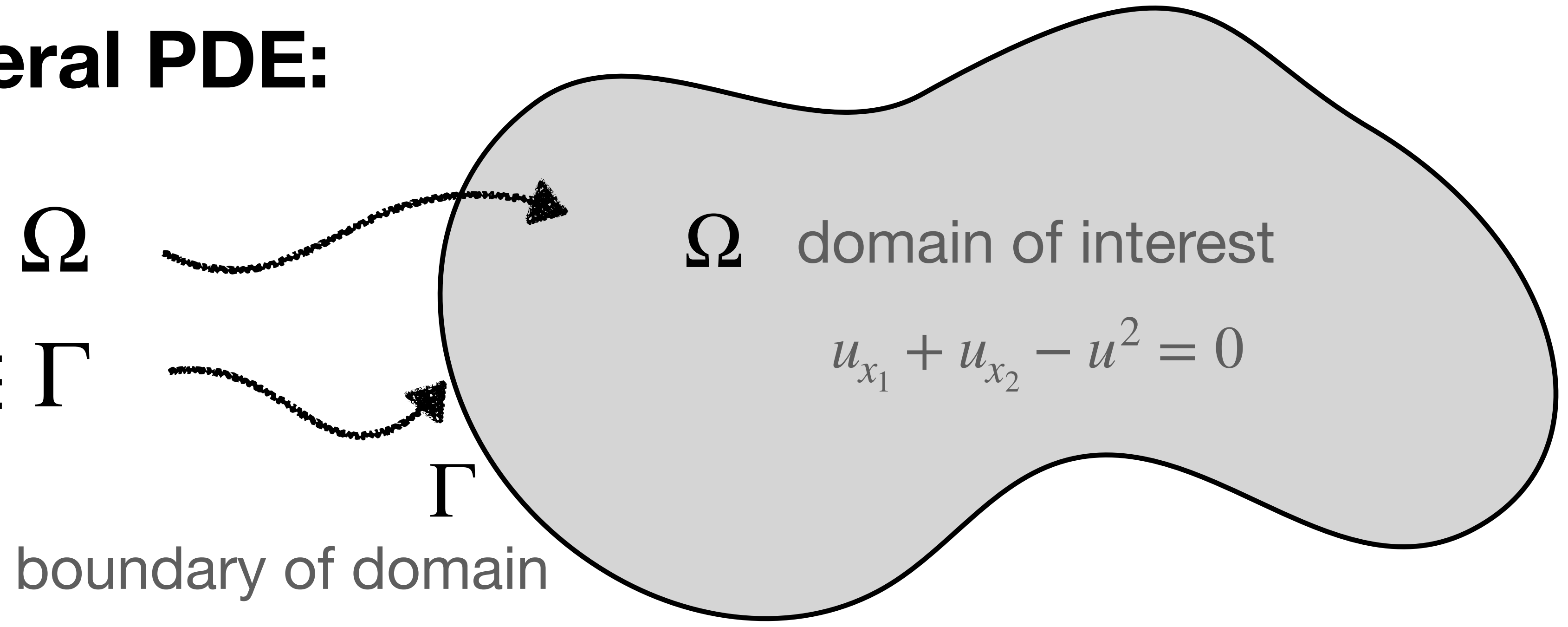
$$\hat{u}(x;\theta) \approx u(x)$$



But seriously, how do we learn equations?

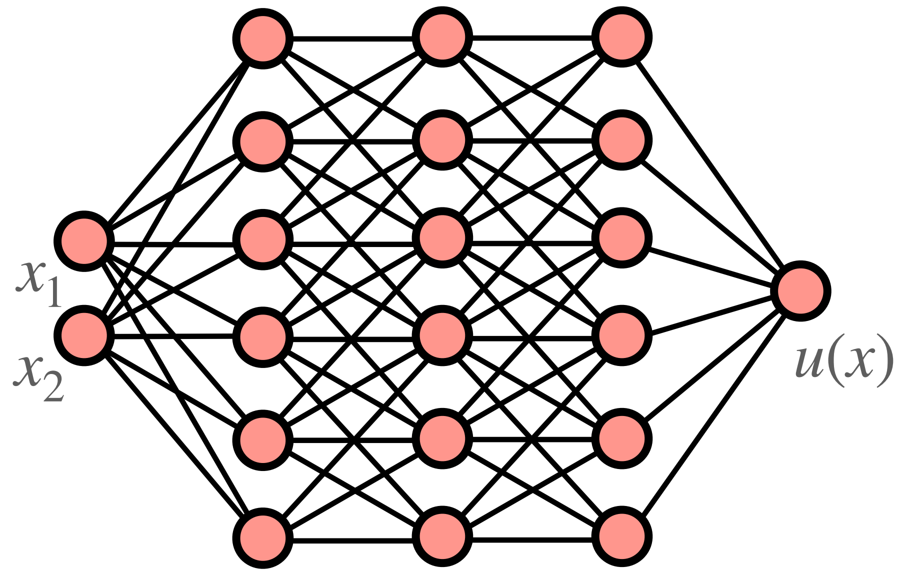
1. **Consider the general PDE:**

$$\mathfrak{F}[u](x) = 0, \quad x \in \Omega$$
$$\mathfrak{B}[u](x) = 0, \quad x \in \Gamma$$

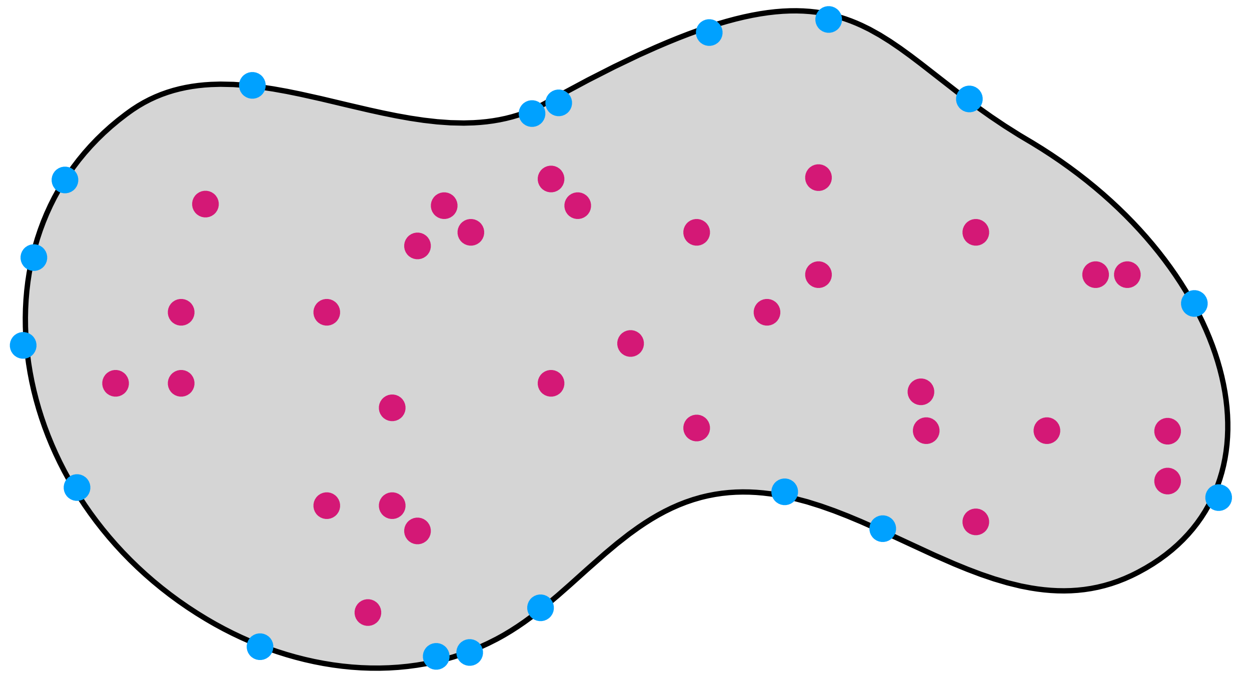


2. **Build a NN to approximate $u(x)$**

$$\hat{u}(x;\theta) \approx u(x)$$



3. **Distribute colocation points in the domain and boundary**



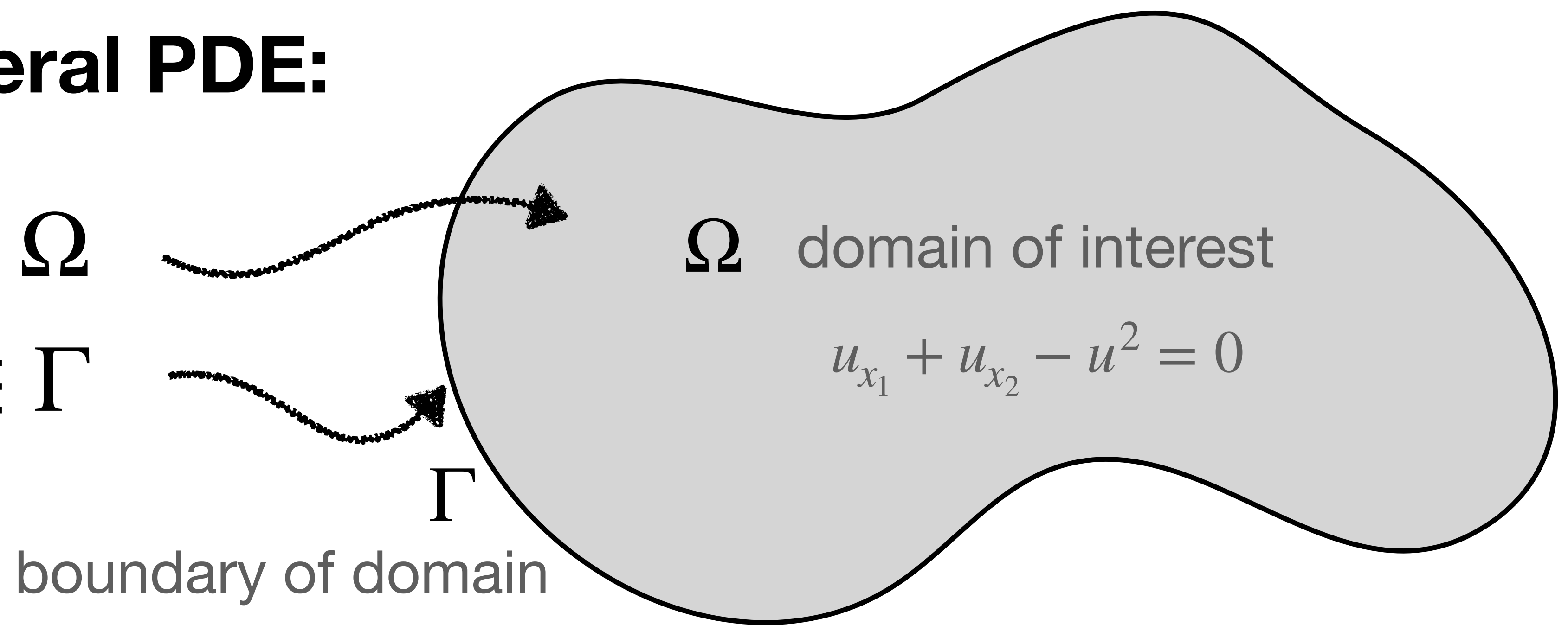
$$\hat{\Omega} = \{x_i : x_i \in \Omega\}$$

$$\hat{\Gamma} = \{x_i : x_i \in \Gamma\}$$

But seriously, how do we learn equations?

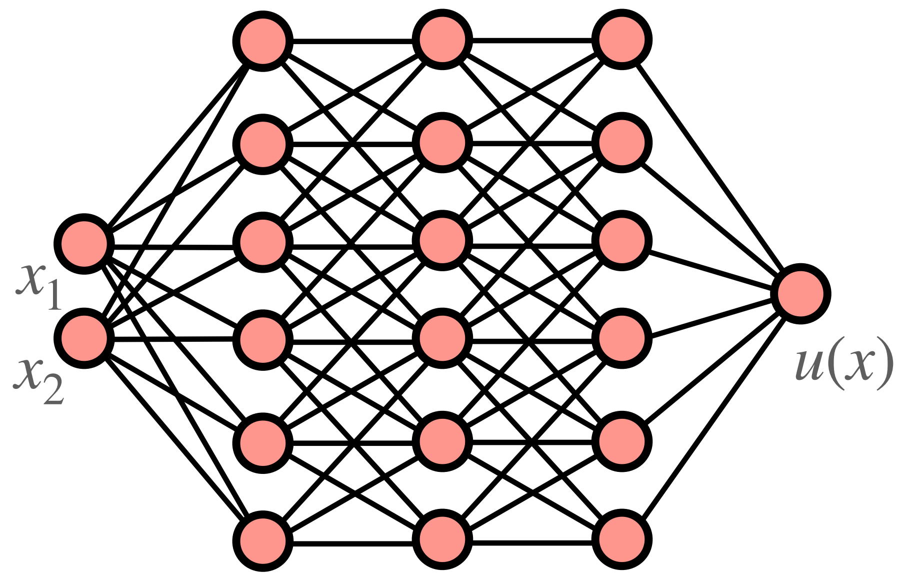
1. Consider the general PDE:

$$\mathfrak{F}[u](x) = 0, \quad x \in \Omega$$
$$\mathfrak{B}[u](x) = 0, \quad x \in \Gamma$$

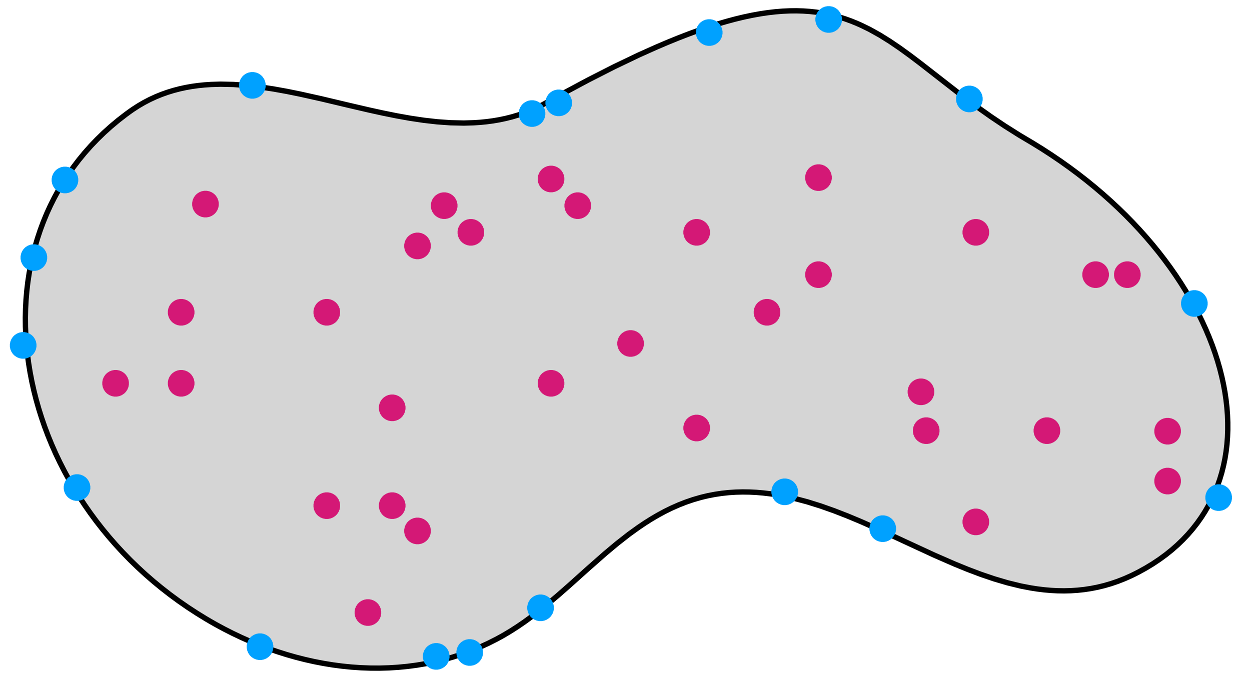


2. Build a NN to approximate $u(x)$

$$\hat{u}(x;\theta) \approx u(x)$$



3. Distribute colocation points in the domain and boundary

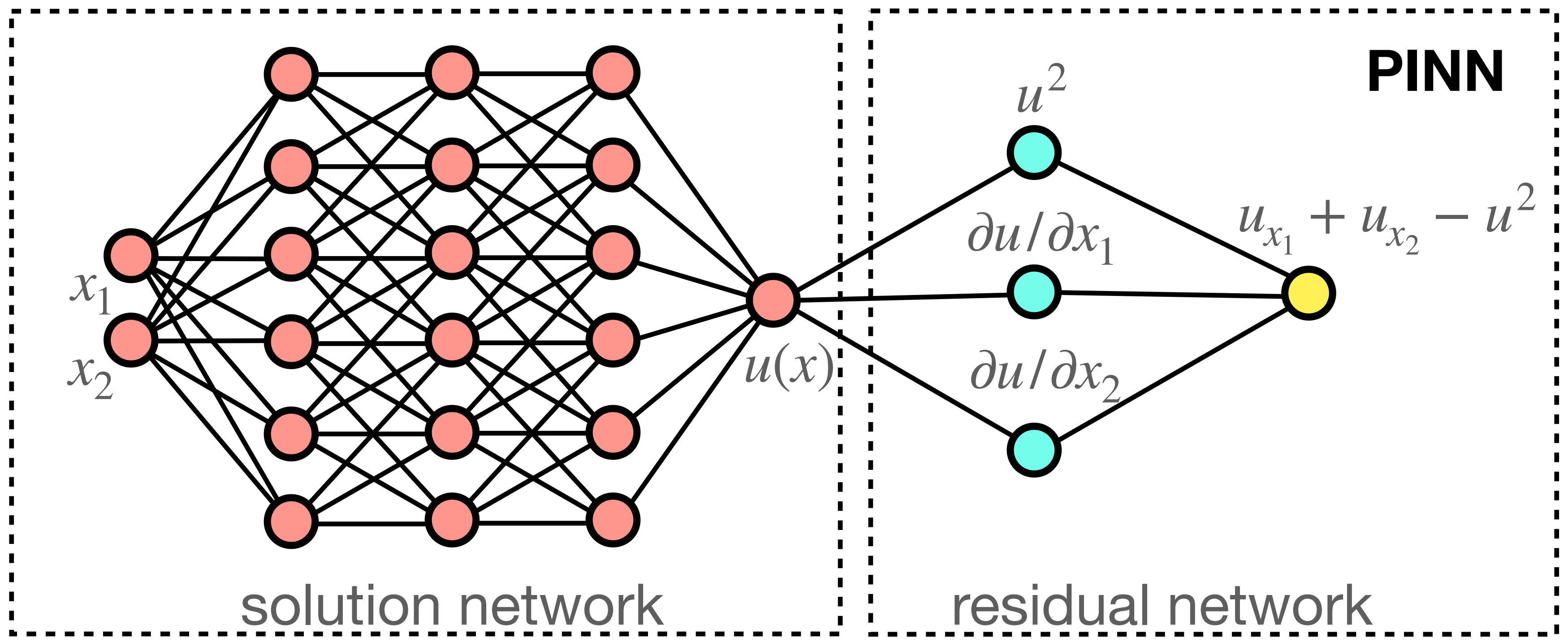


$$\hat{\Omega} = \{x_i : x_i \in \Omega\}$$

$$\hat{\Gamma} = \{x_i : x_i \in \Gamma\}$$

4. Construct loss function from residual operators

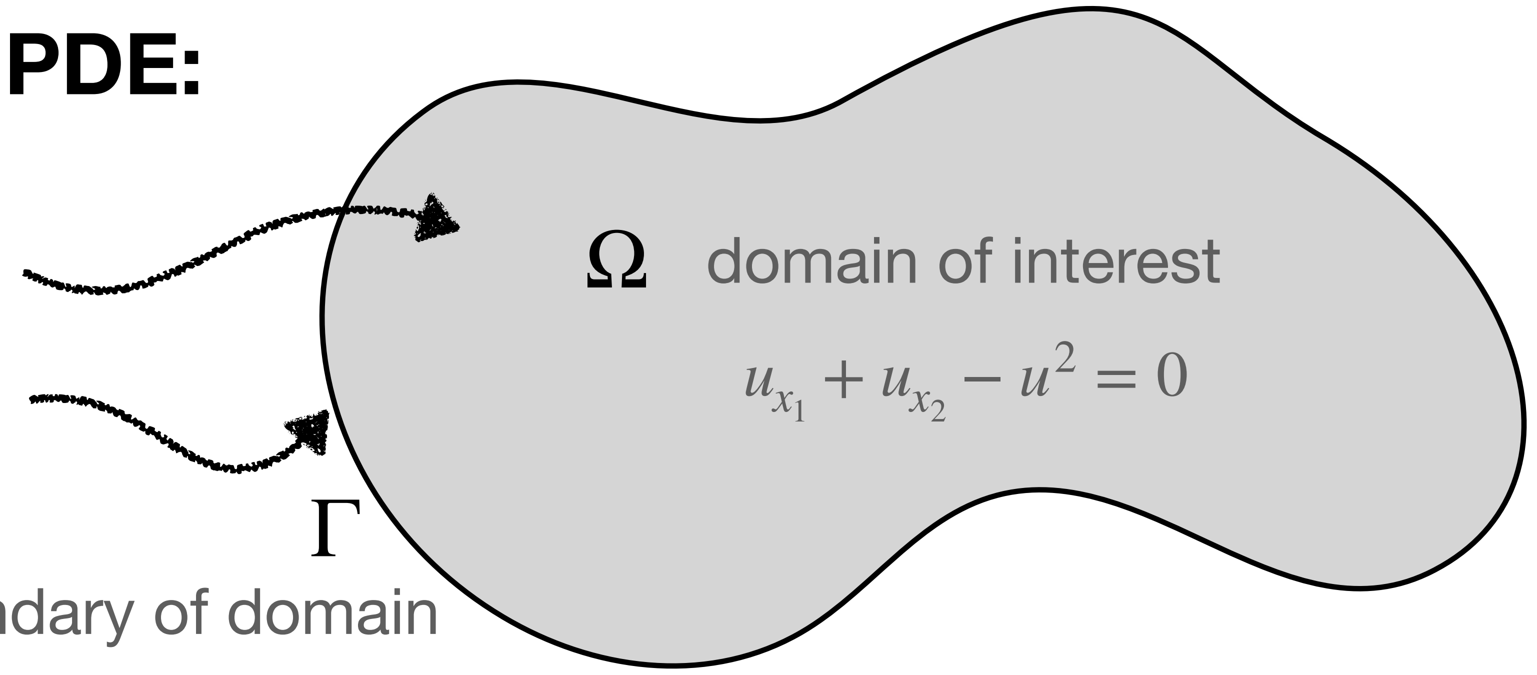
$$\mathscr{L}(\theta) = \frac{1}{|\hat{\Omega}|}\sum_{x_i \in \hat{\Omega}} |\mathfrak{F}[\hat{u}](x_i;\theta)|^2 + \frac{\eta}{|\hat{\Gamma}|}\sum_{x_i \in \hat{\Gamma}} |\mathfrak{B}[\hat{u}](x_i;\theta)|^2$$



But seriously, how do we learn equations?

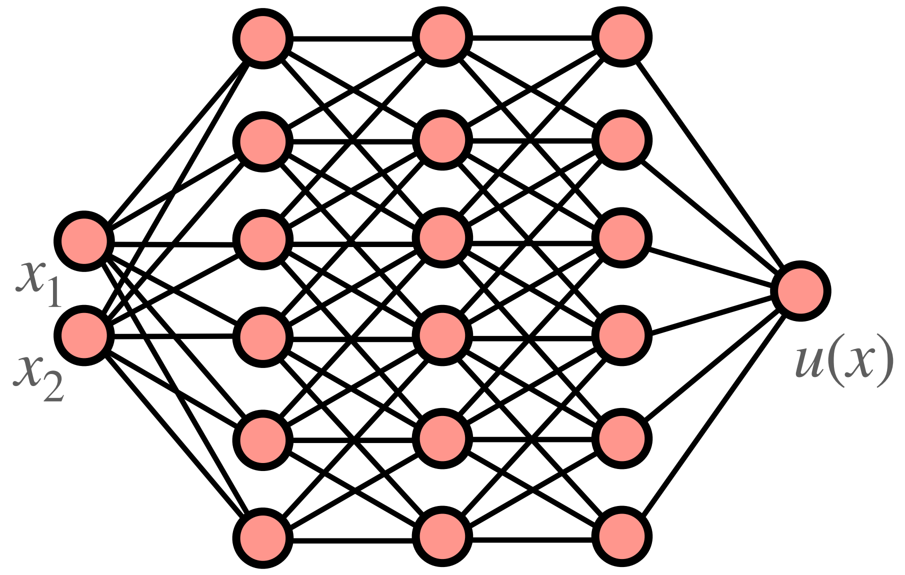
1. **Consider the general PDE:**

$$\mathfrak{F}[u](x) = 0, \quad x \in \Omega$$
$$\mathfrak{B}[u](x) = 0, \quad x \in \Gamma$$

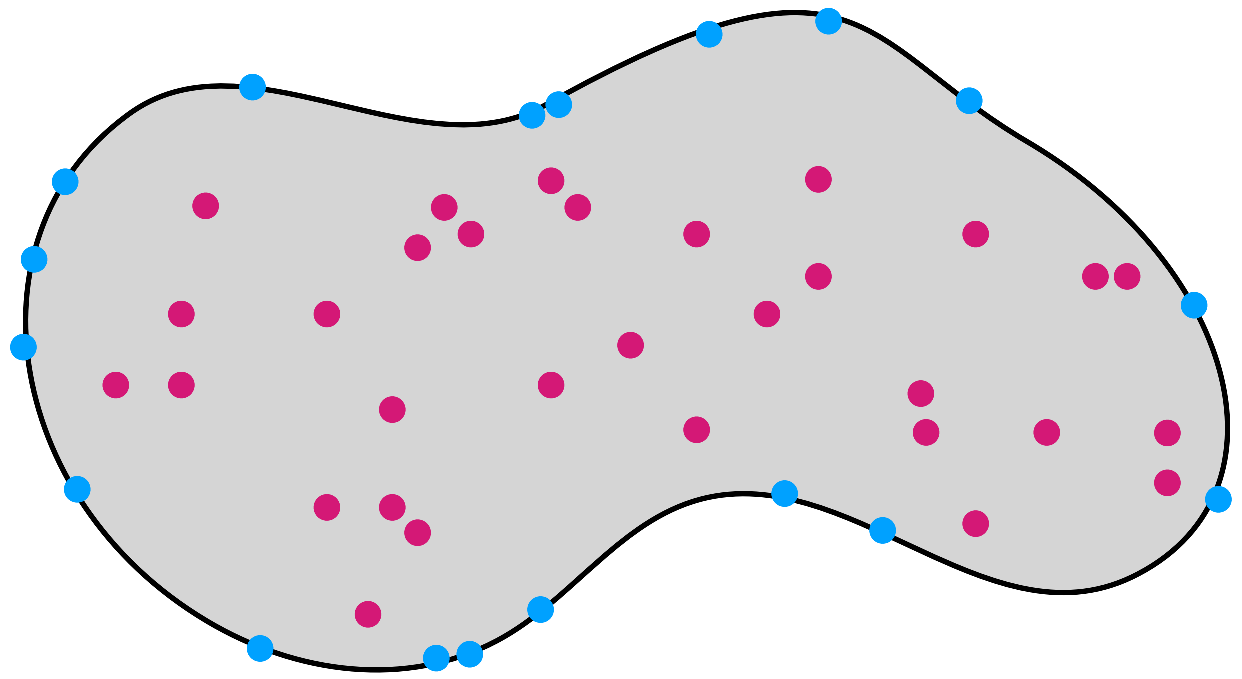


2. **Build a NN to approximate $u(x)$**

$$\hat{u}(x;\theta) \approx u(x)$$



3. **Distribute colocation points in the domain and boundary**

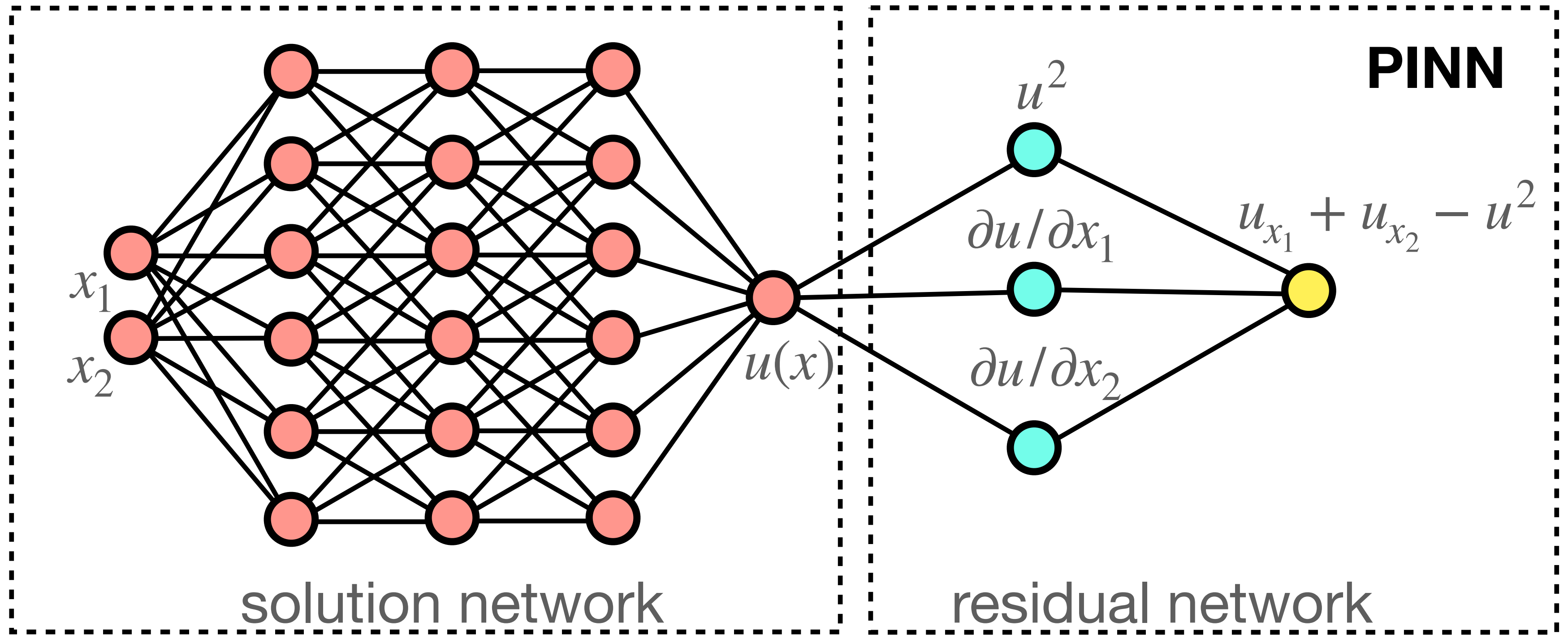


$$\hat{\Omega} = \{x_i : x_i \in \Omega\}$$

$$\hat{\Gamma} = \{x_i : x_i \in \Gamma\}$$

4. **Construct loss function from residual operators**

$$\mathscr{L}(\theta) = \frac{1}{|\hat{\Omega}|} \sum_{x_i \in \hat{\Omega}} |\mathfrak{F}[\hat{u}](x_i;\theta)|^2 + \frac{\eta}{|\hat{\Gamma}|} \sum_{x_i \in \hat{\Gamma}} |\mathfrak{B}[\hat{u}](x_i;\theta)|^2$$



5. **Minimize the loss function with respect to network parameters**

$$\hat{u} = \hat{u}(x;\theta^*), \quad \underset{\theta}{\operatorname{argmin}}\, \mathscr{L}(\theta)$$

Navier-Stokes equations for perfect gas

- Interested in assessing the heating predictions obtained with neural networks in “simple” configurations at high speed
- Previous literature is not concerned with heating

Governing equations.

$$\frac{\partial \boldsymbol{U}}{\partial t} + \frac{\partial \boldsymbol{F}}{\partial x} + \frac{\partial \boldsymbol{G}}{\partial y} = \frac{\partial \boldsymbol{F}^v}{\partial x} + \frac{\partial \boldsymbol{G}^v}{\partial y}, \quad \forall (x, y) \in \boldsymbol{\Omega}$$

$$\boldsymbol{U} = \begin{bmatrix} \rho \\ \rho u \\ \rho v \\ \rho E \end{bmatrix}, \quad \boldsymbol{F} = \begin{bmatrix} \rho u \\ \rho u^2 + p \\ \rho u v \\ \rho u H \end{bmatrix}, \quad \boldsymbol{G} = \begin{bmatrix} \rho v \\ \rho u v \\ \rho v^2 + p \\ \rho v H \end{bmatrix}, \quad \boldsymbol{F}^v = \begin{bmatrix} 0 \\ \tau_{xx} \\ \tau_{xy} \\ \tau_{xx} u + \tau_{xy} v - q_x \end{bmatrix}, \quad \boldsymbol{G}^v = \begin{bmatrix} 0 \\ \tau_{yx} \\ \tau_{yy} \\ \tau_{yx} u + \tau_{yy} v - q_y \end{bmatrix}$$

$$p = \frac{\rho T}{\gamma M_\infty^2}, \quad E = \frac{1}{\gamma - 1}\frac{p}{\rho} + \frac{u^2 + v^2}{2}$$

$$\tau_{xx} = \hat{\mu}\left(\frac{4}{3}\frac{\partial u}{\partial x} - \frac{2}{3}\frac{\partial v}{\partial y}\right), \quad \tau_{yy} = \hat{\mu}\left(\frac{4}{3}\frac{\partial v}{\partial y} - \frac{2}{3}\frac{\partial u}{\partial x}\right), \quad \tau_{xy} = \tau_{yx} = \hat{\mu}\left(\frac{\partial v}{\partial x} + \frac{\partial u}{\partial y}\right), \quad q_x = -k\frac{\partial T}{\partial x}, \quad q_y = -k\frac{\partial T}{\partial y}$$

$$\hat{\mu} = \frac{1}{\mathrm{Re}_\infty}\frac{C + T_\infty}{C + T_\infty T} T^{3/2}, \quad k = \frac{\hat{\mu}}{(\gamma - 1) M_\infty^2 \mathrm{Pr}}$$

Loss function in Python code.

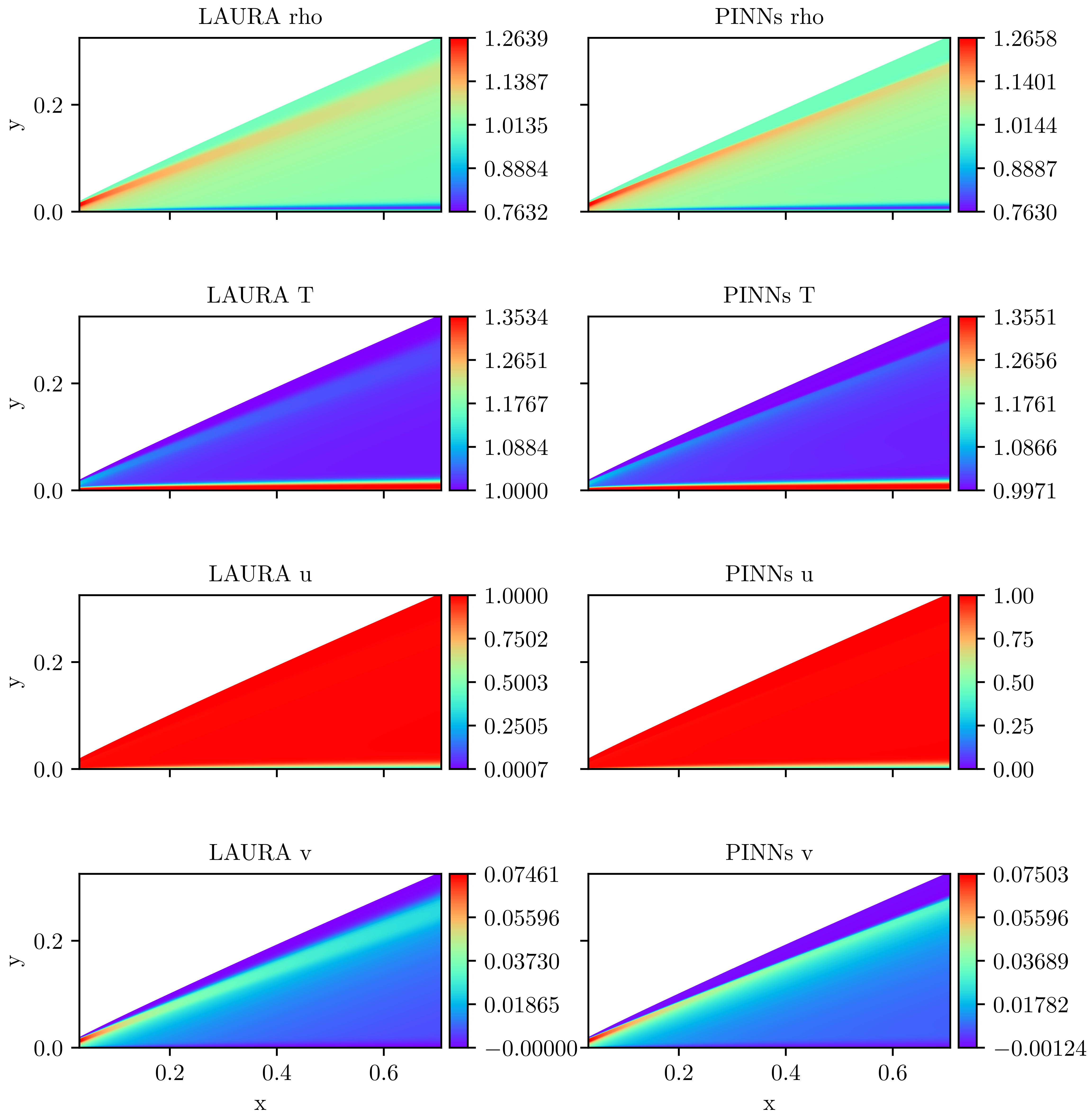
```python
def steady_navier_stokes_2d(coords, prim_vars):
    rho = prim_vars[:,0:1]
    T = prim_vars[:,1:2]
    u = prim_vars[:,2:3]
    v = prim_vars[:,3:]
    p = rho*T/(gamma*M_inf**2)

    mu = (s2 + T_inf) * tf.maximum(T,1.0)**1.5 / (s2 + T_inf*tf.maximum(T,1.0))
    k = mu / ((gamma-1) * M_inf**2 * Pr)

    rho_x, rho_y, T_x, T_y, u_x, u_y, v_x, v_y = gradients(prim_vars, coords)
    p_x, p_y = [dde.grad.jacobian(p, coords, j=j) for j in range(2)]

    tauxx = mu * ((4.0/3.0)*u_x - (2.0/3.0)*v_y)
    tauyy = mu * ((4.0/3.0)*v_y - (2.0/3.0)*u_x)
    tauxy = mu * (u_y + v_x)

    qx = -k * T_x
    qy = -k * T_y

    tauxx_x = dde.grad.jacobian(tauxx, coords, j=0)
    tauxy_x, tauxy_y = [dde.grad.jacobian(tauxy, coords, j=j) for j in range(2)]
    tauyy_y = dde.grad.jacobian(tauyy, coords, j=1)

    qx_x = dde.grad.jacobian(qx, coords, j=0)
    qy_y = dde.grad.jacobian(qy, coords, j=1)

    mass   = rho*(u_x + v_y) + u*rho_x + v*rho_y
    x_mtm  = rho*(u*u_x + v*u_y) + p_x - (tauxx_x + tauxy_y)/Re_inf
    y_mtm  = rho*(u*v_x + v*v_y) + p_y - (tauxy_x + tauyy_y)/Re_inf
    energy = (
        rho*(u*u*u_x + u*v*(v_x+u_y) + v*v*v_y) + gamma/(gamma-1.0)*(
            u*p_x + v*p_y - T*(u*rho_x + v*rho_y)/(gamma*M_inf**2)
        ) - (
            u*tauxx_x + tauxx*u_x + v*tauxy_x + tauxy*v_x +
            u*tauxy_y + tauxy*u_y + v*tauyy_y + tauyy*v_y -
            qx_x - qy_y
        ) / Re_inf
    )

    return [mass, x_mtm, y_mtm, energy]
```

Comparison with LAURA for flat plate

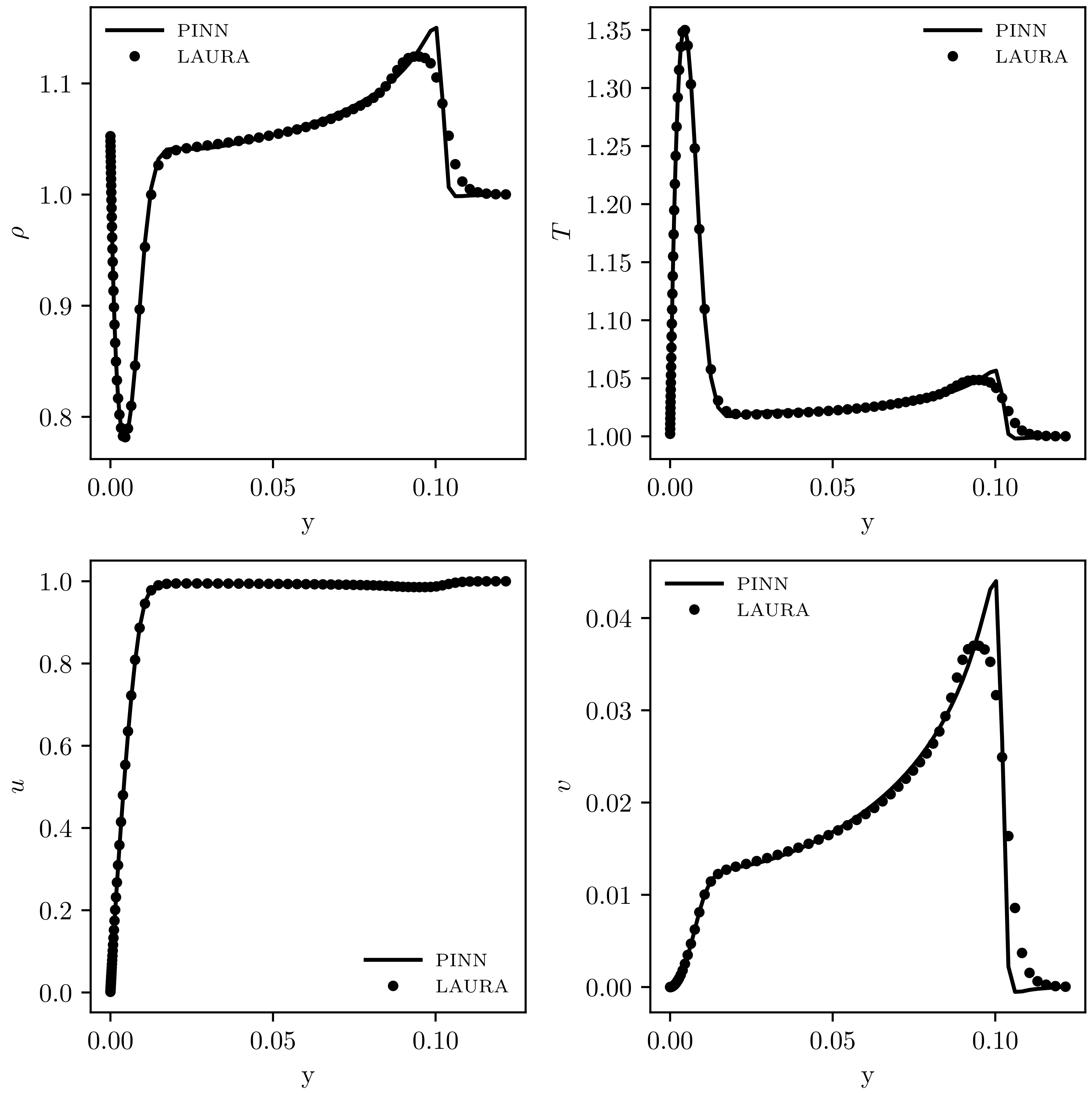
- Freestream conditions

M_∞	Re_∞	T_∞ [K]	T_{wall} [K]	γ	C [K]	Pr
3.0	5.0×10^4	300.0	300.0	1.4	110.33	0.72

- Network architecture and training
  - Dense feed-forward network, 6 hidden layers with 32 nodes
  - Layer-wise adaptive activation function
  - 50,000 Adam iterations with learning rate of 0.001
  - Further converged with L-BFGS algorithm
- LAURA results
  - 81x227 node grid
  - Mesh adaptation to resolve shock



Comparison with LAURA for flat plate



Wall-normal slice at $x \approx 0.25$

- Boundary and shock layers well resolved with PINN
- Heat flux computed along the entire wall (continuous function) by taking gradient of temperature solution network
- Does not require gradient approximation/interpolation as with CFD solution