Machine Learning in Structural Mechanics?

Prof. J. Morlier

SA-I

Sustainable Aerostructures & Interactions

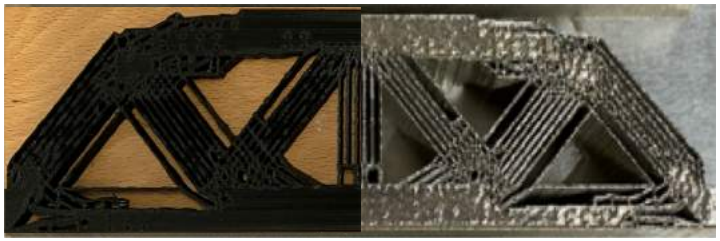
Mainly extracted from SUPAERO's MDO course

https://github.com/jomorlier/mdocourse

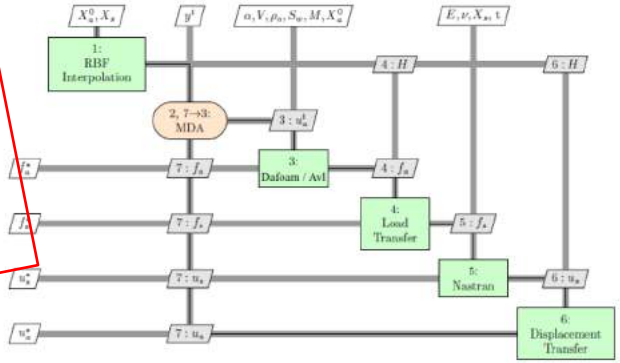
My Research

- 6 PhDs, 3 MsCs

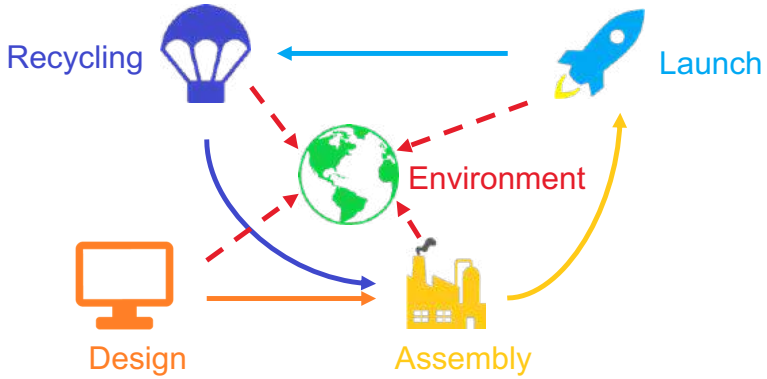
Digital fabrication



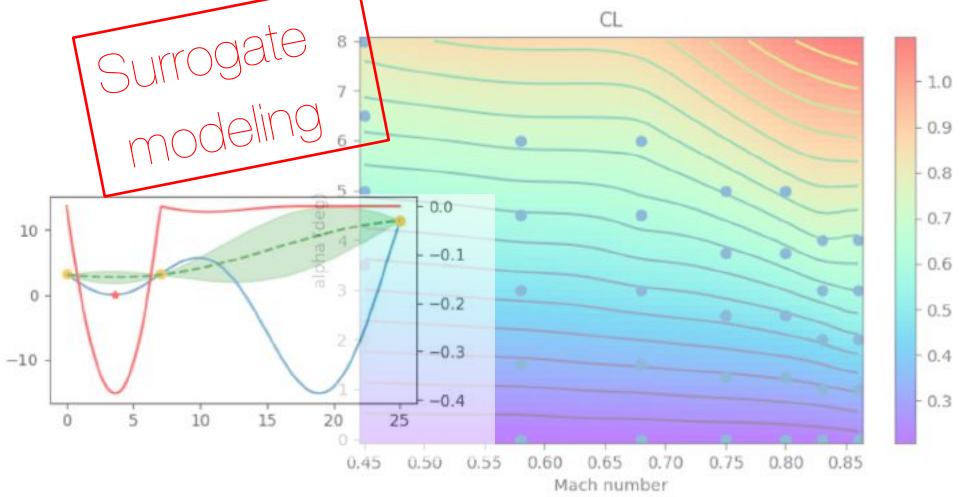
MDO for aerostructures design



Ecodesign



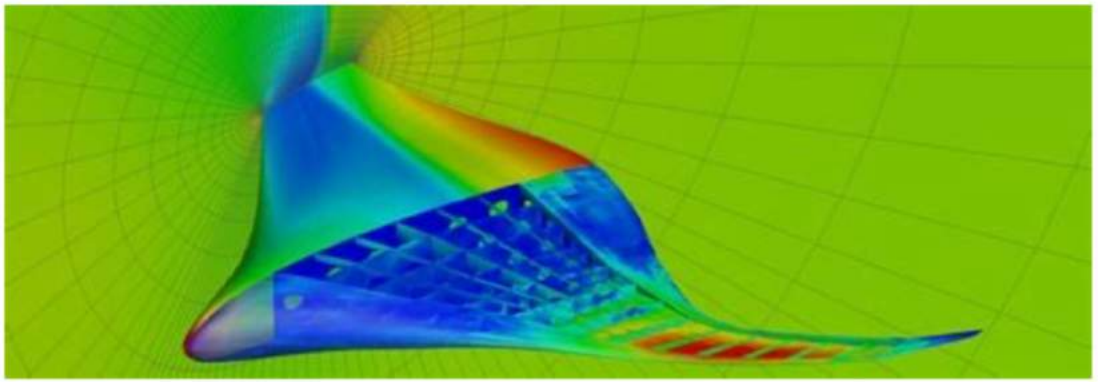
Surrogate modeling



AIRBUS ONERA THE FRENCH AEROSPACE LAB

RMIT UNIVERSITY

Popularization



https://www.linkedin.com/pulse/optimization-mdo-connecting-people-joseph-morlier/

http://mdolab.engin.umich.edu

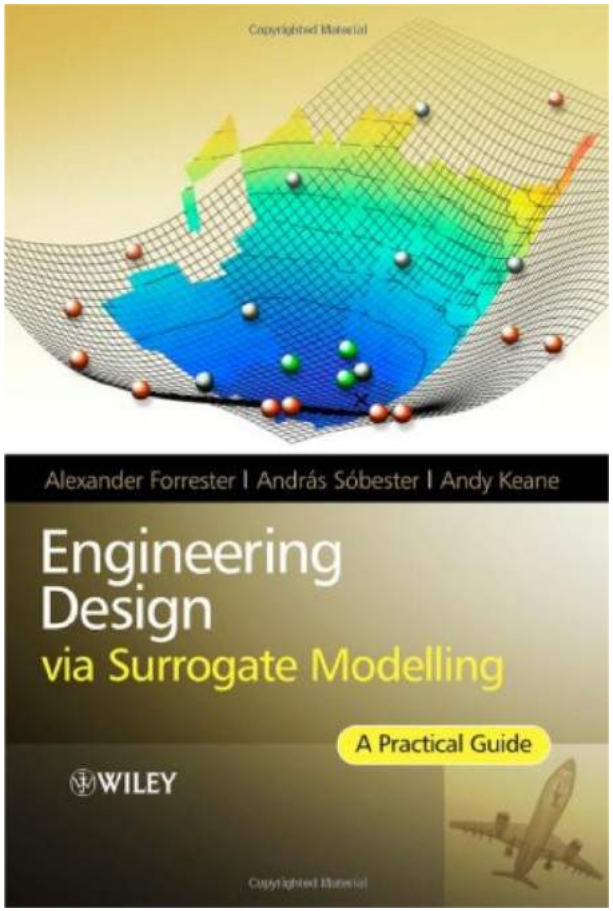
Optimization [MDO] for connecting people?

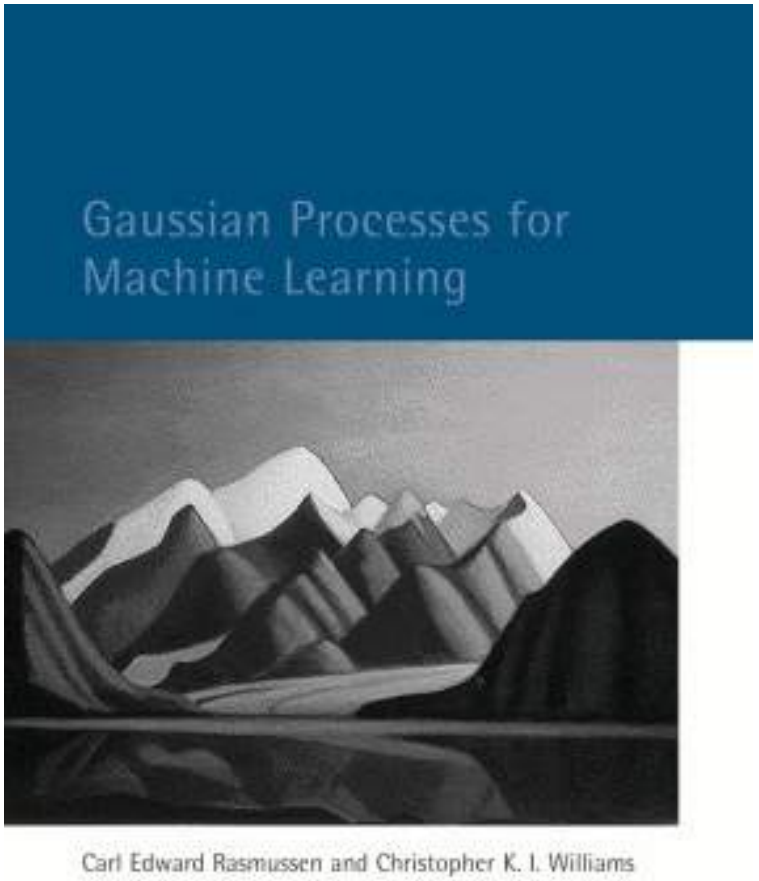
Publié le 14 février 2019 Modifier l'article | Voir les stats

joseph morlier
Professor in Structural and Multidisciplinary Design Optimization, ... any idea?
2 articles

74 31 3 0

XO


Alexander Forrester | András Sóbester | Andy Keane
Engineering Design
via Surrogate Modelling
A Practical Guide
WILEY

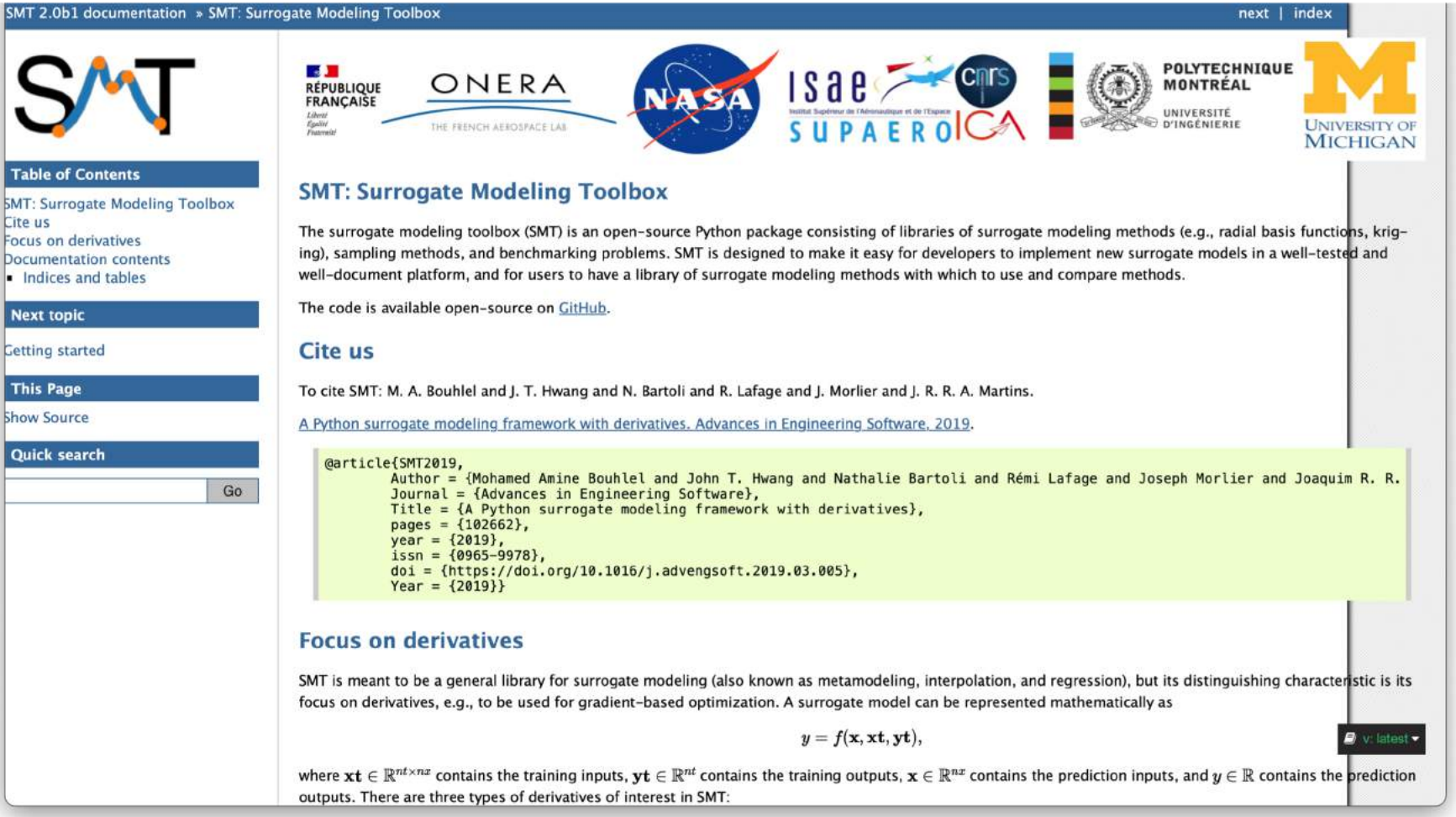

Gaussian Processes for Machine Learning
Carl Edward Rasmussen and Christopher K. I. Williams

X_0 in practice

Since 2017

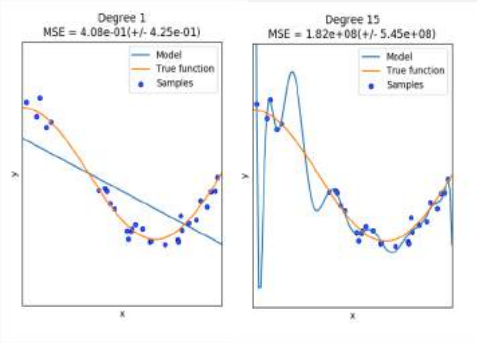
https://smt.readthedocs.io/en/latest

https://github.com/SMTorg/smt



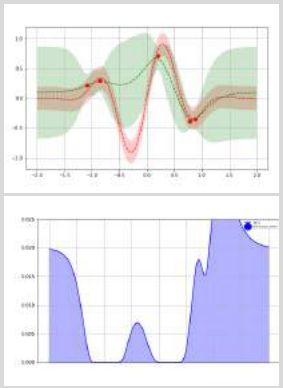
PROs and CONs (Thanks MonolithAI)

Linear and polynomial Regressions



- Some of the most basic models: fitting a polynomial to the data
- They might not capture all the complexity that is in the data
- They are explainable (e.g. via equations), which make it really attractive to some industries
- Main parameters: degree of polynomial (finding the "right compromise")

Gaussian Process Regressions



- More complex statistical process that fits a smooth function through the points, based on gaussian distributions
- Doesn't scale well with large data sets
- Work well with small data sets
  Simple to train, less prone to overfitting
  Always return uncertainty
- Main parameters: kernels (e.g. RBF, defines the smoothness of the model)

GAUSSIAN PROCESS (GP)

OPTIMIZATION AT FIXED BUDGET ++ (FOR EXPENSIVE CODE)

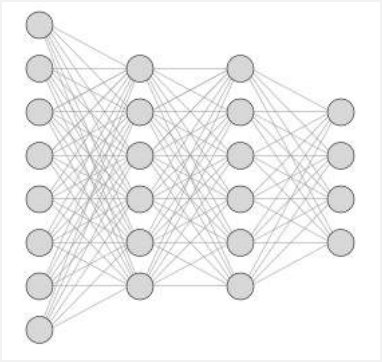
ENRICHMENT (INFILL) ++

MULTIFIDELITY (COKRIGING) ++

UNCERTAINTIES ESTIMATION ++ → LEAD TO BAYESIAN OPTIMIZATION & UQ

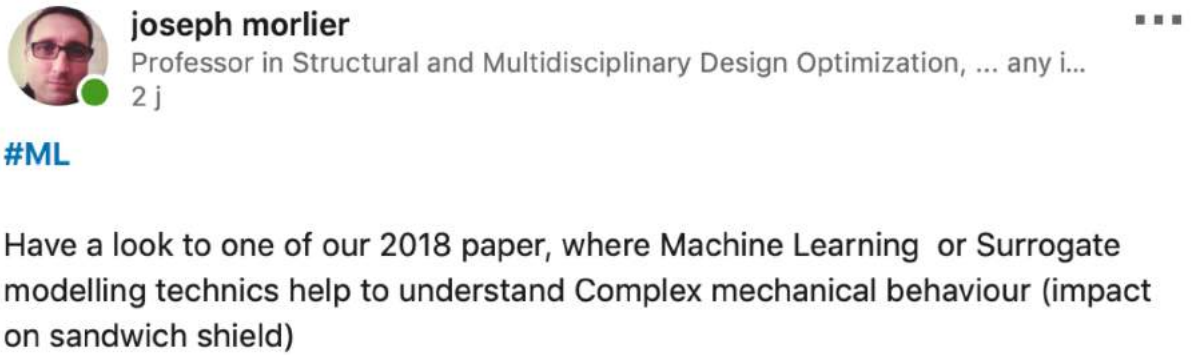
EXPLAINABILITY

Neural Networks



- Algorithms based on a system of neurons, based on biological neural network (brain)
- Could return uncertainty but requires work
  Very flexible: prone to overfit
  Not very good on small data sets
- Very good to tackle very large data sets
  Flexible: can fit most data set if big enough
  Handle non-linearity well
- Main parameters: architecture (number of layers, neurons), number of training steps

Arnaud Wilhelm' s PhD
Back in 2014-2017



joseph morlier
Professor in Structural and Multidisciplinary Design Optimization, ... any i...
2 j

#ML

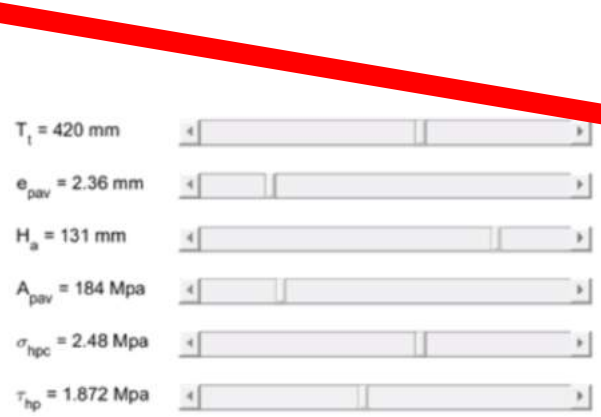
Have a look to one of our 2018 paper, where Machine Learning or Surrogate modelling technics help to understand Complex mechanical behaviour (impact on sandwich shield)

https://lnkd.in/dr_WSqA

RSM is Response Surface Modeling or Metamodeling or Surrogate modeling

y=f(x)

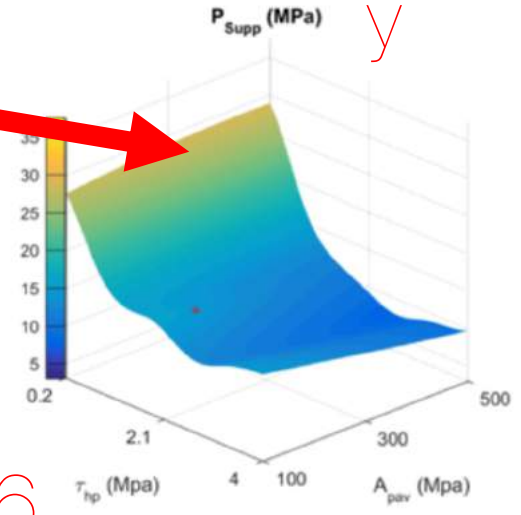
x



x_6

(Expensive) simulation

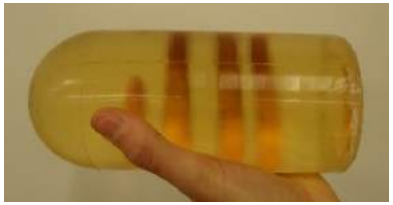
y



x_4

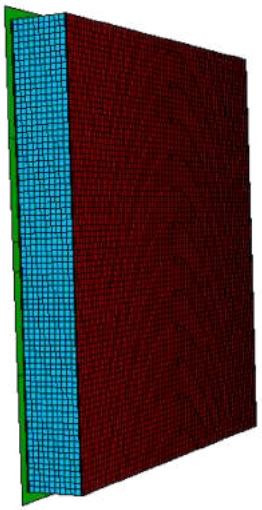
Complex model

- Test

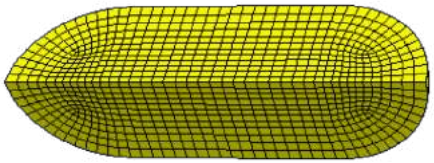


vs

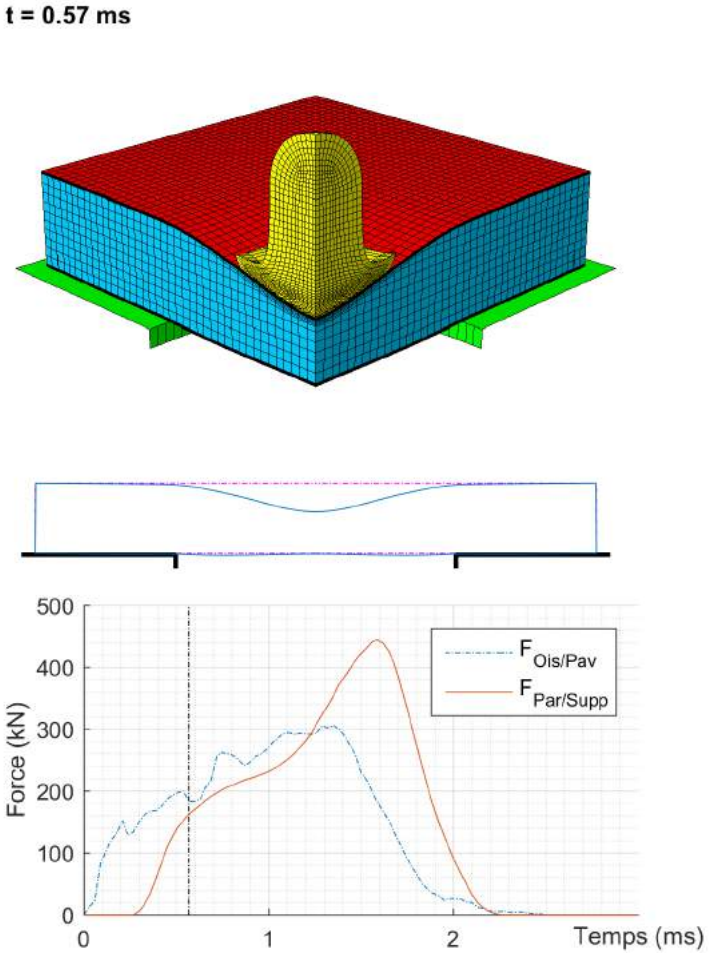
Model

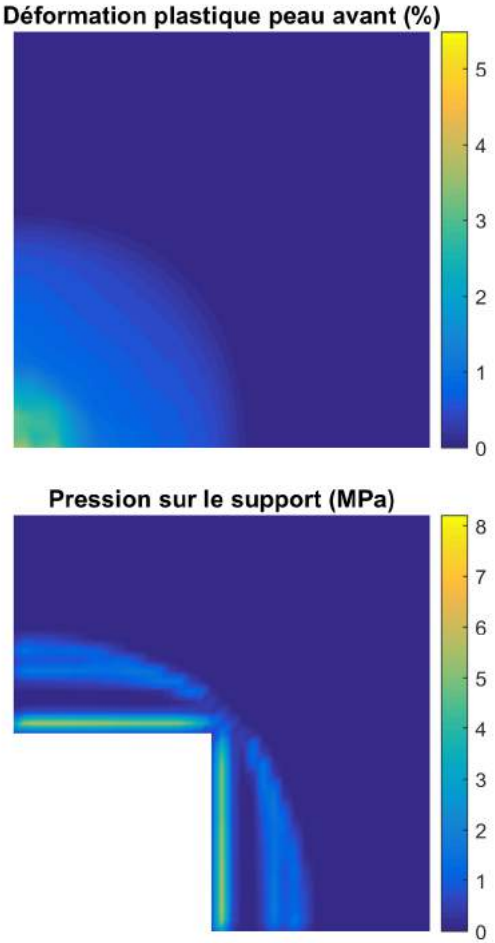


ABAQUS/Explicit



Simulation (really costly)





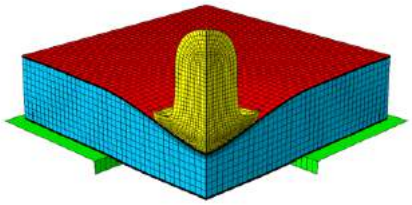
What is Surrogate Modeling ?

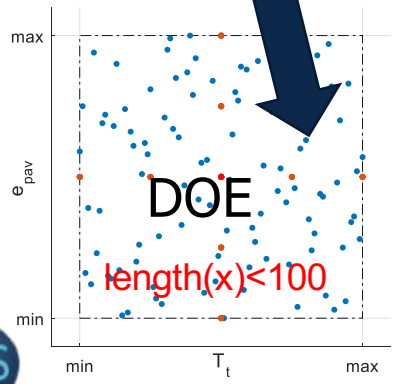
A. Forrester, A. Sobester, and A. Keane. "Engineering Design via Surrogate Modelling: A Practical Guide". Coll. John Wiley & Sons (2008).

A surrogate model of a function is an approximation of Expensive Computer simulation: It's a supervised learning process in AI.

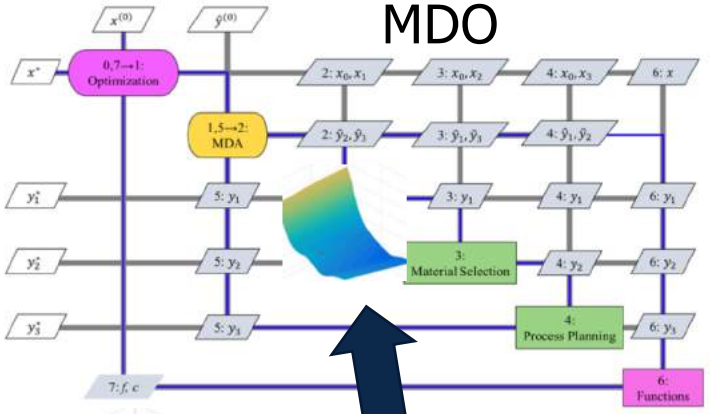
As the surrogate is less costly to evaluate it can be used as a "fast" code in a Multidisciplinary Design Optimization loop. [or do Uncertainty Quantification or do Bayesian Optimization etc…]

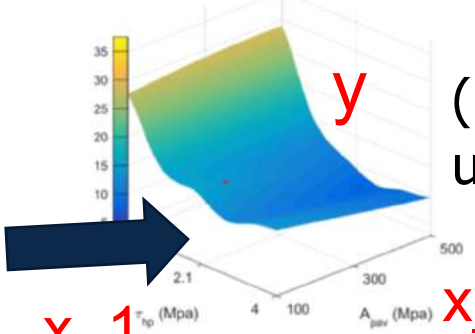
(Expensive) simulation





MDO





(Fast) Surrogate using SMT y=f(x)

Our approach: is Collaborative !

AEROSPACE ENGINEERING
UNIVERSITY of MICHIGAN
NASA Isae SUPAERO ICA
ONERA
THE FRENCH AEROSPACE LAB

OpenSource code: github.com/SMTorg/smt
Documentation: smt.readthedocs.io

Au programme des 3 heures

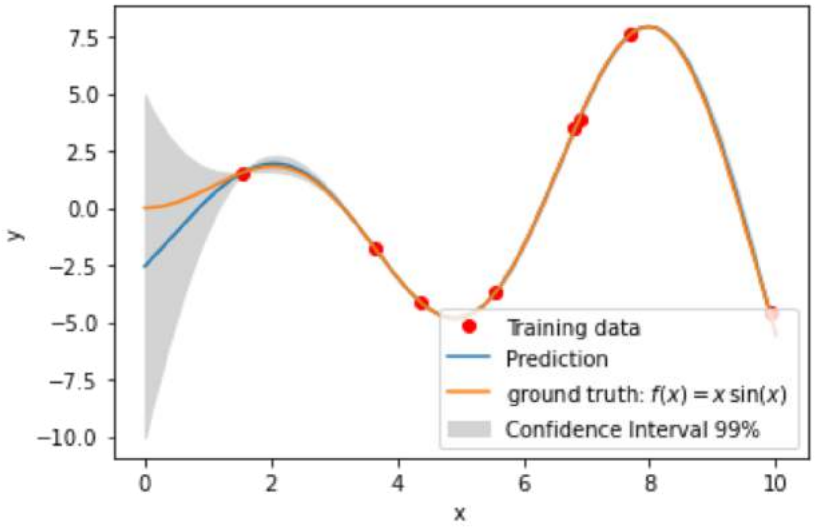
Part 1. Régression par processus gaussiens
Introduction aux processus gaussiens, fondements mathématiques en 30’ Exercice: Je programme mon GPR 30’
Part 2. processus gaussiens versus réseaux de neurones
Illustration en grande dimension, hyper parameter tuning, BO en 30’ Exercice:
J’utilise des toolboxes sur des exemples plus complexes 45’
Part 3. Physics Informed Neural Networks {PINN}
Qu’est ce qu’un PINN connaissant un NN? en 15’ Exercice:
Entrainer un PINN pour simuler le système (oscillateur harmonique amorti) en 30’
Bonus: Entrainer un PINN pour inverser les paramètres sous-jacents

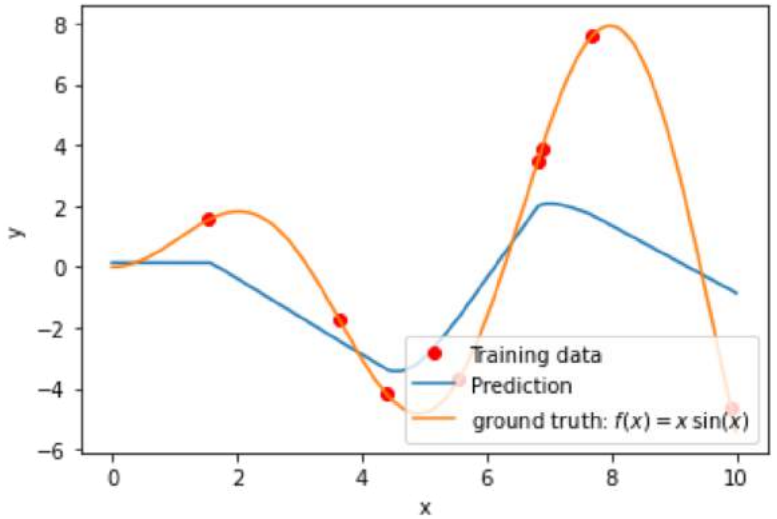
Travaux pratiques (Python)

sous colab Pour aller plus loin: Tutoriels SMT sur les processus gaussiens multi-fidélité, l'optimisation bayesienne
https://github.com/SMTorg/smt/tree/master/tutorial

1/Comparaison sur une simple fonction 1D : x*sin(x). Sait-on à priori si l'on dispose d'assez de points ? Quel est l'effet des hyperparamètres (noyaux, degrés des polynômes, …) ? Comparaison moindres carrés avec modèle linéaire/quadratique.
2/Entraîner un réseau de neurones (et/ou un processus gaussien) sur un dataset donné (sur une poutre encastrée).

GPR vs NN





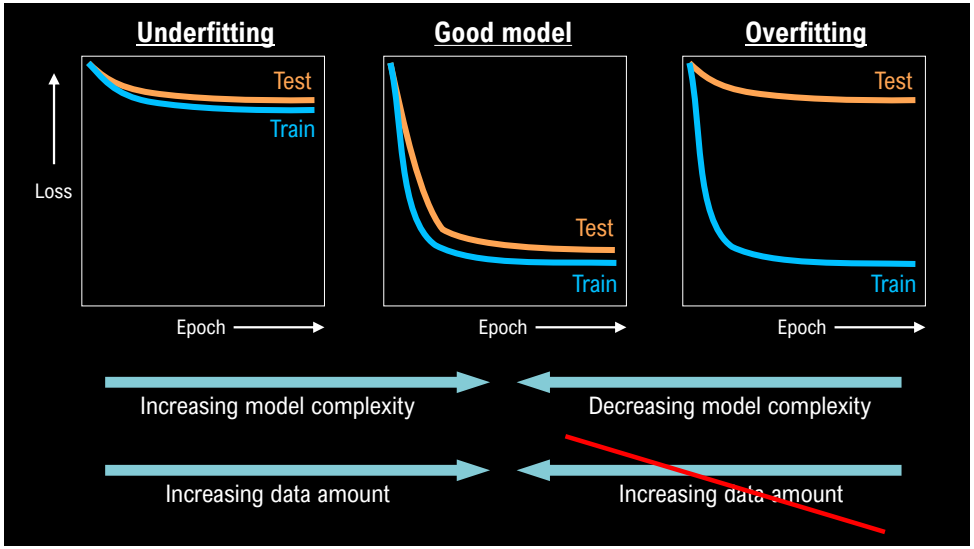
Travaux pratiques (Python)

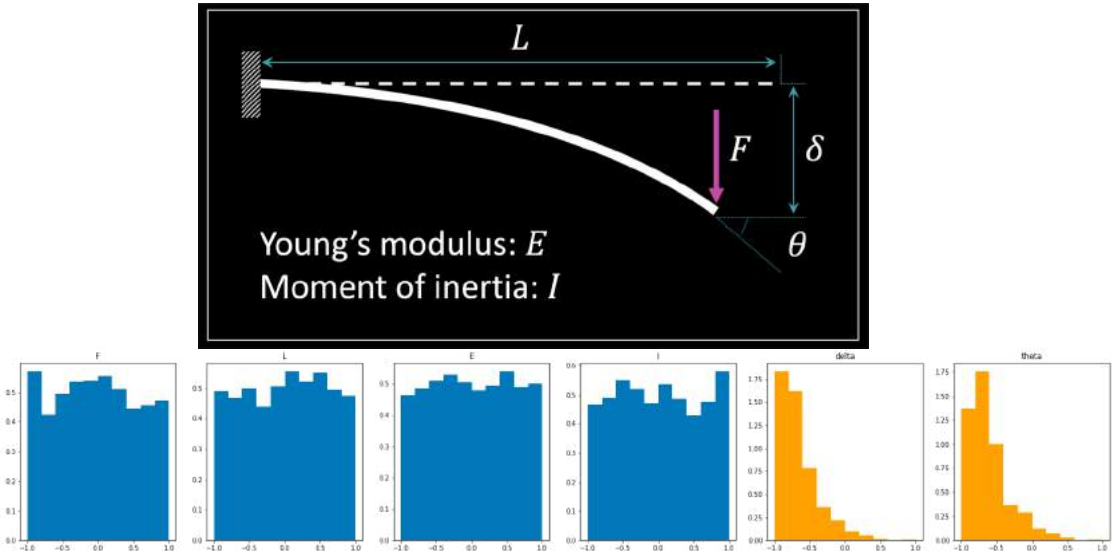
sous colab

1/Best activation function (link to derivative)

https://github.com/christianversloot/machine-learning-articles/blob/main/using-relu-sigmoid-and-tanh-with-pytorch-ignite-and-lightning.md

2/Question on data generation (experiments / cheap code / costly code





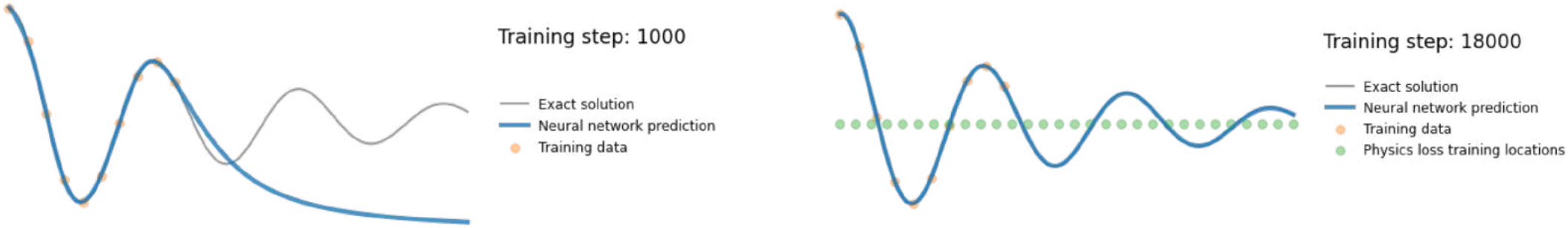
Travaux pratiques (Python)

sous colab

Pour aller plus loin: tutoriels SciANN en mécanique des solides, vibrations

https://github.com/sciann/sciann-applications

NN vs PINN



Part 1. Régression par processus gaussiens

Part 2. processus gaussiens versus réseaux de neurones

Part 3. Physics Informed Neural Networks {PINN}

ML vs Engineering

Kriging (Pionneer)	Gaussian Processes (link with AI)
Developed by Daniel Krige – 1951; formalized by Georges Mathéron in the 60's (Mines Paris)	Neural network with infinite neurons tend to Gaussian Process 1994

Krige, D. G., 1951, A statistical approach to some basic mine valuation problems on the Witwatersrand: J. Chem. Metal. Min. Soc. South Africa, v. 52, p. 119–139.

Matheron, G., 1963b, Principles of geostatistics: Economic Geol., v. 58, p. 1246–1266.

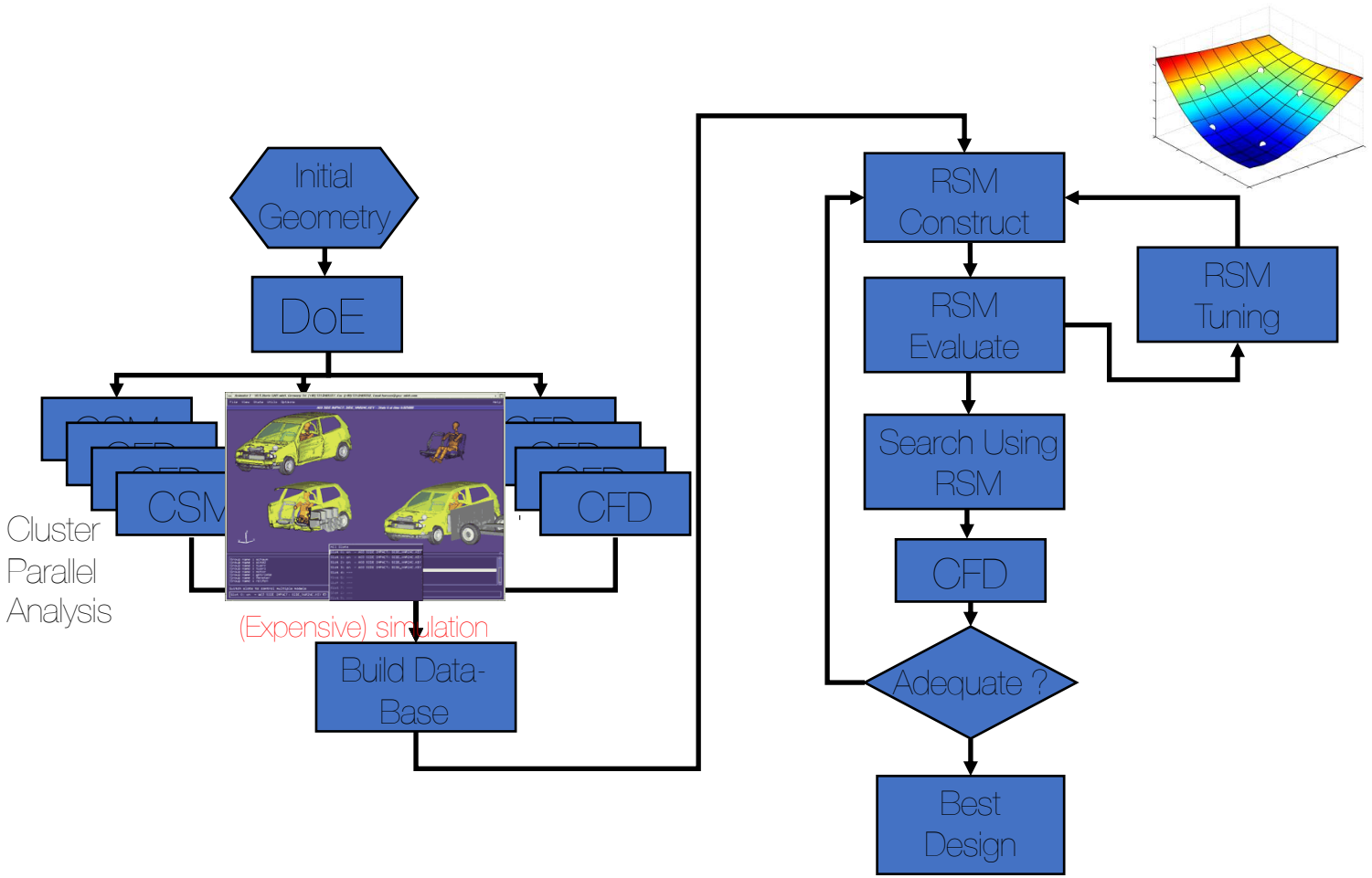
Neal, R. Priors for infinite networks. Tech. rep., University of Toronto, 1994.

Williams, C. K. I., and Rasmussen, C. E. Gaussian processes for regression. *Advances in Neural Information Processing Systems 8* (1996), 514–520.

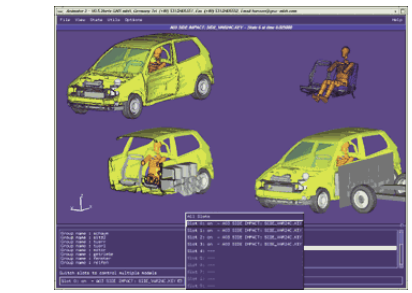
http://extrapolated-art.com

Qualitative claims such as "ML works OK for interpolation but doesn't work for extrapolation" are wrong.
https://arxiv.org/abs/2110.09485

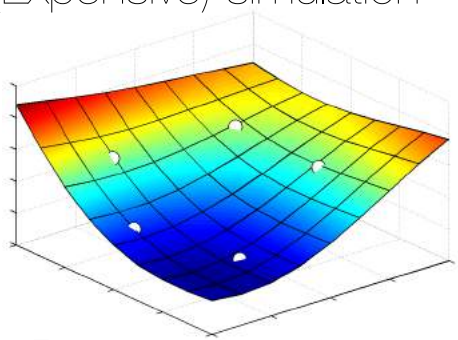
Surrogate


Initial Geometry
DoE
CSM
CFD
Cluster Parallel Analysis
(Expensive) simulation
Build Data-Base
RSM Construct
RSM Evaluate
RSM Tuning
Search Using RSM
CFD
Adequate ?
Best Design

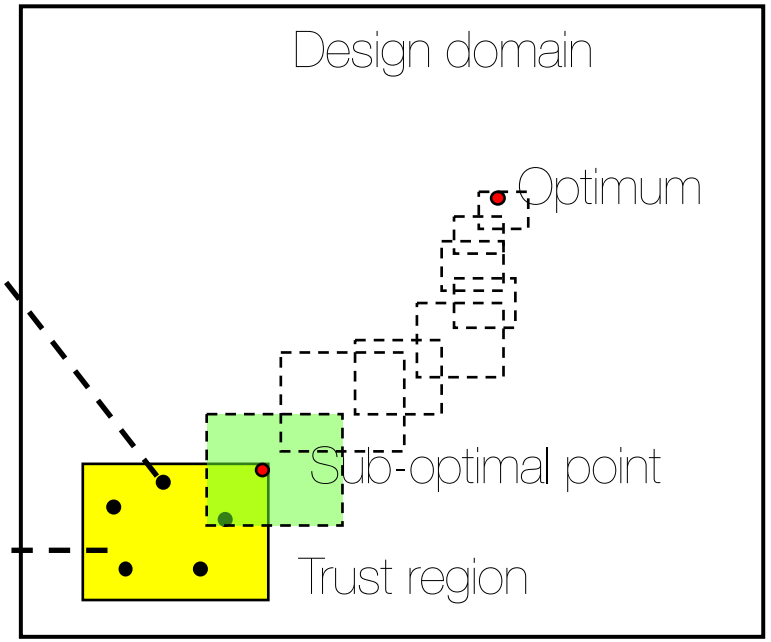
Surrogate Based Optimization by Trust Region



(Expensive) simulation



Response surface



Surrogate Models

- A surrogate model of a function is an approximation of the function that is **much** less costly to evaluate
- The surrogate model can then be used to help direct the search for the optimum of the real objective function

Fitting Surrogate Models

- Given a set of design points and function evaluations,

$$X = \left\{ \mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots, \mathbf{x}^{(m)} \right\} \qquad \mathbf{y} = \left\{ y^{(1)}, y^{(2)}, \ldots, y^{(m)} \right\}$$

- Find the model parameters which minimize error

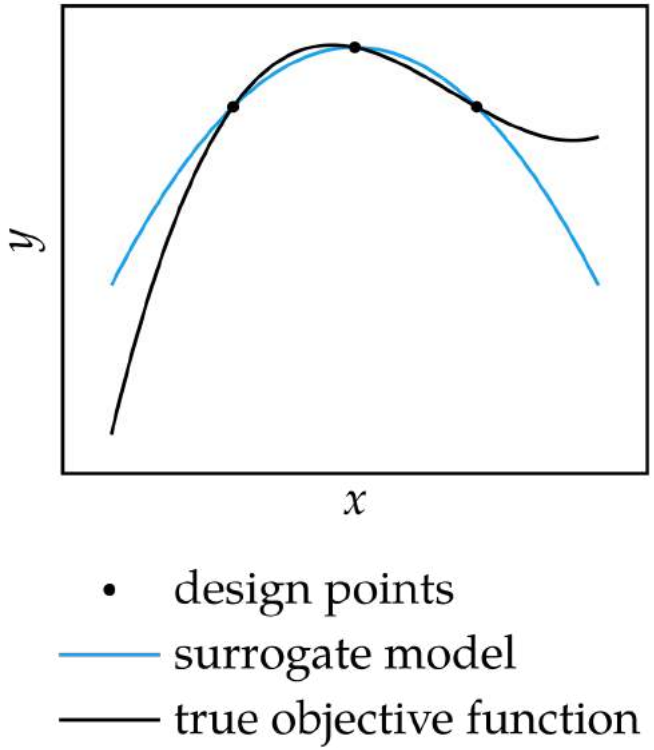
$$\underset{\theta}{\text{minimize}} \quad \|\mathbf{y} - \hat{\mathbf{y}}\|_p$$

$$\hat{\mathbf{y}} = \left\{ \hat{f}_\theta(\mathbf{x}^{(1)}), \hat{f}_\theta(\mathbf{x}^{(2)}), \ldots, \hat{f}_\theta(\mathbf{x}^{(m)}) \right\}$$

- This optimization problem is called regression

Fitting Surrogate Models

- Example: approximating f with a quadratic function



Basis Functions

- Any surrogate model represented as a linear combination of basis functions can be fit using regression

$$\underset{\theta}{\text{minimize}} \quad \|\mathbf{y} - \mathbf{B}\theta\|_2^2 \qquad \theta = \mathbf{B}^{+}\mathbf{y}$$

inv(B)	**linalg.inv(B)**	**inverse of square 2D array B**
pinv(B)	**linalg.pinv(B)**	**pseudo-inverse of 2D array B**
y\B	**linalg.solve(y, B) if a is square; linalg.lstsq(y, B) other wise**	**solution of B Θ = y for Θ**

Basis Functions: Polynomials

- A simple 1-D polynomial model of degree k has the form

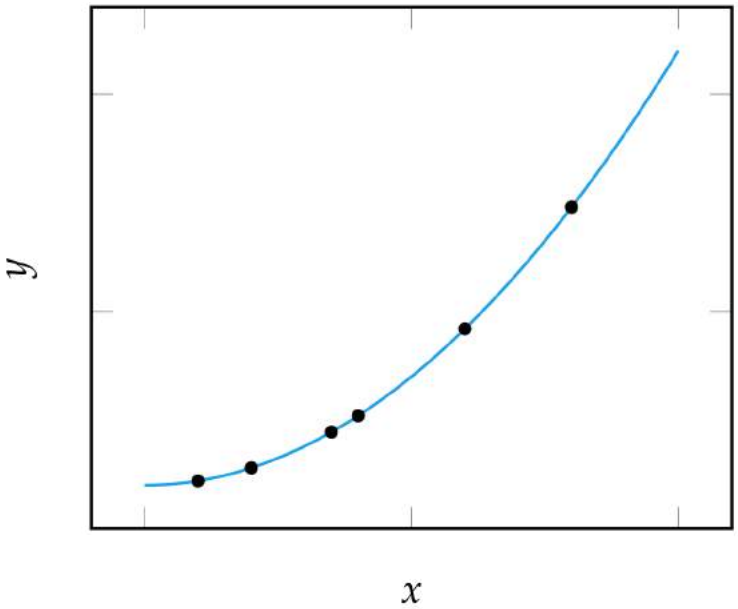
$$\hat{f}(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \cdots + \theta_k x^k = \sum_{i=0}^{k} \theta_i x^i$$

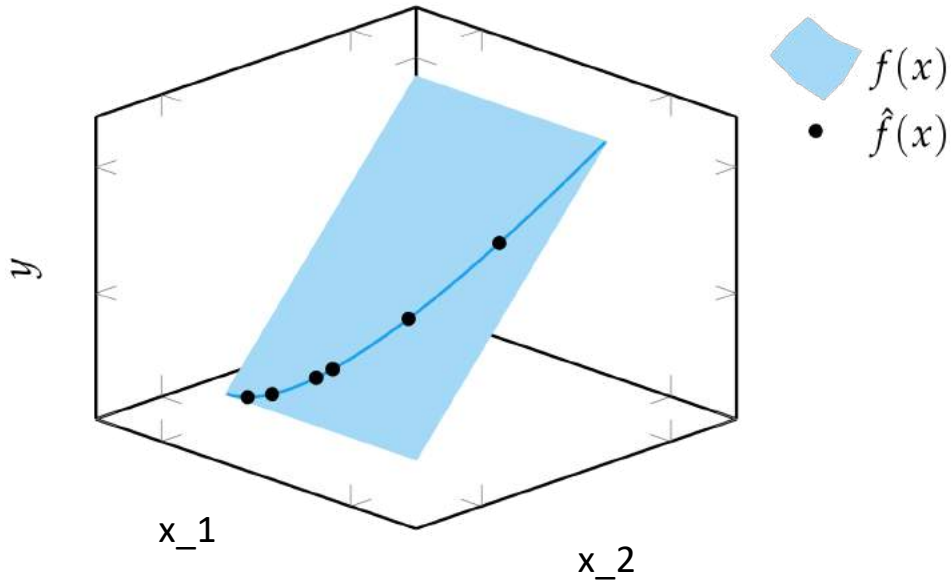
- In 2-D, the basis function has the form

$$b_{ij}(\mathbf{x}) = x_1^i x_2^j \ \text{ for } i, j \in \{0, \ldots, k\}, \ \ i + j \leq k$$

Basis Functions: Polynomials

- Polynomial surrogate models are fit with linear regression, so in a sufficiently high-dimensional space, each polynomial model is always linear





Model Selection

- After a model is fit to data, its quality still must be evaluated
- Data used to fit a model is called **training data**
- Models are compared based on generalization error which is the predictive error **on data not in the training set**

Model Selection

- One way to quantify generalization error is as the expected squared error of predictions

$$\epsilon_{\text{gen}} = \mathbb{E}_{\mathbf{x}\sim\mathcal{X}}\left[\left(f(\mathbf{x}) - \hat{f}(\mathbf{x})\right)^2\right]$$

- Since this requires knowledge of the original function value at unknown points, it can be tempting to estimate generalization error using training error, which is the mean squared error (MSE) of the model compared to the training data

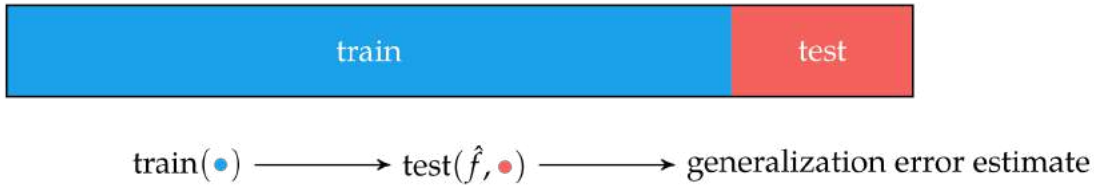
$$\epsilon_{\text{train}} = \frac{1}{m}\sum_{i=1}^{m}\left(f(\mathbf{x}^{(i)}) - \hat{f}(\mathbf{x}^{(i)})\right)^2$$

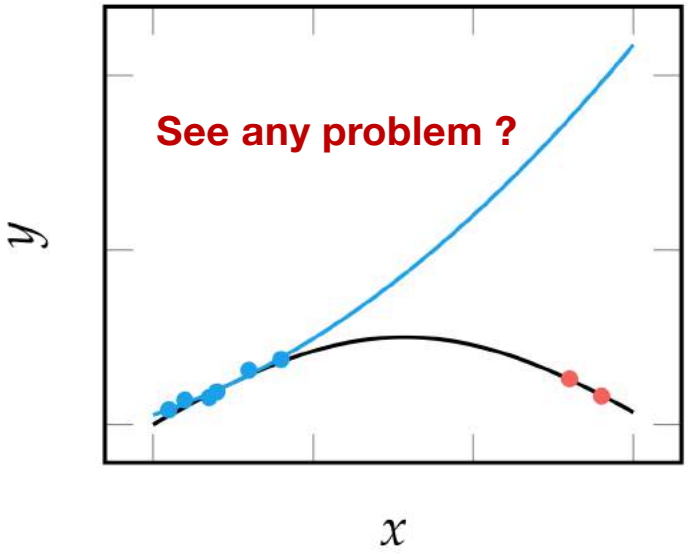
- **Models with low training error can still perform poorly outside of the regions containing training data**

Model Selection: Holdout

- The holdout method estimates generalization **error by partitioning available data into a test set and a training set.**
- Data from the test set is used to fit the model, and data from the training set is used to estimate generalization error.
- The model designer must decide how to partition the data into these two sets:
  - If training set is too small, the model will be poor
  - If training set is too large, the generalization error estimate will be poor
- In random subsampling, the holdout method is applied multiple times with randomly selected test-train partitions, with the generalization error averaged over all runs

Model Selection: Holdout



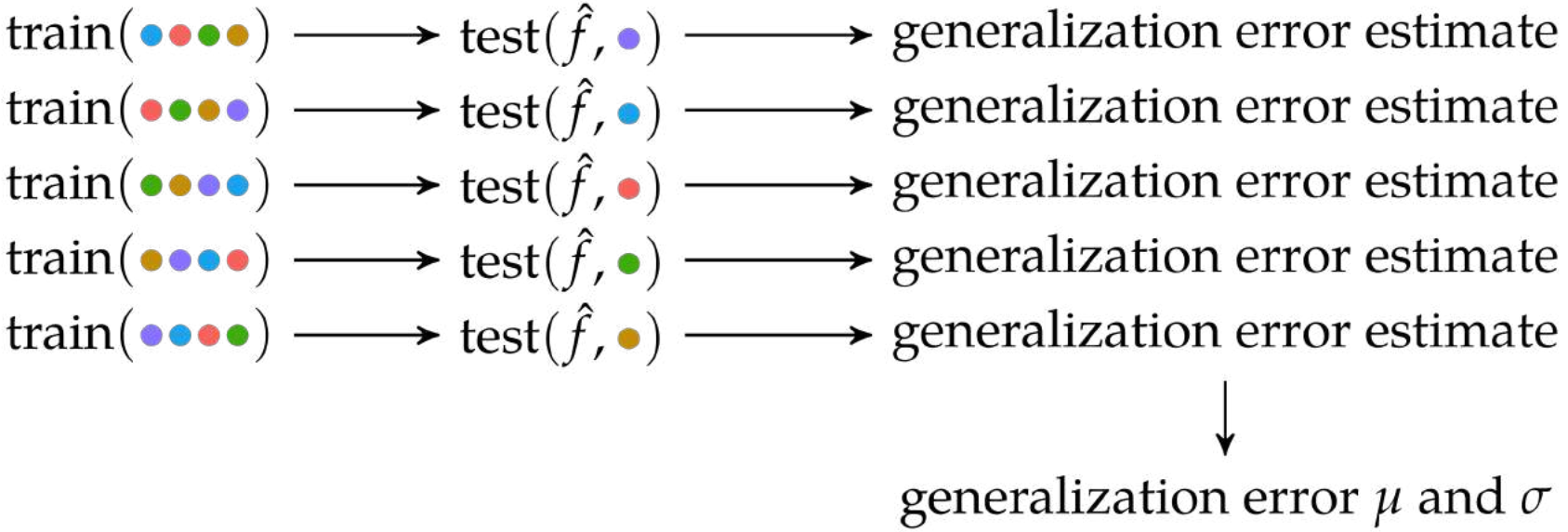


$$\epsilon_{\text{holdout}} = \frac{1}{h} \sum_{(\mathbf{x},y)\in\mathcal{D}_h} \left(y - \hat{f}(\mathbf{x})\right)^2$$

Model Selection: Cross Validation

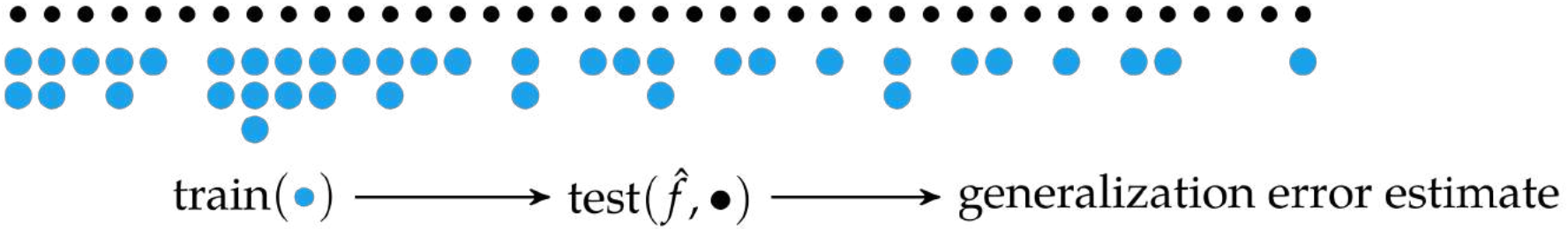
- Data can be more efficiently utilized for training and validation using k-fold cross validation
- The original dataset is **randomly partitioned into k subsets**
- At the ith iteration, the ith partition is selected as the holdout set
- The remaining $k-1$ partitions are used as the training set to generate a model and compute the ith generalization error μ_i
- The k values of μ are used to compute a mean and standard deviation

Model Selection: Cross Validation



Model Selection: Bootstrap

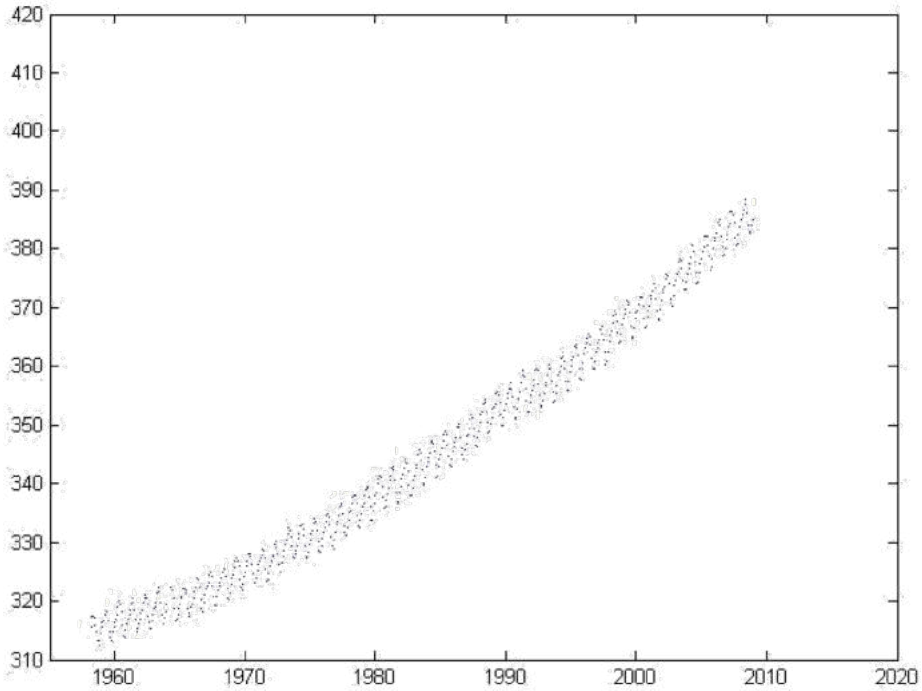
- The bootstrap method averages the generalization error by training on b bootstrap samples evaluated on the entire training set
- Each bootstrap sample is generated by randomly selecting m data points with replacement from a dataset of size m, meaning some data points can be selected multiple times in the same bootstrap sample



Summary

- Surrogate models are function approximations that can be optimized instead of the true, potentially expensive objective function
- Many surrogate models can be represented using a linear combination of basis functions
- Model selection involves a bias-variance tradeoff between models with low complexity that cannot capture important trends and models with high complexity that overfit to noise
- Generalization error can be estimated using techniques such as holdout, k-fold cross validation, and the bootstrap

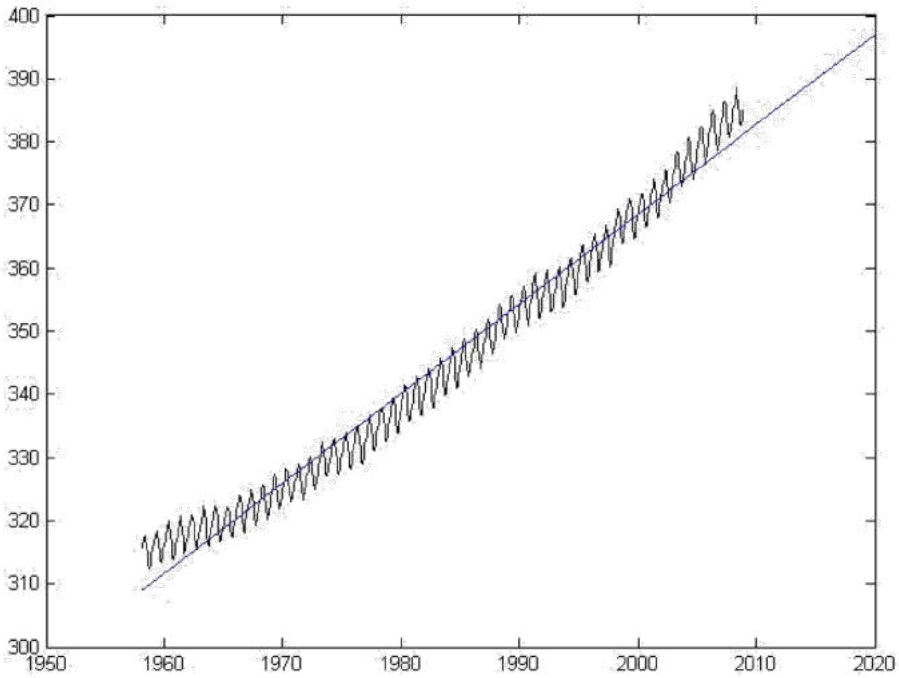
Limit of linear models for prediction



Month-wise data of Co2 concentration in atmosphere at Hawaii

Image Source: http://mlg.eng.cam.ac.uk/teaching/4f13/1314/

Example – Linear Regression



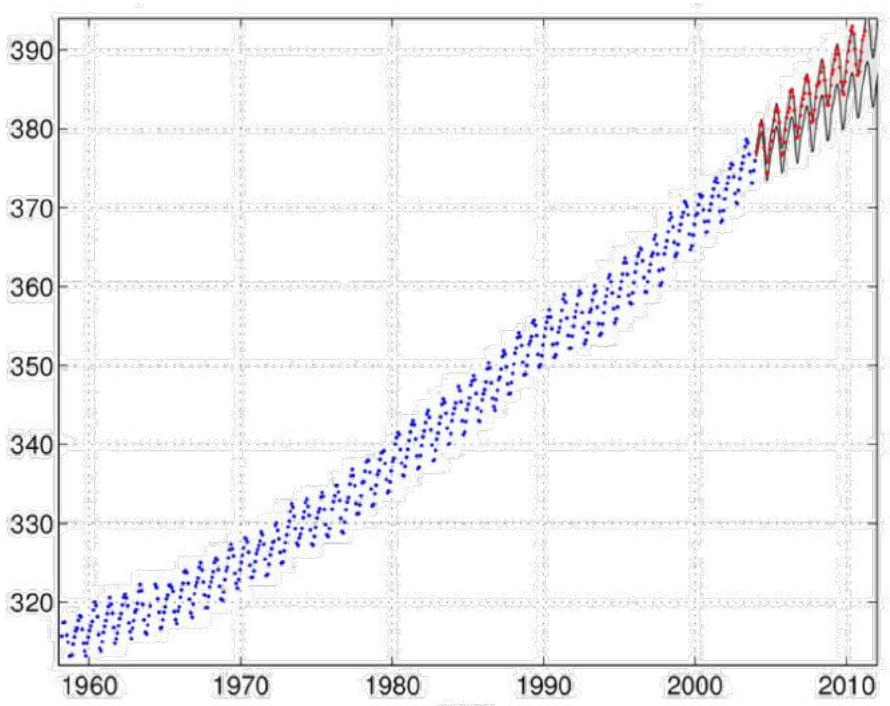
Should we choose a polynomial?

What degree of polynomial should we choose? (overfitting)

For a given degree, what parameters of polynomial should we choose

Image Source: http://mlg.eng.cam.ac.uk/teaching/4f13/1314/

Example – Gaussian Process



Predicted variance after year 2005 in grey, real data-points in red

Image Source: http://mlg.eng.cam.ac.uk/teaching/4f13/1314/

Probabilistic Surrogate Models

- It is often useful to quantify confidence in a surrogate model
- One approach is to use a probabilistic model that quantifies our confidence
- **A common probabilistic model is the Gaussian process (or Kriging)**

Gaussian Distribution

- Also called normal distribution
- Univariate Gaussian distribution is parameterized by mean μ and variance σ^2
- Multivariate Gaussian distribution is parameterized by mean vector μ and covariance matrix $\sum$
- Probability density at x is given by

$$x \sim \mathcal{N}(\mu, \Sigma)$$

$$\mathcal{N}(\mathbf{x} \mid \boldsymbol{\mu}, \boldsymbol{\Sigma}) = (2\pi)^{-n/2} |\boldsymbol{\Sigma}|^{-1/2} \exp\left(-\frac{1}{2} (\mathbf{x} - \boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-1} (\mathbf{x} - \boldsymbol{\mu}) \right)$$

mid (;)

exp(·) = e^·

Gaussian Distribution

- Sampling a value from a Gaussian is written as

$$\mathbf{x} \sim \mathcal{N}(\mu, \Sigma)$$

- Two jointly Gaussian random variables a and b are written as

$$\begin{bmatrix} \mathbf{a} \\ \mathbf{b} \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \mu_{\mathbf{a}} \\ \mu_{\mathbf{b}} \end{bmatrix}, \begin{bmatrix} \mathbf{A} & \mathbf{C} \\ \mathbf{C}^{\top} & \mathbf{B} \end{bmatrix} \right)$$

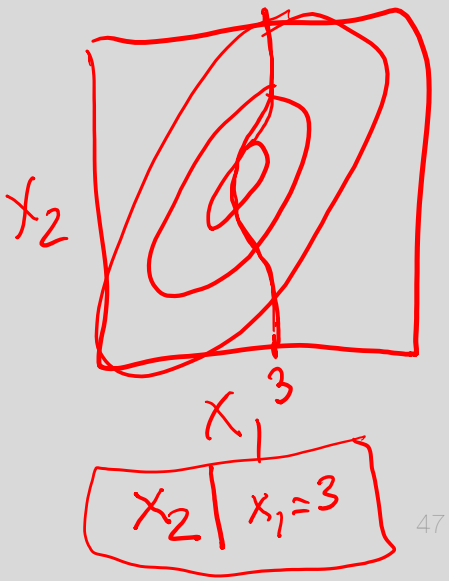
- Each variable's marginal distribution is written as

$$\mathbf{a} \sim \mathcal{N}(\mu_{\mathbf{a}}, \mathbf{A}) \qquad \mathbf{b} \sim \mathcal{N}(\mu_{\mathbf{b}}, \mathbf{B})$$

- A variable's conditional distribution is written as

$$\mathbf{a} \mid \mathbf{b} \sim \mathcal{N}\left(\mu_{\mathbf{a} \mid \mathbf{b}}, \Sigma_{\mathbf{a} \mid \mathbf{b}}\right)$$

$p(x_2 \mid x_1 = 3)$



Gaussian Distribution

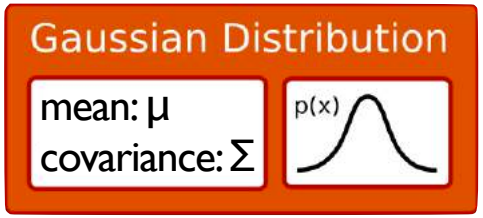
- Given the full set of parameters μ and ∑, the parameters for conditional distributions can be easily computed

$$\mu_{\mathbf{a}\mid\mathbf{b}} = \mu_{\mathbf{a}} + \mathbf{C}\mathbf{B}^{-1}(\mathbf{b} - \mu_{\mathbf{b}})$$

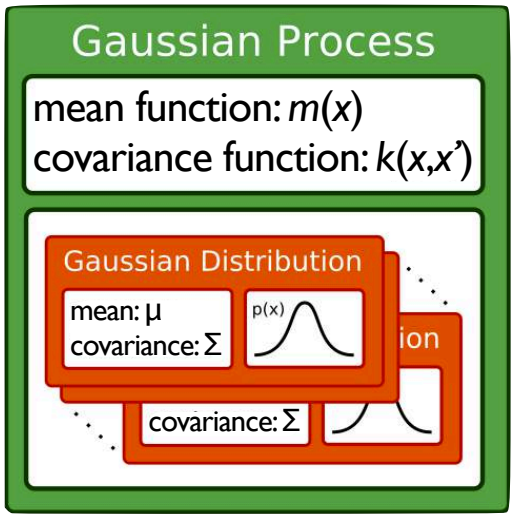
$$\Sigma_{\mathbf{a}\mid\mathbf{b}} = \mathbf{A} - \mathbf{C}\mathbf{B}^{-1}\mathbf{C}^{\mathsf{T}}$$

GD versus GP

- vs. Gaussian Distribution



$$X \sim \mathcal{G}(\mu,\Sigma)$$



$$f = (f_1, \ldots f_n)^T \sim \mathcal{GP}(m(x),k(x,x'))$$

Prediction

$$\mu_{a \mid b} = \mu_a + \mathbf{C}\mathbf{B}^{-1}(\mathbf{b} - \mu_b)$$
$$\Sigma_{a \mid b} = \mathbf{A} - \mathbf{C}\mathbf{B}^{-1}\mathbf{C}^{\intercal}$$

- The predicted values can be found using the conditional distribution

$$\hat{\mathbf{y}} \mid \mathbf{y} \sim \mathcal{N}\left(\underbrace{\mathbf{m}(X^*) + \mathbf{K}(X^*, X)\mathbf{K}(X, X)^{-1}(\mathbf{y} - \mathbf{m}(X))}_{\text{mean}}, \underbrace{\mathbf{K}(X^*, X^*) - \mathbf{K}(X^*, X)\mathbf{K}(X, X)^{-1}\mathbf{K}(X, X^*)}_{\text{covariance}}\right)$$

using the conditional distribution formulas previously presented

- The predicted mean and variance of $\hat{y}$ is

$$\begin{aligned}\hat{\mu}(\mathbf{x}) &= m(\mathbf{x}) + \mathbf{K}(\mathbf{x}, X)\mathbf{K}(X, X)^{-1}(\mathbf{y} - \mathbf{m}(X)) \\ &= m(\mathbf{x}) + \theta^{\intercal}\mathbf{K}(X, \mathbf{x}) \\ \hat{v}(\mathbf{x}) &= \mathbf{K}(\mathbf{x}, \mathbf{x}) - \mathbf{K}(\mathbf{x}, X)\mathbf{K}(X, X)^{-1}\mathbf{K}(X, \mathbf{x})\end{aligned}$$

Gaussian Processes

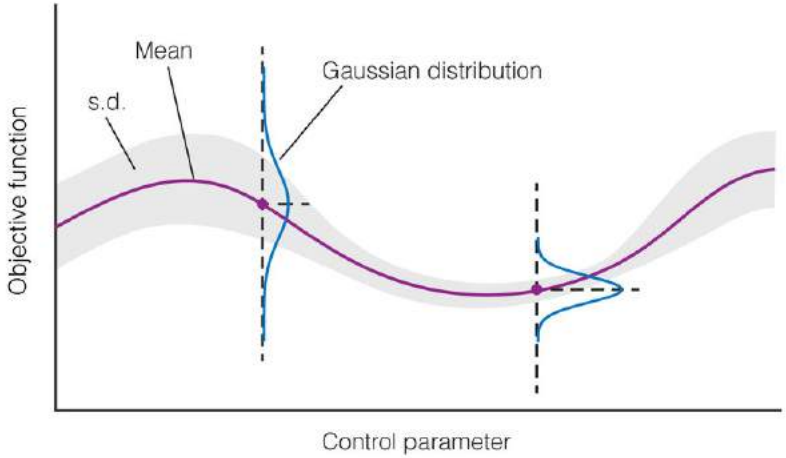
$k(x, x') = \varphi(x)^T \varphi(x')$

$\varphi(x)$:a nonlinear feature space mapping

k is a symmetric function of its arguments so that

$k(x, x') = k(x', x)$

- A Gaussian Process extends the idea of
- Gaussian distributions to functions



- For any finite set of points, the distribution ov as

$$\begin{bmatrix} y_1 \\ \vdots \\ y_m \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} m(\mathbf{x}^{(1)}) \\ \vdots \\ m(\mathbf{x}^{(m)}) \end{bmatrix}, \begin{bmatrix} k(\mathbf{x}^{(1)}, \mathbf{x}^{(1)}) & \cdots & k(\mathbf{x}^{(1)}, \mathbf{x}^{(m)}) \\ \vdots & \ddots & \vdots \\ k(\mathbf{x}^{(m)}, \mathbf{x}^{(1)}) & \cdots & k(\mathbf{x}^{(m)}, \mathbf{x}^{(m)}) \end{bmatrix} \right)$$

Here, m(x) is the mean function and k(x, x') is the covariance function or kernel

Supervised learning
+
Normal distribution
=

Mais comment obtenir:

- K les covariances entre les points d'entraînement ?
- $K_\star$ les covariances entre entraînement et test ?
- $K_{\star\star}$ les covariances entre test ?

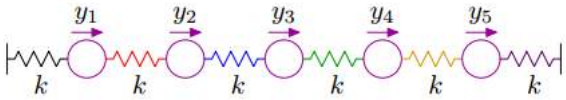
Soit $D = \{\mathbf{x}^i, y^i\}_{i=1}^N$ nos données et $\mathbf{x}_t^1 \dots \mathbf{x}_t^T$ les points que l'on veut inférer.
On a établit que (en simplifiant en fixant la moyenne à 0) :

$$p\left(\begin{pmatrix} y^1 \\ \vdots \\ y^N \\ y_t^1 \\ \vdots \\ y_t^T \end{pmatrix} | \mathbf{x}^1, \mathbf{x}^2, \dots, \mathbf{x}^N, \mathbf{x}_t^1, \dots, \mathbf{x}_t^T\right) \sim \mathcal{N}(0, \Sigma),\ \Sigma = \begin{bmatrix} K & K_\star \\ K_\star^t & K_{\star\star} \end{bmatrix}$$

Alors

$$p(y_t^1 \dots y_t^T | \mathbf{y}, \mathbf{x}^1, \dots \mathbf{x}_t^T) \sim \mathcal{N}(K_\star^t K^{-1}\mathbf{y}, K_{\star\star} - K_\star^t K^{-1} K_\star)$$

Quiz on the meaning of *K*

- Example: 

- Which is the covariance matrix, and which the inverse covariance?

- Which is the covariance matrix, and which the inverse covariance?

$$\frac{k}{T}\begin{bmatrix} 2 & -1 & 0 & 0 & 0 \\ -1 & 2 & -1 & 0 & 0 \\ 0 & -1 & 2 & -1 & 0 \\ 0 & 0 & -1 & 2 & -1 \\ 0 & 0 & 0 & -1 & 2 \end{bmatrix} \qquad \frac{T}{k}\begin{bmatrix} 0.83 & 0.67 & 0.50 & 0.33 & 0.17 \\ 0.67 & 1.33 & 1.00 & 0.67 & 0.33 \\ 0.50 & 1.00 & 1.50 & 1.00 & 0.50 \\ 0.33 & 0.67 & 1.00 & 1.33 & 0.67 \\ 0.17 & 0.33 & 0.50 & 0.67 & 0.83 \end{bmatrix}$$

Matrice de covariance = Kernel !

Ce que l'on veut pour la matrice de covariance :

- qu'elle soit symétrique ! (i influence j comme j influence i)
- deux points "similaires" doivent avoir une corrélation forte : $Cov(\mathbf{x}^i, \mathbf{x}^j)$ grand
- deux points "dissimilaires" doivent avoir une corrélation faible : $Cov(\mathbf{x}^i, \mathbf{x}^j)$ petit
- qu'elle soit semi-défini positive
- $Cov(\mathbf{x}^i, \mathbf{x}^i)$ doit dénoté la variance en ce point

$\Rightarrow$ Très similaire à la notion de noyaux en SVM ! Autant utiliser une fonction noyau pour encoder la covariance ...

Covariances typiques :

- Squared Exponential : $K(\mathbf{x}^1, \mathbf{x}^2) = \sigma^2 e^{-\frac{1}{2}\left(\frac{\|\mathbf{x}^1 - \mathbf{x}^2\|^2}{\lambda}\right)}$
- Linéaire : $K(\mathbf{x}^1, \mathbf{x}^2) = \lambda + \langle \mathbf{x}^1, \mathbf{x}^2 \rangle$
- Periodic : $K(\mathbf{x}^1, \mathbf{x}^2) = \sigma^2 e^{-\frac{2sin^2(\frac{\|\mathbf{x}^1 - \mathbf{x}^2\|}{2})}{\lambda^2}}$

Sometimes people express probabilistic models in terms of energy functions that are minimized in the most probable configuration. For example, in regression with cubic splines, a regularizer is defined which describes the energy that a steel ruler would have if bent into the shape of the curve.

Such models usually have the form:

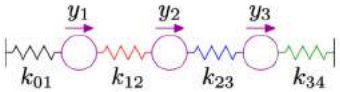
$$P(\mathbf{y}) = \frac{1}{Z} e^{-\frac{E(\mathbf{y})}{T}}, \tag{29}$$

and in simple cases, the energy $E(\mathbf{y})$ may be a quadratic function of $\mathbf{y}$, such as

$$E(\mathbf{y}) = -\sum_{i<j} J_{ij} y_i y_j + \sum_i a_i y_i^2 \tag{30}$$

If so, then the distribution is a Gaussian (just like (1)), and the 'couplings' J_{ij} are minus the coefficients in the *inverse* covariance matrix.

As a simple example, consider a set of three masses coupled by springs, and subjected to thermal perturbations.



THREE MASSES, FOUR SPRINGS

The equilbrium positions are $(y_1, y_2, y_3, y_4) = (0, 0, 0, 0)$, and the spring constants are k_{ij}. The extension of the second spring is $y_2 - y_1$. The energy of this system is

$$\begin{aligned} E(\mathbf{y}) &= \frac{1}{2}k_{01}y_1^2 + \frac{1}{2}k_{12}(y_2 - y_1)^2 + \frac{1}{2}k_{23}(y_3 - y_2)^2 + \frac{1}{2}k_{34}y_3^2 \\ &= \frac{1}{2}\begin{bmatrix} y_1 & y_2 & y_3 \end{bmatrix} \begin{bmatrix} k_{01} + k_{12} & -k_{12} & 0 \\ -k_{12} & k_{12} + k_{23} & -k_{23} \\ 0 & -k_{23} & k_{23} + k_{34} \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} \end{aligned}$$

So at temperature T, the probability distribution of the displacements is Gaussian with inverse covariance matrix

$$\frac{1}{T}\begin{bmatrix} k_{01}+k_{12} & -k_{12} & 0 \\ -k_{12} & k_{12}+k_{23} & -k_{23} \\ 0 & -k_{23} & k_{23}+k_{34} \end{bmatrix} \tag{31}$$

Notice that there are 0 entries between displacements y_1 and y_3, the two masses that are not directly coupled by a spring.

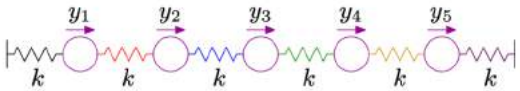


Figure 1. FIVE MASSES, SIX SPRINGS

So inverse covariance matrices are sometimes very sparse. If we have five masses in a row connected by identical springs k for example, then

$$\mathbf{K}^{-1} = \frac{k}{T}\begin{bmatrix} 2 & -1 & 0 & 0 & 0 \\ -1 & 2 & -1 & 0 & 0 \\ 0 & -1 & 2 & -1 & 0 \\ 0 & 0 & -1 & 2 & -1 \\ 0 & 0 & 0 & -1 & 2 \end{bmatrix}. \tag{32}$$

But this sparsity doesn't carry over to the covariance matrix, which is

$$\mathbf{K} = \frac{T}{k}\begin{bmatrix} 0.83 & 0.67 & 0.50 & 0.33 & 0.17 \\ 0.67 & 1.33 & 1.00 & 0.67 & 0.33 \\ 0.50 & 1.00 & 1.50 & 1.00 & 0.50 \\ 0.33 & 0.67 & 1.00 & 1.33 & 0.67 \\ 0.17 & 0.33 & 0.50 & 0.67 & 0.83 \end{bmatrix}. \tag{33}$$

Et si on introduit du bruit ?

Bruit additif gaussien

- On observe $y^i = f(\mathbf{x}^i) + \epsilon_i$, avec ϵ_i indépendant et suivant $\mathcal{N}(0, \sigma^2)$
- La covariance est changée en $\hat{\Sigma}$:

$$\hat{\Sigma}_{ij} = \mathbb{E}[(f(\mathbf{x}^i) + \epsilon_i)(f(\mathbf{x}^j) + \epsilon_j)] = \mathbb{E}[f(\mathbf{x}^i)f(\mathbf{x}^j)] + \mathbb{E}[f(\mathbf{x}^j)]\mathbb{E}[\epsilon_i] + \mathbb{E}[f(\mathbf{x}^i)]\mathbb{E}[\epsilon_j] + \mathbb{E}[\epsilon_i\epsilon_j]$$

- Pour $i \neq j$, $\mathbb{E}[\epsilon_i] = 0, \mathbb{E}[\epsilon_i\epsilon_j] = \mathbb{E}[\epsilon_i]\mathbb{E}[\epsilon_j] = 0$

$$\hat{\Sigma_{ij}} = \mathbb{E}[f(\mathbf{x}^i)f(\mathbf{x}^j)] = \Sigma_{ij}$$

- Pour $i = j$, $\mathbb{E}[\epsilon_i] = 0$, $\mathbb{E}[\epsilon_i^2] = \sigma^2$

$$\hat{\Sigma_{ii}} = \mathbb{E}[f(\mathbf{x}^i)f(\mathbf{x}^i)] + \mathbb{E}[\epsilon_i^2] = \Sigma_{ii} + \sigma^2$$

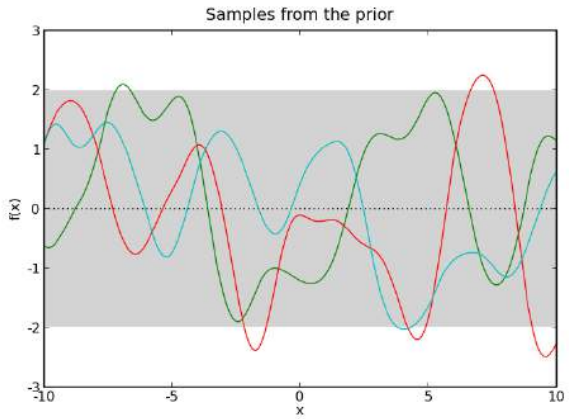
- Donc $\hat{\Sigma} = \Sigma + \sigma^2\boldsymbol{I}$

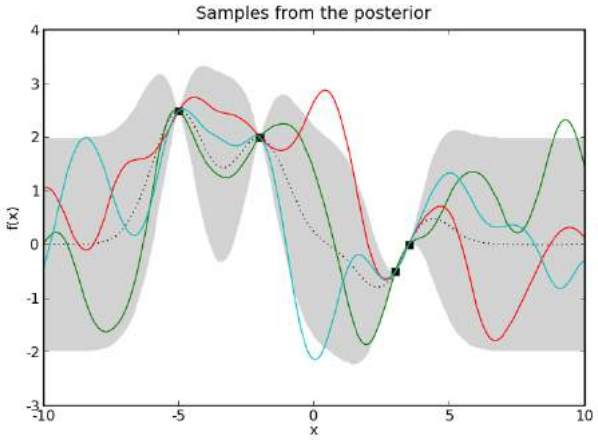
Formule de la régression GP dans le cas général

$$p(y_t^1 \dots y_t^T | \mathbf{y}, \mathbf{x}^1, \dots \mathbf{x}_t^T) \sim \mathcal{N}(K_\star^t(K + \sigma^2\boldsymbol{I})^{-1}\mathbf{y},\ K_{\star\star} - K_\star^t(K + \sigma^2\boldsymbol{I})^{-1}K_\star)$$

$\Rightarrow$ Régression à noyaux !

- Mais avec l'information sur l'incertitude liée à la prédiction !





Définition

- La fonction f dont on cherche la prédiction est vue comme une collection de points $\boldsymbol{f}$ (les mesures en différents points) potentiellement infinie.
- Un processus gaussien est une collection de variables aléatoires (potentiellement infinie) tels que la distribution jointe de tout sous-ensemble de ces variables est une gaussienne multivariée :

$$f \sim GP(\mu, k)$$

avec $\mu(\mathbf{x})$ et $k(\mathbf{x}^1, \mathbf{x}^2)$ sont les fonctions de moyenne et de covariance.

- On cherche à estimer la distribution $P(\boldsymbol{f_t}|\mathbf{x}_t, D)$ en utilisant un prior GP : $P(\boldsymbol{f}|\mathbf{x}) \sim \mathcal{N}(\mu, \Sigma)$ et en le conditionnant par les données d'entraînement D afin de modéliser la distribution jointe $\boldsymbol{f}$ des points d'entraînement et $\boldsymbol{f}_t$ les points de test.

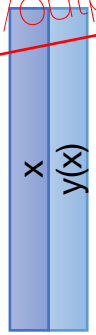
- The covariance function defines how smoothly the (latent) function *f* varies from a given *x*.
- The data points "anchor" the function *f* at specific *x* locations.

Matrix view of Gaussian Process

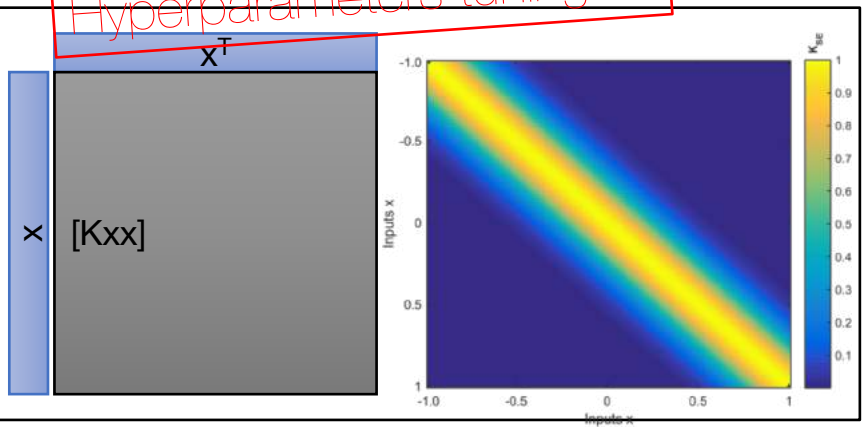
1/ Get your inputs/outputs data

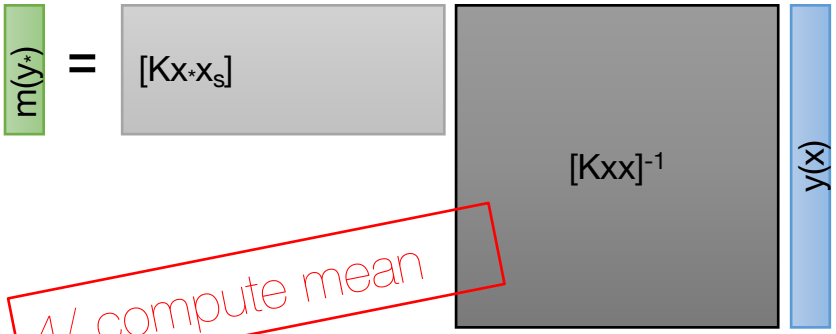
2/ You wan to predict at x*

3/ Choose a Kernel/Construct Kxx and Hyperparameters tuning



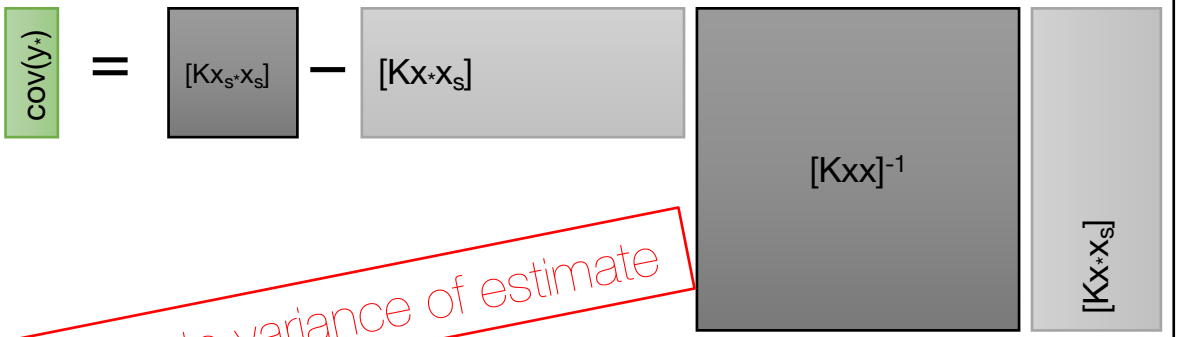
$$k(x, x') = \theta_1{}^2 \exp\left(-\frac{(x - x')^2}{2\theta_2{}^2}\right)$$





4/ compute mean

$$\boldsymbol{m(x_*) = K_*[Kxx]^{-1}y}$$



4/ compute variance of estimate

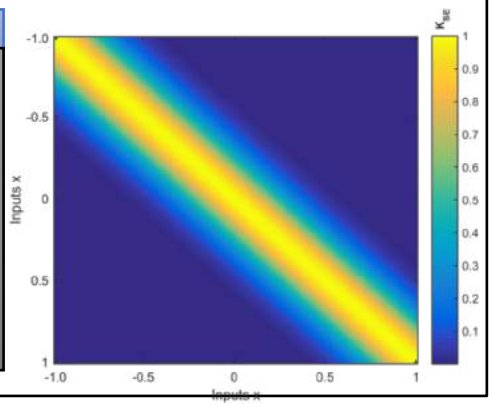
$$\boldsymbol{var(x_*, x'_*) = K_{**} - K_*^T[Kxx]^{-1}K_*}$$

Matrix view of Gaussian Process

x y(x)

x*

$k(x, x') = \theta_1{}^2 \exp(-\frac{(x - x')^2}{2\theta_2{}^2})$ = x [Kxx] x^T



m(y*) = [Kx*xs] [Kxx]$^{-1}$ y(x)

posterior_mean = covXXs @ np.linalg.inv(covXX_noisy) @ y

cov(y*) = [Kxs*xs] − [Kx*xs] [Kxx]$^{-1}$ [Kx*xs]

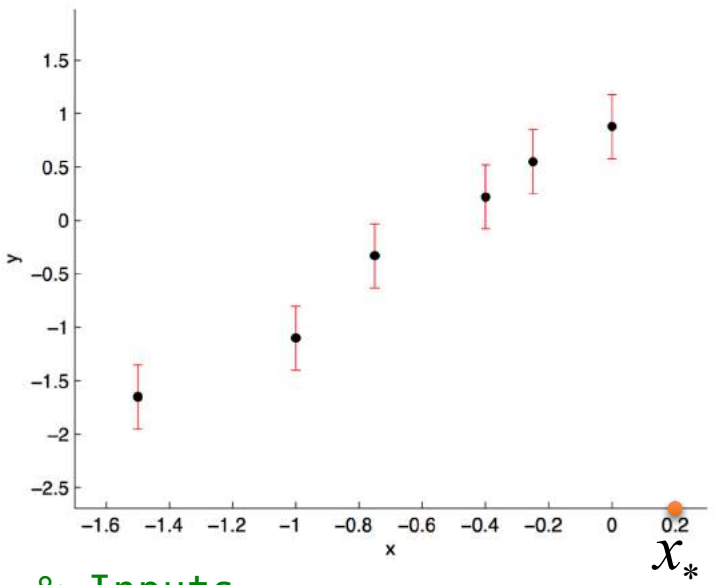
posterior_cov = covXsXs − covXXs @ np.linalg.inv(covXX_noisy) @

Example

Gaussian Processes for Regression A Quick Introduction, M.Ebden, August 2008.

$$y \sim N(0, K)$$

$$k(x, x') = \sigma_f^2 \exp\left[-\frac{(x - x')^2}{2l^2}\right] + \sigma_n \delta(x, x') \qquad \delta(x,x') = \begin{cases} 1, & x = x' \\ 0, & x \neq x' \end{cases}$$




```
% Inputs
X = [-1.5 -1 -.75 -.4 -.25 0]';
% Outputs
y = [-1.6 -1.10 -.4 .2 0.6 1]';
```

(0.20, ?)

$$K = \begin{bmatrix} k(x_1, x_1) & k(x_1, x_2) & \cdots & k(x_1, x_n) \\ k(x_2, x_1) & k(x_2, x_2) & \cdots & k(x_2, x_n) \\ \vdots & \vdots & \ddots & \vdots \\ k(x_n, x_1) & k(x_n, x_2) & \cdots & k(x_n, x_n) \end{bmatrix}$$

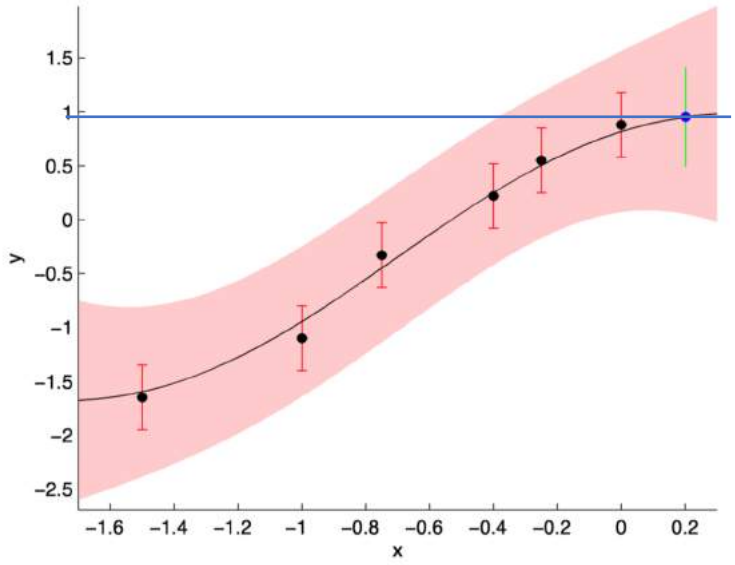
```
% Taille 6*6
```

$$K_* = \begin{bmatrix} k(x_*, x_1) & k(x_*, x_2) & \cdots & k(x_*, x_n) \end{bmatrix} \qquad K_{**} = k(x_*, x_*)$$

$$\begin{bmatrix} \mathbf{y} \\ y_* \end{bmatrix} \sim \mathcal{N}\left(\mathbf{0}, \begin{bmatrix} K & K_*^{\mathrm{T}} \\ K_* & K_{**} \end{bmatrix}\right)$$

$$y_* | \mathbf{y} \sim \mathcal{N}(K_* K^{-1} \mathbf{y}, K_{**} - K_* K^{-1} K_*^{\mathrm{T}})$$

$$\bar{y}_* = K_* K^{-1} \mathbf{y} \qquad \mathrm{var}(y_*) = K_{**} - K_* K^{-1} K_*^{\mathrm{T}}$$



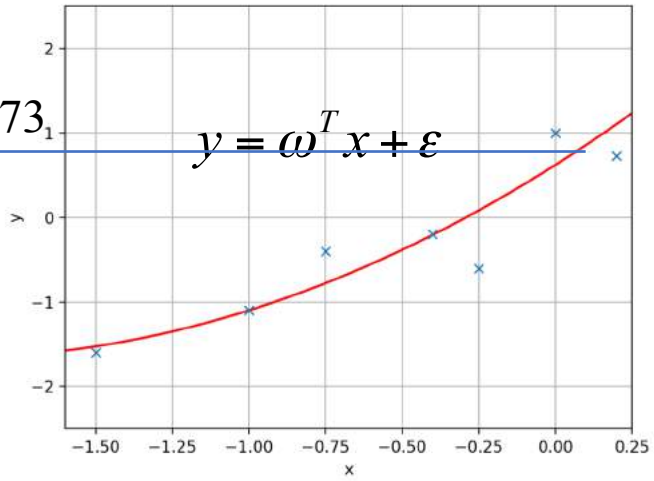
$$\sigma_n = 0.3$$

$$K = \begin{bmatrix} 1.70 & 1.42 & 1.21 & 0.87 & 0.72 & 0.51 \\ 1.42 & 1.70 & 1.56 & 1.34 & 1.21 & 0.97 \\ 1.21 & 1.56 & 1.70 & 1.51 & 1.42 & 1.21 \\ 0.87 & 1.34 & 1.51 & 1.70 & 1.59 & 1.48 \\ 0.72 & 1.21 & 1.42 & 1.59 & 1.70 & 1.56 \\ 0.51 & 0.97 & 1.21 & 1.48 & 1.56 & 1.70 \end{bmatrix}$$

$$K_{**} = 1.70 \qquad K_* = [0.38 \quad 0.79 \quad 1.03 \quad 1.35 \quad 1.46 \quad 1.58]$$

$$\bar{y}_* = 0.95 \qquad \mathrm{var}(y_*) = 0.21.$$

$\hat{y}_* = 0.73$



http://htmlpreview.github.io/?https://github.com/jomorlier/mdocourse/blob/master/GP_Tutorial/GP_Tutorial.html

How to start?

Use Google's Colab (no installation required, but times out if idle):

1. go to **https://colab.research.google.com**
2. login
3. File > Open notebook
4. click on Github (no need to login or authorize anything)
5. paste the git link: **https://github.com/jomorlier/IA_CNRS_ICA**
6. click search and then click on the notebook

Part 1. Régression par processus gaussiens

Part 2. processus gaussiens versus réseaux de neurones

Part 3. Physics Informed Neural Networks {PINN}

Optimizing Marginal Likelihood (ML)

$$\text{ML} = log\big(p(y|X,\theta)\big) = -\frac{1}{2}y^T K^{-1} y \; - \frac{1}{2} log|K| \; - \frac{n}{2} log(2\pi)$$

- It is a combination of **data-fit term**, a **complexity penalty** term and a **normalization term**

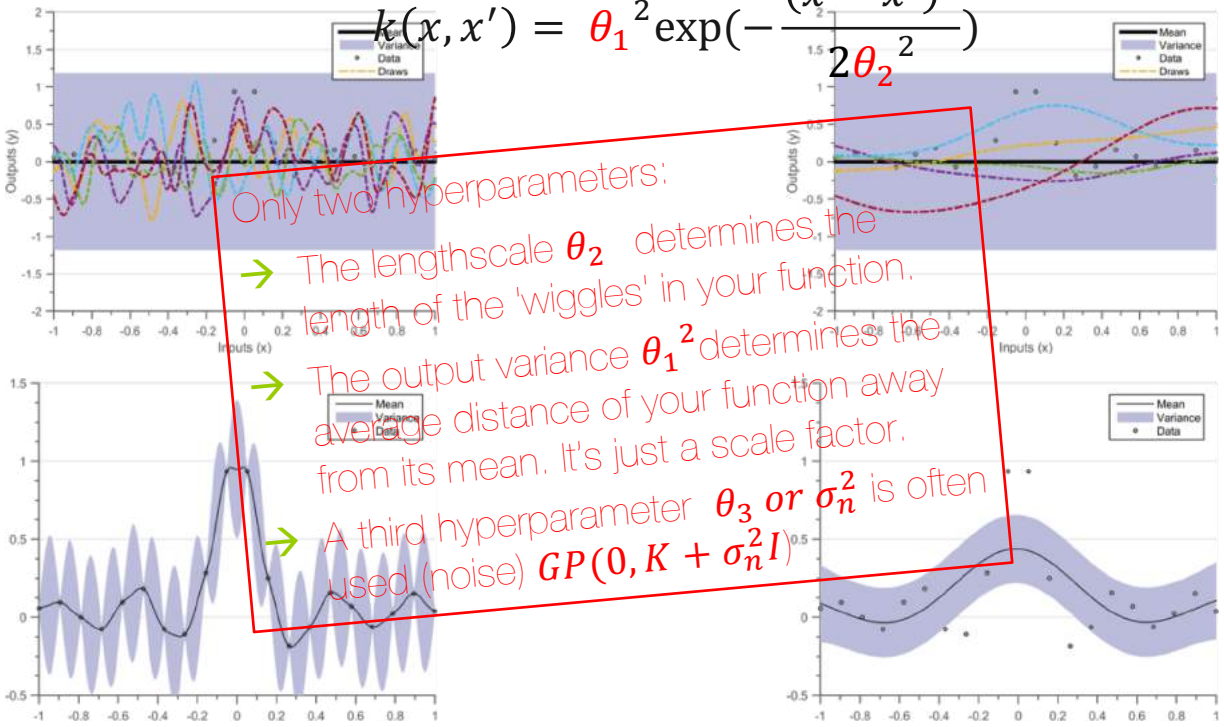
$$k(x,x') = {\theta_1}^2 \exp\left(-\frac{(x-x')^2}{2{\theta_2}^2}\right)$$

Only two hyperparameters:

→ The lengthscale θ_2 determines the length of the 'wiggles' in your function.

→ The output variance ${\theta_1}^2$ determines the average distance of your function away from its mean. It's just a scale factor.

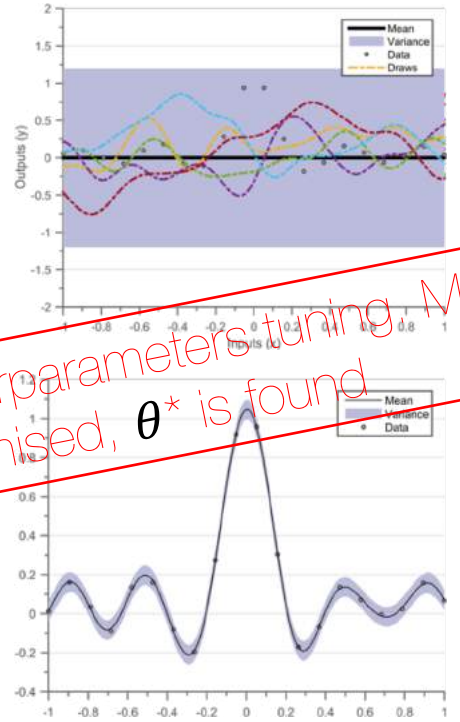
→ A third hyperparameter θ_3 *or* σ_n^2 is often used (noise) $GP(0, K + \sigma_n^2 I)$



ML = -8.2

ML = -35.3

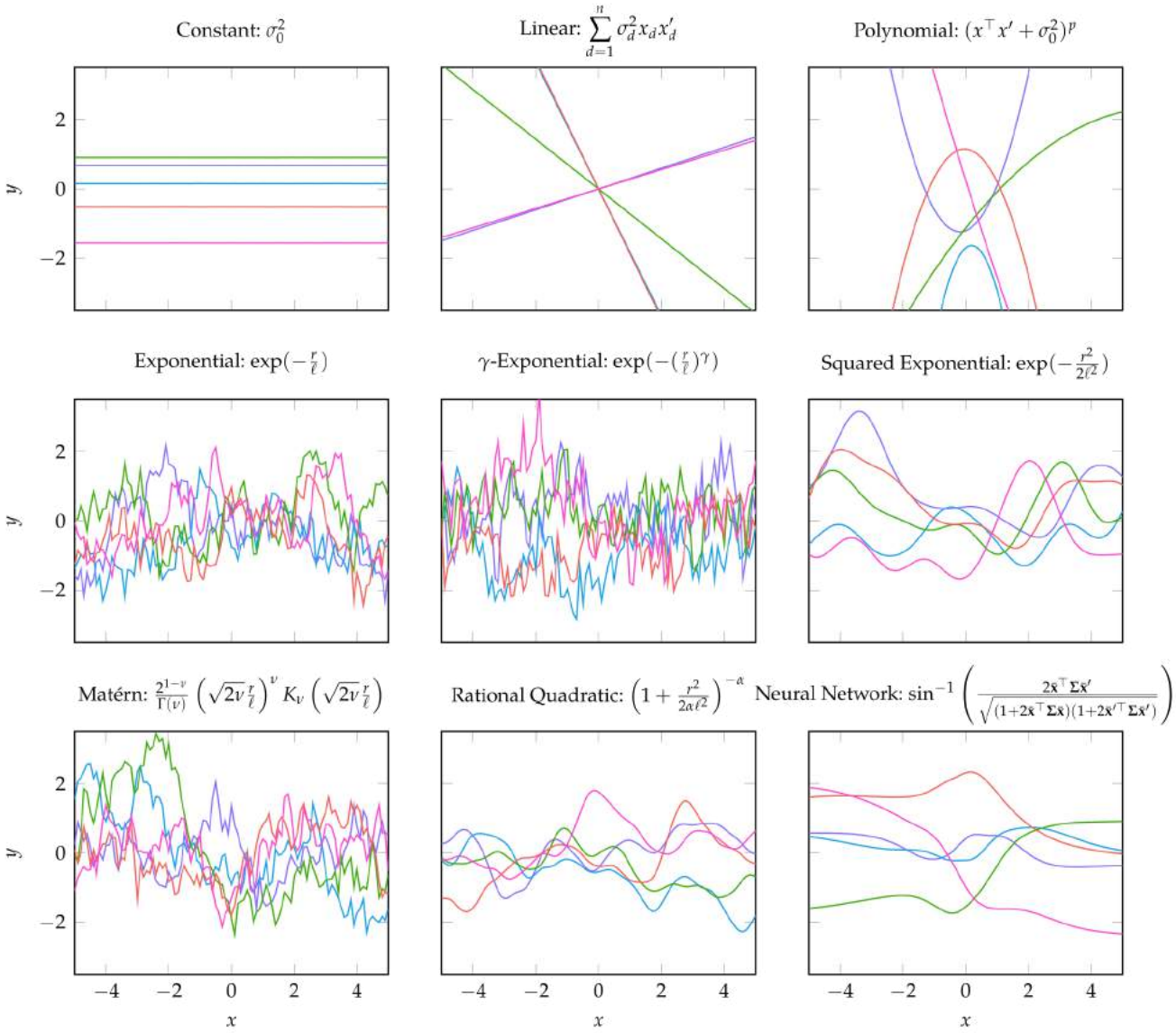
3/ Hyperparameters tuning: ML is maximised, θ^* is found



ML = 6,04

Gaussian Processes

- Examples of the same Gaussian process with different kernel functions

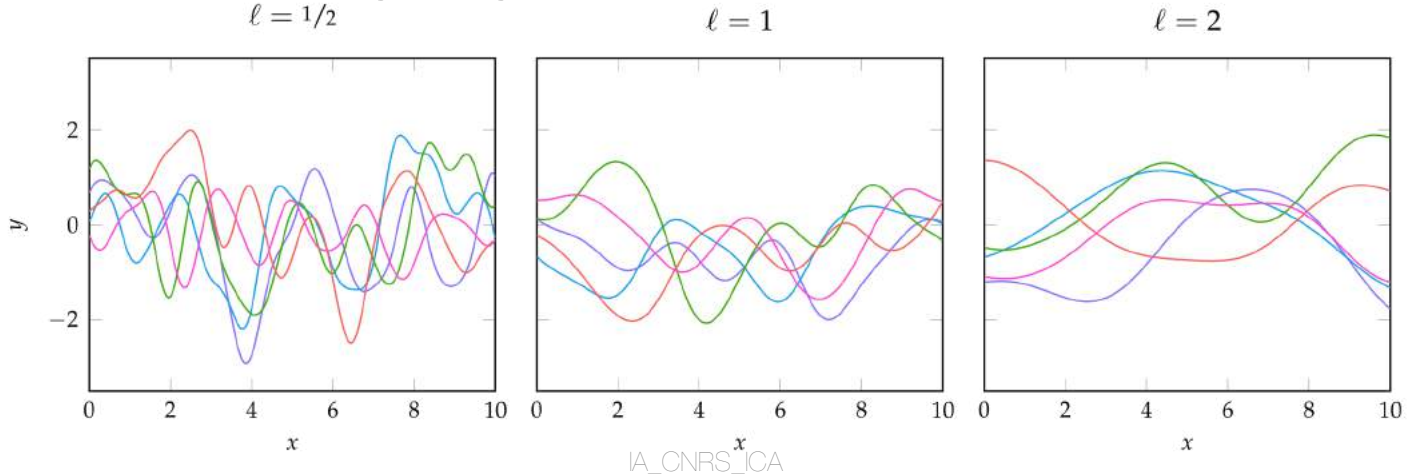


Gaussian Processes

- A common kernel function is the squared exponential kernel

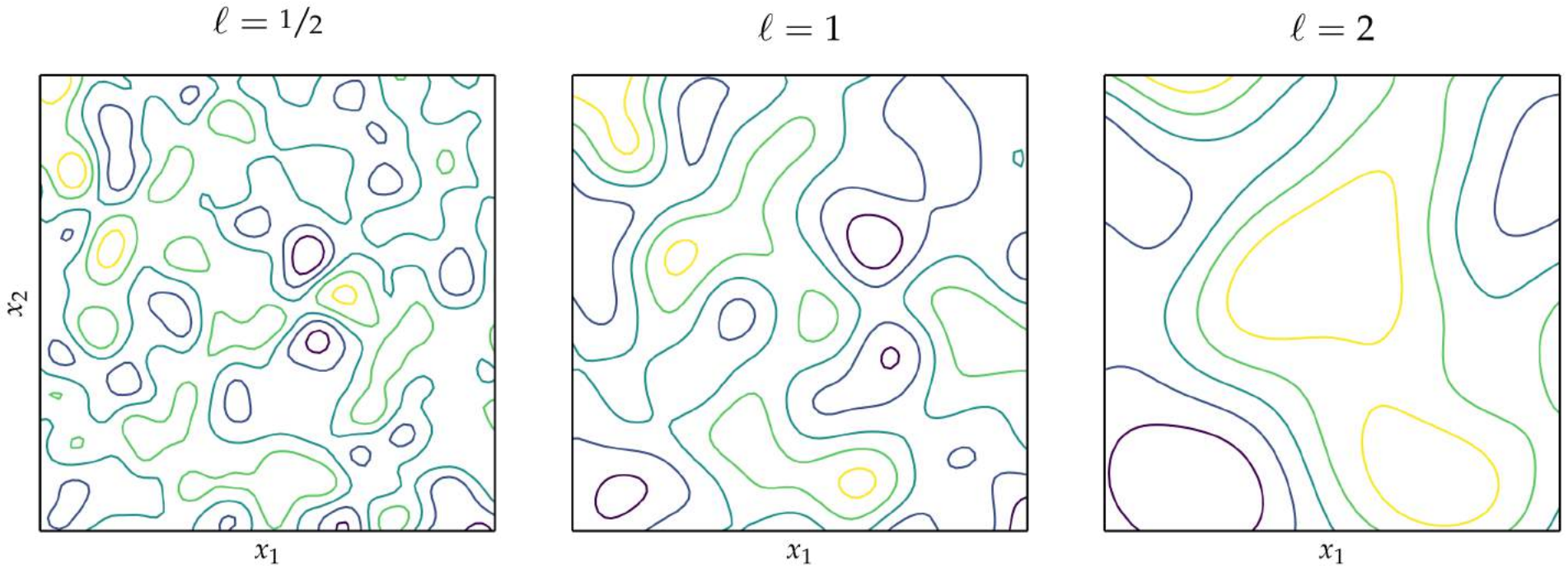
$$k(x, x') = \exp\left(-\frac{(x - x')^2}{2\ell^2}\right)$$

where ℓ is the characteristic length-scale, which is the distance required for the function to change significantly



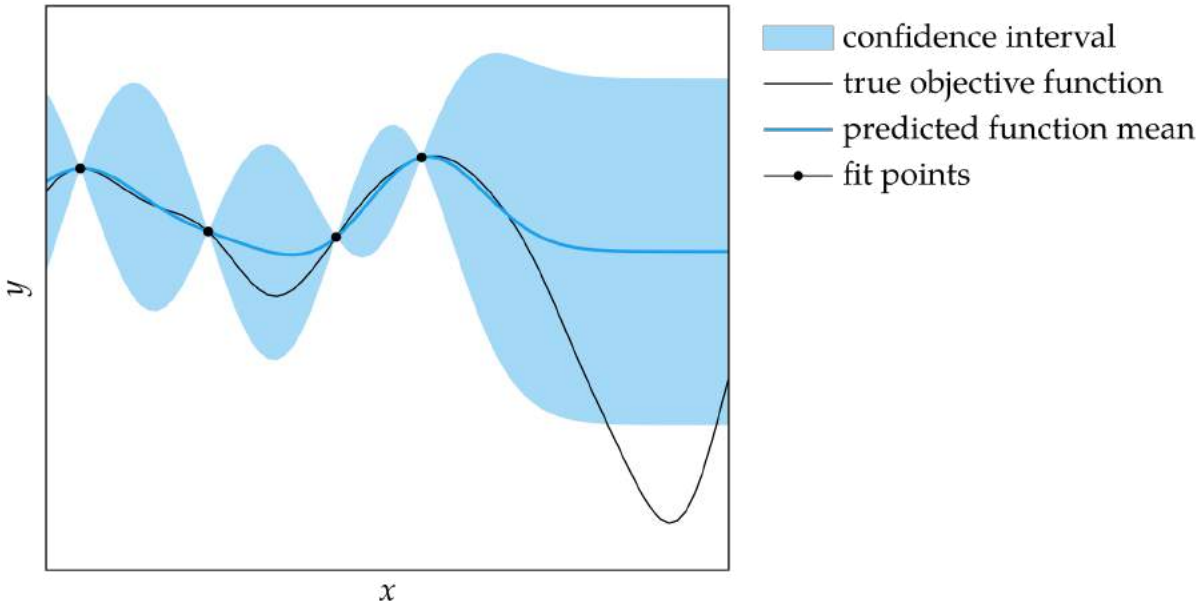
Gaussian Processes

- Example of multivariate Gaussian with squared-exponential kernel at different length scales



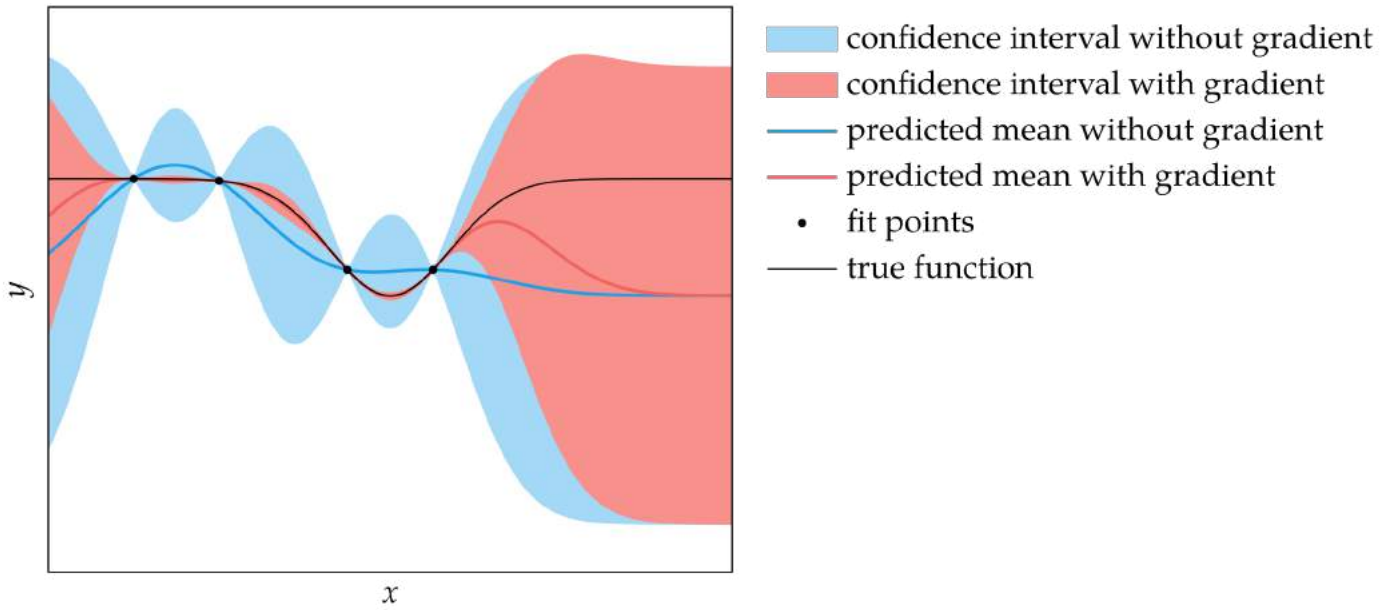
Prediction

- Using variance to compute standard deviation, the predicted mean and standard deviation can be computed at any point
- This enables calculation of the 95% confidence region



Gradient Measurements

- If function gradient evaluations can be made as well, the process can be extended to include gradient predictions for higher prediction fidelity



Summary

- Gaussian processes are probability distributions over functions
- The choice of kernel affects the smoothness of the functions sampled from a Gaussian process
- The multivariate normal distribution has analytic conditional and marginal distributions
- We can compute the mean and standard deviation of our prediction of an objective function at a particular design point given a set of past evaluations

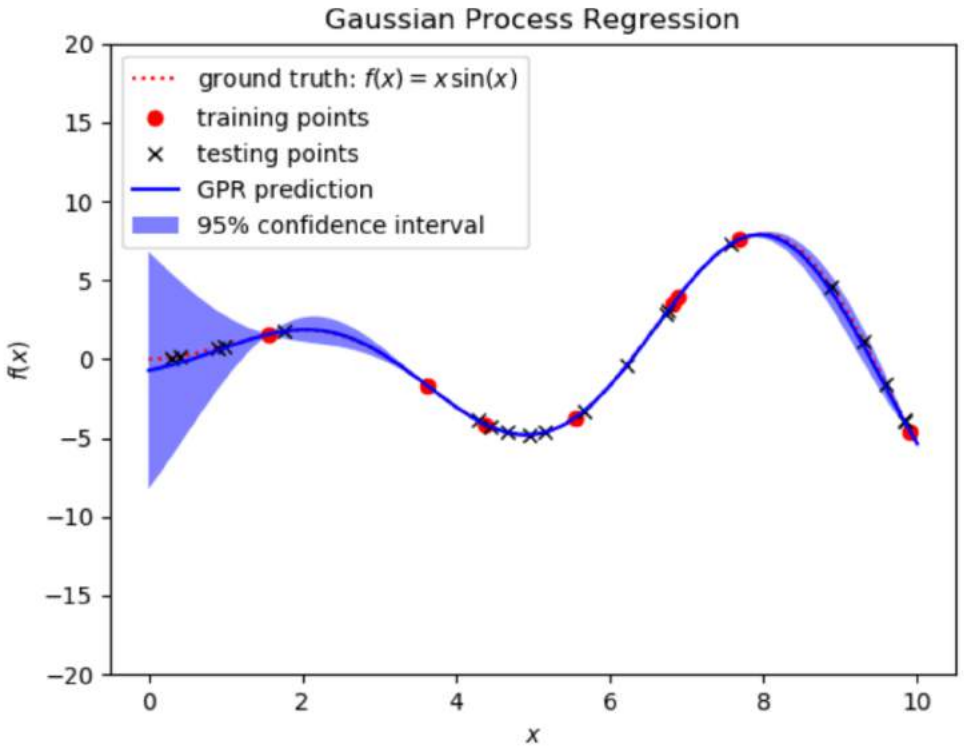
Choosing an ML algorithm

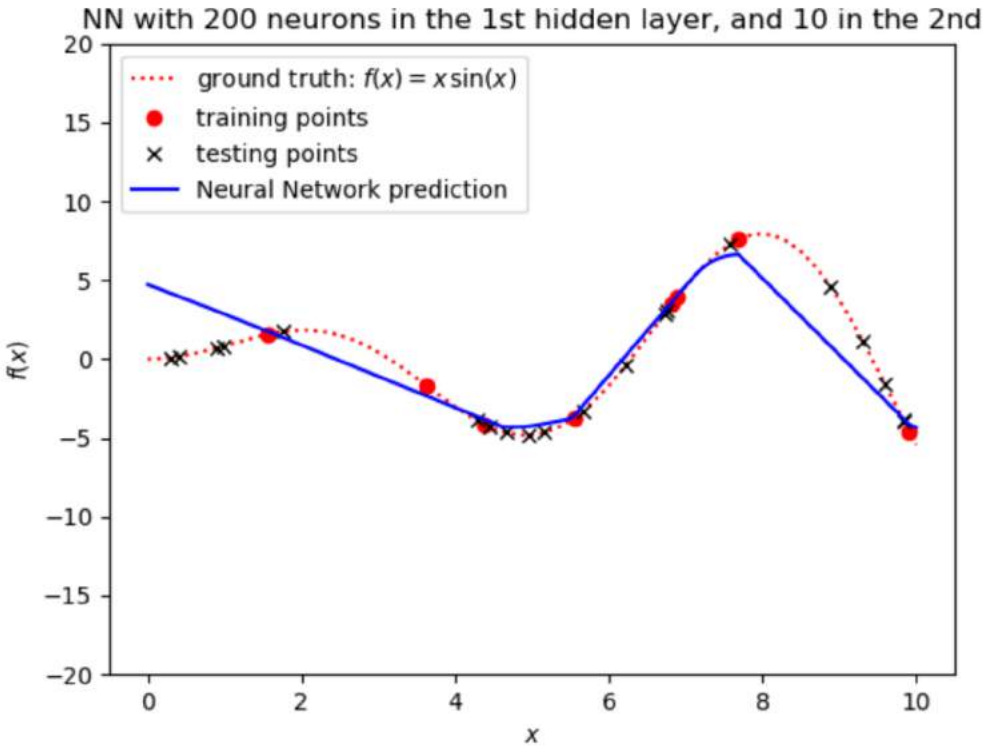
- **Training involves finding this vast amount of weights such that they fit the data accordingly**. Neural networks are very simple algorithms, so they are extremely scalable (they can easily deal with millions of data points).

- Gaussian Processes (GPs): a Bayesian or probabilistic ML algorithm, meaning that it predicts the response as well as the uncertainty. **This algorithm has outstanding regression capabilities and, in general, is easy to train**. Unfortunately, this algorithm is poorly scalable, so it is limited to small datasets (approximately 10,000 points).

GP vs NN

As can be observed, GPs provide an excellent approximation of the function x*sin(x) even when using only 8 points, and they also quantify the uncertainty away from the training data.

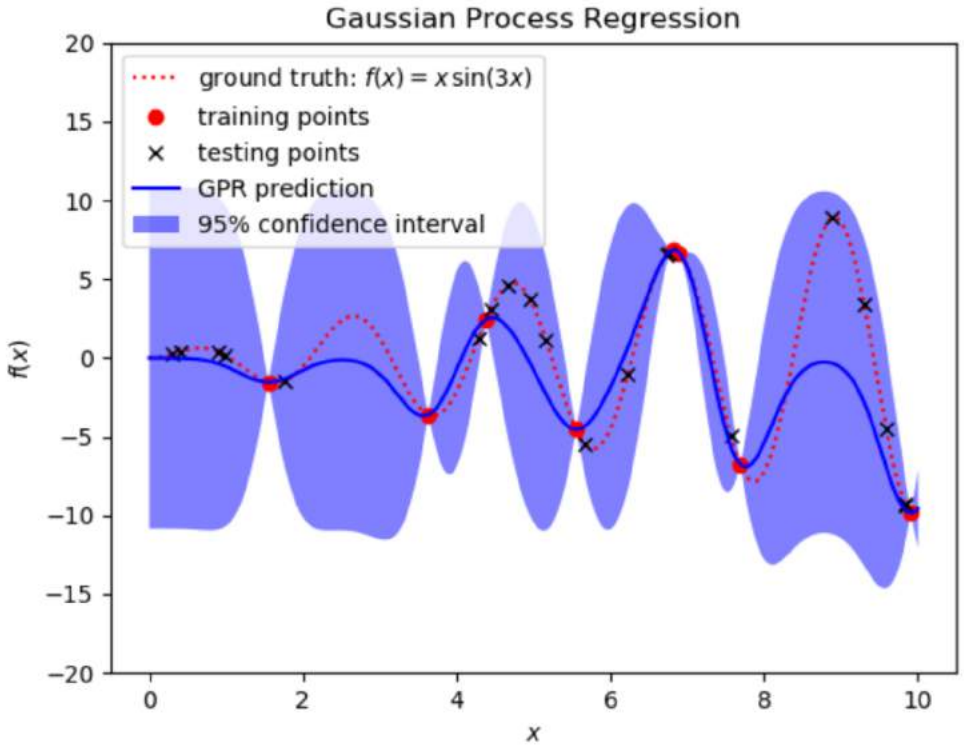


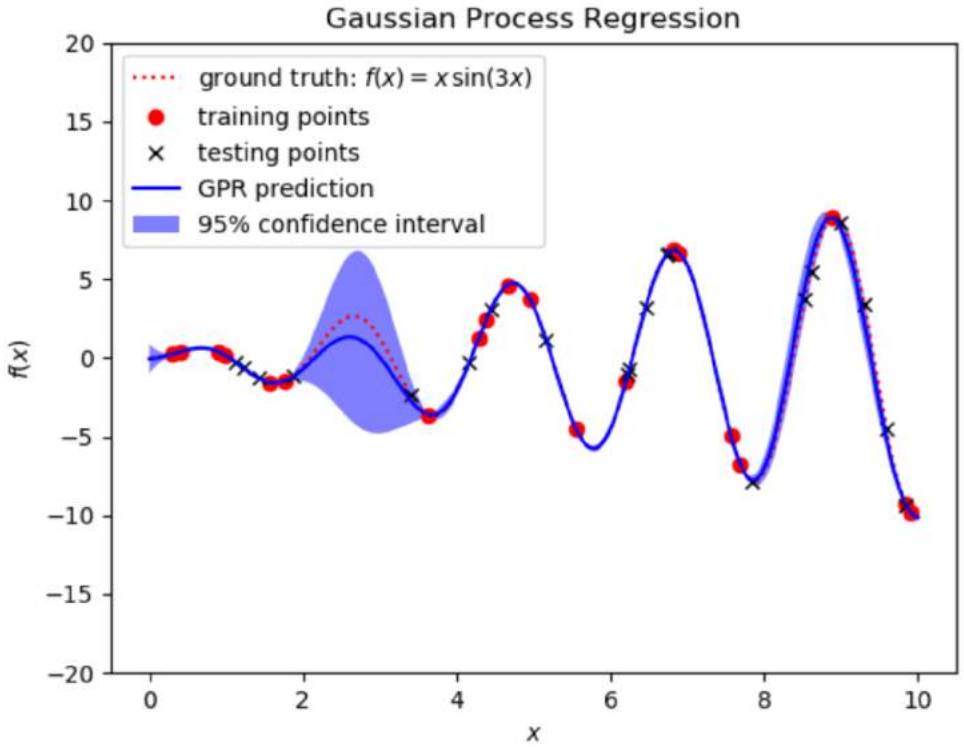


Do we know a priori if there are "enough" training points?

we have enough training data by iteratively assessing the error of our approximation against the test data.

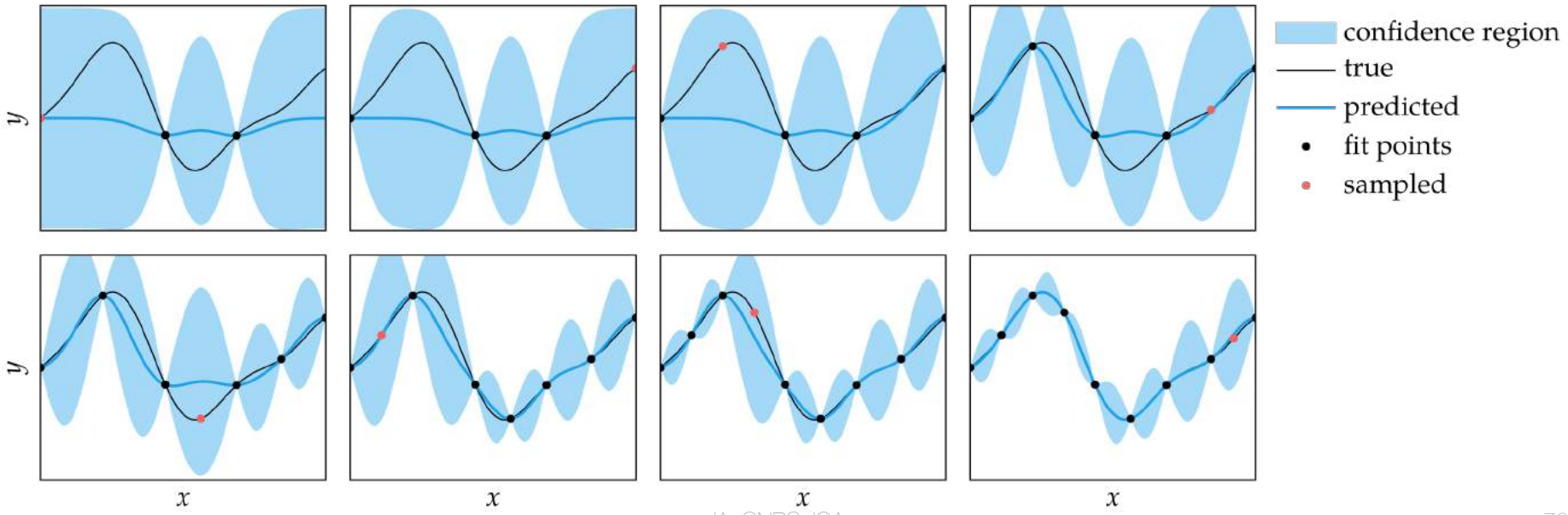
In practice, the training data size may need to be very large when our problem has many input variables due to the curse of dimensionality: every additional input variable causes an exponential growth of the volume that needs to be sampled.





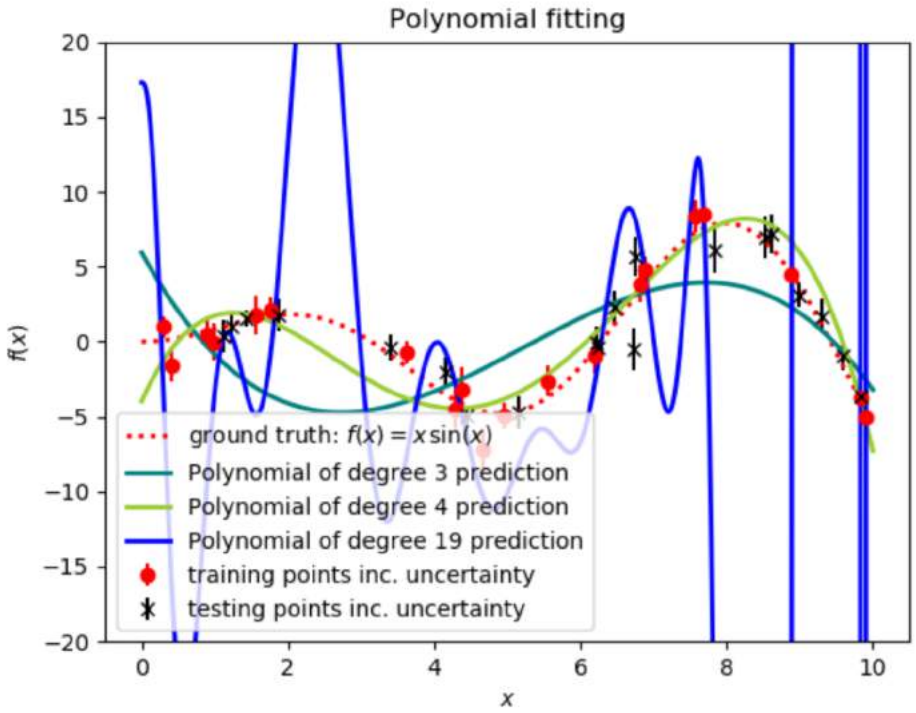
Error-Based Exploration

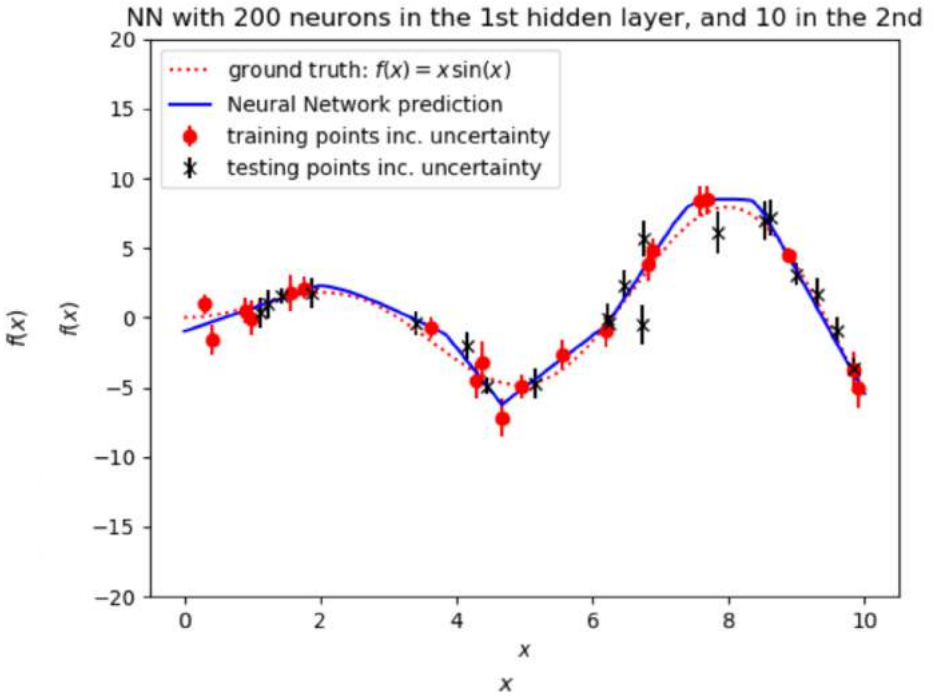
- Focuses exclusively on exploration
- For Gaussian processes, error-based exploration simply minimizes the maximum standard deviation within a specified domain



What happens if my data is noisy or uncertain?

Noisy datasets are very common in Engineering. Data coming from experiments is often dependent on environmental fluctuations, material defects, etc. Note even with simulations : structures undergoing buckling depend on the geometric imperfections considered in the model.



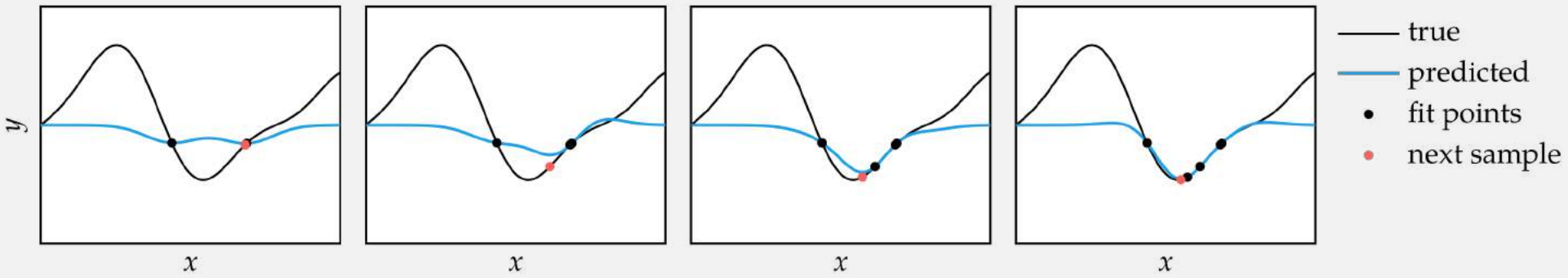


Surrogate Optimization

- Given a surrogate model with both prediction and confidence parameters, an optimization procedure must balance the search for the expected optimal point and decreasing uncertainty
- In other words, the optimization algorithm must balance exploitation with exploration

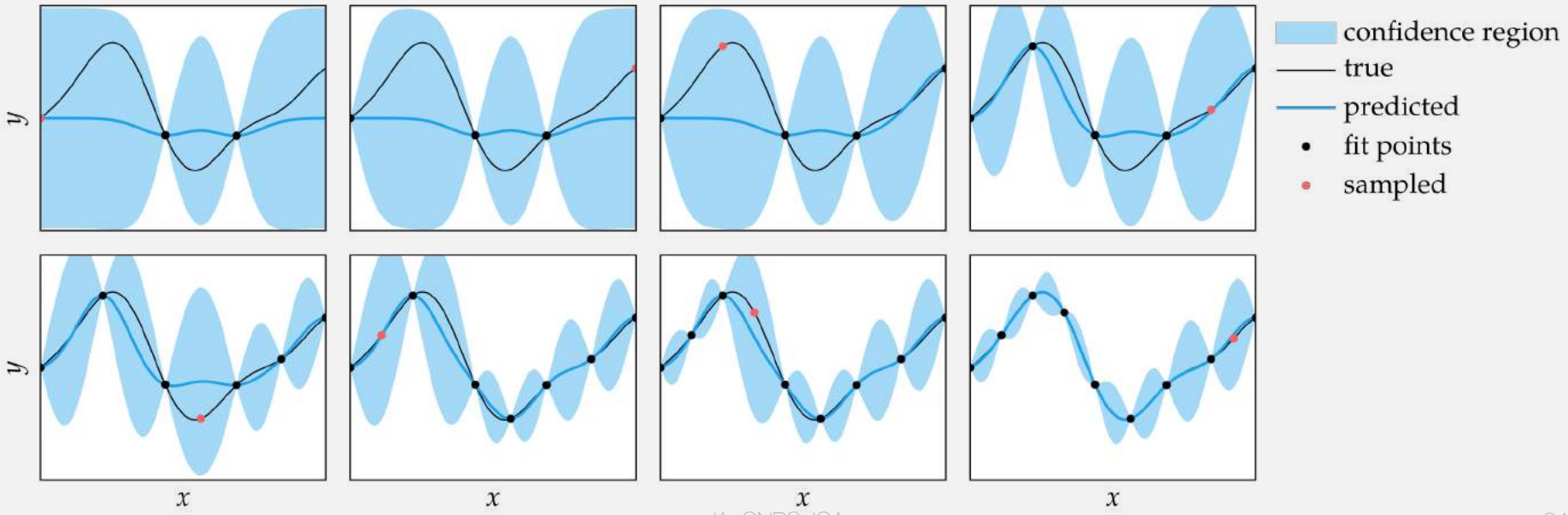
Prediction-Based Exploration

- Focuses exclusively on exploitation, also called greedy approach
- When using a Gaussian process surrogate model, prediction-based exploration simply optimizes over the mean function and ignores uncertainty



Error-Based Exploration

- Focuses exclusively on exploration
- For Gaussian processes, error-based exploration simply minimizes the maximum standard deviation within a specified domain

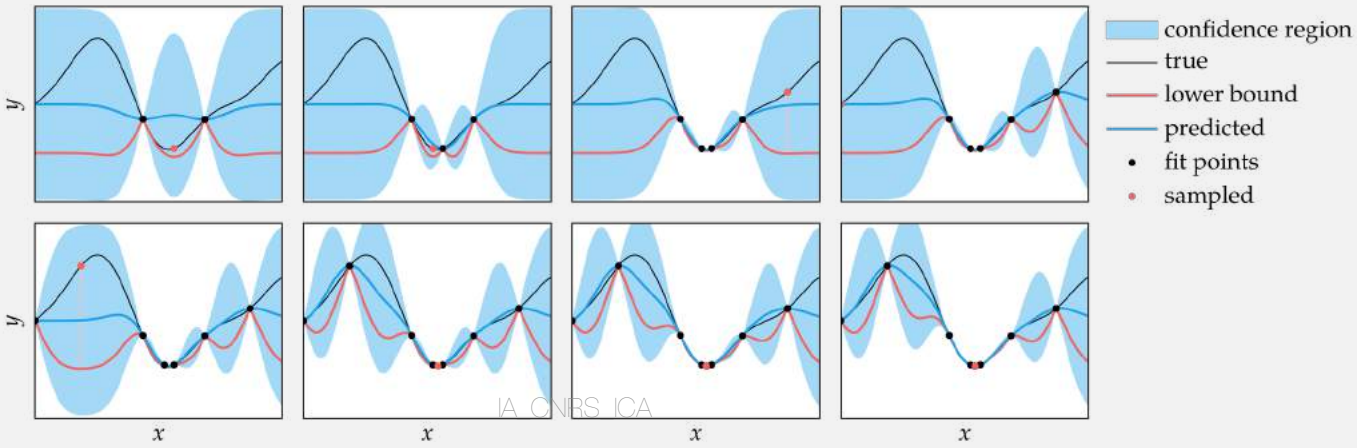


Lower Confidence Bound Exploration

- Tradeoff between exploration and exploitation
- The next sample minimizes the lower confidence bound of the objective function

$$LB(\mathbf{x}) = \hat{\mu}(\mathbf{x}) - \alpha\hat{\sigma}(\mathbf{x})$$

where $\alpha \geq 0$ is the tradeoff parameter



Probability of Improvement Exploration

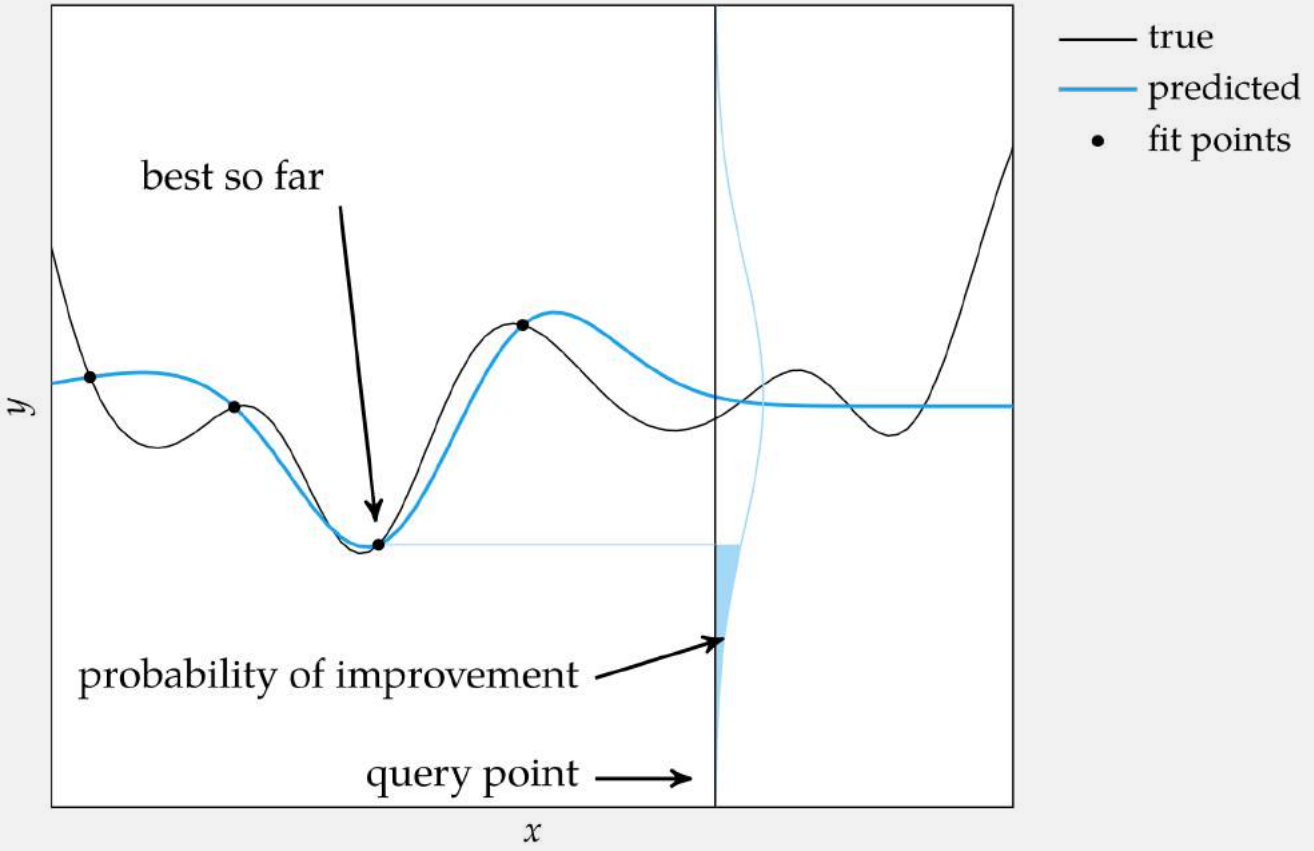
- Searches at the location with the highest probability of improvement
- Improvement is defined as

$$I(y) = \begin{cases} y_{\min} - y & \text{if } y < y_{\min} \\ 0 & \text{otherwise} \end{cases}$$

- The probability of improvement is defined as

$$P(y < y_{\min}) = \int_{-\infty}^{y_{\min}} \mathcal{N}(y \mid \hat{\mu}, \hat{\sigma}) dy$$
$$= \Phi\left(\frac{y_{\min} - \hat{\mu}}{\hat{\sigma}}\right)$$

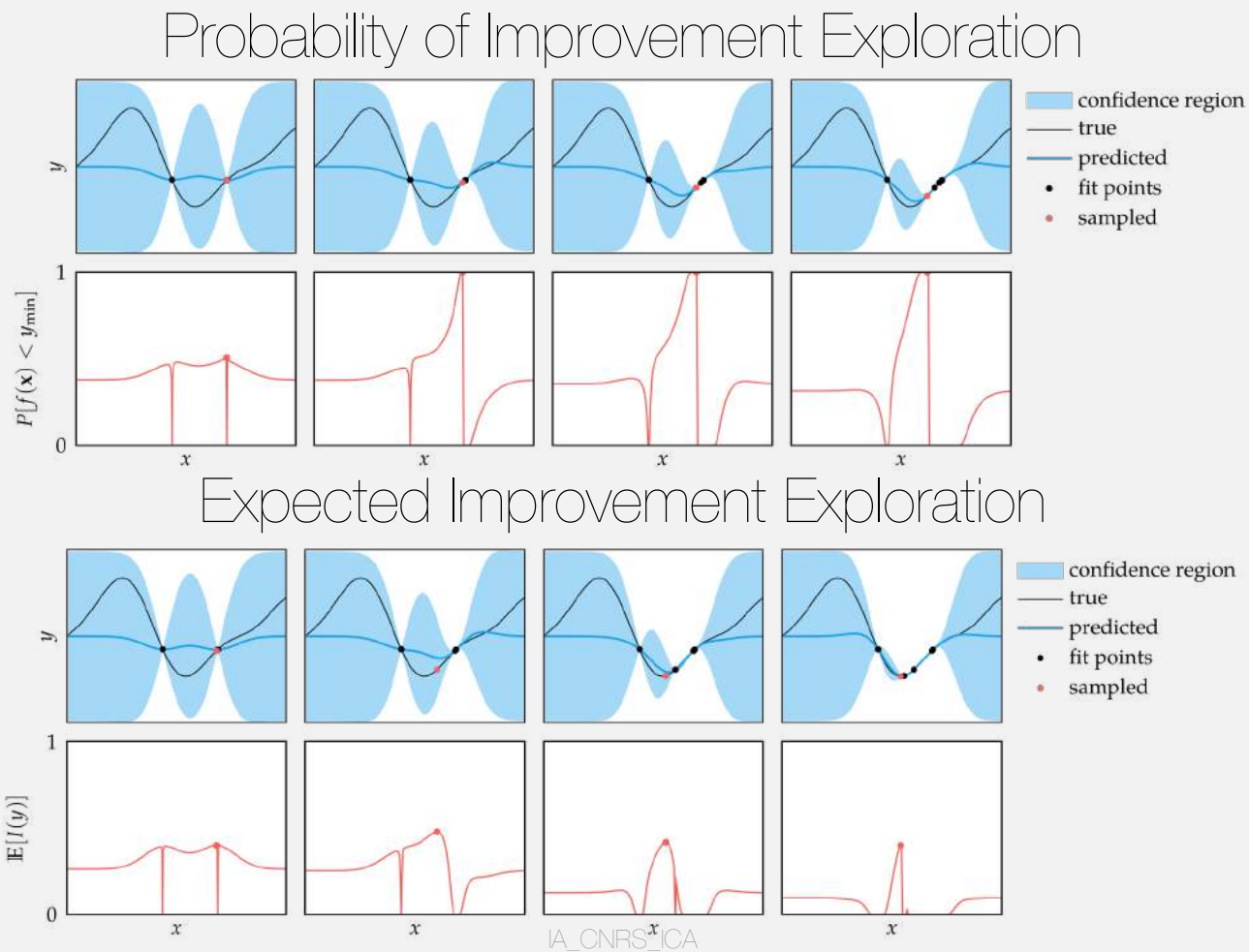
Probability of Improvement Exploration



Expected Improvement Exploration

- While probability of improvement exploration seeks to ensure some expected improvement at each step, expected improvement exploration seeks to maximize expected improvement at each step

Expected Improvement Exploration



Summary

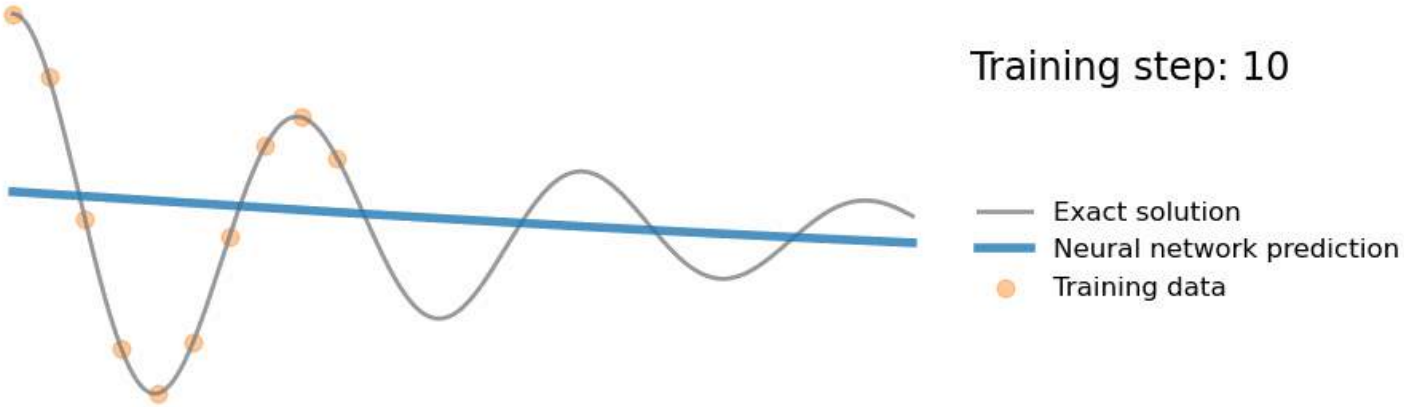
- Gaussian processes can be used to guide the optimization process using a variety of strategies that use estimates of quantities such as the lower confidence bound, probability of improvement, and expected improvement
- Some problems do not allow for the evaluation of unsafe designs, in which case we can use safe exploration strategies that rely on Gaussian processes

Part 1. Régression par processus gaussiens
Part 2. processus gaussiens versus réseaux de neurones

Part 3. Physics Informed Neural Networks {PINN}

PINN Can we embedd physics in NN?

https://github.com/benmoseley/harmonic-oscillator-pinn



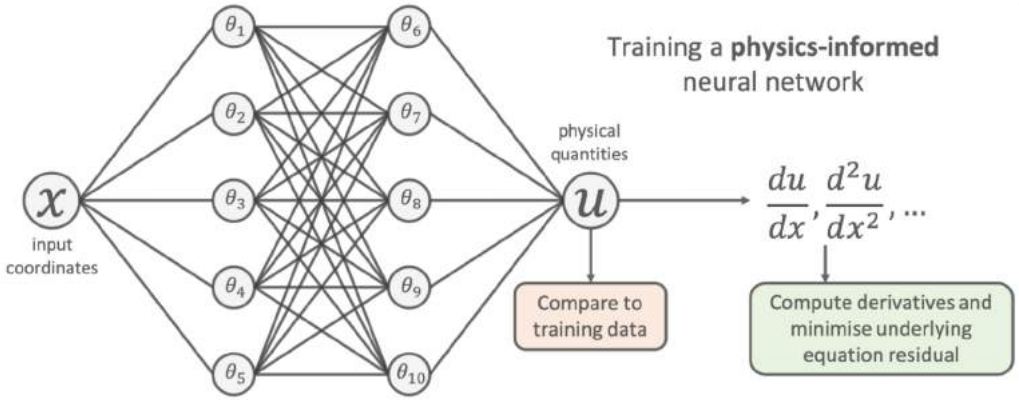
You can see that whilst the neural network accurately models the physical process within the vicinity of the experimental data, it fails to generalise away from this training data. By only relying on the data, one could argue it hasn't truly "understood" the scientific problem.

Scientific machine learning (SciML)

- we know that the underlying physics can be described by the following differential equation:

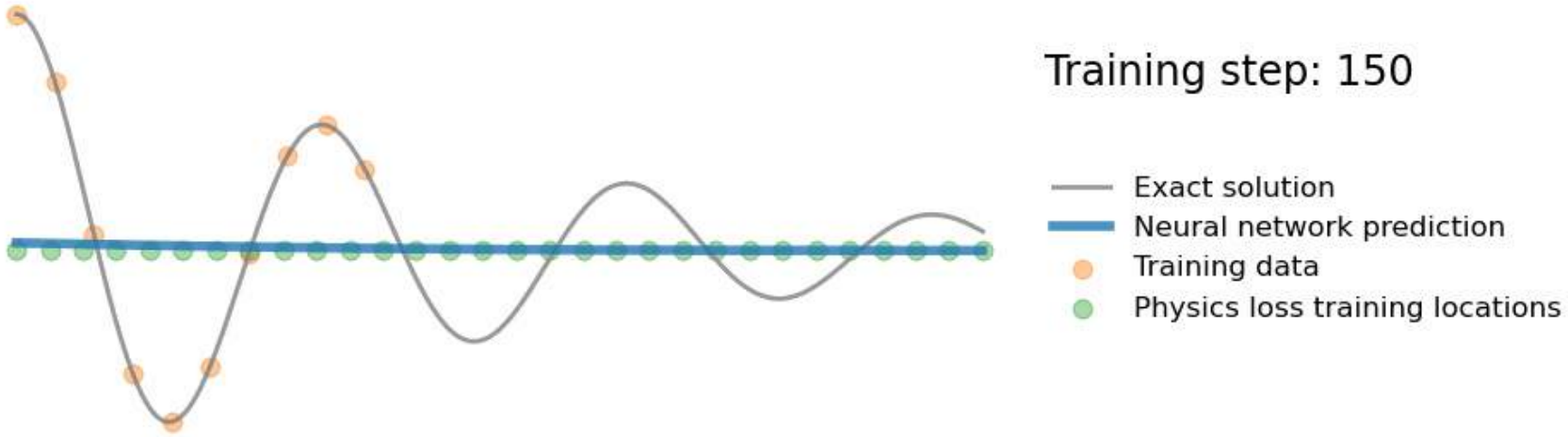
$$m\frac{d^2u}{dx^2} + \mu\frac{du}{dx} + ku = 0$$

The idea is very simple: add the known differential equations directly into the loss function when training the neural network.



$$\min \frac{1}{N}\sum_i^N (u_{\mathrm{NN}}(x_i;\theta) - u_{\mathrm{true}}(x_i))^2 + \frac{1}{M}\sum_j^M \left(\left[m\frac{d^2}{dx^2} + \mu\frac{d}{dx} + k\right] u_{\mathrm{NN}}(x_j;\theta)\right)^2$$

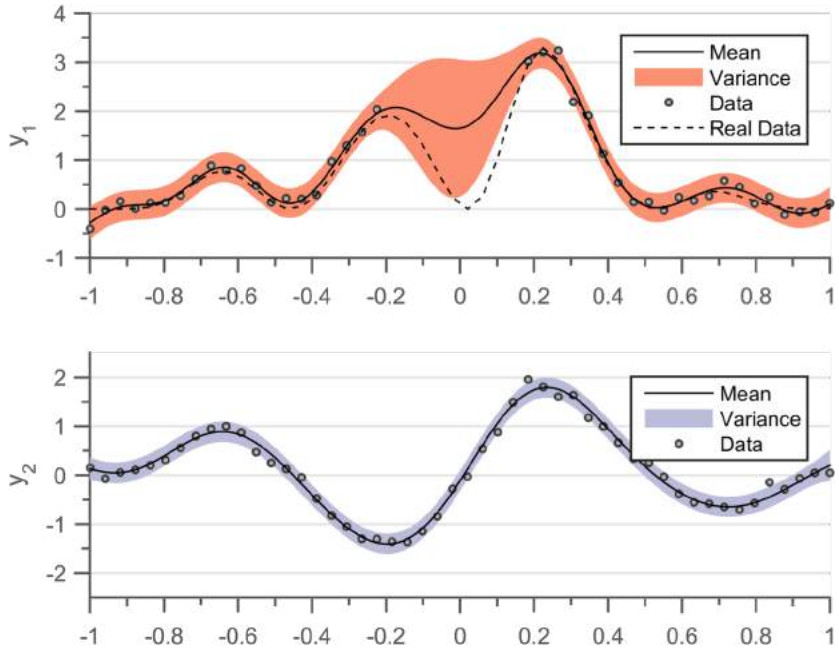
Prior Knowledge



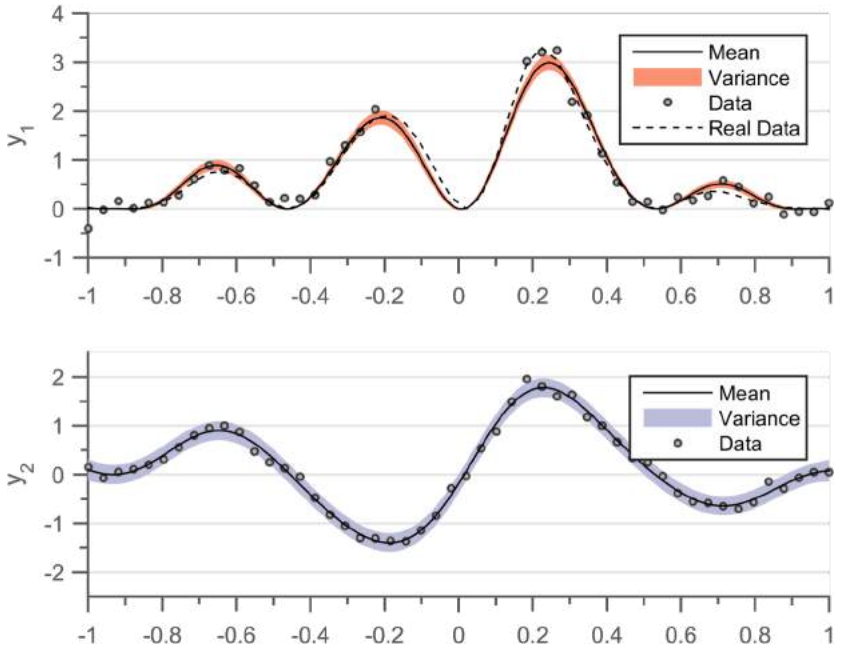
What we did few years ago before PINN and SciML…

https://github.com/ankitchiplunkar/thesis_isae

Faulty sensor



Independent GPs

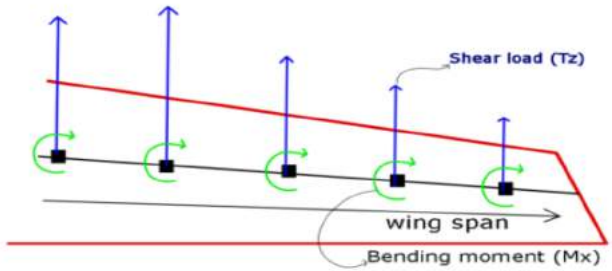


Related GPs

What we did few years ago before PINN and SciML…

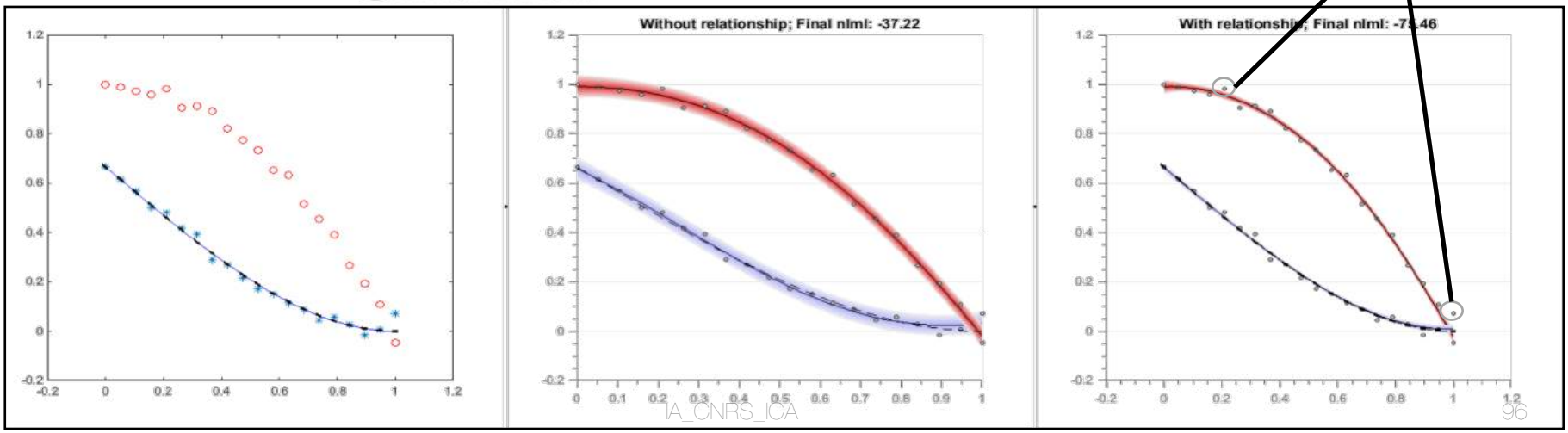
https://github.com/ankitchiplunkar/thesis_isae

Flight test - Relationship between Tz and Mx



$$Mx = \int_{\eta}^{\eta_{edge}} (x - \eta)\, Tz\, dx$$

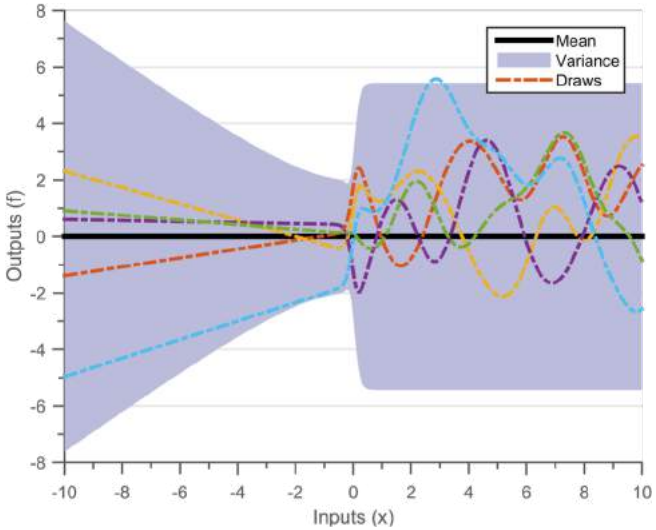
Wrong data point

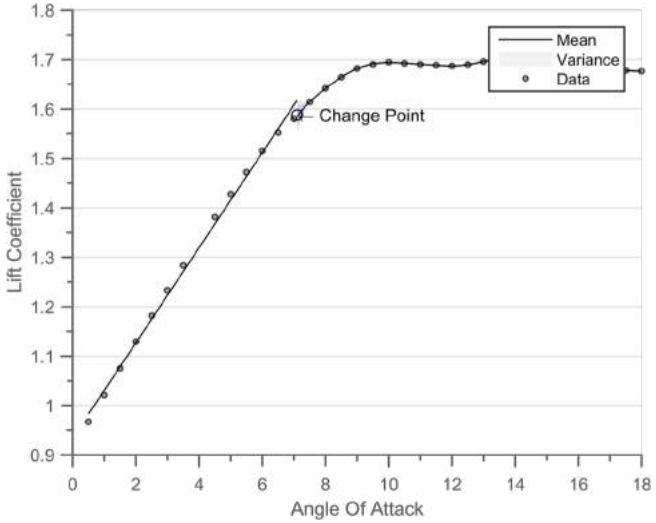


What we did few years ago before PINN and SciML…

https://github.com/ankitchiplunkar/thesis_isae

Identification of nonlinearity





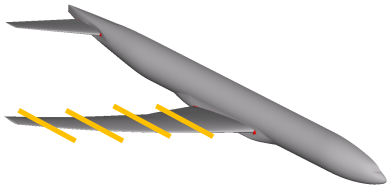
$$k_{CP}(k_1, k_2, x_1, x_2) = sigm(x_1)k_1 sigm(x_2) + (1 - sigm(x_1))k_2(1 - sigm(x_2))$$

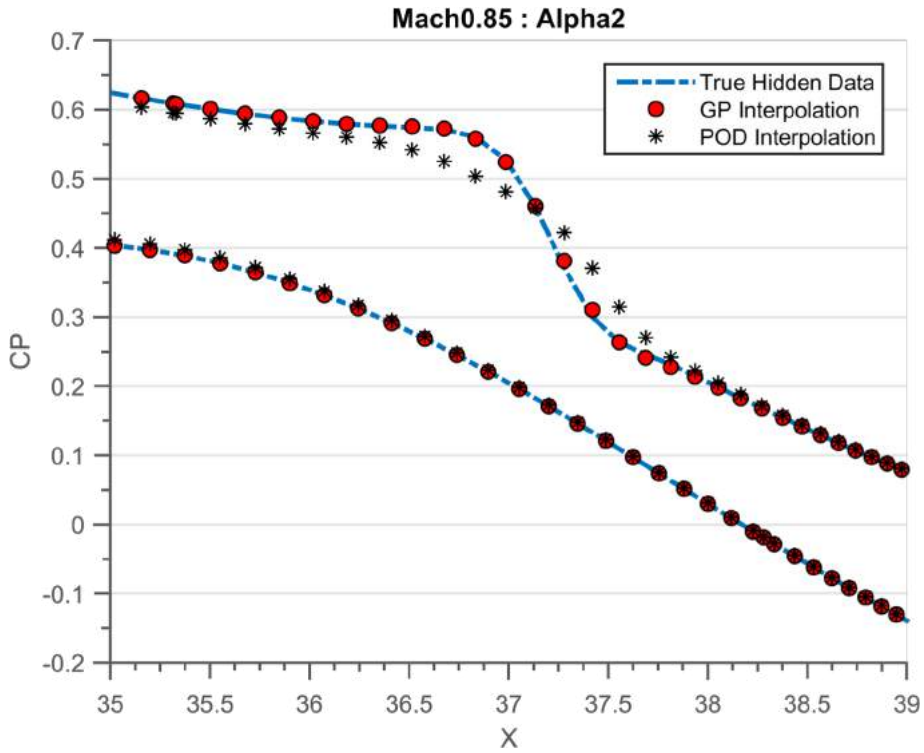
- Estimate change in pattern
- Change-point kernel is unstable
- Use global optimization
- Also possible estimate change-point by calculating the posterior of hyperparameters

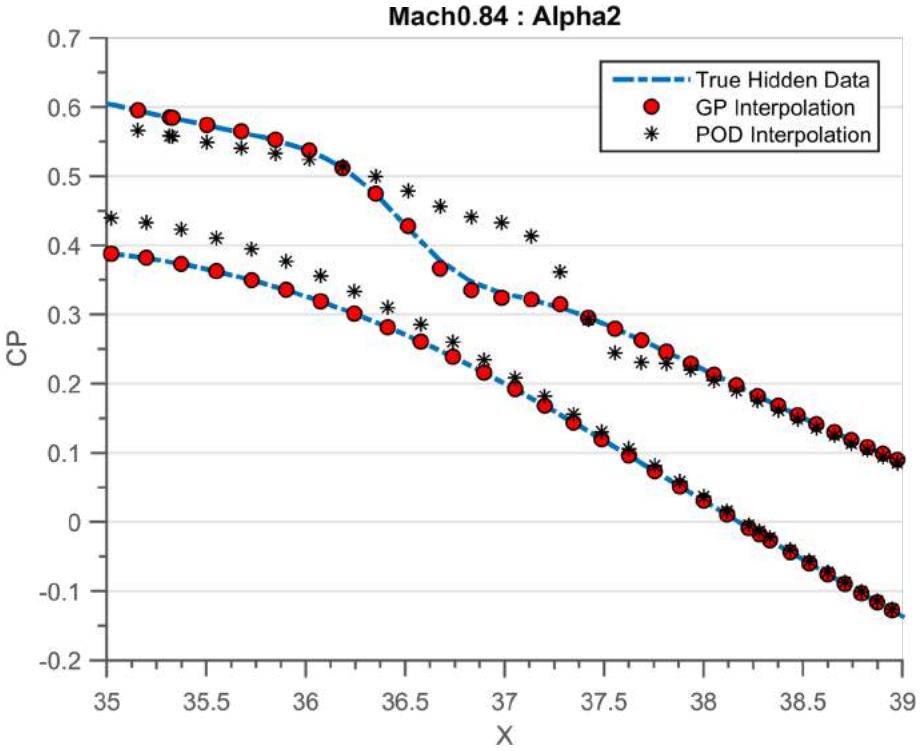
What we did few years ago before PINN and SciML…

https://github.com/ankitchiplunkar/thesis_isae

Comparison of results: Cut1 (y/B = 0,105)







To finish Complementary materials online

CHALLENGE # deeper understanding

https://thegradient.pub/gaussian-process-not-quite-for-dummies/

https://www.cs.toronto.edu/~duvenaud/cookbook/

https://distill.pub/2019/visual-exploration-gaussian-processes/

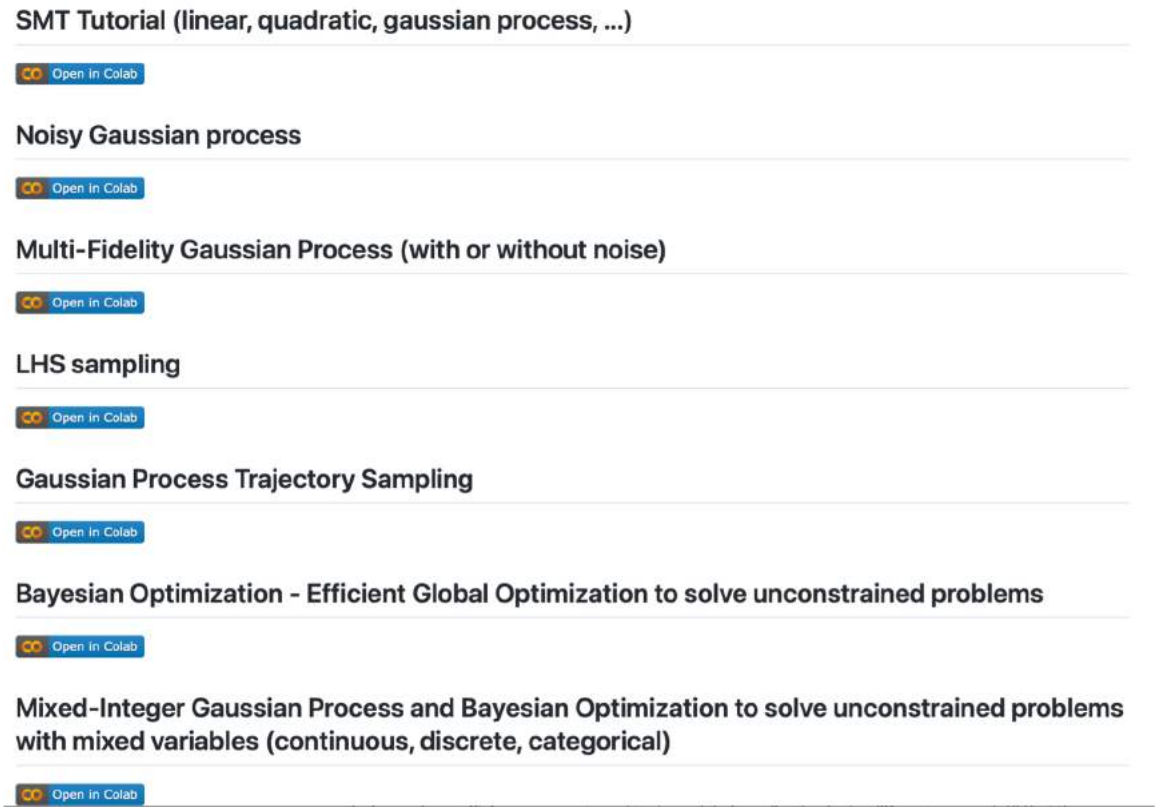
https://sandipanweb.wordpress.com/2020/12/08/gaussian-process-regression-with-python/

To conclude

1. If you are interested in **AI4E** please join us on linkedin.

2. Have a look to SMT tutorials: https://github.com/SMTorg/smt/tree/master/tutorial#readme

3. **Contact: joseph.morlier@isae-supaero.fr**



Think JULIA

https://sinews.siam.org/Details-Page/scientific-machine-learning-how-julia-employs-differentiable-programming-to-do-it-best

Automatic Differentiation

- Evaluate a function and compute partial derivatives simultaneously using the chain rule of differentiation

$$\frac{d}{dx} f(g(x)) = \frac{d}{dx}(f \circ g)(x) = \frac{df}{dg}\frac{dg}{dx}$$

AD . . . is Computer Sciences

A program is composed of elementary operations like addition, subtraction, multiplication, and division.

Consider the function $f(a,b) = \ln(ab + \max(a,2))$. If we want to compute the partial derivative with respect to a at a point, we need to apply the chain rule several times:[9]

$$
\begin{aligned}
\frac{\partial f}{\partial a} &= \frac{\partial}{\partial a} \ln(ab + \max(a,2)) \\
&= \frac{1}{ab + \max(a,2)} \frac{\partial}{\partial a}(ab + \max(a,2)) \\
&= \frac{1}{ab + \max(a,2)} \left[\frac{\partial (ab)}{\partial a} + \frac{\partial \max(a,2)}{\partial a} \right] \\
&= \frac{1}{ab + \max(a,2)} \left[\left(b\frac{\partial a}{\partial a} + a\frac{\partial b}{\partial a} \right) + \left((2 > a)\frac{\partial 2}{\partial a} + (2 < a)\frac{\partial a}{\partial a} \right) \right] \\
&= \frac{1}{ab + \max(a,2)} [b + (2 < a)]
\end{aligned}
$$

One example

- Forward Accumulation is equivalent to expanding a function using the chain rule and computing the derivatives inside-out
- Requires n-passes to compute n-dimensional gradient

$$f(a,b) = \ln(ab + \max(a, 2))$$

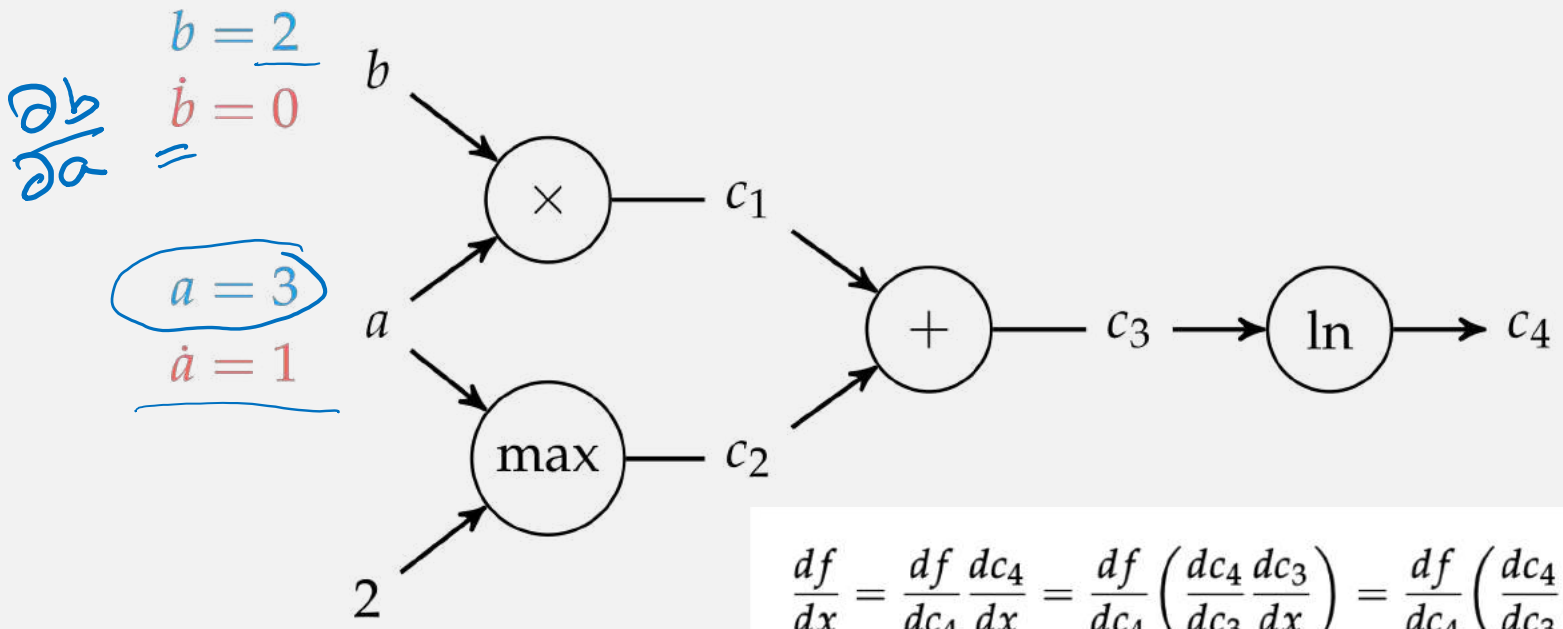
$$\frac{\partial f}{\partial a}(3,2)$$

AD computational graphs

$$\frac{\partial f(3,2)}{\partial a}$$

- Forward Accumulation

$$f(a,b) = \ln(ab + \max(a, 2))$$

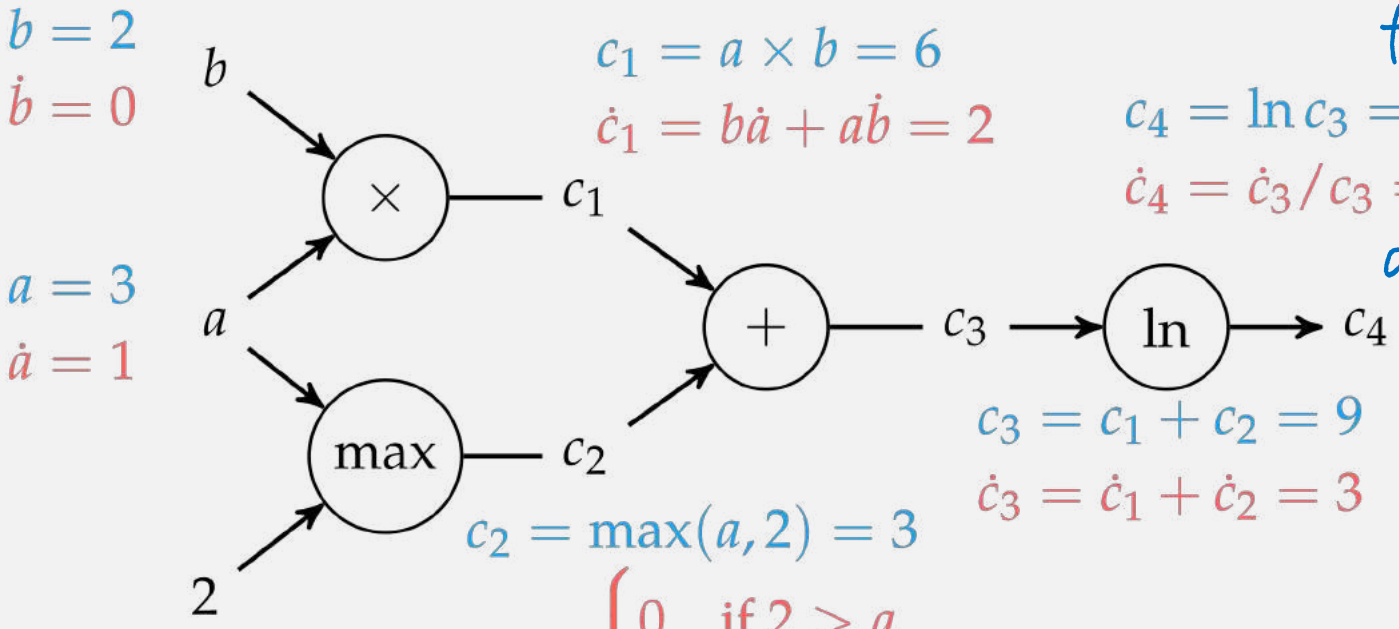


$$\frac{df}{dx} = \frac{df}{dc_4}\frac{dc_4}{dx} = \frac{df}{dc_4}\left(\frac{dc_4}{dc_3}\frac{dc_3}{dx}\right) = \frac{df}{dc_4}\left(\frac{dc_4}{dc_3}\left(\frac{dc_3}{dc_2}\frac{dc_2}{dx} + \frac{dc_3}{dc_1}\frac{dc_1}{dx}\right)\right)$$

Automatic Differentiation

- Forward Accumulation

$$f(a,b) = \ln(ab + \max(a, 2))$$



In Julia

The `ForwardDiff.jl` package supports an extensive set of mathematical operations and additionally provides gradients and Hessians.

```
julia> using ForwardDiff
julia> a = ForwardDiff.Dual(3,1);
julia> b = ForwardDiff.Dual(2,0);
julia> log(a*b + max(a,2))
Dual{Nothing}(2.1972245773362196,0.3333333333333333)
```

AD Reverse

Forward accumulation requires n passes in order to compute an n-dimensional gradient. *Reverse accumulation*[11] requires only a single run in order to compute a complete gradient but requires two passes through the graph: a *forward pass* during which necessary intermediate values are computed and a *backward pass* which computes the gradient. Reverse accumulation is often preferred over forward accumulation when gradients are needed, though care must be taken on memory-constrained systems when the computational graph is very large.[12]

Like forward accumulation, reverse accumulation will compute the partial derivative with respect to the chosen target variable but iteratively substitutes the outer function instead:

$$\frac{df}{dx} = \frac{df}{dc_4}\frac{dc_4}{dx} = \left(\frac{df}{dc_3}\frac{dc_3}{dc_4}\right)\frac{dc_4}{dx} = \left(\left(\frac{df}{dc_2}\frac{dc_2}{dc_3} + \frac{df}{dc_1}\frac{dc_1}{dc_3}\right)\frac{dc_3}{dc_4}\right)\frac{dc_4}{dx}$$

Reverse

$$\frac{df}{dx} = \frac{df}{dc_4}\frac{dc_4}{dx} = \frac{df}{dc_4}\left(\frac{dc_4}{dc_3}\frac{dc_3}{dx}\right) = \frac{df}{dc_4}\left(\frac{dc_4}{dc_3}\left(\frac{dc_3}{dc_2}\frac{dc_2}{dx} + \frac{dc_3}{dc_1}\frac{dc_1}{dx}\right)\right)$$

Forward

[11] S. Linnainmaa, "The Representation of the Cumulative Rounding Error of an Algorithm as a Taylor Expansion of the Local Rounding Errors," Master's thesis, University of Helsinki, 1970.

[12] Reverse accumulation is central to the backpropagation algorithm used to train neural networks. D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning Representations by Back-Propagating Errors," *Nature*, vol. 323, pp. 533–536, 1986.

In Julia

The `Zygote.jl` package provides automatic differentiation in the form of reverse-accumulation. Here the `gradient` function is used to automatically generate the backwards pass through the source code of `f` to obtain the gradient.

```
julia> import Zygote: gradient
julia> f(a, b) = log(a*b + max(a,2));
julia> gradient(f, 3.0, 2.0)
(0.3333333333333333, 0.3333333333333333)
```

Summary

Monolithic
Black boxes input and outputs

Algorithmic differentiation
Lines of code code variables

When using the forward mode, for each intermediate variable in the algorithm, a variation due to one input variable is carried through. This is very similar to the way the complex-step method works. To illustrate this, suppose we want to differentiate the multiplication operation, $f = x_1 x_2$, with respect to x_1. Table 4.4 compares how the differentiation would be performed using either automatic differentiation or the complex-step method.

Automatic	Complex-Step
$\Delta x_1 = 1$	$h_1 = 10^{-20}$
$\Delta x_2 = 0$	$h_2 = 0$
$f = x_1 x_2$	$f = (x_1 + ih_1)(x_2 + ih_2)$
$\Delta f = x_1 \Delta x_2 + x_2 \Delta x_1$	$f = x_1 x_2 - h_1 h_2 + i(x_1 h_2 + x_2 h_1)$
$\mathrm{d}f/\mathrm{d}x_1 = \Delta f$	$\mathrm{d}f/\mathrm{d}x_1 = \mathrm{Im}\, f/h$

- Complex step method can eliminate the effect of subtractive cancellation error when taking small steps
- AD: Reverse accumulation is performed in single run using two passes over an n-dimensional function (forward and back)
- Note: this is central to the backpropagation algorithm used to train neural networks

Link with NN

- To clarify, "**reverse** mode" AD is efficient when you have a functions f(x) with **small number of outputs fi and many inputs xj** (in computing ∂fi/∂xj), i.e. for functions mapping x∈Rm to f∈Rn with **n≪m** .

→(For example, in neural-network training where you want the derivative of one loss function (n=1) with respect to millions (m) of network parameters. (The "manual" application of such a technique is also known as an adjoint method 19, and in the neural-net case it is called backpropagation.)

- In contrast, **forward**-mode AD (as in ForwardDiff.jl) is better when there is a small number of inputs and a *large* number of outputs, i.e. **when n≫m** , i.e. when you are computing *many* functions of a *few* variables.
- Thus, ReverseDiff is generally a better choice for gradients, but Jacobians and Hessians are trickier to determine.