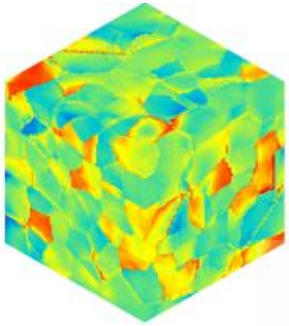
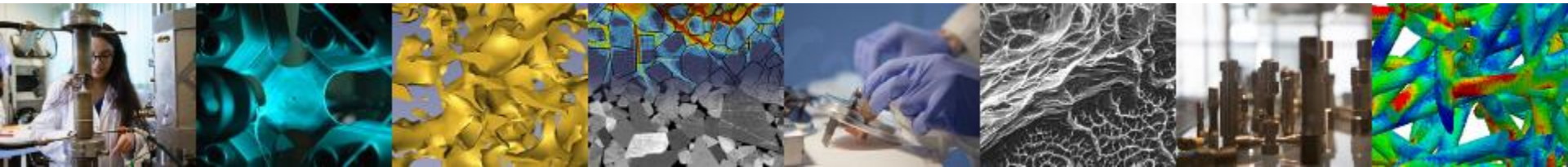


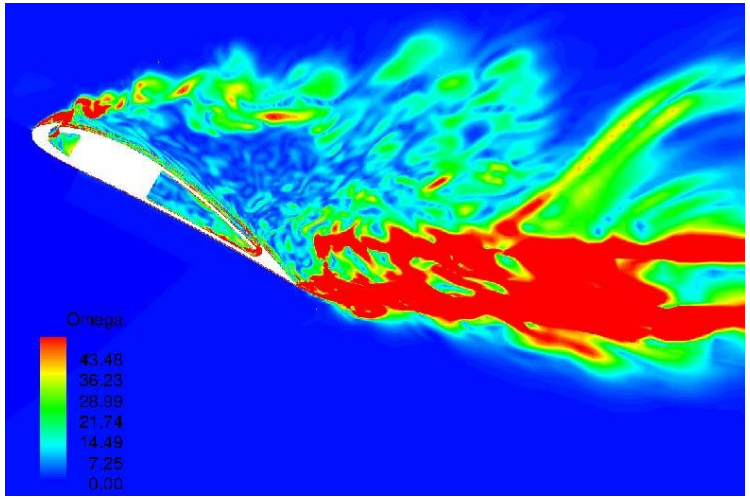
Introduction to Machine Learning for Surrogate Modelling

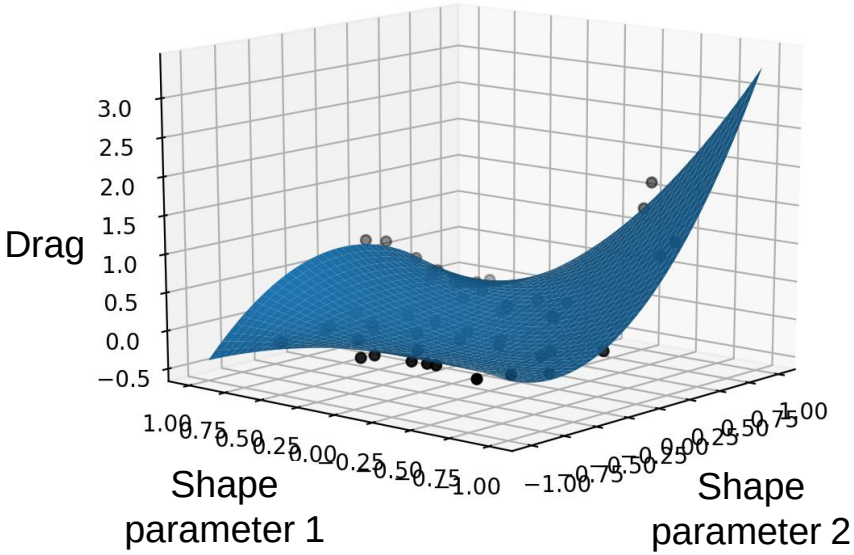
Pierre Kerfriden

Centre des Matériaux, Mines Paris - PSL University, France



Surrogate modelling



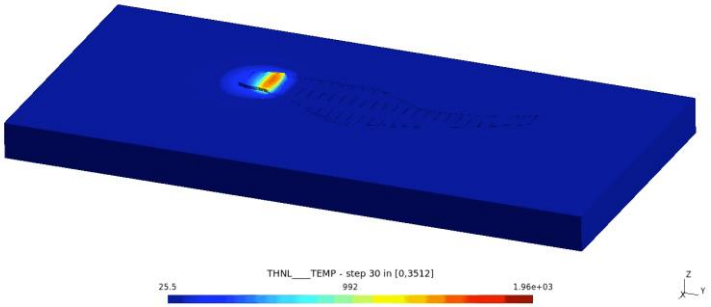


Surrogate/meta modelling used for tasks that necessitates repeated calls to an expensive numerical simulator, subject to parameter variations, *e.g.*(i) system optimisation (ii) calibration & inverse problems (iiii) sensitivity analysis and uncertainty propagation

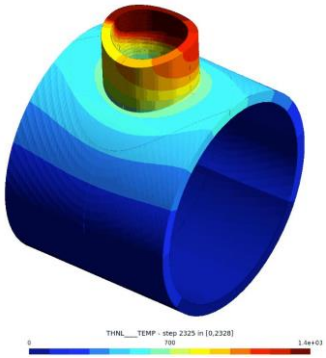
Meta-modelling strategies considers the numerical simulator as a black-box. They (A) evaluate the relationship between model parameters and model outputs at sample points. Then (B) regression models from machine learning are used to interpolate model outputs between samples. Finally (C), the aforementioned tasks may be performed using the inexpensive regression model as a surrogate.

Virtual process control (1/2)

- WAAM manufacturing process: use welding robots in a layer-by-layer fashion for metal 3D printing
  - to build large 3D parts



  - Add new 3D features to existing parts



PhD thesis of S. Hilal (EDF R&D / Mines Paris)

- Anticipated role of simulations
  - Virtual process optimisation
  - Virtual feasibility studies (deformations/stresses in existing parts)

Virtual process control (2/2)

- **Model calibration** requires running simulation hundreds of times

Monte Carlo sampling

$a \sim \mathcal{U}(3.0,18.0)$

$f_r \sim \mathcal{U}(1.10,1.7)$

$c \sim \mathcal{U}(1.0,10.0)$

$\eta \sim \mathcal{U}(0.50,0.95)$

$\varepsilon_0 \sim \mathcal{U}(0.2,0.8)$

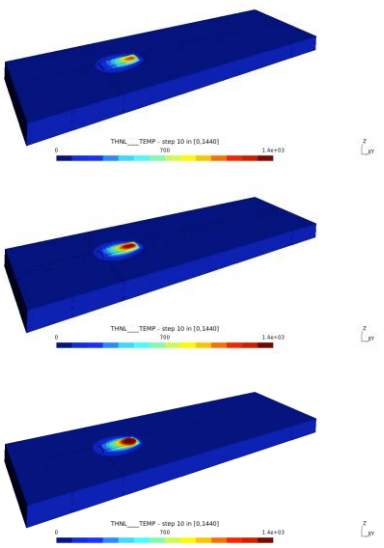
$\varepsilon_1 \sim \mathcal{U}(0.0,1.0)$

$h_0 \sim \mathcal{U}(0.0e{-}6,30.0e{-}6)$

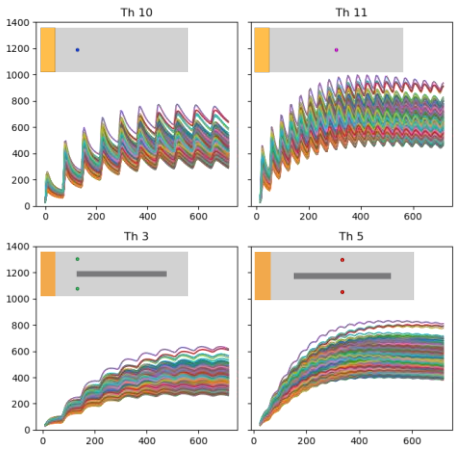
$h_1 \sim \mathcal{U}(0.0,1.0)$

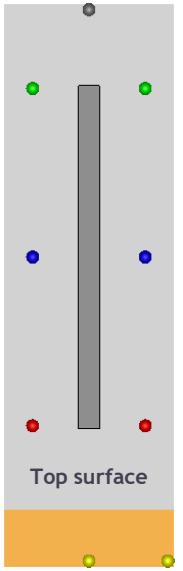
$c_{loss} \sim \mathcal{U}(0.0e{-}6,300.0e{-}6)$

4 thermal simulations
20 beads



Simulations post-processing (400 realisations)





1 thermo mechanical simulation = 100 CPU hours

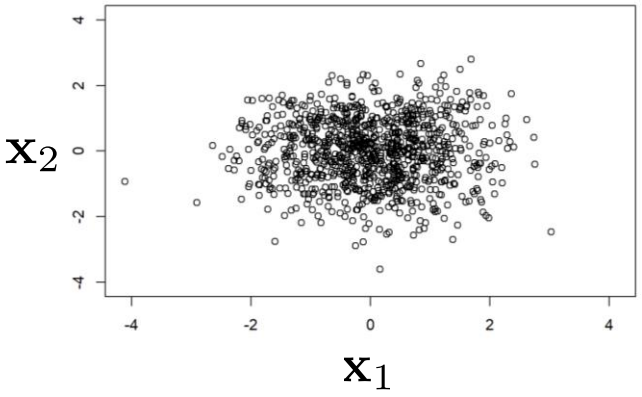
- **Virtual optimisation** runs calibrated model over hundreds of configurations

<u>Digital twinning</u> strategies go one step further and perform calibrations repeatedly, online, together with uncertainty propagation &/or optimisation

UQ and sensitivity analysis

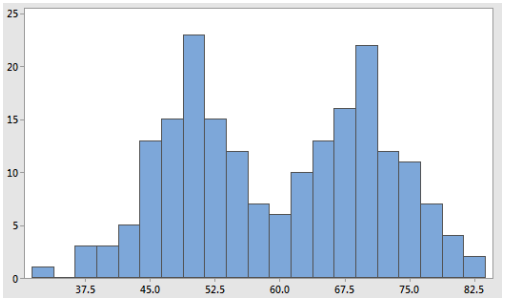
- Uncertainty propagation (+ probabilistic inverse problems & optimisation under uncertainty)

Uncertain parameter distribution

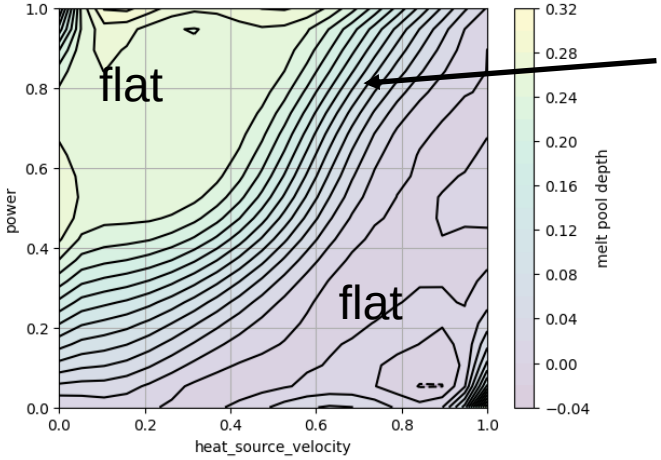


simulator

Quantity of interest



- Sensitivity analysis



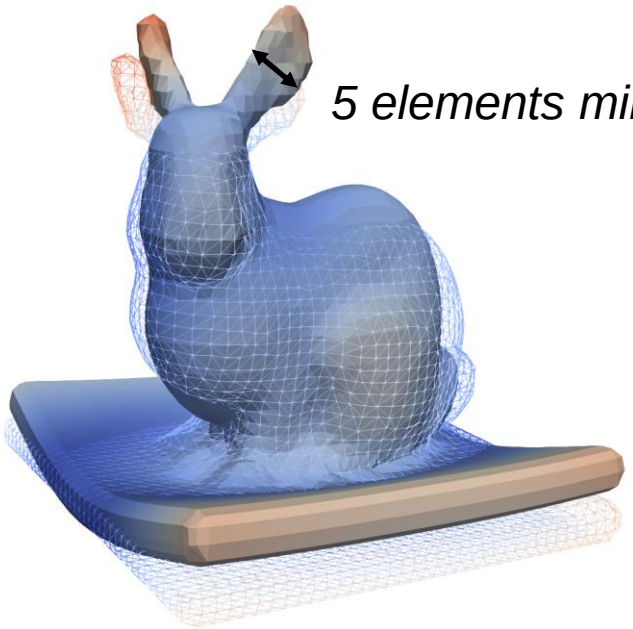
Active linear subspace

Solving PDE-based models

- Solution of linear systems of equations from FEM / FD / FV : o(N) operations for ideal/optimised solvers and up to o(N^2).
  - → At minima, cost increases 1/(spatial resolution)^3 in 3D and 1/res^4 in 4D
  - → Orders of magnitude : 1M DOFs FEM system: 16 GB, 10s - 1mn for steady-state heat diffusion $-\Delta u = f$ + BC

- Computer architectures
  - Boosted desktop with 64 + GB RAM
  - Memory-distributed HPC

- Software
  - Commercial: expensive (several k€ p.a.)
  - Academic: need specialist knowledge (MSc/PhD level) for installation AND use



Current (exotic) trend: solving PDEs with TensorFlow / PyTorch

https://tinyurl.com/3u7mcawr

Outlines

1. Designs of experiments (DoE)

2. Polynomial surrogate modelling
   a. Tensor-product polynomial approximation spaces
   b. Sparse grids interpolation
   c. Sparse polynomial regression

3. Control of (polynomial) surrogate models
   a. Regularisation methods
   b. Model selection

4. Non-parametric surrogate modelling, Gaussian Processes

5. Beyond response surface methodologies using Deep Learning

6. Physics-Informed Neural Nets : surrogate models without numerical solver

Meta-Modelling / DoE

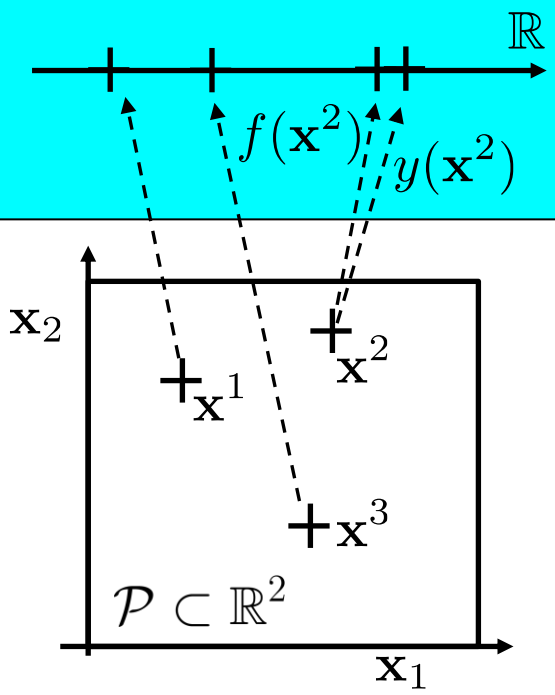
We wish to replace an expensive computer model (or an experiment)

$$y: \quad \mathcal{P} \subset \mathbb{R}^n \rightarrow \mathbb{R}$$
$$\mathbf{x} \mapsto y(\mathbf{x})$$

by a simpler regression model $f(x) \approx y(x)$

based on *a **small*** set of model evaluations

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



- How do we design $f(\mathbf{x})$ so that
  - the meta-model is cheap to evaluate
  - The meta-model is accurate for all parameters
  - The number of model evaluations is small
- How do we construct the dataset?

  → Design of experiment, active learning, space-filling designs

Yet another machine learning task?

Just an interpolation / regression task, but…

1. We can choose where/when to sample new data as needed
2. We can request as much data as we want
3. Data points are expensive and we start from scratch
4. **In most cases, we do not know which part of the parameter space will be of interest to the user, *i.e.* we need be accurate over its entirety**

Space-filling designs - overview

- Design of experiments for parameter space $\mathcal{P} \subset \mathbb{R}^n$:
  Set of points where the model output
  is to be calculated

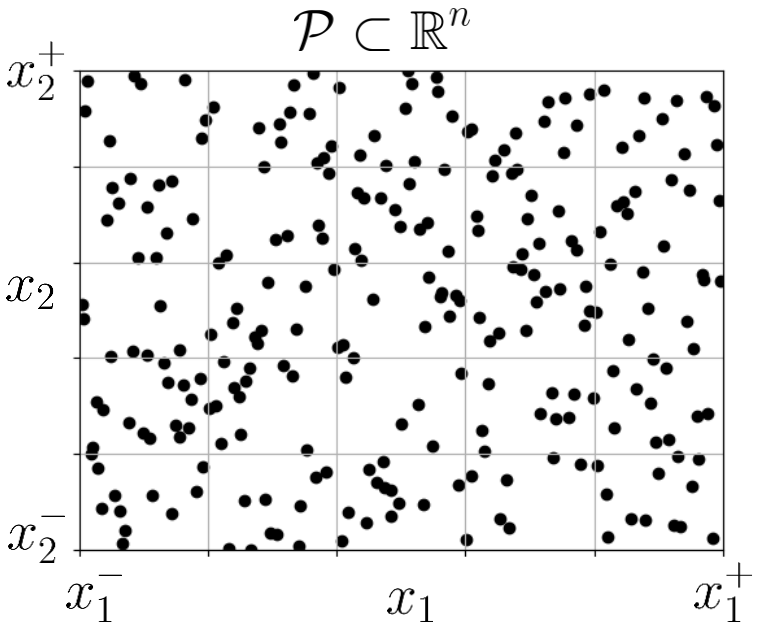
- Should cover the input space
  without leaving too many gaps

- Naive approaches:

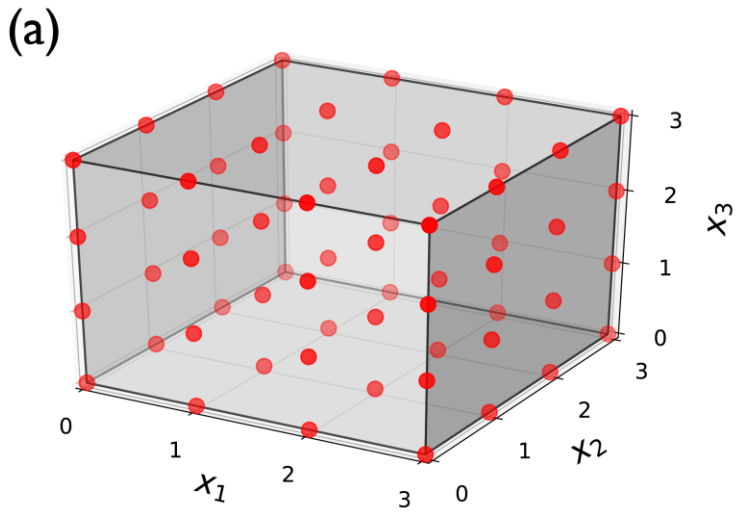
  – Uniform random sampling likely to leav

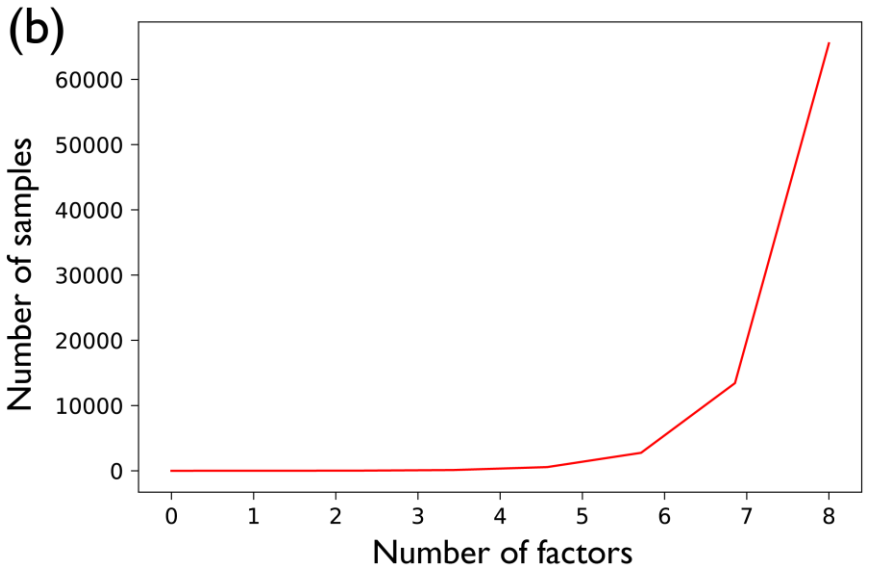
  → Needs stratification

  – Full factorial DOE impractical **due to curse of dimensionality**



Curse of dimensionality

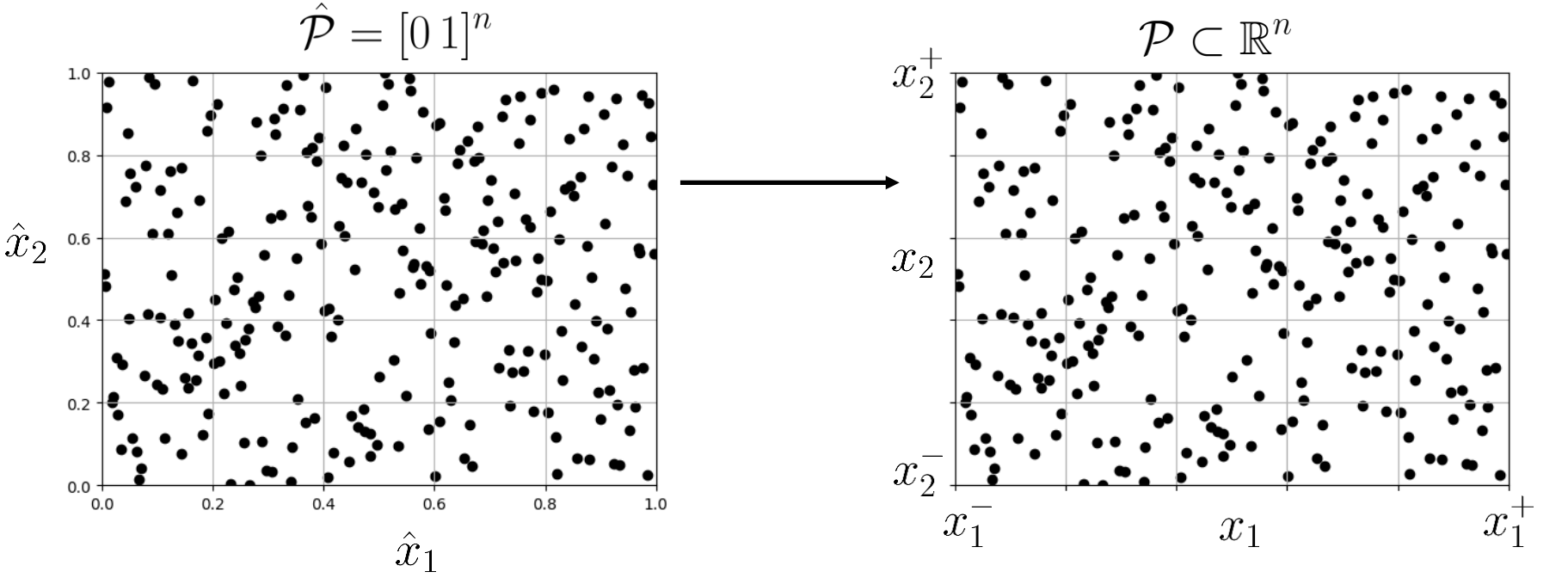




4 samples in 1D
16 samples in 2D
64 samples in 3D
256 samples in 4D
1024 samples in 5D
…

Space-filling designs

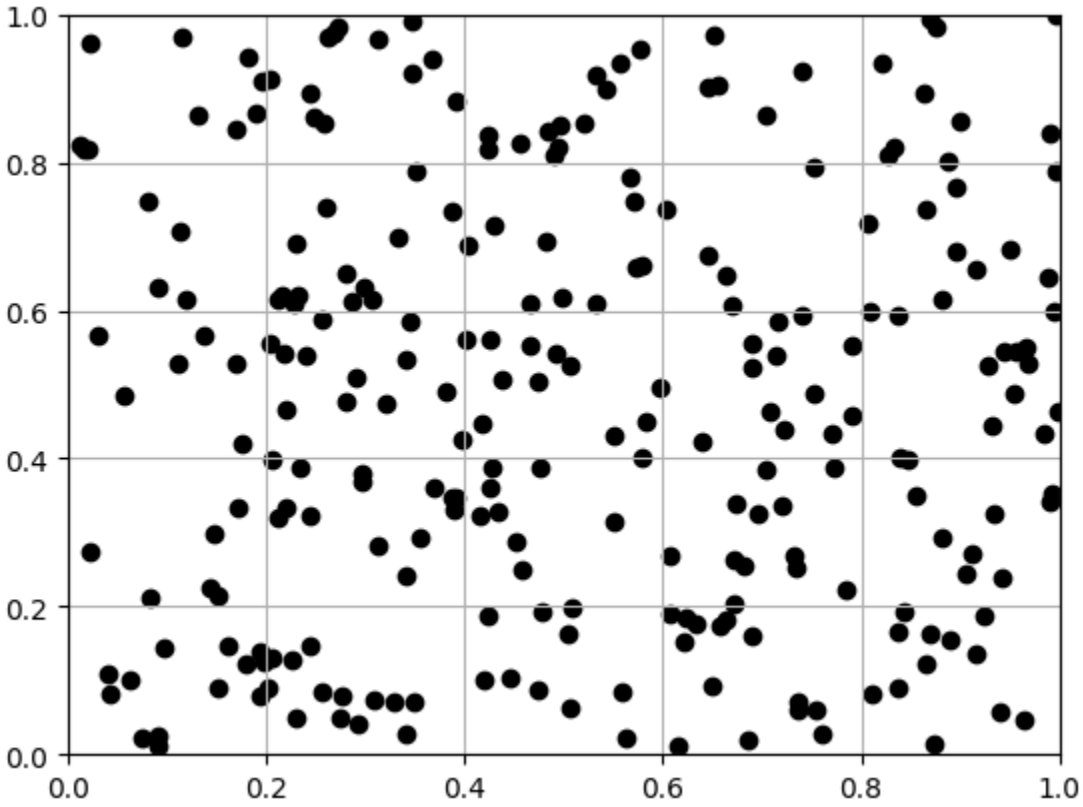
- Parameter space of interest $\mathcal{P} \subset \mathbb{R}^n$ obtained by transformation of reference parameter space $\hat{\mathcal{P}} = [0\,1]^n$



$$x_i = (x_i^+ - x_i^-)\hat{x}_i + x_i^- \in \mathcal{P}^i = [x_i^-\ x_i^+]$$

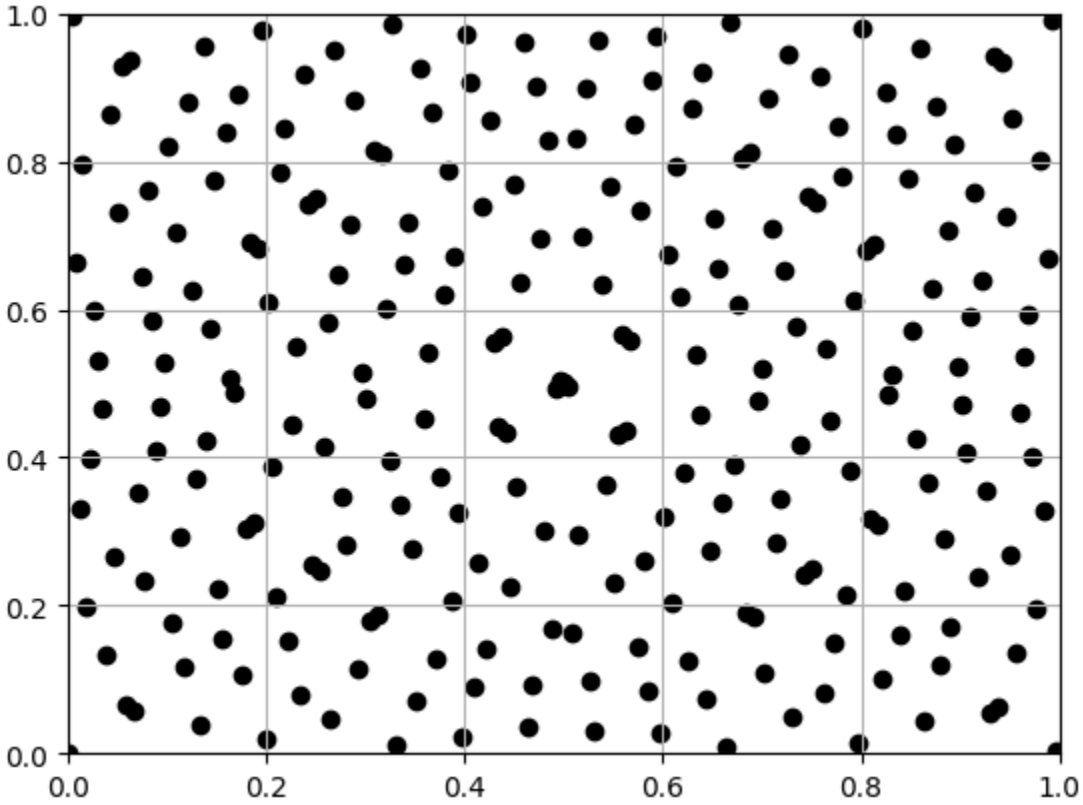
- Space filling design defined over reference parameter space, and samples transformed afterwards before calling simulation model

Random sampling



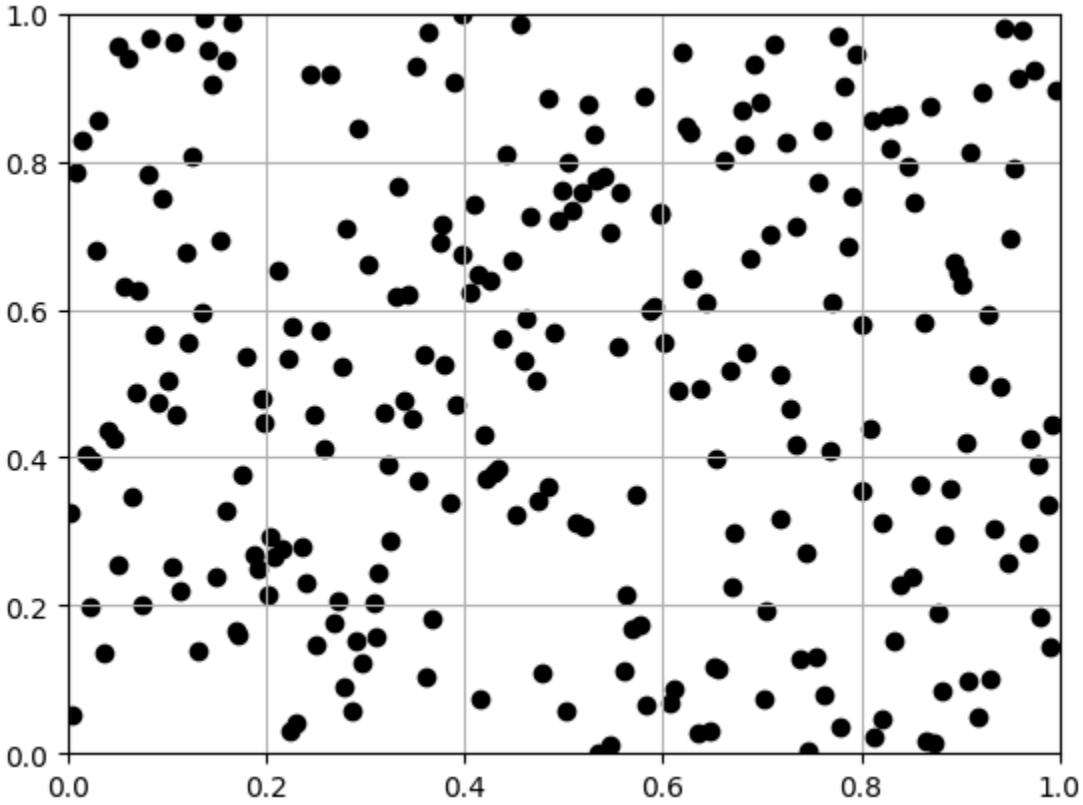

```
X = np.random.rand(N,n)
```

Quasi-random sampling



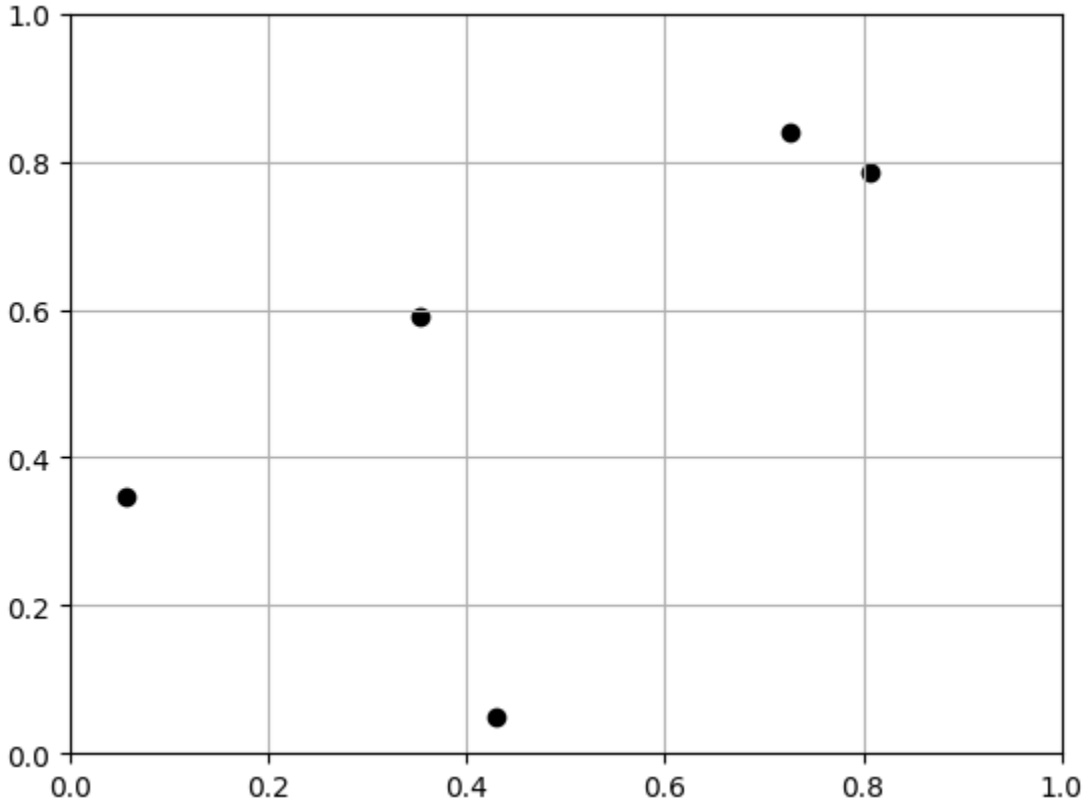
Low-discrepency deterministic sequences of vectors such as those delivered by Sobol sequences

Latin hypercube sampling (1/2)



Stratified random sampling

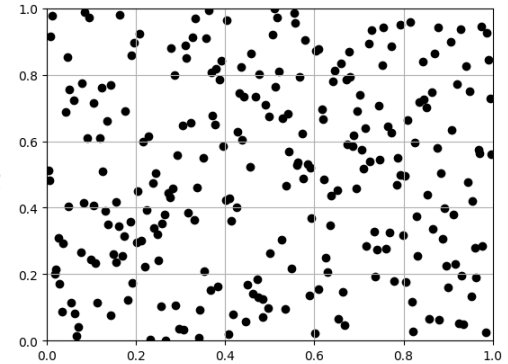
Latin hypercube sampling (2/2)

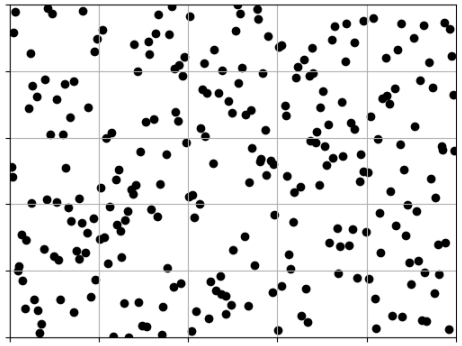


1 randomly placed point per line and column of a regular NxN subdivision of the unit hypercube

Nonlinear space transformations

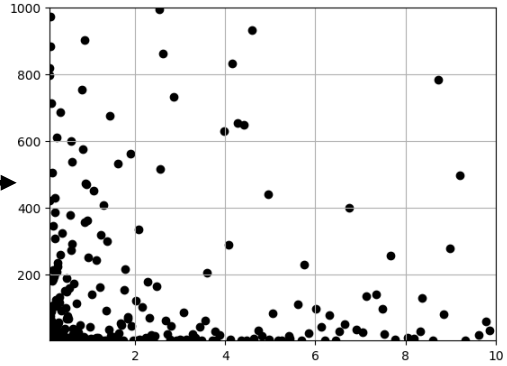
$\hat{\mathcal{P}} = [0\,1]^n$





$\tilde{x}_i = (x_i^+ - x_i^-)\hat{x}_i + x_i^-$

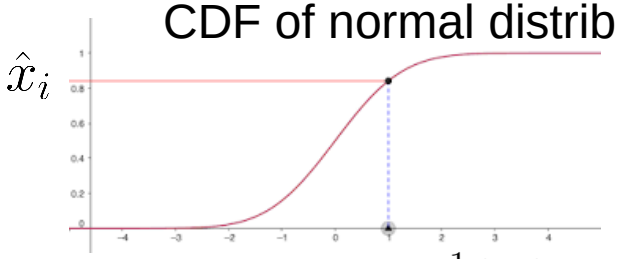
$\mathcal{P} \subset \mathbb{R}^n$



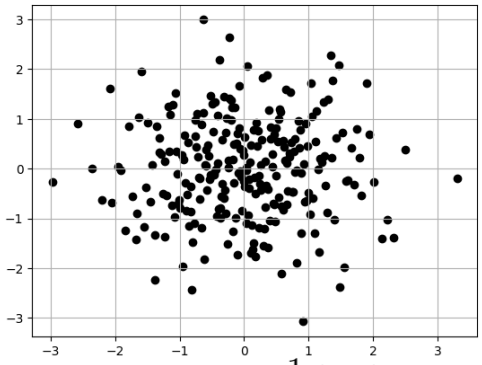
$\tilde{x}_i = 10^{\tilde{x}_i}$

Logarithmic sampling

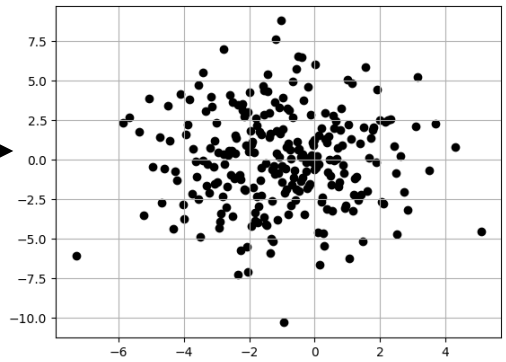
CDF of normal distrib



$\tilde{x}_i = F^{-1}(\hat{x}_i)$



$\tilde{x}_i = F^{-1}(\hat{x}_i)$

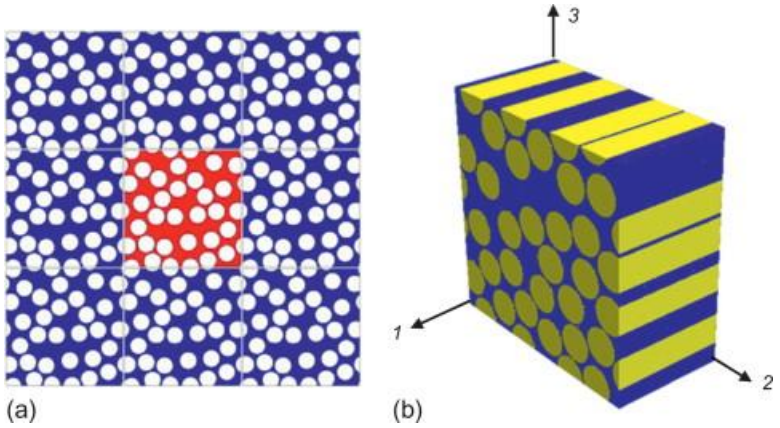


$x_i = a_i\tilde{x}_i + b_i$

Gaussian sampling

Why scale parameters differently?

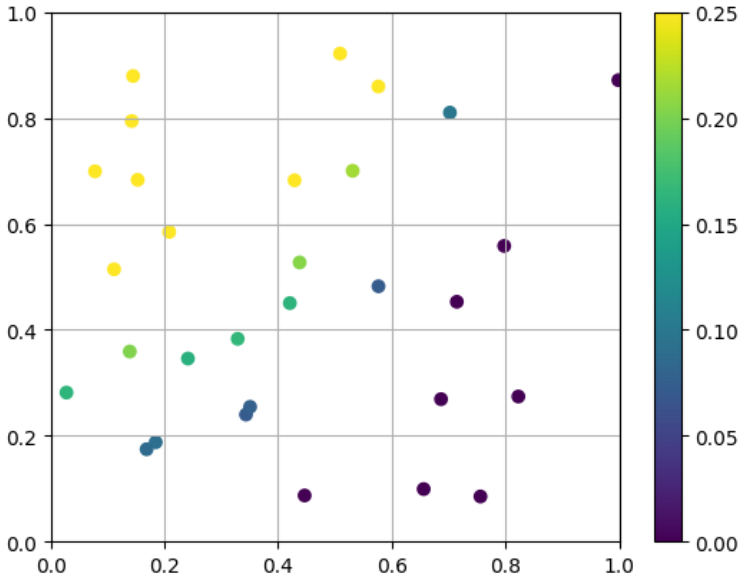
- Why logarithmic sampling?



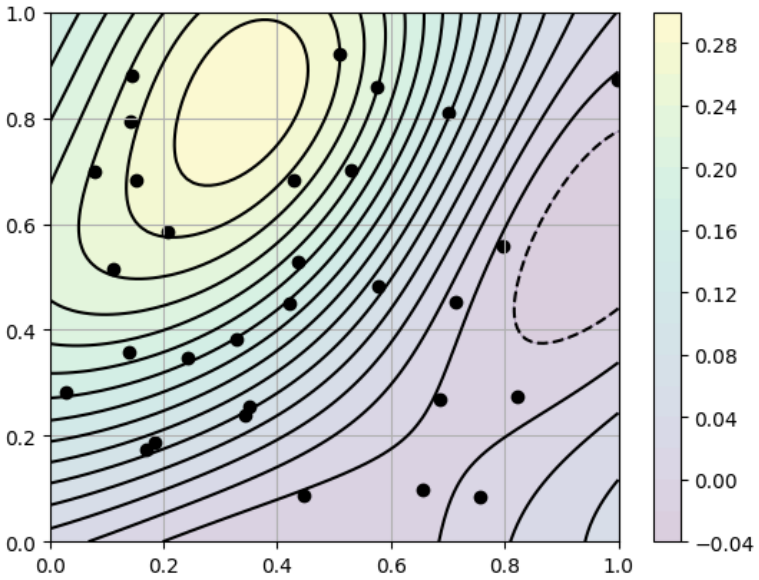
  - Want to compute overall stifness of this composite material with elastic contrast c = E_{fiber} / E_{matrix} as parameter
  - For 0. <= c <= 1. we have soft inclusions, for 1< c < infty, we have hard inclusions
  - To sample as many examples with soft and hard inclusions, we can sample log(c) uniformly over [-a a]

- Why gaussian sampling? No strict bounds provided, instead we have a mean and standard deviation (i.e. uncertainty around the mean)

Regression in reference parameter space



Compute output of simulator at DOE points



Meta-model : regression
(or interpolation)

Two main regression methods are used to build meta-models

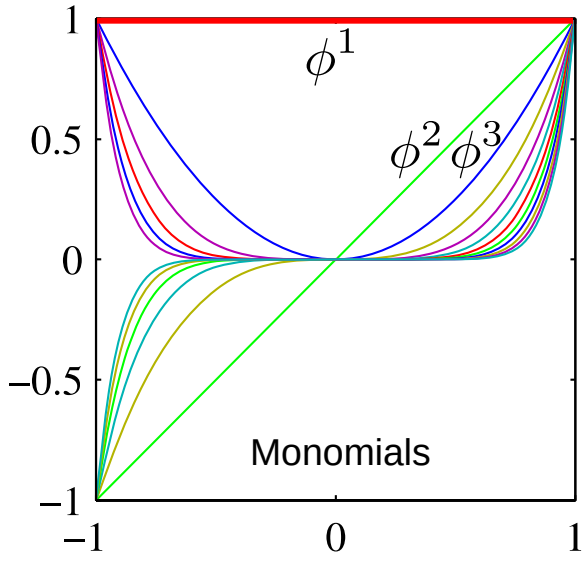
- Polynomial regression (also called polynomial chaos in UQ communities)
- Gaussian Process regression (Kriging)
  (neural networks and regression trees also used but less popular)

Outlines

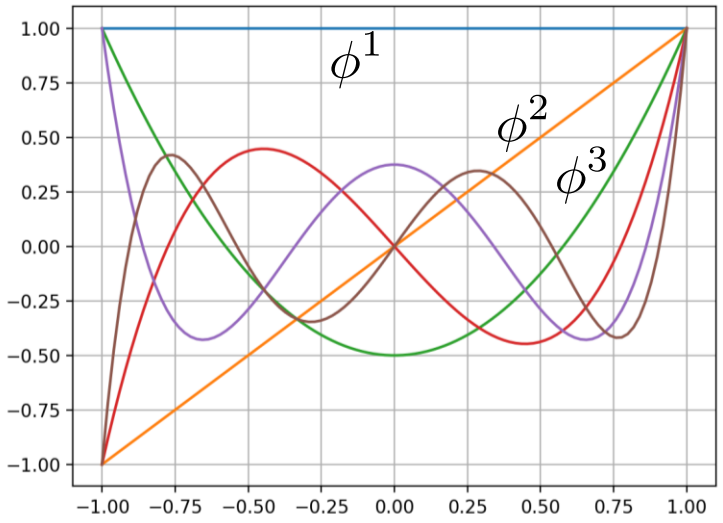
1. Designs of experiments (DoE)
2. Polynomial surrogate modelling
   a. Tensor-product polynomial approximation spaces
   b. Sparse grids interpolation
   c. Sparse polynomial regression
3. Control of (polynomial) surrogate models
   a. Regularisation methods
   b. Model selection
4. Non-parametric surrogate modelling, Gaussian Processes
5. Beyond response surface methodologies using Deep Learning
6. Physics-Informed Neural Nets : surrogate models without numerical solver

Polynomial regression / interpolation

- Surrogate model: $y(\mathbf{x}) \approx f(\mathbf{x})$ $\quad f(\mathbf{x}) = \sum_{i=1}^{N} \phi^i(\mathbf{x}) w_i$



monomials



Legendre polynomials (recombination of monomials to ensure orthogonality over [-1 1])

→ Features are nonlinear functions of the input but the model is a linear combination of thoses explicitly defined nonlinear features

Polynomial bases for $n_{inputs} > 1$

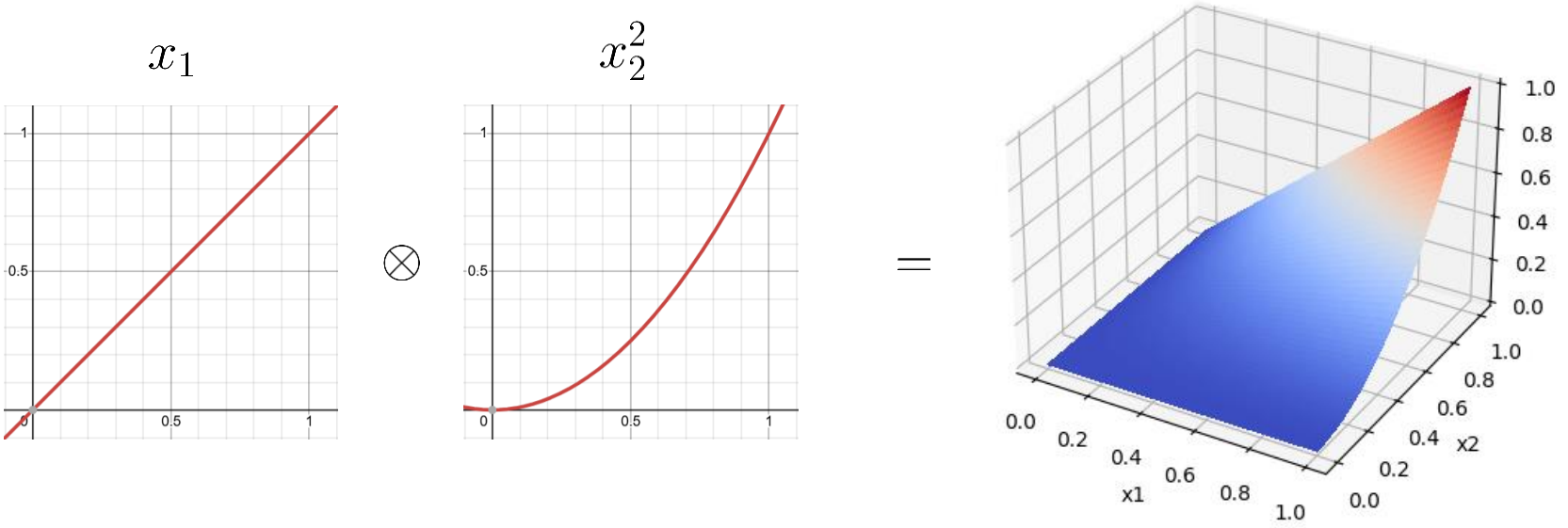
Polynomial shape functions obtained by tensorisation of the polynomial bases defined in dimension 1

$n_{inputs} = 2$ $\quad \phi_{ij}(x_1, x_2) = \phi_i(x_1)\phi_j(x_2)$

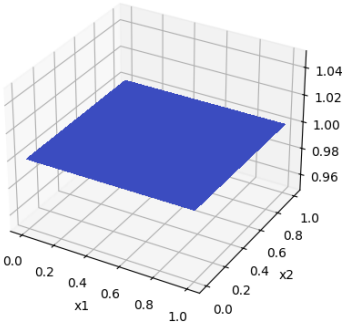
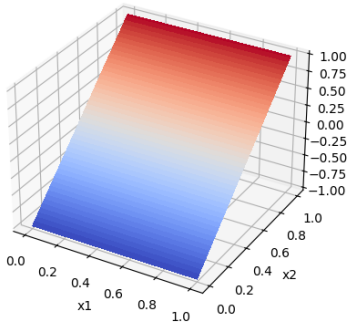
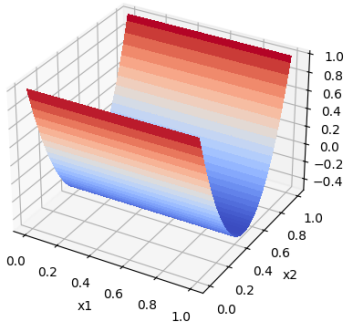
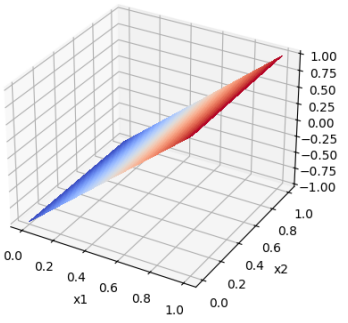
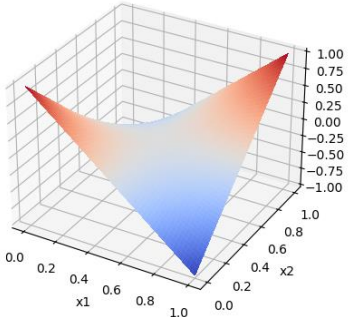
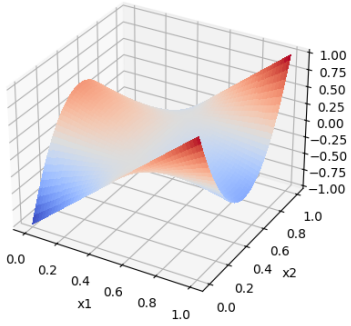
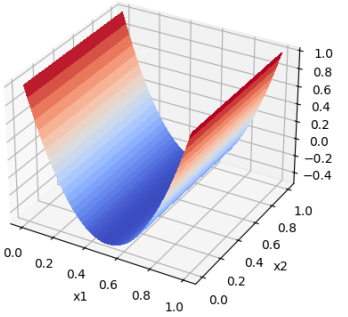
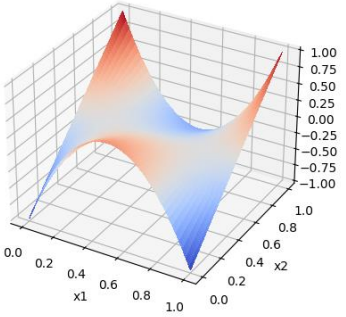
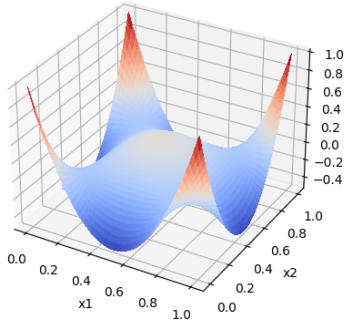
$n_{inputs} = 3$ $\quad \phi_{ijk}(x_1, x_2, x_3) = \phi_i(x_1)\phi_j(x_2)\phi_k(x_3)$

Example: product of monomials of order 1 in coordinate 1 and order 2 in coordinate 2

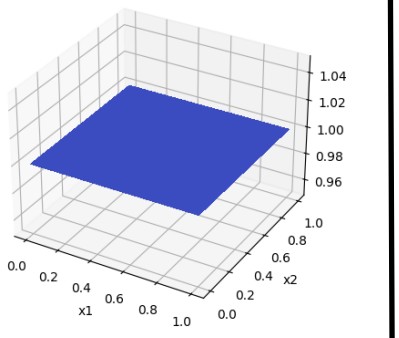
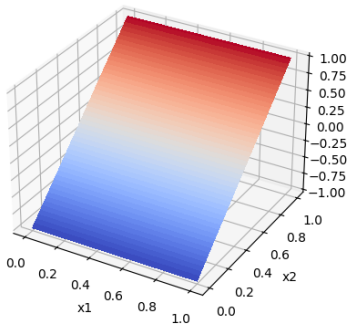
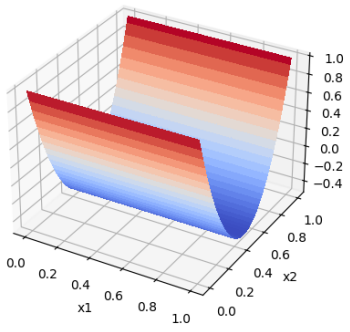
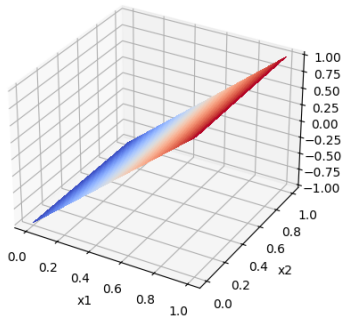
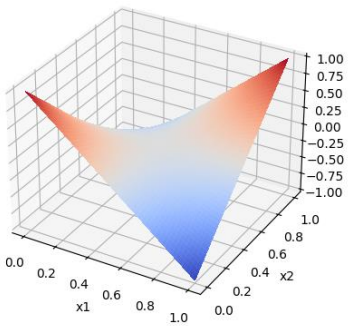
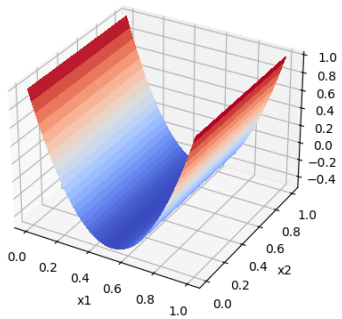
$$\phi_{23}(x_1, x_2) = \phi_2(x_1)\phi_3(x_2)$$
$$\phi_{23}(x_1, x_2) = x_1(x_2)^2$$



Polynomial basis for $N_{inputs} > 1$

order in x_2 / order in x_1	0	1	2
0			
1			
2			

Polynomial basis for $N_{inputs} > 1$

order in x_2 / order in x_1	0	1	2
0			
1			
2			

Polynomial basis of order 2 usually defined as the set of polynomial products such that the sum of orders is lower than 2

Polynomial models

- Polynomial model: $f(\mathbf{x}) = \sum_{i=1}^{N} \phi^i(\mathbf{x}) w_i$ $\qquad y(\mathbf{x}) \approx f(\mathbf{x})$

- Model evaluation at m sample points:

$$\begin{pmatrix} f(\mathbf{x}^1) \\ f(\mathbf{x}^2) \\ \dots \\ f(\mathbf{x}^m) \end{pmatrix} = \begin{pmatrix} \phi^1(\mathbf{x}^1) & \phi^2(\mathbf{x}^1) & \dots & \phi^N(\mathbf{x}^1) \\ \phi^1(\mathbf{x}^2) & \phi^2(\mathbf{x}^2) & \dots & \phi^N(\mathbf{x}^2) \\ & & \dots & \\ \phi^1(\mathbf{x}^m) & \phi^2(\mathbf{x}^m) & \dots & \phi^N(\mathbf{x}^m) \end{pmatrix} \begin{pmatrix} w_1 \\ w_2 \\ \dots \\ w_N \end{pmatrix}$$

($\boldsymbol{\phi}$ and $\mathbf{w}$ labels)

Interpolation:

$\forall i \in [1\ N], \quad f(\mathbf{x}^i) = y^i$

Needs N interpolation points optimally positioned using results of interpolation therory

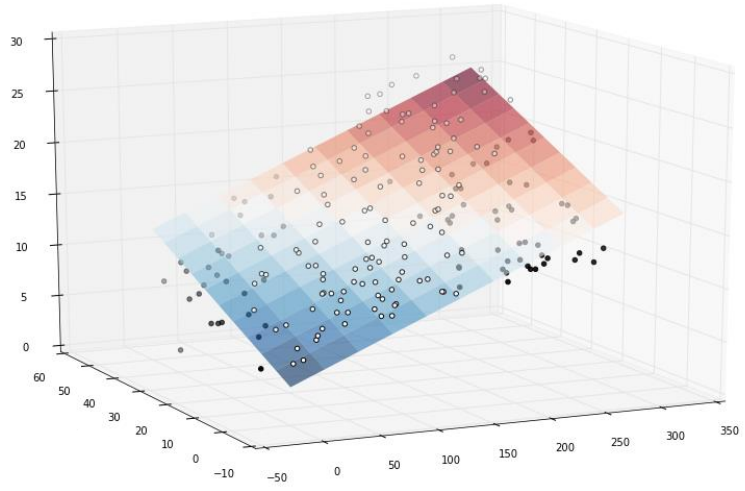
Regression : minimise

$$E(\mathbf{w}) = \frac{1}{2} \sum_{i=1}^{m} \left(y^i - f(\mathbf{x}^i; \mathbf{w}) \right)^2$$

Controlled via statistical model selection methods

Examples

Linear regression in dimension 2



$$\phi = \begin{pmatrix} 1 & x_1^1 & x_2^1 \\ 1 & x_1^2 & x_2^2 \\ & \dots & \\ 1 & x_1^m & x_2^m \end{pmatrix}$$

Vandermonde matrix

Quadratic polynomial regression in 1D

$$\phi = \begin{pmatrix} 1 & x^1 & (x^1)^2 \\ 1 & x^2 & (x^2)^2 \\ & \dots & \\ 1 & x^m & (x^m)^2 \end{pmatrix}$$

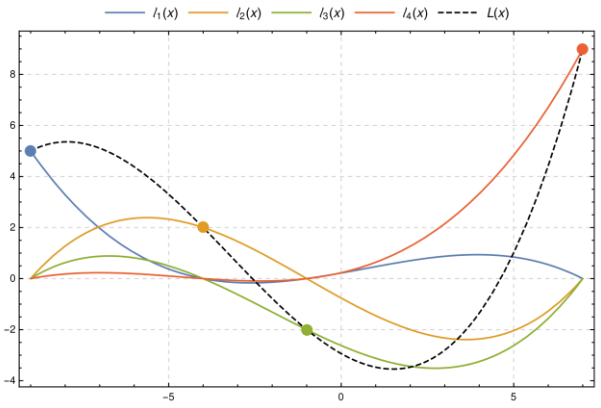
Note: polynomial interpolation

- Interpolation theorem:

  Given $n+1$ distinct points $x_0, x_1, \ldots, x_n$ and corresponding values $y_0, y_1, \ldots, y_n$, there exists a unique polynomial of degree at most k that interpolates the data $\{(x_0, y_0), \ldots, (x_n, y_n)\}$.[2]

- Lagrange polynomials

$$L(x) = \sum_{j=0}^{k} y^j \ell^j(x) \qquad \ell^j(x) = \prod_{\substack{0 \le m \le k \\ m \ne j}} \frac{x - x_m}{x_j - x_m}$$



- Polynomial interpolation error:

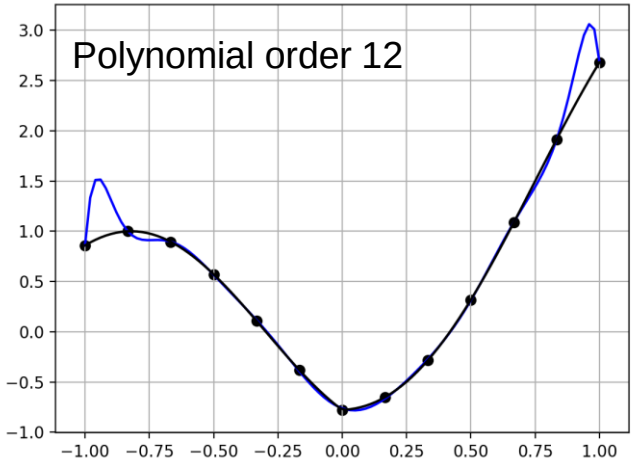
$$\max_{x \in [a,b]} |f(x) - p_n(x)| \le \max_{x \in [a,b]} |\omega_{n+1}(x)| \cdot \frac{\max_{x \in [a,b]} |f^{(n+1)}(x)|}{(n+1)!}$$

$$\omega_{n+1}(x) \equiv (x - x_0)(x - x_1) \cdots (x - x_n) = \prod_{j=0}^{n} (x - x_j)$$

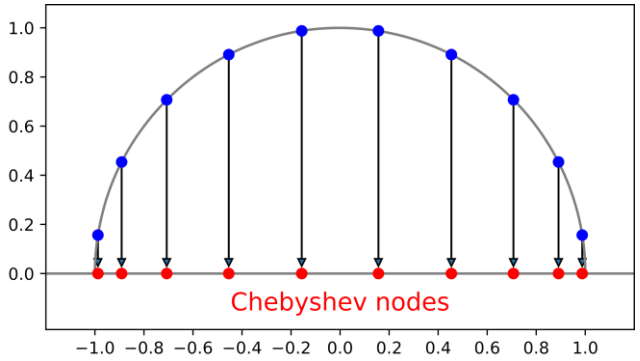
https://en.wikipedia.org/wiki/Mean_value_theorem_(divided_differences)

Runge's phenomenon

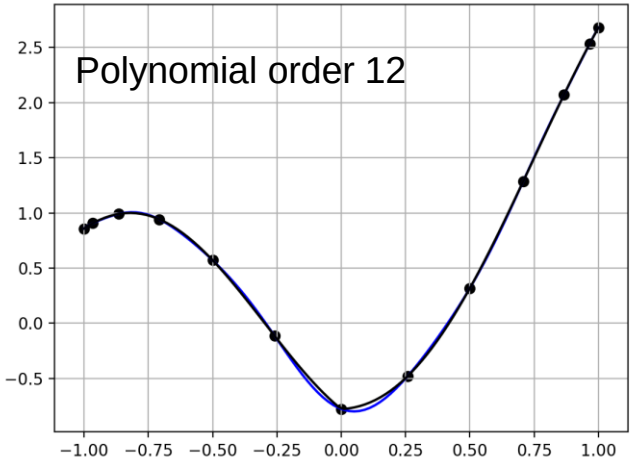
Newton-Cotes interpolation



Curtis-Clenshaw quadrature points:



Curtis-Clenshaw interpolation

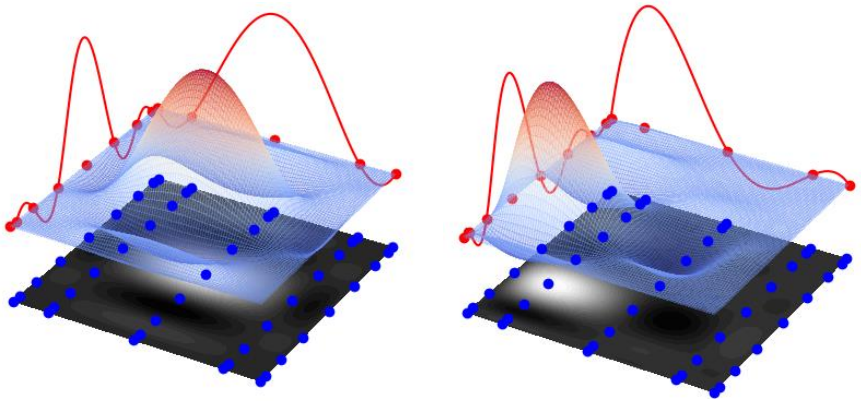


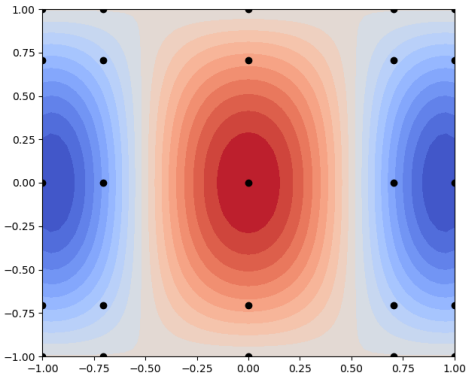
Nested rule: the rule for order *N* uses a subset of the points from order 2N

Tensor product interpolation

- Multiplication of 1D Lagrange polynomials yields a family N-dimensional Lagrange polynomials

$$f(\mathbf{x}) = \sum_{\boldsymbol{j} \leq \boldsymbol{\beta}} f(\mathbf{x}^{(\boldsymbol{j})}) \prod_{i \in [d]} \phi_j(x_i) \qquad \phi_j(x) = \prod_{k=1, k \neq j}^{m} \frac{x - x^{(k)}}{x^{(j)} - x^{(k)}}$$





- Needs a tensor-product grid of interpolation points (here Curtis-Clenshaw but arbitrary set of points in every individual dimension)
- Limitations
  - Inflexible in terms of sampling point adaptivity (one additionnal point in a particular dimension adds an hyperplane in dimension N-1)
  - Curse of dimensionality

Curse of Dimensionality

Tensor Product Monomial Basis

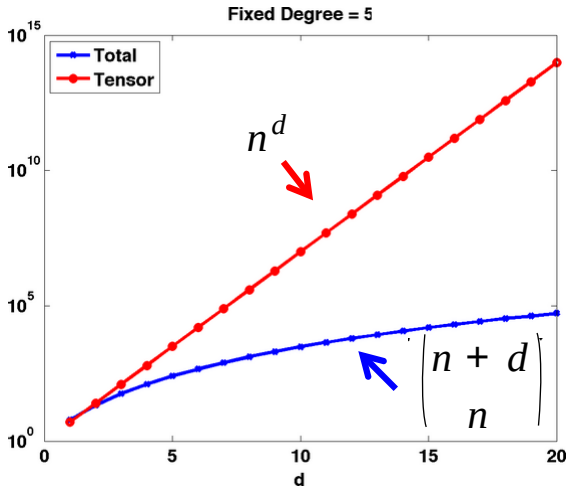
y^2	x^1y^2	x^2y^2
y^1	x^1y^1	x^2y^1
1	x^1	x^2

dimension of the polynomial space! n^d

Total Degree Monomial Basis!

y^2		
y^1	x^1y^1	
1	x^1	x^2

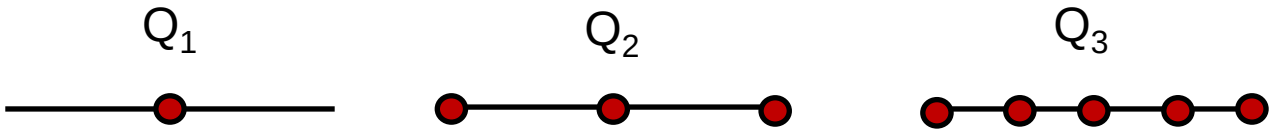
dimension of the polynomial space! $\binom{n+d}{d}$



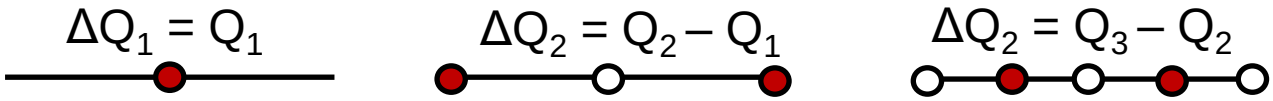
Adapted from P. Constantine

Sparse grids : principle (1/3)

- Define hierarchical / nested grids for each dimension

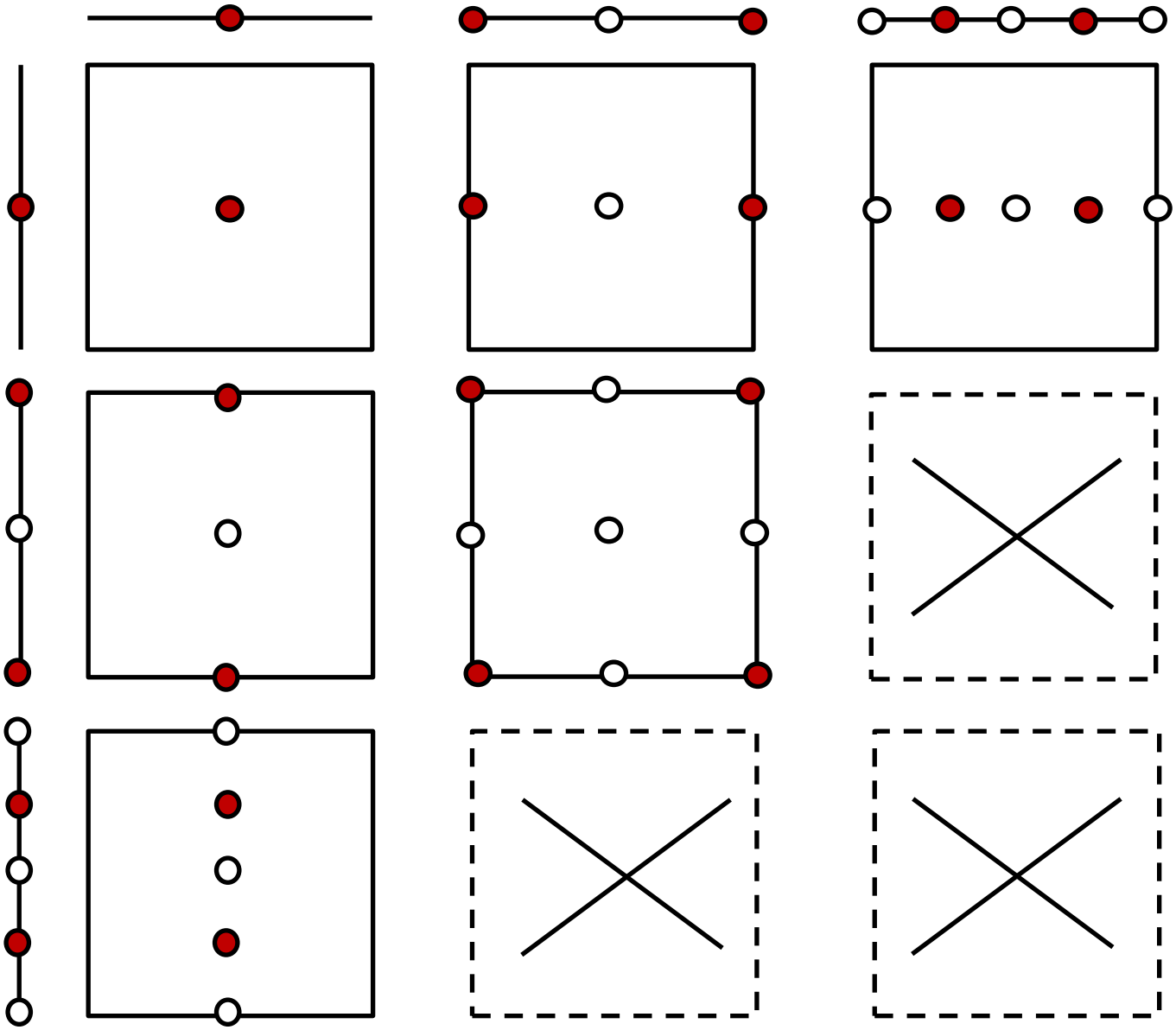


- Rewrite as differences of grids between two successive levels

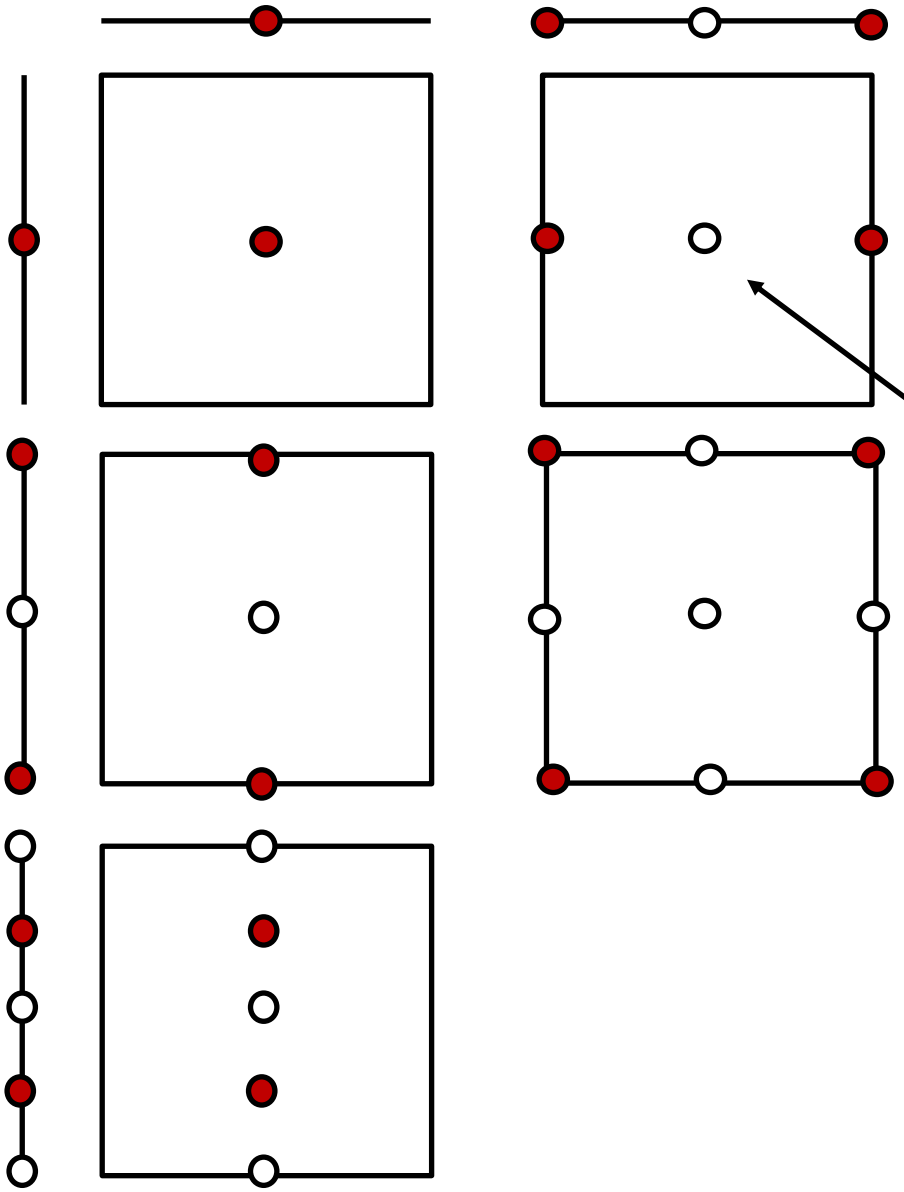


- Tensorise accross dimensions

Sparse grids : principle (2/3)



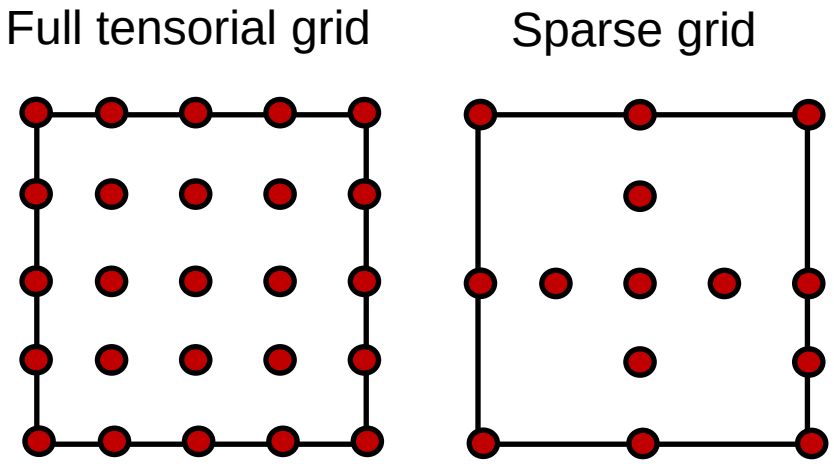
Sparse grids : principle (2/3)



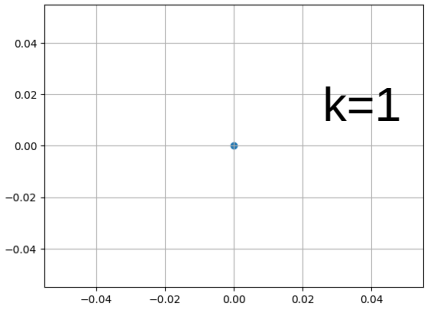
Order 2 in x_1, order 0 in x_2
Interpolate using tensorised Lagrange polynomials that vanish at all points « o » that exist at coarser grids levels

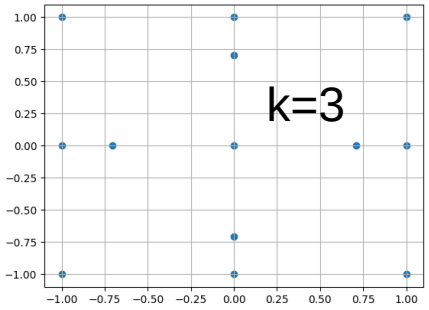
Sum over terms of the sparse grid. Interpolation is guaranteed, as only one of the terme is non-vanishing for each point of the grid

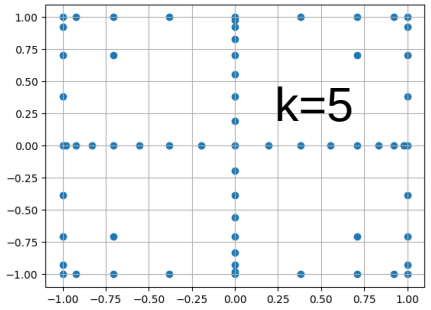
Sparse grids : principle (3/3)

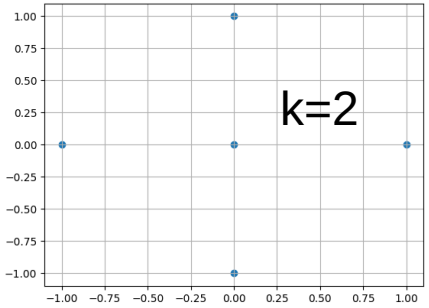


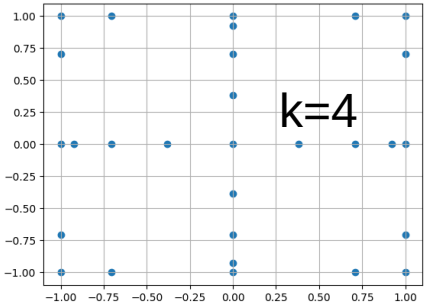
Curtis-Clenshaw sparse grids

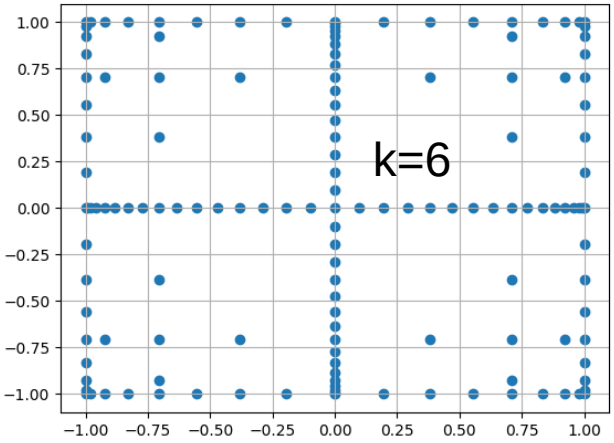




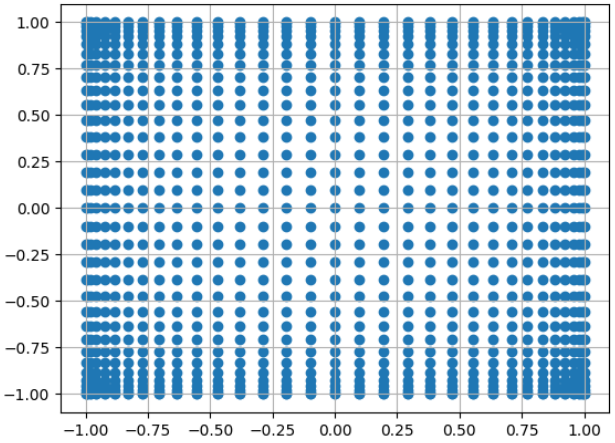




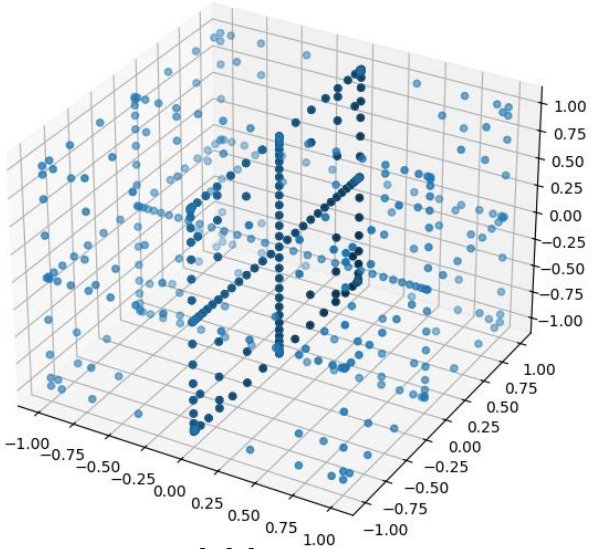




m = 145



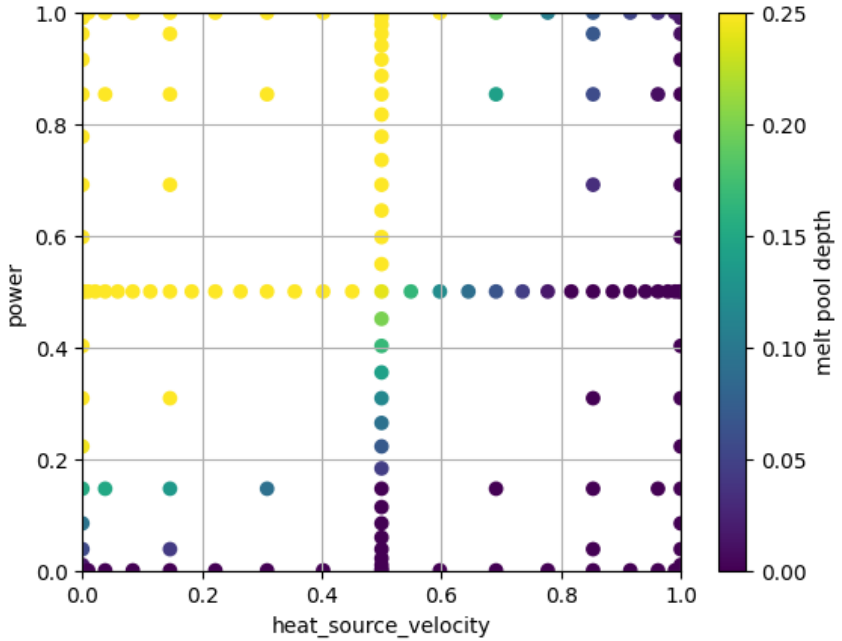
m = 1089



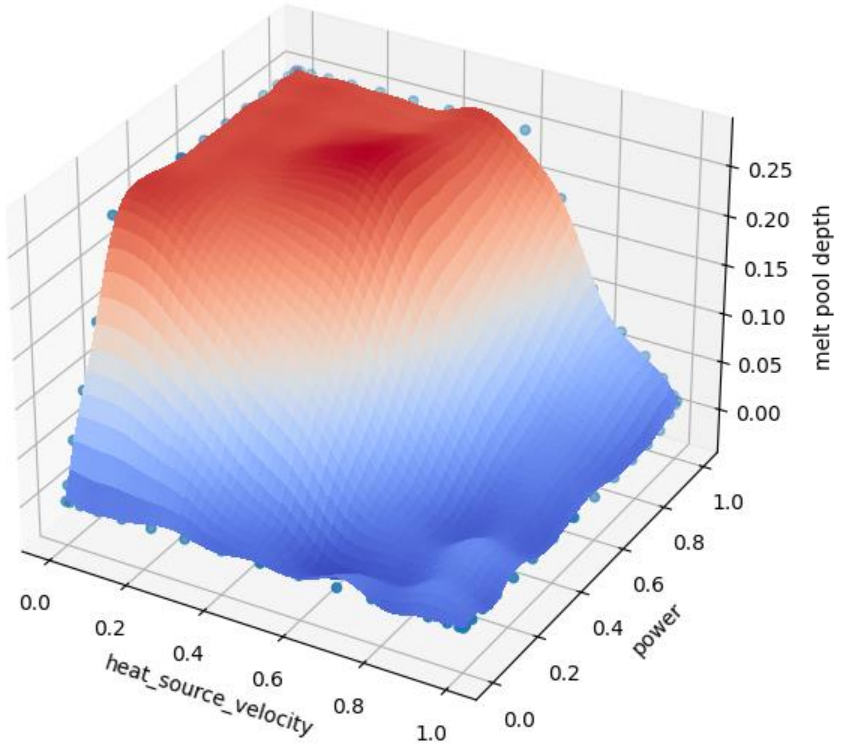
m = 441

Interpolation results

Sparse grid k=6 with sampled values

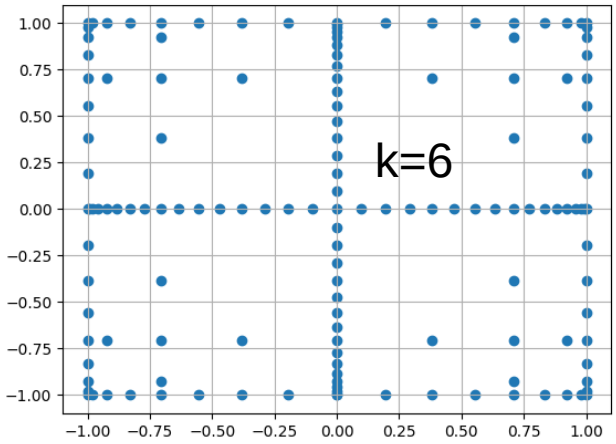


Interpolation over regular grid

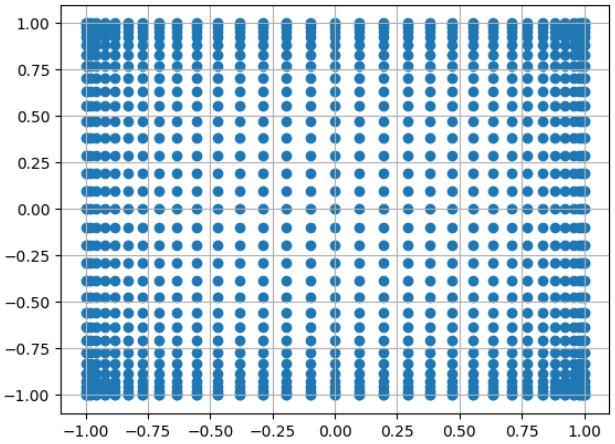


Sparse grids : pros & cons

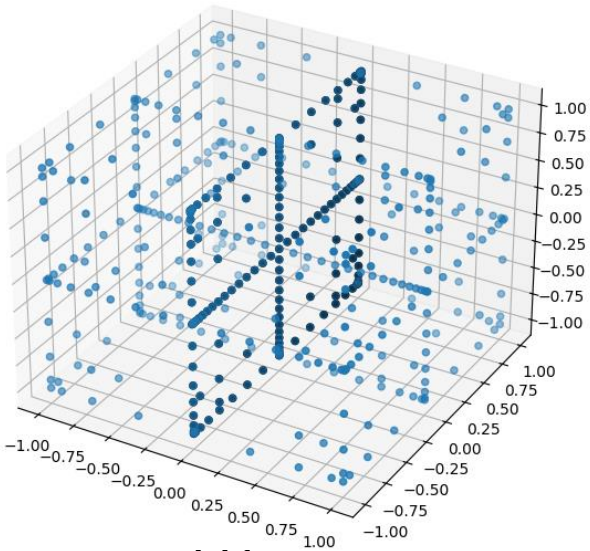
- Advantages :
  - Deterministic
  - Interpolatory (exact at model evaluation points)
- Limitations :
  - number of points still increases fast with parameter dimension N
  - Number of points increases fast with refinement level k
  - → Adaptivity dimension-per-dimension faisable but non-trivial
  - → More stringent sparsification rules than « sum of levels lower than » exist
  - Some overfitting in regions that have been « sparsified »



m = 145



m = 1089



m = 441

Regression : linear least-squares

- Least-square error formulation

$$\mathbf{w}^\star = \arg\min_{\mathbf{w}} E(\mathbf{w}) \qquad E(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{m}\left(y^i - f(\mathbf{x}^i;\mathbf{w})\right)^2$$

$$E(\mathbf{w}) = \frac{1}{2}\|\mathbf{y} - \boldsymbol{\phi}\mathbf{w}\|^2$$

Has always at least one solution! Not interpolatory in general

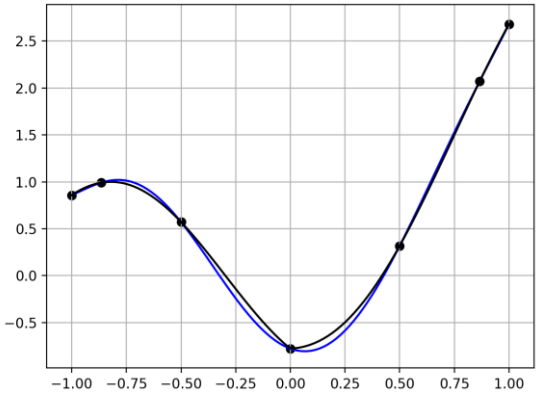
$$\begin{pmatrix} f(\mathbf{x}^1) \\ f(\mathbf{x}^2) \\ \dots \\ f(\mathbf{x}^m) \end{pmatrix} = \underbrace{\begin{pmatrix} \phi^1(\mathbf{x}^1) & \phi^2(\mathbf{x}^1) & \dots & \phi^N(\mathbf{x}^1) \\ \phi^1(\mathbf{x}^2) & \phi^2(\mathbf{x}^2) & \dots & \phi^N(\mathbf{x}^2) \\ & & \dots & \\ \phi^1(\mathbf{x}^m) & \phi^2(\mathbf{x}^m) & \dots & \phi^N(\mathbf{x}^m) \end{pmatrix}}_{\boldsymbol{\phi}} \underbrace{\begin{pmatrix} w_1 \\ w_2 \\ \dots \\ w_N \end{pmatrix}}_{\mathbf{w}}$$

- Solution:

$$\mathbf{w}^\star = \left(\boldsymbol{\phi}^T\boldsymbol{\phi}\right)^{-1}\boldsymbol{\phi}^T\mathbf{y}$$

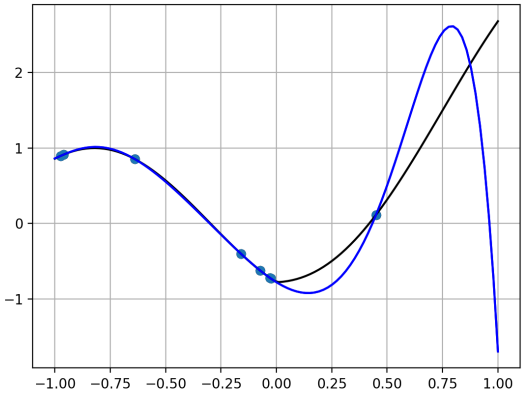
Examples in 1D

Polynomial order 6



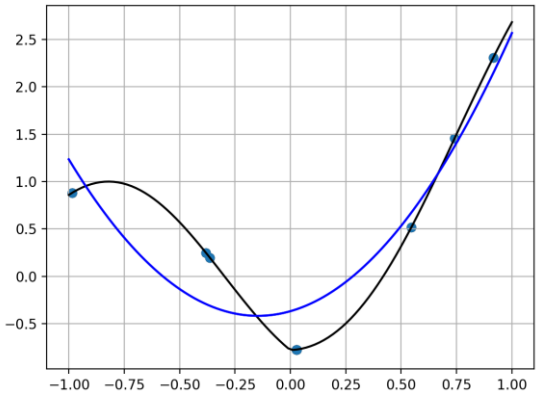
Curtis-Clenshaw, interpolation scheme (optimally placed points)

Polynomial order 6

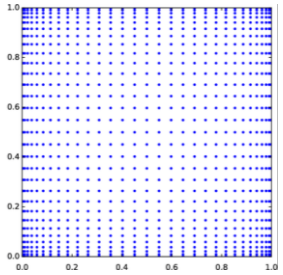


Random sampling … erroneous in poorly explored regions, unstable extrapolation

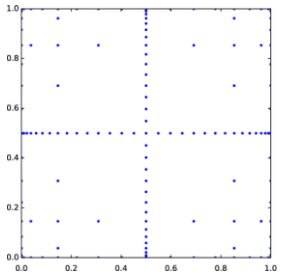
Polynomial order 2



not interpolatory but stable



Full factorial Curtis-Chenshaw, CoD ☹



Sparse grids

Restricting expressiveness *a priori* is one way to introduce stability, another one is **regularisation** through added terms to the loss

L_2 regularisation

- Idea: penalise large values of the regression weights
- **Ridge / L2 / Tikhonov regularization**

unregularised RSS

$$\mathbf{w}^\star = \arg\min_{\mathbf{w}} E(\mathbf{w}) = \arg\min_{\mathbf{w}} \left(\tilde{E}(\mathbf{w}) + E_0(\mathbf{w}) \right)$$

regularisation term

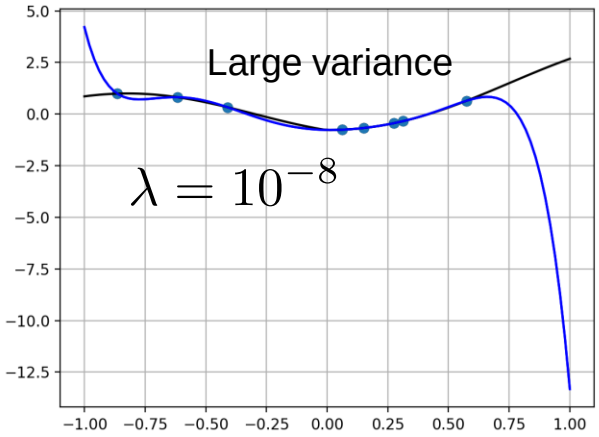
$$E(\mathbf{w}) = \tilde{E}(\mathbf{w}) + E_0(\mathbf{w}) = \tilde{E}(\mathbf{w}) + \frac{\lambda}{2}\|\mathbf{w}\|_2^2$$

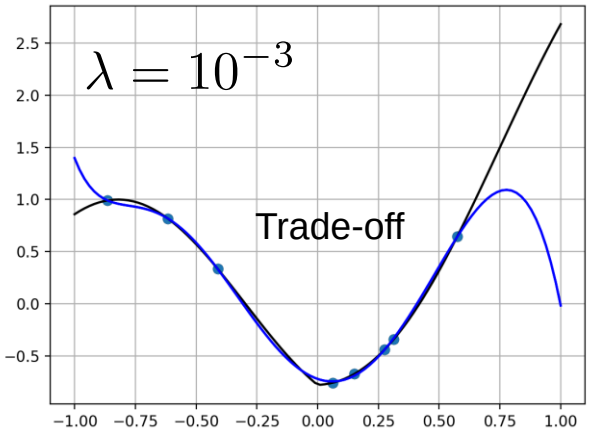
$$\mathbf{w}^\star = \left(\boldsymbol{\phi}^T \boldsymbol{\phi} + \lambda \mathbf{I}_d \right)^{-1} \boldsymbol{\phi}^T \mathbf{y}$$

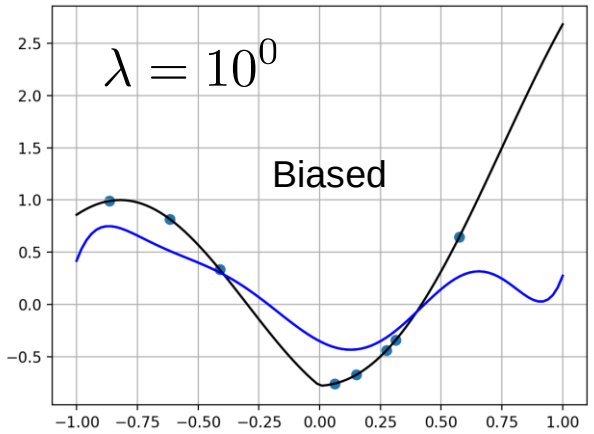
regularisation coefficient

- Theorem: if the covariance matrix is not invertible, then the ridge-regularised one is invertible (prove it at home using the SVD)
  - → Variance reduced but bias towards weight vectors of smaller size

L_2 regularisation







How can you choose the regularization parameter ?

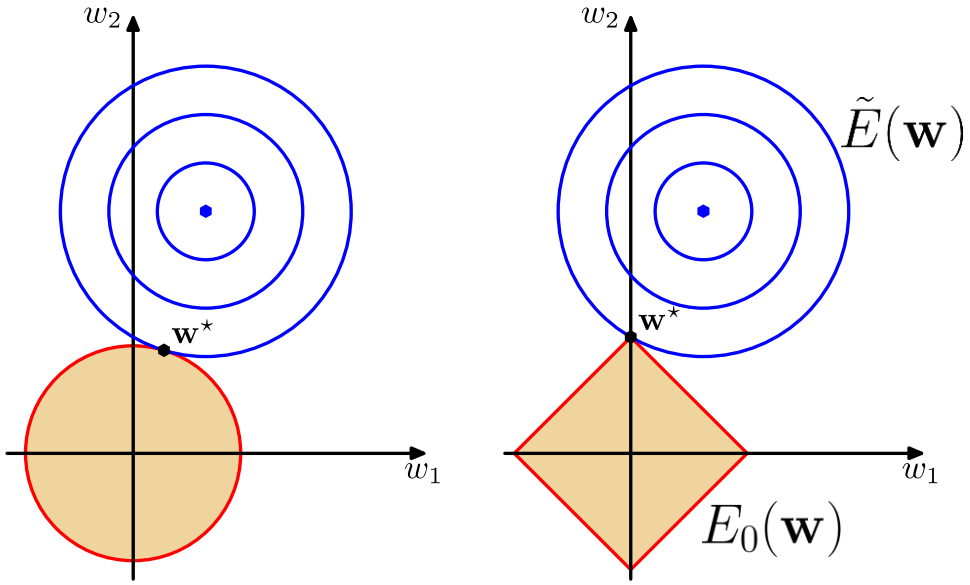
Variance reduction: sparse sensing

- Try finding smallest number of shape functions (sparse solutions)
- On way to achieve this is LASSO or L_1 regularisation

$$\mathbf{w}^\star = \arg\min_{\mathbf{w}} E(\mathbf{w}) = \arg\min_{\mathbf{w}} \left(\tilde{E}(\mathbf{w}) + E_0(\mathbf{w}) \right)$$

$$E(\mathbf{w}) = \tilde{E}(\mathbf{w}) + E_0(\mathbf{w}) = \frac{1}{2}\|\mathbf{e}(\mathbf{w})\|_2^2 + \frac{\lambda}{2}\|\mathbf{w}\|_1$$

$$\|\mathbf{w}\|_p = \left(\sum_{i=1}^{N} |\mathbf{w}_i|^p \right)^{\frac{1}{p}}$$



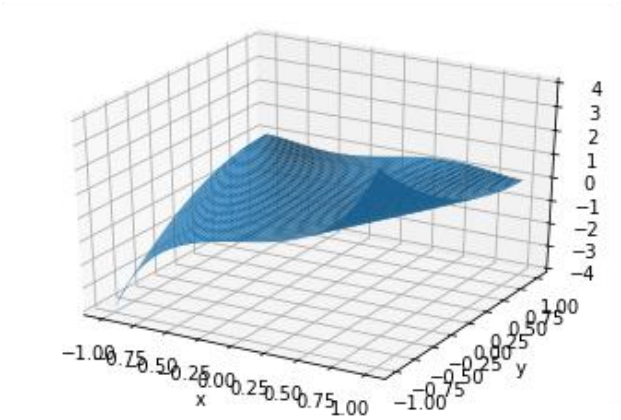
L1-norm term encourages weight vectors with vanishing components

However, the least-square problem is now nonlinear and non-smooth. Specialised iterative solvers required!

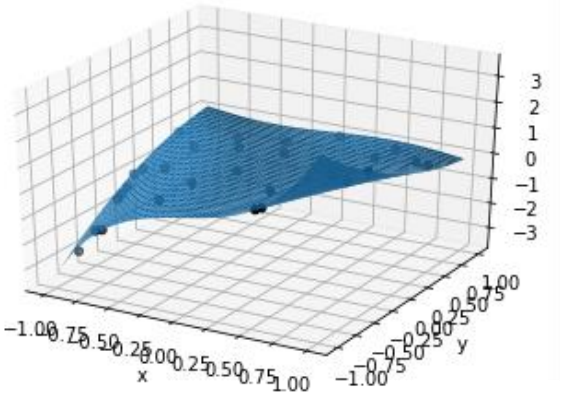
https://xavierbourretsicotte.github.io/lasso_derivation.html

L1-regularised polynomial regression

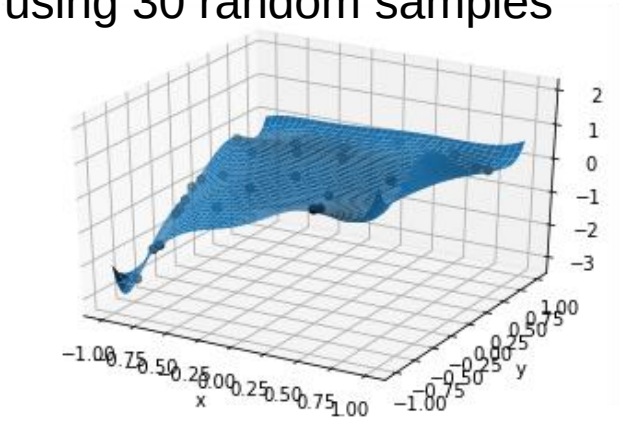
Truth



Order 6 polynomial (full factorial), using 30 random samples, L1-regularised



Order 6 polynomial (full factorial), using 30 random samples



Weights (mostly zeros)

```
[-0.00464736  0.74423948 -0.          0.39366551 -0.          0.02476349
 -0.          0.00968563 -1.15600968  0.         -0.58354303  0.
 -0.          0.         -0.          0.44318502 -0.         -0.
 -0.         -0.         -0.          0.          0.          0.
  0.          0.          0.          0.          0.         -0.
  0.         -0.          0.         -0.         -0.         -0.
  0.         -0.          0.         -0.          0.         -0.
  0.         -0.          0.         -0.          0.         -0.
  0.        ]
```

Supervised feature engineering!

https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Lasso.html

Polynomial Chaos

- PC minimises the (weighted) integral of error over the input domain

$$\mathbf{w}^\star = \arg\min_{\mathbf{w}} \left(\left(y(\mathbf{x}) - \sum_{i=1}^{N} \phi^i(\mathbf{x}) w_i, y(\mathbf{x}) - \sum_{i=1}^{N} \phi^i(\mathbf{x}) w_i \right)_{[-1\ 1]^n} \right)$$

$$\longrightarrow \quad \sum_{i=1}^{N} \left(\phi^i(\mathbf{x}), \phi^j(\mathbf{x}) \right)_{[-1\ 1]^n} w_i = \sum_{i=1}^{N} \left(y(\mathbf{x}), \phi^j(\mathbf{x}) \right)_{[-1\ 1]^n}$$

- If orthogonal polynomials are used (*e.g.* Legendre), we find

$$\forall i, \quad w_i = \left(y(\mathbf{x}), \phi^i(\mathbf{x}) \right)_{[-1\ 1]^n}$$

→ Can be evaluated, *e.g.* by Monte-Carlo sampling, no matrix inversion

- **Regression is a Monte-Carlo approximation of PC metric**

$$\mathbf{w}^\star = \arg\min_{\mathbf{w}} \left(\sum_{j=1}^{m} \left(y(\mathbf{x}^j) - \sum_{i=1}^{N} \phi^i(\mathbf{x}^j) w_i \right)^2 \right)$$

Outlines

1. Designs of experiments (DoE)

2. Polynomial surrogate modelling
   a. Tensor-product polynomial approximation spaces
   b. Sparse grids interpolation
   c. Sparse polynomial regression

3. Control of (polynomial) surrogate models
   a. Regularisation methods
   b. Model selection

4. Non-parametric surrogate modelling, Gaussian Processes

5. Beyond response surface methodologies using Deep Learning

6. Physics-Informed Neural Nets : surrogate models without numerical solver

Measure of accuracy?

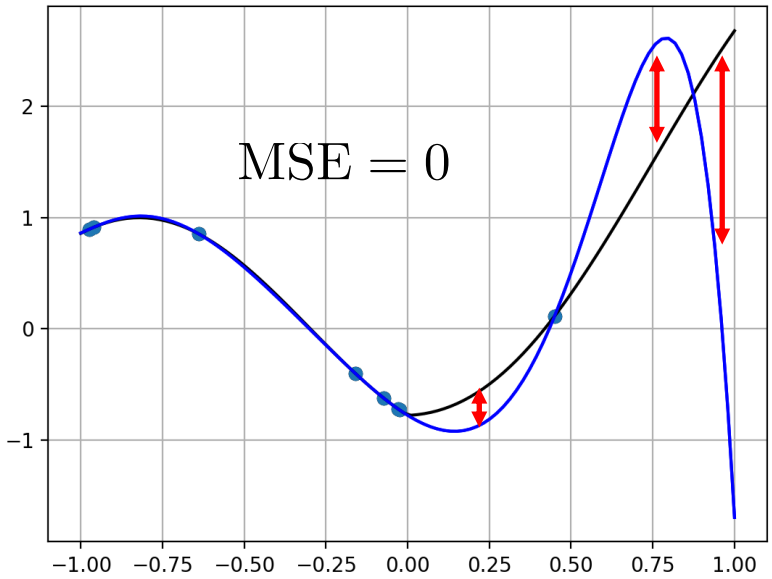
$$E(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{m}\left(y^i - f(\mathbf{x}^i)\right)^2$$

$$\mathrm{MSE}(\mathbf{w}) = \frac{1}{m}\sum_{i=1}^{m}\left(y^i - f(\mathbf{x}^i)\right)^2$$

$$R^2 = 1 - \frac{\mathrm{MSE}(\mathbf{w})}{\frac{1}{m}\|\mathbf{y} - \mathrm{mean}(\mathbf{y})\mathbf{1}\|^2}$$

Loss functions are notoriously bad measures of performance on future predictions

MSE = mean squared error
R^2 = coefficient of determination



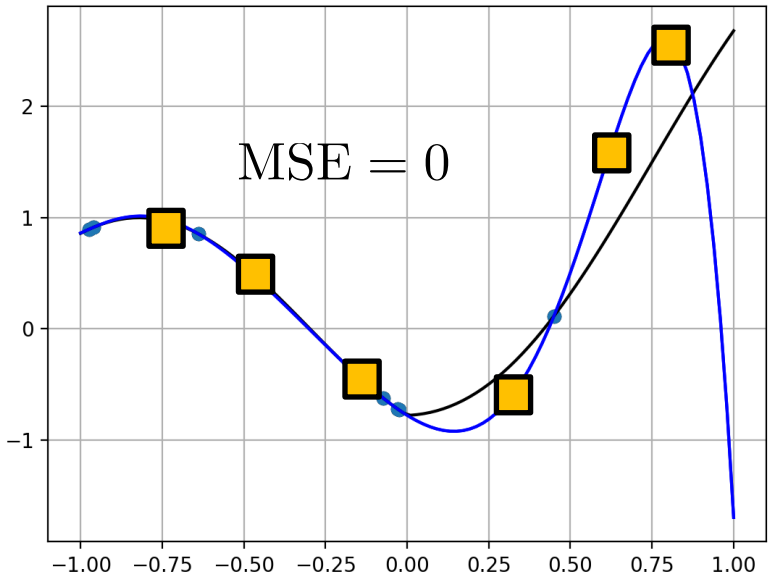
Large error for vanishing MSE = **overfitting**

Order 7 polynomial regression with 6 datapoints

Generalisation error

Compute error indicators on validation set

$$\mathrm{MSE}_{\mathrm{val}}(\mathbf{w}) = \frac{1}{m_{\mathrm{val}}} \sum_{i=1}^{m_{\mathrm{val}}} \left(y^i_{\mathrm{val}} - f(\mathbf{x}^i_{\mathrm{val}})\right)^2$$



Validation set

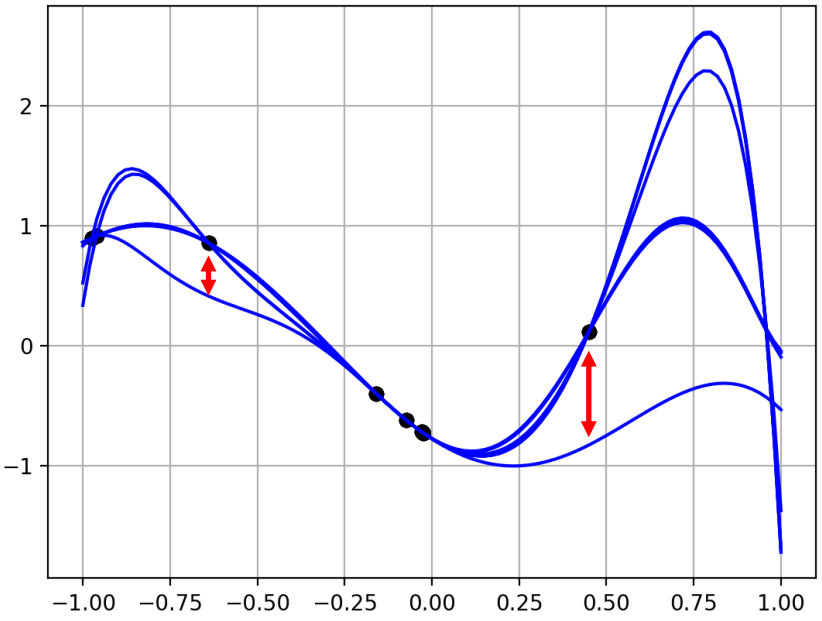
⟶ We may want to use more efficient strategies as data points are too valuable not to use them to improve model fitting

Cross-validation (LOOCV)

Fake the availability of a validation set. For every datapoint, fit model without that data point and measure the corresponding prediction error

$$\mathrm{MSE}_{cv}(\mathbf{w}) = \frac{1}{m}\sum_{i=1}^{m}\left(y^i - \boldsymbol{\phi}\mathbf{w}^i_-\right)^2$$

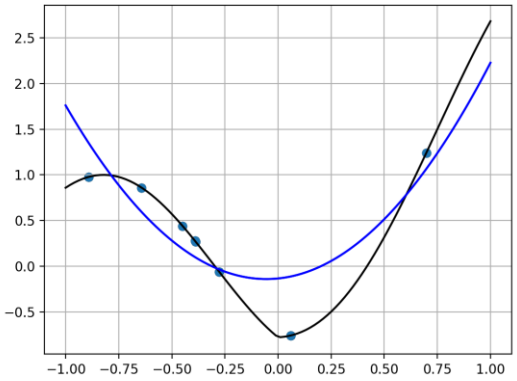
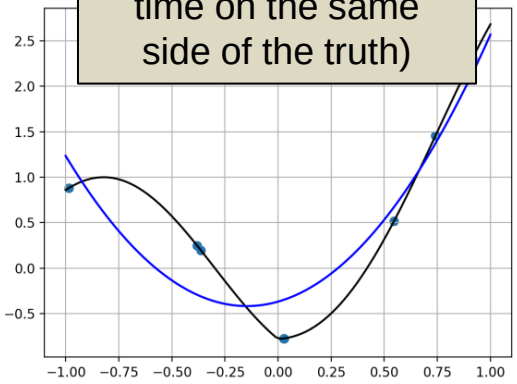
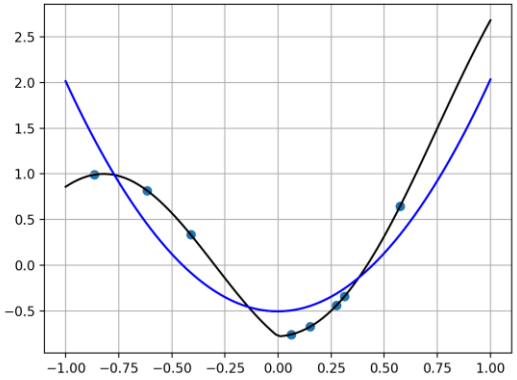
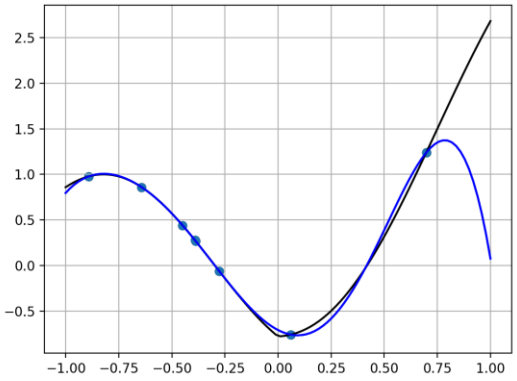
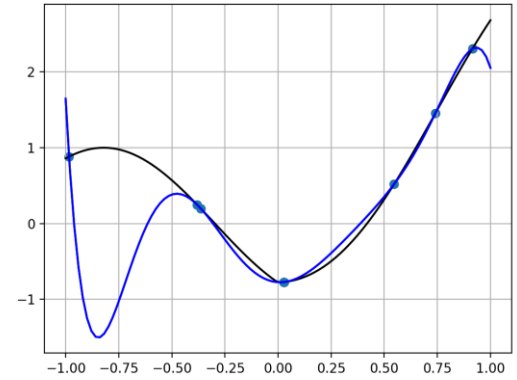
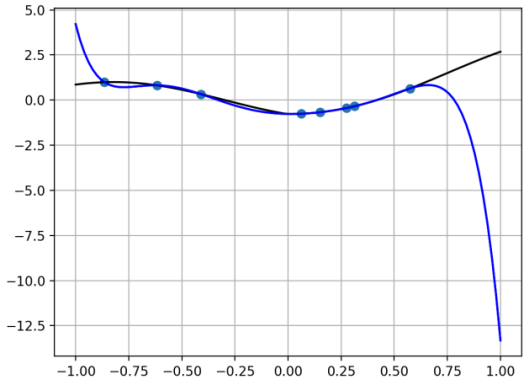
$$\mathbf{w}^i_- = \arg\min_{\mathbf{w}} \frac{1}{2}\sum_{j=1\ldots m, j\neq i}\left(y^j - \boldsymbol{\phi}(\mathbf{x}^j)\mathbf{w}\right)^2$$



Remember that for regular nested grids, we looked at grid points that were one level finer than that used for interpolation/regression

⟶ Expensive procedure, exists in cheaper forms (10-fold CV)

Bias-Variance trade-off (1/2)

	Data set 1	Data set 2	Data set 3
Polynomial order 2		 Biased (most of the time on the same side of the truth)	
Polynomial order 6		 Large variance	

8 datapoints in all cases

Bias-Variance trade-off (2/2)

$$\mathbb{E}\left((y(\mathbf{x}) - f(\mathbf{x}))^2\right) = \mathbb{E}\left((y(\mathbf{x}) - \mathbb{E}(y(\mathbf{x})) + \mathbb{E}(y(\mathbf{x})) - f(\mathbf{x}))^2\right)$$

$$\mathbb{E}\left((y(\mathbf{x}) - f(\mathbf{x}))^2\right) = \underbrace{\mathbb{E}\left((y(\mathbf{x}) - \mathbb{E}(y(\mathbf{x}))\right)^2)}_{\text{Irreducible (zero here)}} + \underbrace{\mathbb{E}\left((\mathbb{E}(y(\mathbf{x})) - f(\mathbf{x}))^2\right)}_{\text{To be controlled}}$$

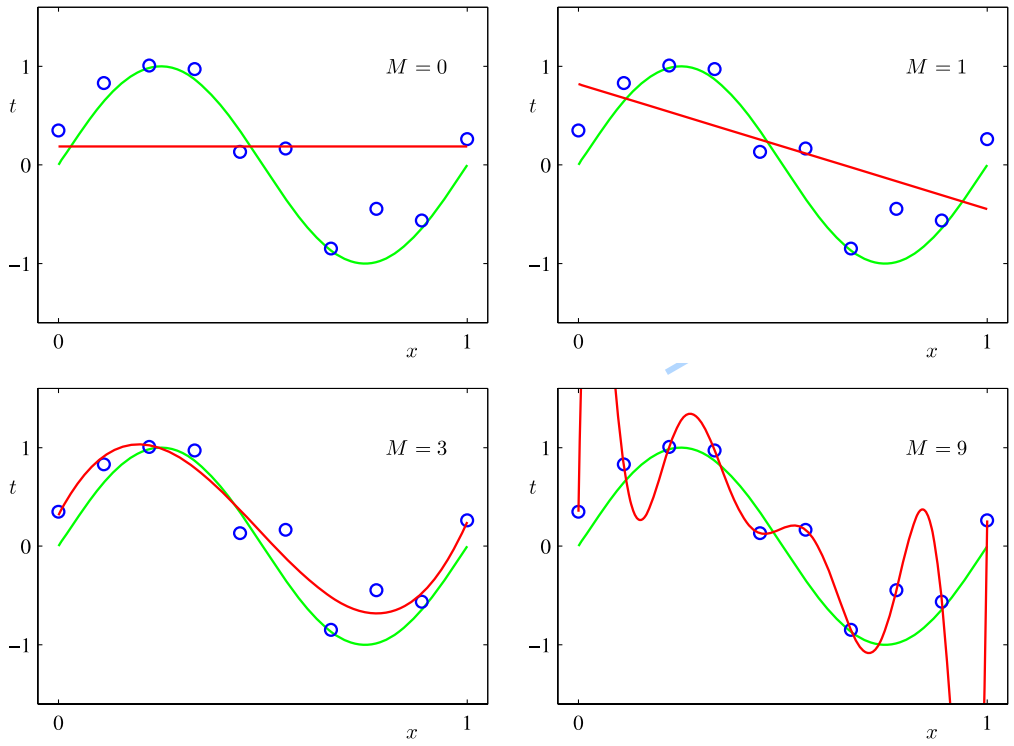
$$\mathbb{E}\left((\mathbb{E}(y(\mathbf{x})) - f(\mathbf{x}))^2\right) = \mathbb{E}\left((\mathbb{E}(y(\mathbf{x})) - \mathbb{E}(f(\mathbf{x})) + \mathbb{E}(f(\mathbf{x})) - f(\mathbf{x}))^2\right)$$

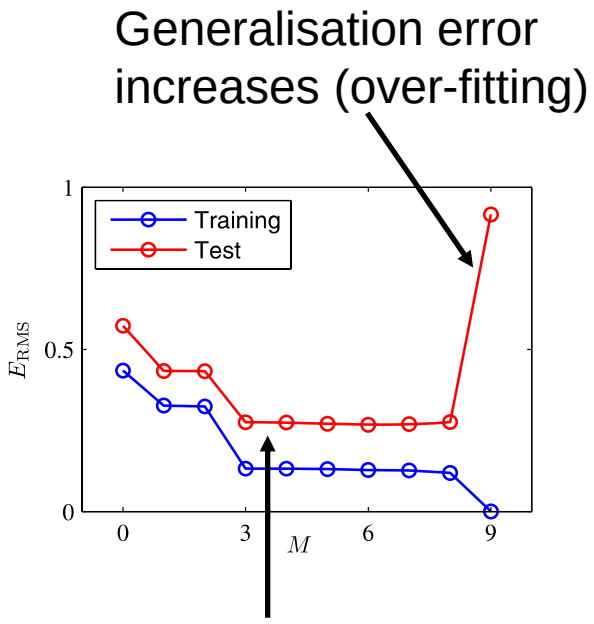
$$\underbrace{\mathbb{E}\left((\mathbb{E}(y(\mathbf{x})) - f(\mathbf{x}))^2\right)}_{\text{Expectation of error}} = \underbrace{\left(\mathbb{E}(y(\mathbf{x})) - \mathbb{E}(f(\mathbf{x}))\right)^2}_{\text{Bias (squared)}} + \underbrace{\mathbb{E}\left(f(\mathbf{x}) - \mathbb{E}(f(\mathbf{x}))\right)^2}_{\text{variance}}$$

Careful, expectations are taken with respect to repeated experiments, which is purely a thought process. In practice, we only have one single dataset = one draw of this virtual experiment

Model selection

Among all the models that can explain the data points, choose the simplest (Occam's razor principle)





Does not generalize well (polynomial order in multiple dimensions, depth AND width of a FCNNet)

Choosing scalar continuous regularisation coefficients may be more efficient than choosing arrays of discrete architectural parameters

Outlines

1. Designs of experiments (DoE)

2. Polynomial surrogate modelling
   a. Tensor-product polynomial approximation spaces
   b. Sparse grids interpolation
   c. Sparse polynomial regression

3. Control of (polynomial) surrogate models
   a. Regularisation methods
   b. Model selection

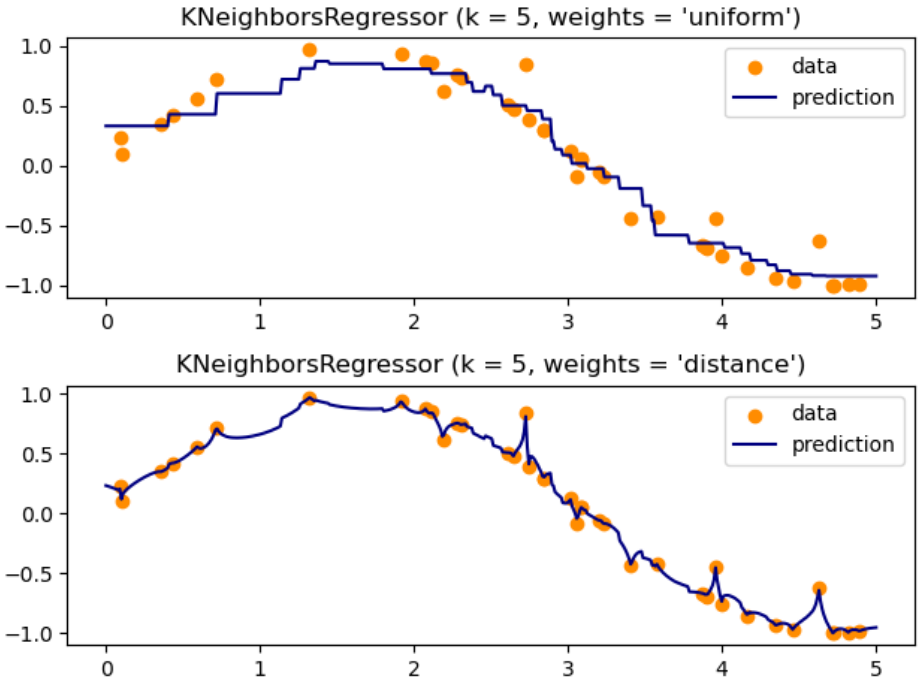
4. Non-parametric surrogate modelling, Gaussian Processes

5. Beyond response surface methodologies using Deep Learning

6. Physics-Informed Neural Nets : surrogate models without numerical solver

Non-parametric regression (1/2)

K-Nearest Neighbors & regression trees (*e.g.* random forest)



Interpolatory KNN via Inverse distance weighting

https://scikit-learn.org/stable/auto_examples/neighbors/plot_regression.html

Hyperparameters: number of neighbours, choice of weighting function

→ Functional representation is learned directly from data (no model selection per say)

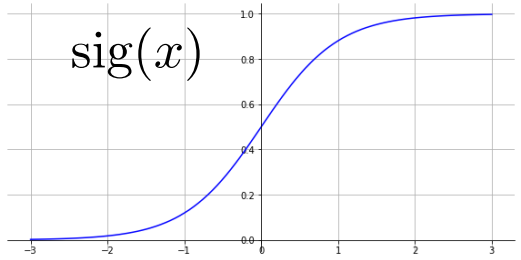
Non-parametric regression (2/2)

Neural Network with one input, one output and one hidden layer

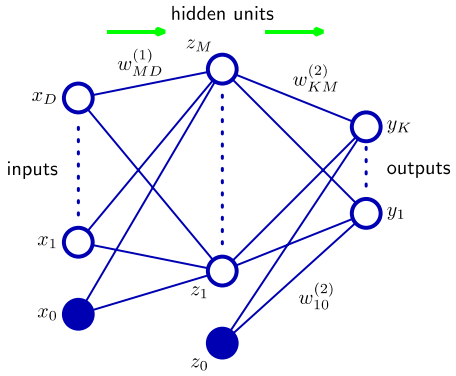
$$y(x) = \sum_{i=1}^{n_\mathrm{h}} w_i^{(2)} a_i(x) + b^{(2)}$$

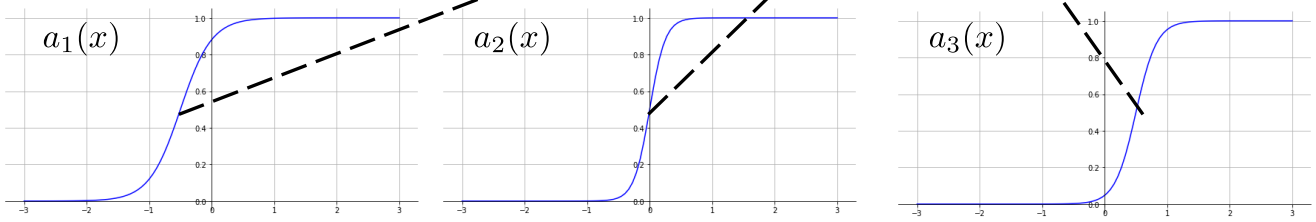
$$a_i(x) = \mathrm{sig}(w_i^{(1)} x + b_i^{(1)})$$

$$\mathrm{sig}(z) = \frac{\tanh(x) + 1}{2}$$



$$y(x) = -1.1 \times a_1(x) + 1.5 \times a_2(x) - 0.6 \times a_3(x) + 0.1$$



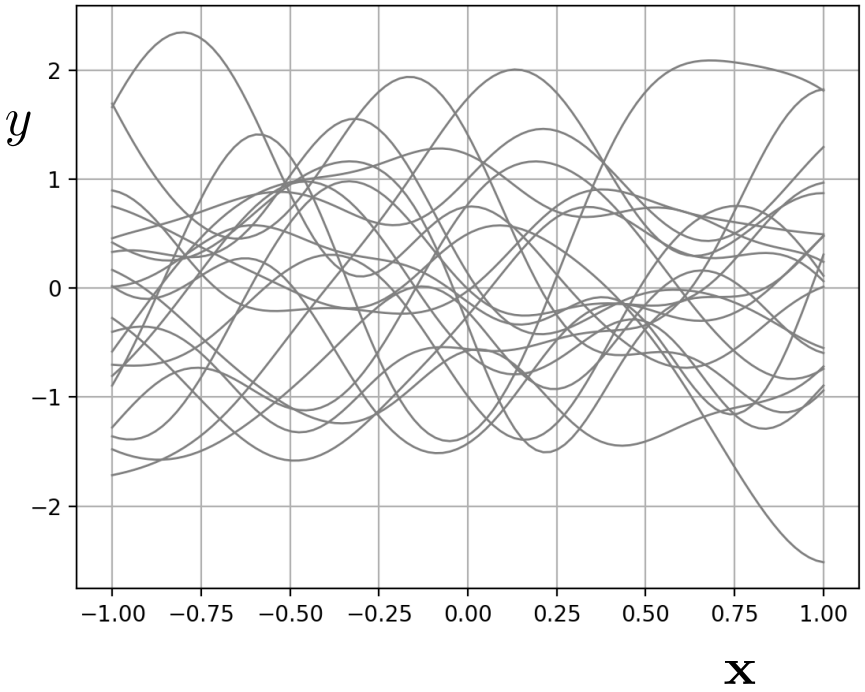


Activations are “sent” by the optimiser in regions of parameter spaces where they are needed to explain variations of the target

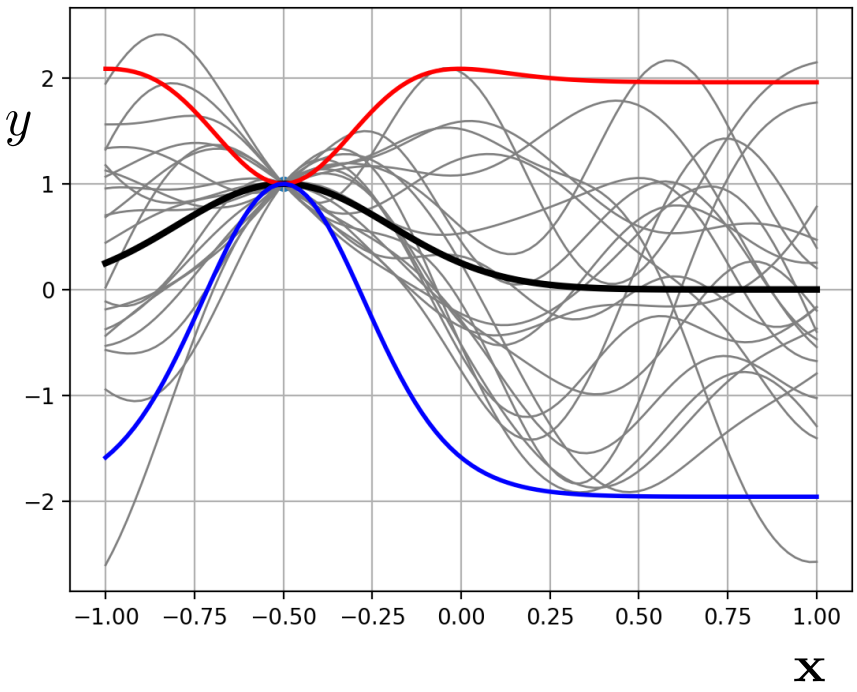
- **Functional representation is learned directly from data (no model selection per say)**
- No free lunch. Nonlinear optimization to fit Neural Nets, large full linear systems to invert for Gaussian Processes, CoD for K-NN and trees, **more data needed as a rule of thumb**

Gaussian Process Regression

Prior over functions. Only smoothness is assumed (covariance structure)



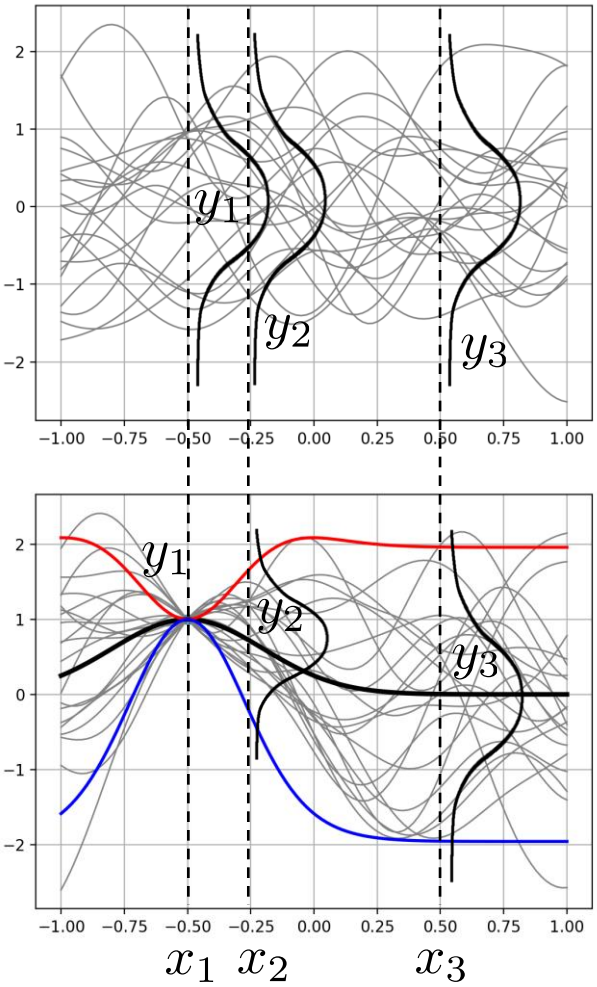
Condition prior to match observations to get posterior distribution over interpolating functions

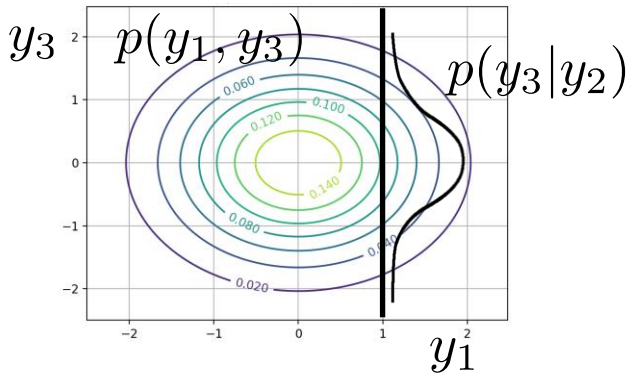


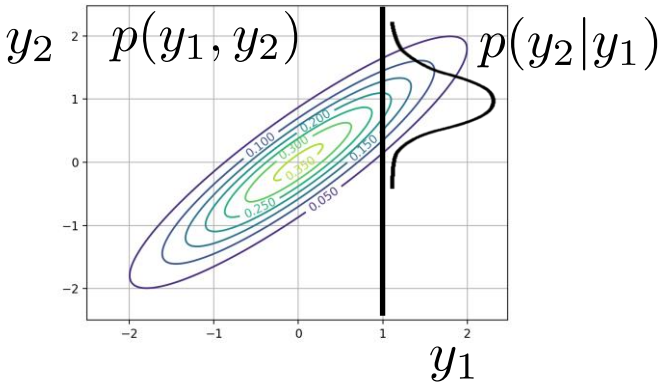
Gaussian Process Regression

$p(\mathbf{Y}) = \mathcal{N}(\mathbf{0}, \mathrm{Cov}(\mathbf{Y}, \mathbf{Y}))$ For any data set

Kernel Covariance: $\mathrm{Cov}(y_i, y_j) = k(\mathbf{x}_i, \mathbf{x}_j) = A \exp\left(-\frac{1}{2l^2}\|\mathbf{x}_i - \mathbf{x}_j\|^2\right)$







Gaussian Process Regression

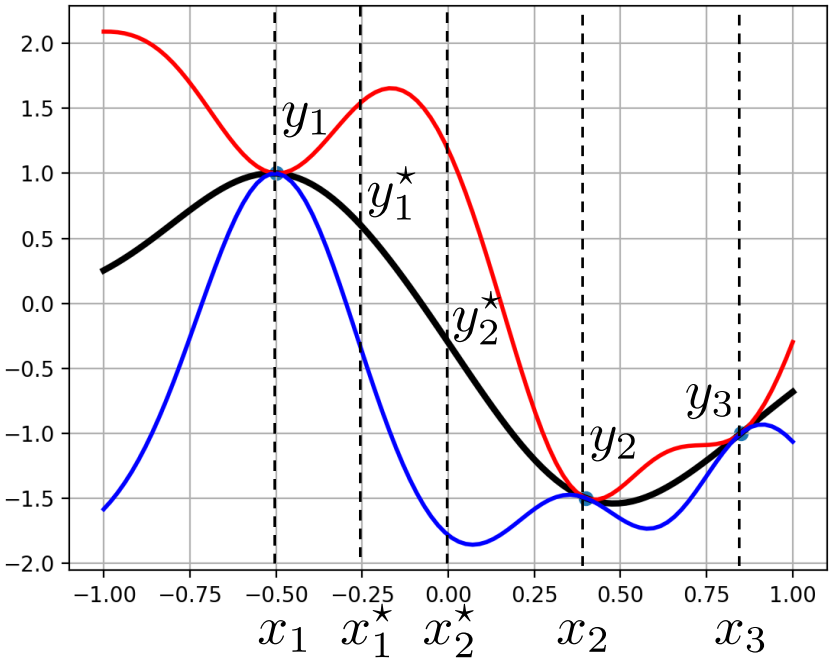
For any test set $\mathbf{X}^\star$ (one point or a full grid for instance), data set $(\mathbf{X}, \mathbf{y})$

$$p(\mathbf{Y}^\star, \mathbf{Y}) = \mathcal{N}\left(\begin{pmatrix}0\\0\end{pmatrix}, \begin{pmatrix}\mathrm{Cov}(\mathbf{Y},\mathbf{Y}) & \mathrm{Cov}(\mathbf{Y},\mathbf{Y}^\star)\\ \mathrm{Cov}(\mathbf{Y}^\star,\mathbf{Y}) & \mathrm{Cov}(\mathbf{Y}^\star,\mathbf{Y}^\star)\end{pmatrix}\right)$$

Predictions : $p(\mathbf{Y}^\star|\mathbf{Y}) = \mathcal{N}(\mathbf{m}^\star, \mathbf{\Sigma}^\star)$ (Bayes' theorem)

$$\mathbf{m}^\star = \mathrm{Cov}(\mathbf{Y}^\star,\mathbf{Y})\left(\mathrm{Cov}(\mathbf{Y},\mathbf{Y})\right)^{-1}\mathbf{Y}$$

$$\mathbf{\Sigma}^\star = \mathrm{Cov}(\mathbf{Y}^\star,\mathbf{Y}^\star) - \mathrm{Cov}(\mathbf{Y}^\star,\mathbf{Y})\left(\mathrm{Cov}(\mathbf{Y},\mathbf{Y})\right)^{-1}\mathrm{Cov}(\mathbf{Y},\mathbf{Y}^\star)$$



$$\mathrm{Cov}(y_i, y_j) = k(\mathbf{x}_i, \mathbf{x}_j) = A\exp\left(-\frac{1}{2l^2}\|\mathbf{x}_i - \mathbf{x}_j\|^2\right)$$

$$\mathrm{Cov}(y_i, y_j^\star) = k(\mathbf{x}_i, \mathbf{x}_j^\star) = A\exp\left(-\frac{1}{2l^2}\|\mathbf{x}_i - \mathbf{x}_j^\star\|^2\right)$$

$$\mathrm{Cov}(y_i^\star, y_j^\star) = k(\mathbf{x}_i^\star, \mathbf{x}_j^\star) = A\exp\left(-\frac{1}{2l^2}\|\mathbf{x}_i^\star - \mathbf{x}_j^\star\|^2\right)$$

Learning hyperparameters

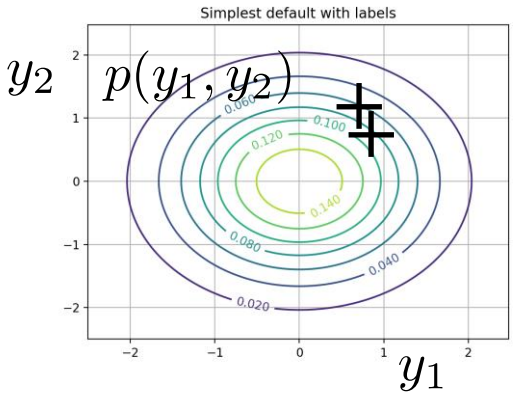
Maximise the data likelihood (model evidence/marginal likelihood)

→ Choose the prior such that the probability of the dataset being generated by the prior is maximized (Bayesian model selection)

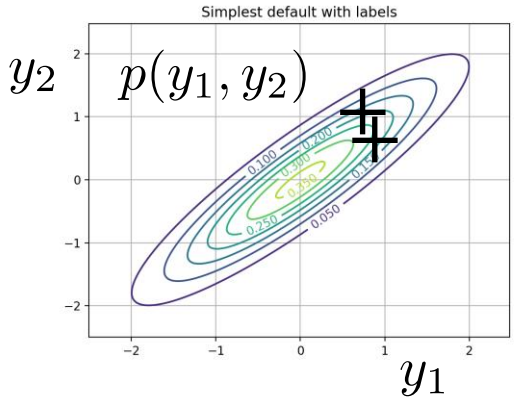
$$\boldsymbol{\theta}^\star = \arg\min_{\boldsymbol{\theta}} \left(\log p(\mathbf{Y};\boldsymbol{\theta})\right) = \arg\min_{\boldsymbol{\theta}} \left(\log p\left(\mathcal{N}(\mathbf{Y};\mathbf{0},\mathrm{Cov}(\mathbf{Y},\mathbf{Y})_{|\boldsymbol{\theta}})\right)\right)$$

$$\log p(\mathbf{Y};\boldsymbol{\theta}) = -\frac{1}{2}\mathbf{Y}^T\left(\mathrm{Cov}(\mathbf{Y},\mathbf{Y})\right)^{-1}\mathbf{Y} - \frac{1}{2}\log\left|\mathrm{Cov}(\mathbf{Y},\mathbf{Y})\right| - \frac{m}{2}\log 2\pi$$

Illustration with 2 datapoints

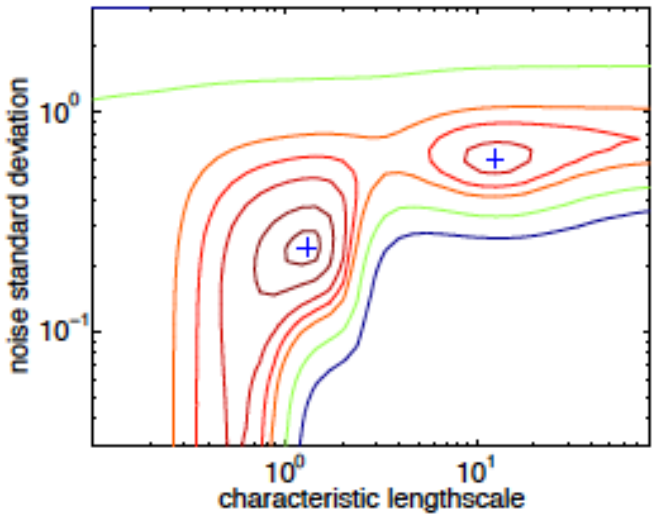


Likely

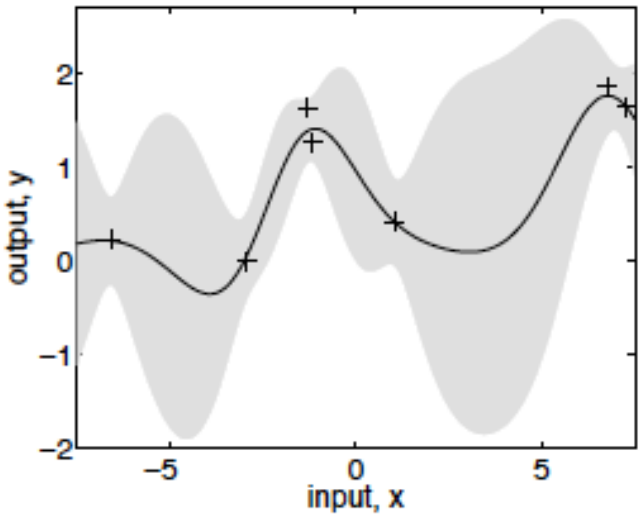


More likely

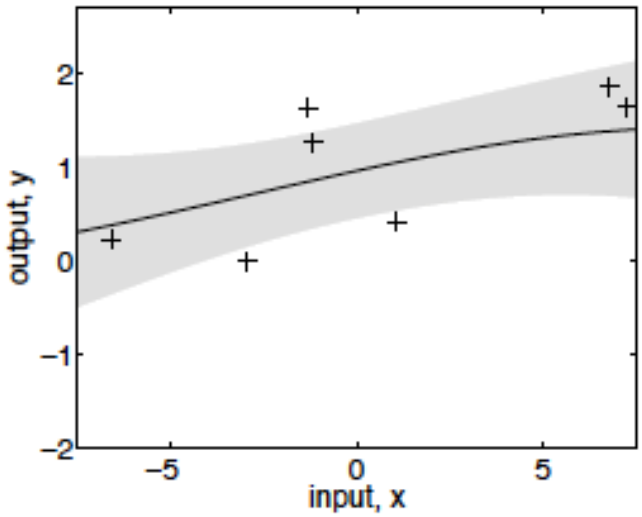
Learning hyperparameters



(a)



(b)

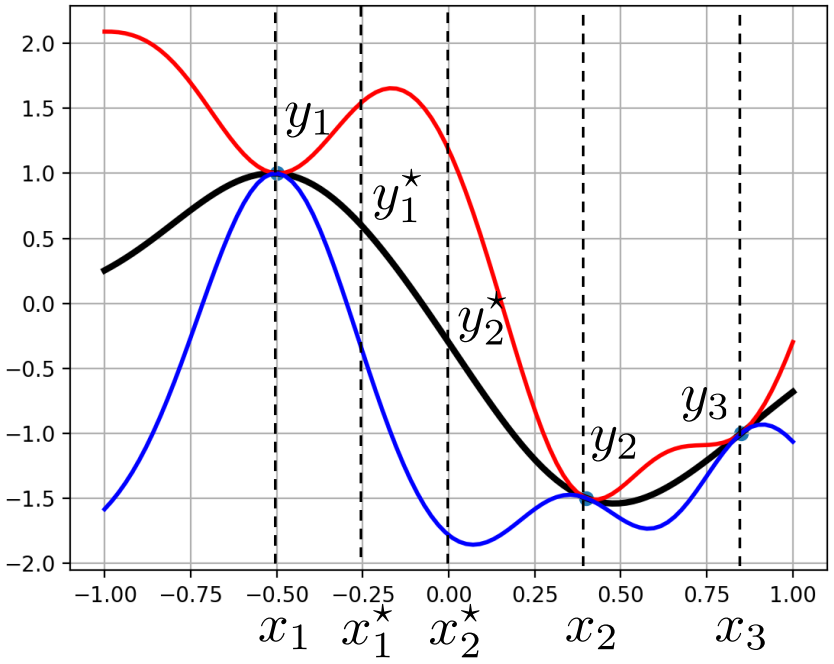


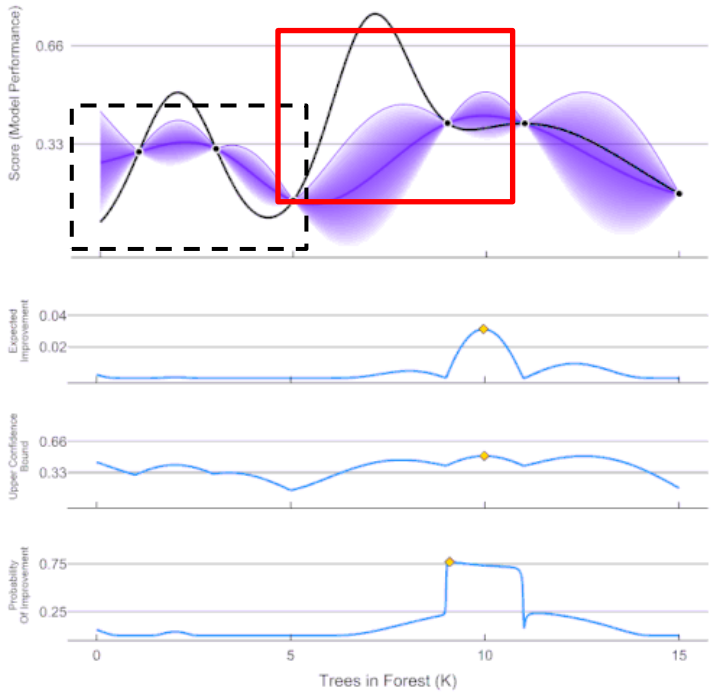
(c)

Select data adaptively for optimization

Not all inputs interesting. Use uncertainty associated with the generative model to guide data acquisition in a greedy, goal-oriented manner

https://en.wikipedia.org/wiki/Bayesian_optimization





Bayesion opimisation: acquire data where changes of improving loss is high, *e.g.* optimisation and exploration performed jointly (gradient versions also exist)

Outlines

1. Designs of experiments (DoE)

2. Polynomial surrogate modelling
   a. Tensor-product polynomial approximation spaces
   b. Sparse grids interpolation
   c. Sparse polynomial regression

3. Control of (polynomial) surrogate models
   a. Regularisation methods
   b. Model selection

4. Non-parametric surrogate modelling, Gaussian Processes

5. Beyond response surface methodologies using Deep Learning

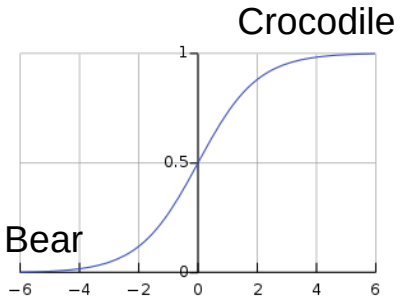
6. Physics-Informed Neural Nets : surrogate models without numerical solver

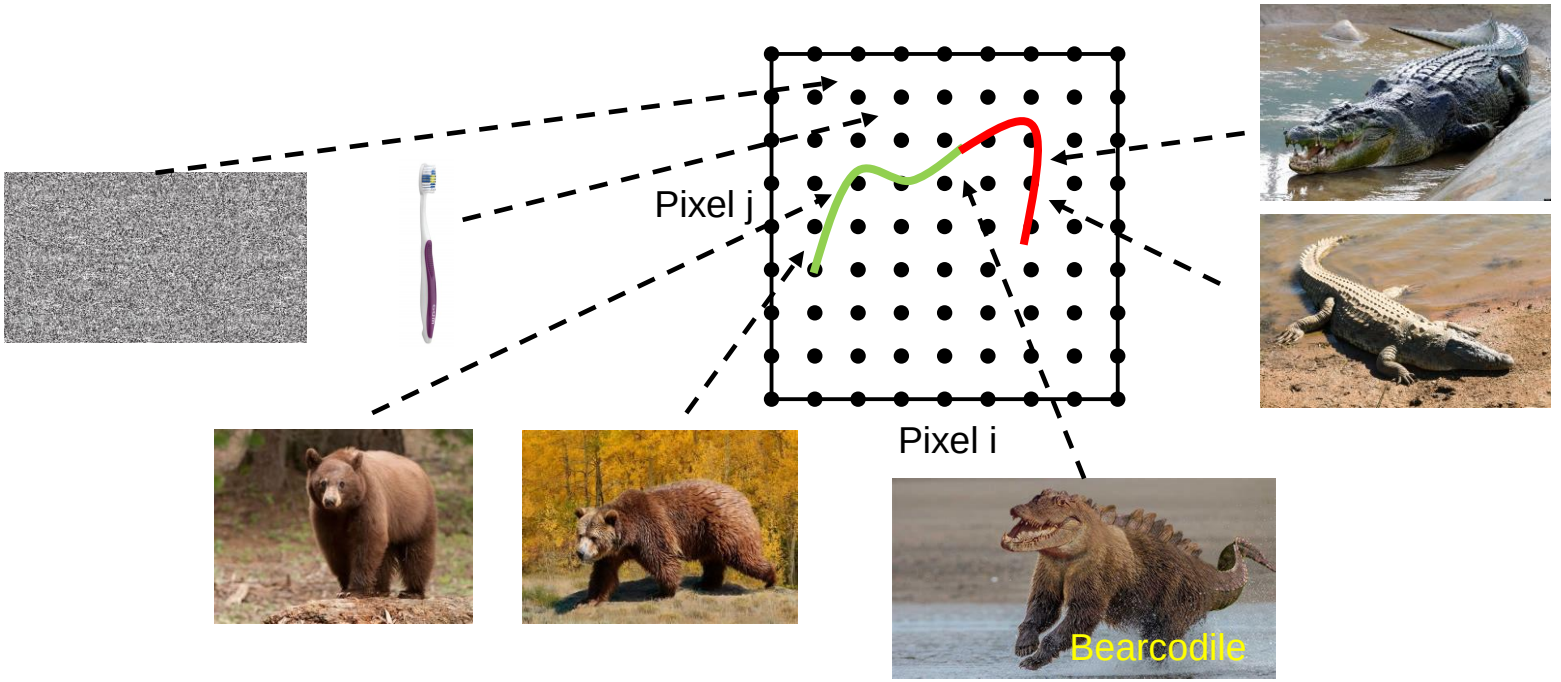
Regression over implicit manifolds

Not all inputs will be of practical interest, *e.g.*

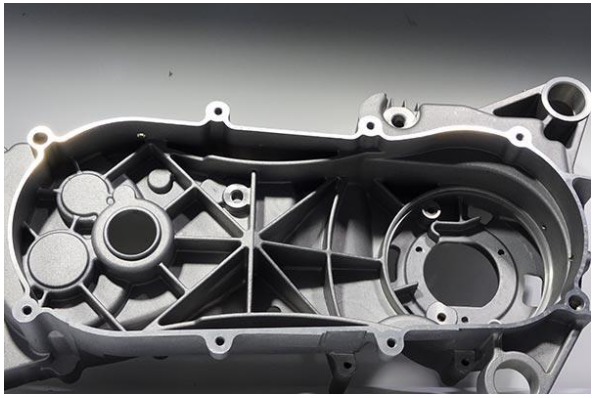
- Unphysical regions of parameter dimensions
- Parameter values that are of no interest within a given set of applications (*e.g.* far from optima)

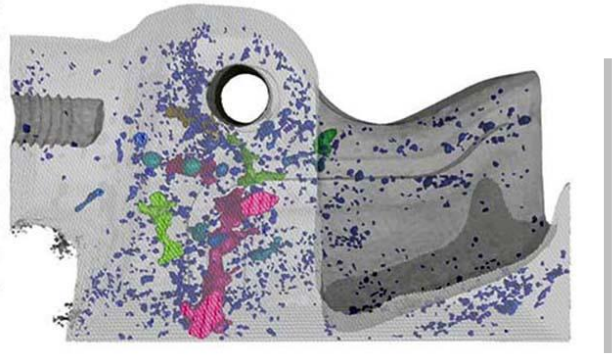
→ Manifolds of interest defined implicitly via training examples = **machine learning**

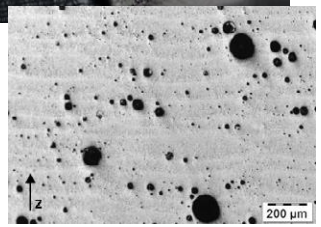




Porosity in casting and 3D printing

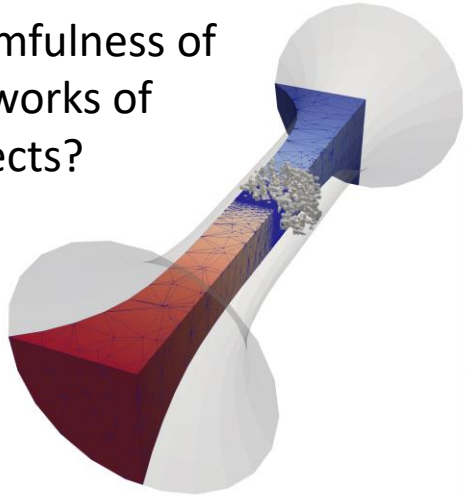


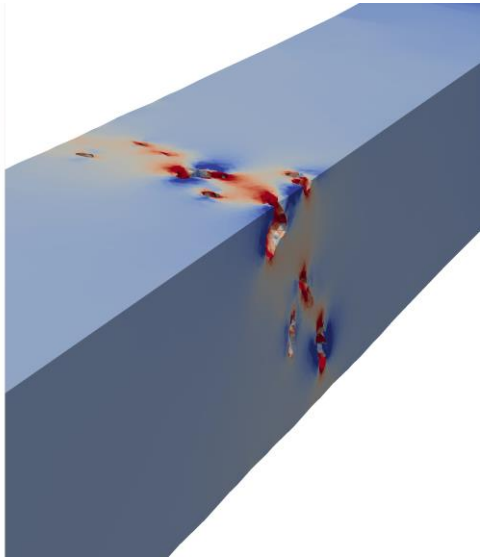


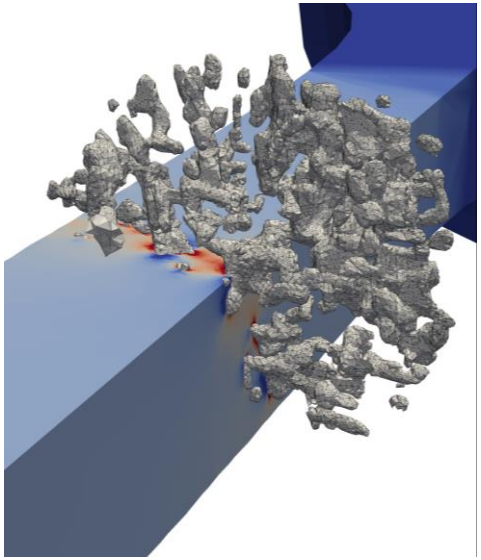


https://www.starrapid.com/blog/porosity-in-pressure-die-casting-and-how-to-control-it/

Harmfulness of networks of defects?





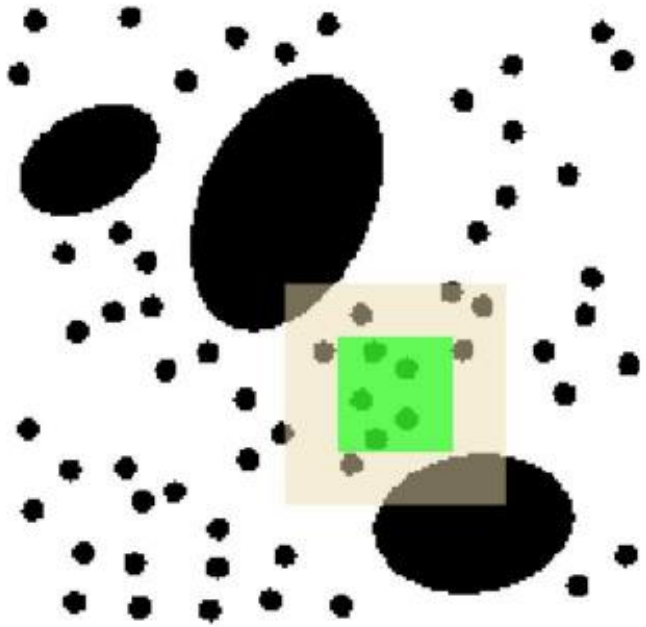


[Kerfriden, (Mines PT), V. Chiaruttini, S. Claus (Onera) & L. Marcin (SAFRAN)]

Can we replace the FEA simulation by a NNet for online inference?

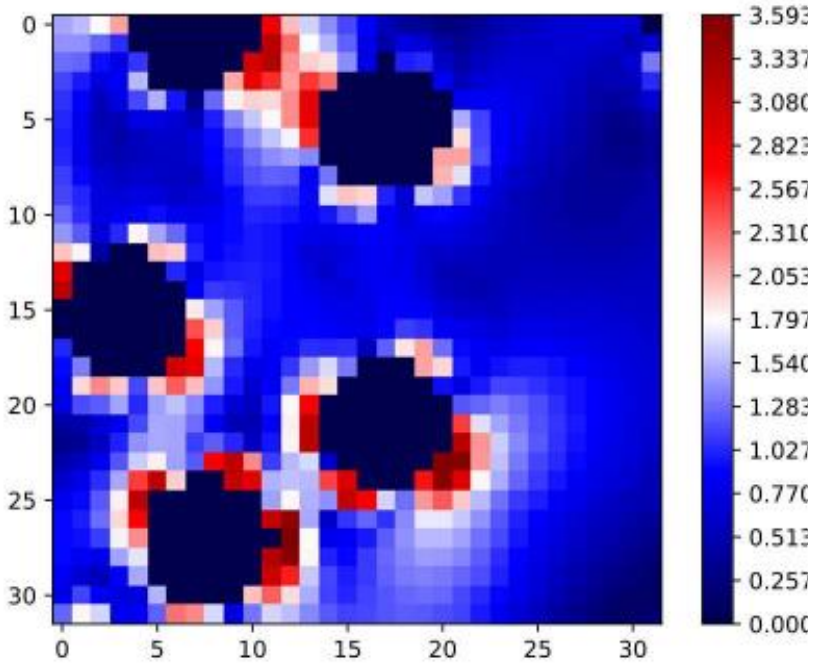
Images as parameters

Input



64 x 64 = 4096 pixels

Output: Von Mises stress measure



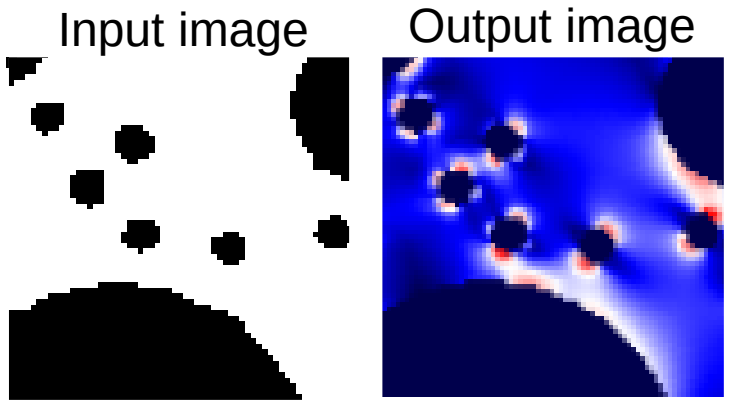
32 x 32 image

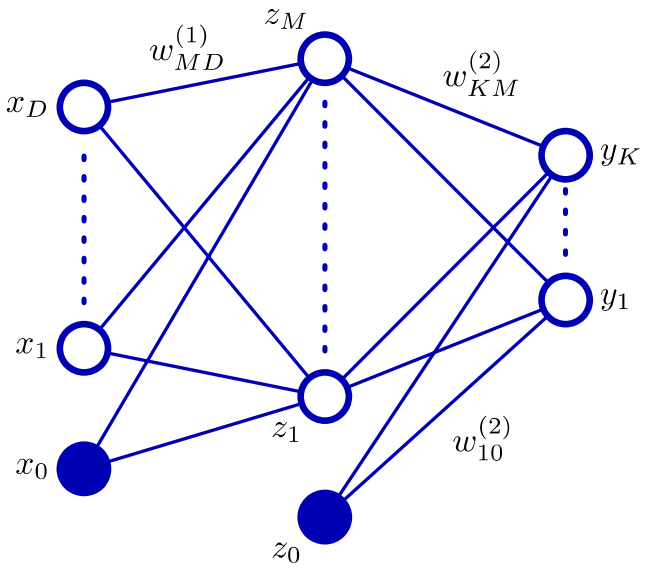
Output dimension is not a problem, but predicting, say, one of the pixel in the output image is a scalar regression problem in dimension 4096!

We hope that feasible inputs live in a small portion of this massive parametric space

Naïve FCN approach for images

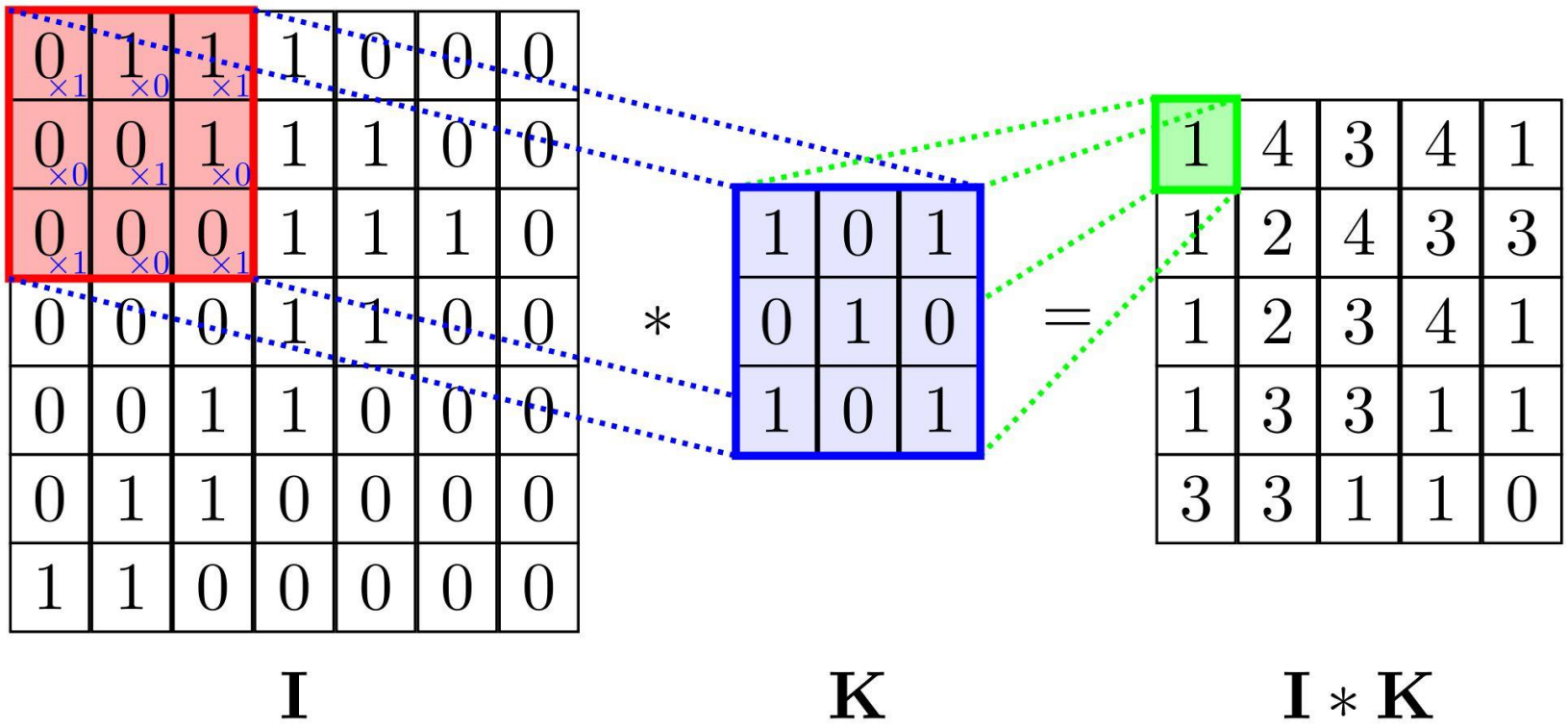
- Voxelise inputs (and outputs)
- Concatenate voxel features in 1D array :
  - N_{voxel} * N_{input} scalar inputs
  - N_{output} scalar outputs for structure-level predictions
  - N_{voxel} * N_{output} output scalars for voxel-level predictions



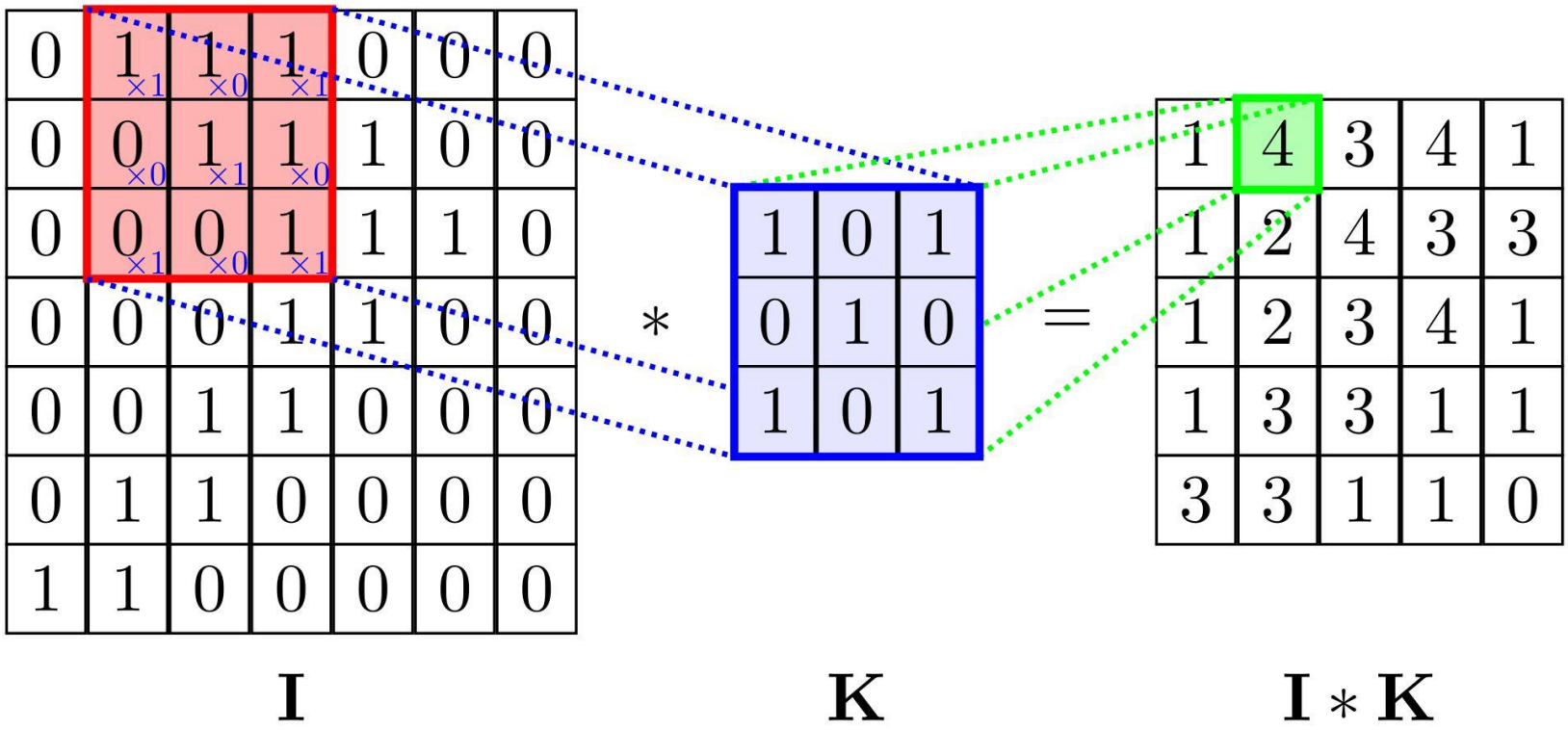


Limitations:
- **Very large number of parameters** O($(N_{voxel}\, N_{input})^2$)
- No use of **connections between voxels**, *spatial proximity is lost*
- No use of **translational invariant properties** (local input pattern may produce the same output effect irrespectively of its spatial position)

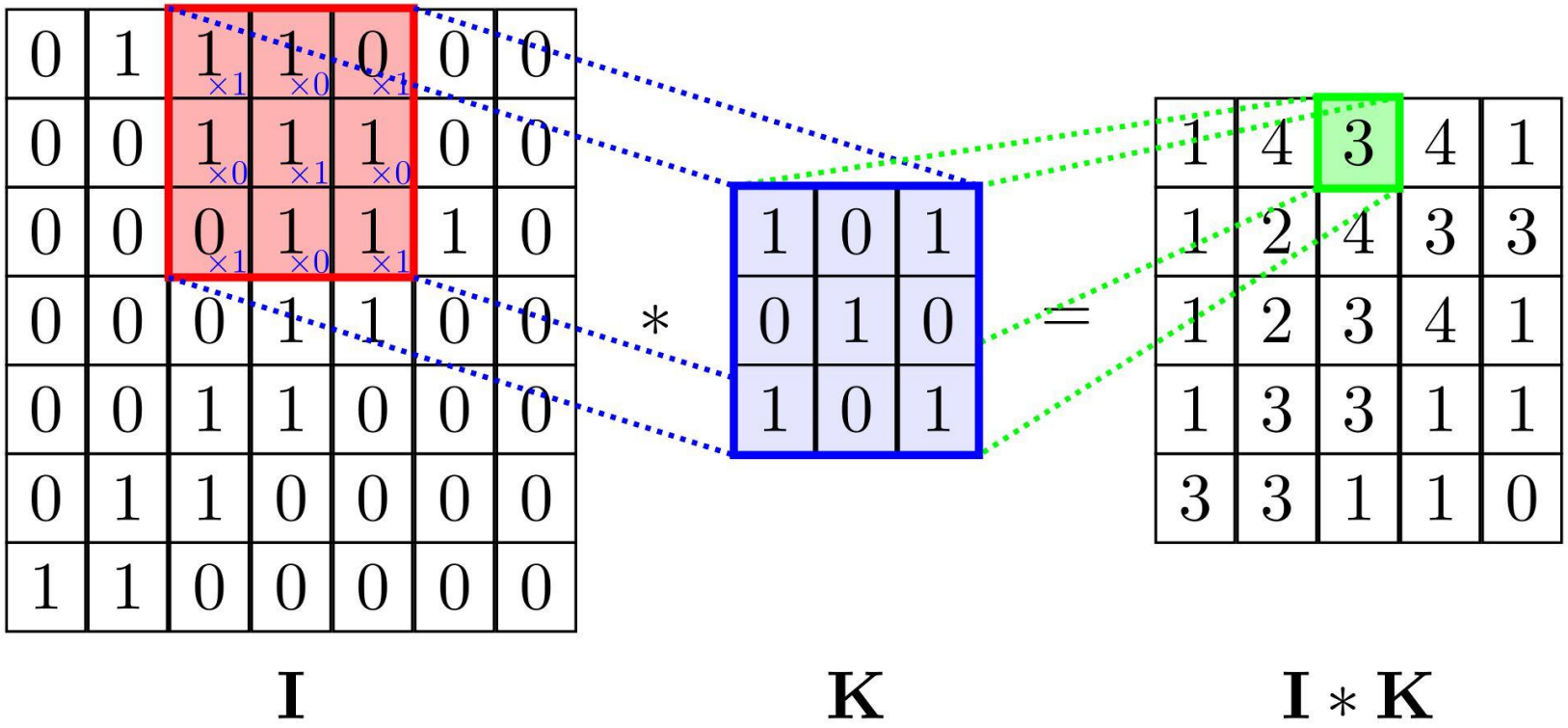
Learnable convolution filters



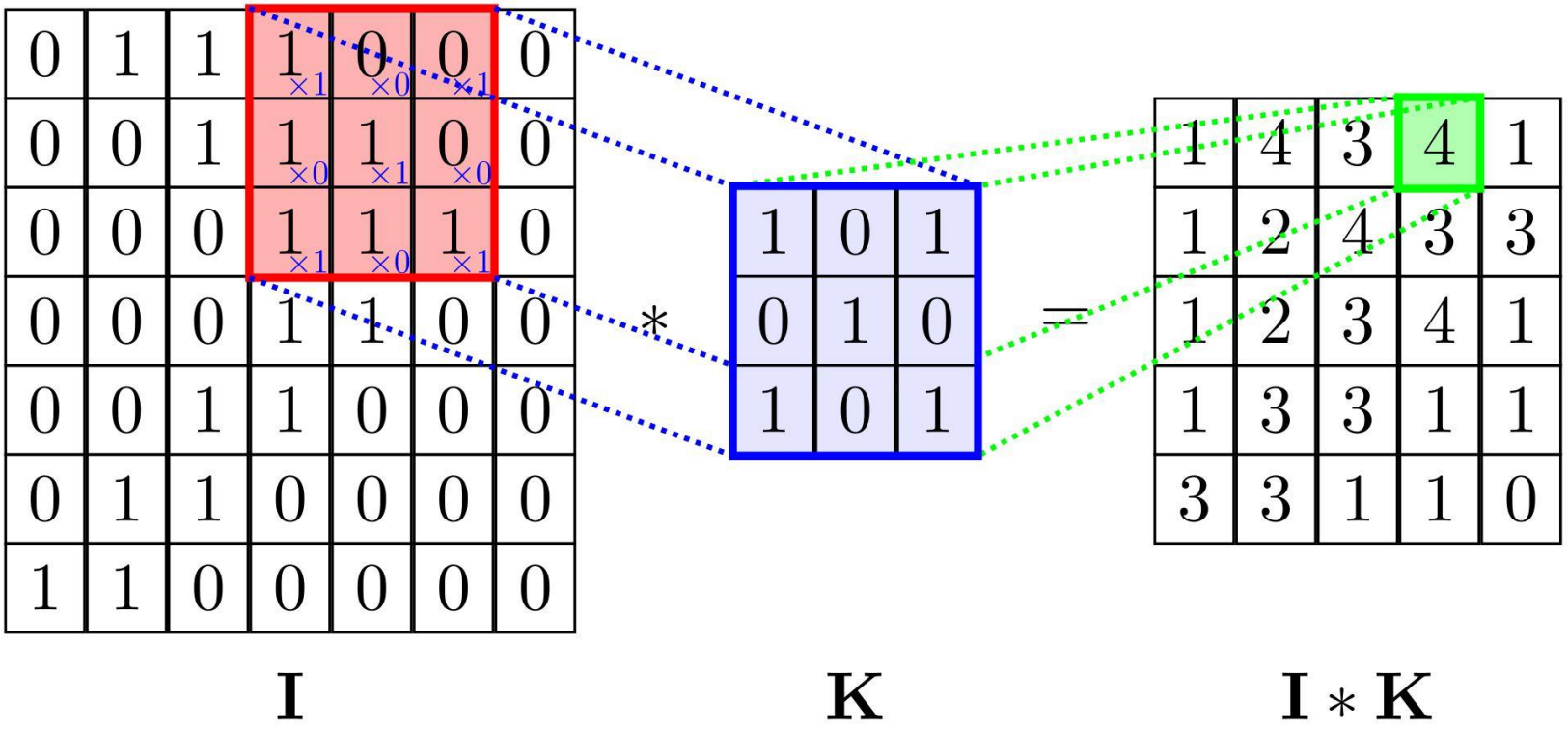
Learnable convolution filters



Learnable convolution filters



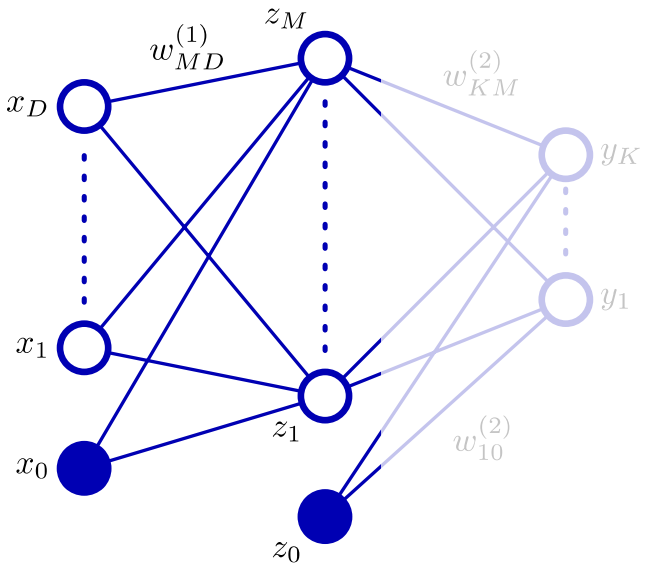
Learnable convolution filters



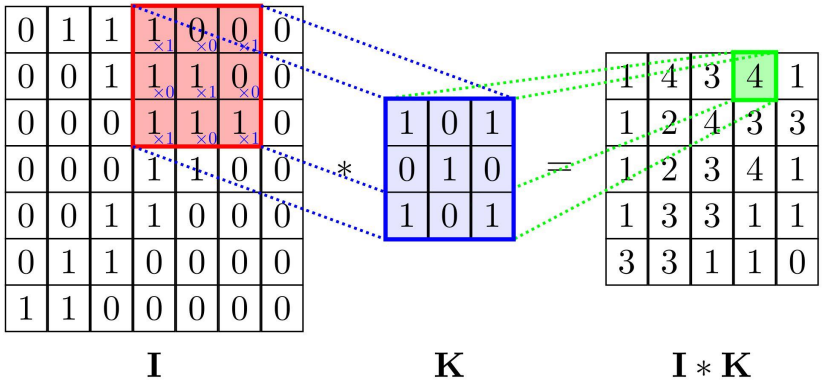
- Weights of convolution filter shared across the entire domain
- Local diffusion of information (stack layers or max-pool for deeper diffusion)
- Translational invariance: a 3x3 input pattern will produce the same output, regardless of the absolute position of the pattern in the input grid

FCN versus CNN

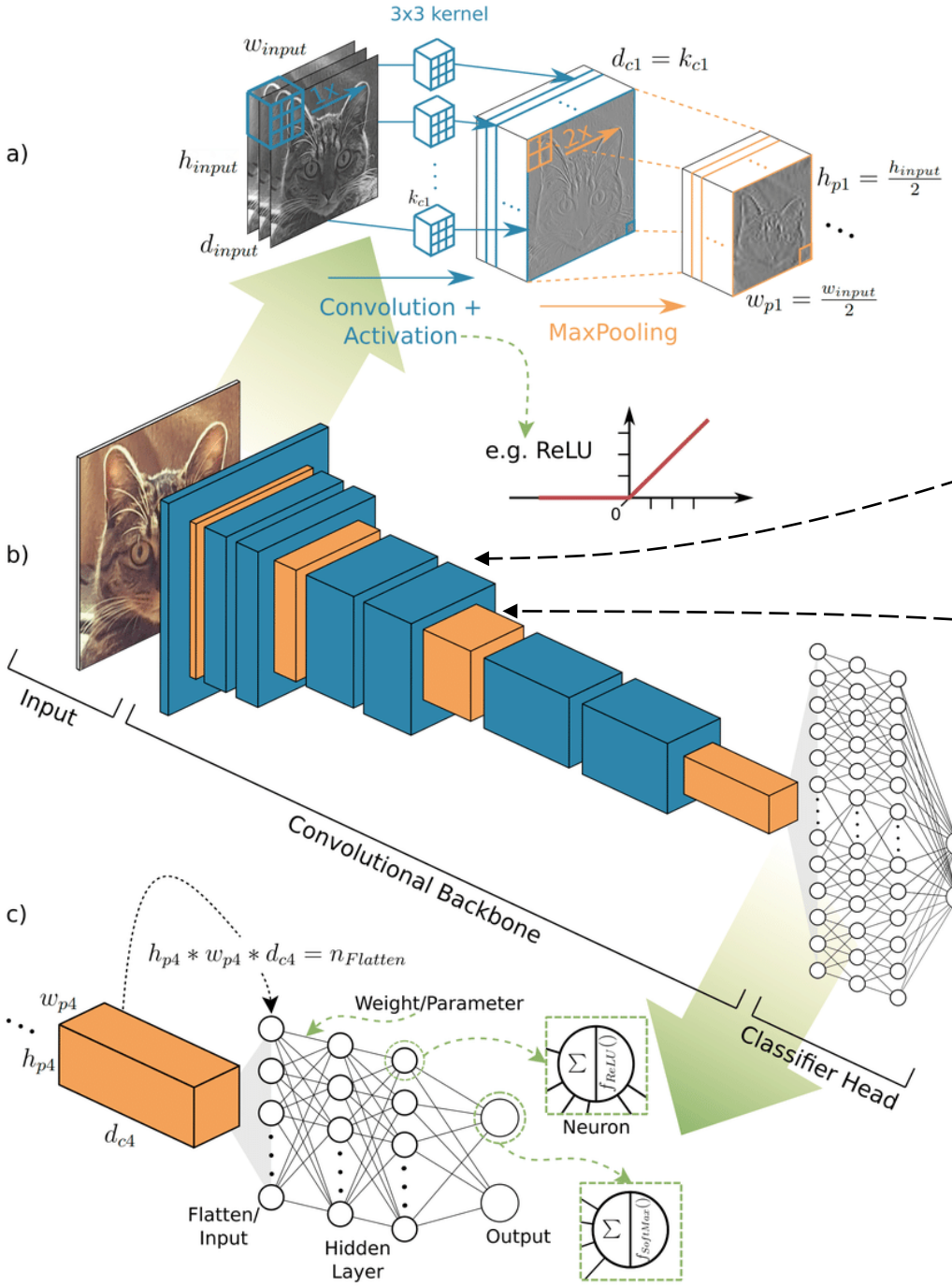
Transformation of one image into another image with O(N^2) weights, spatiality is lost in hidden layers



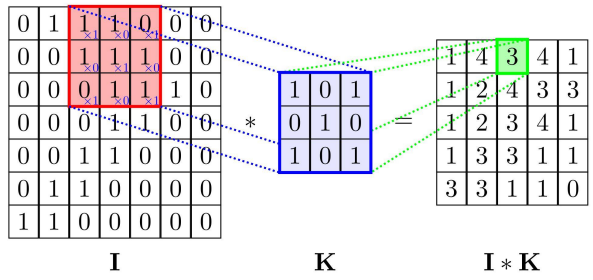
Transformation of one image into another image with O(M) weights, M number of pixels of Kernel K, spatiality is preserved in hidden layers



Chain convolutions to obtain more expressive network and increase receptive size. Use convolutions at different scales, *e.g.* through pooling

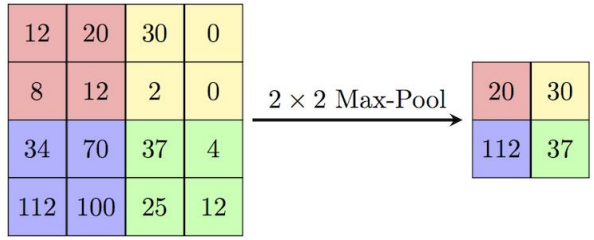


Convolution



translational equivariance

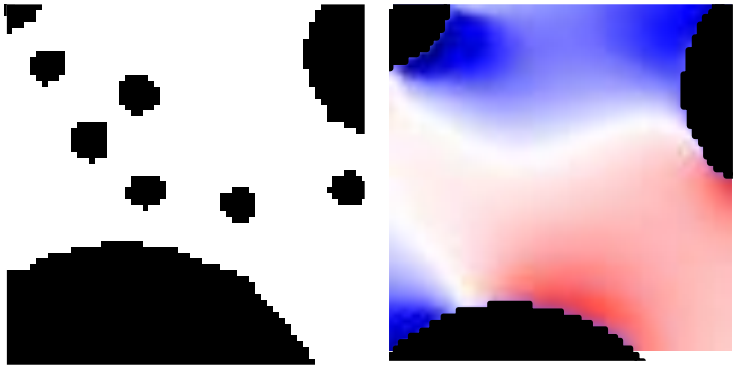
Max pooling



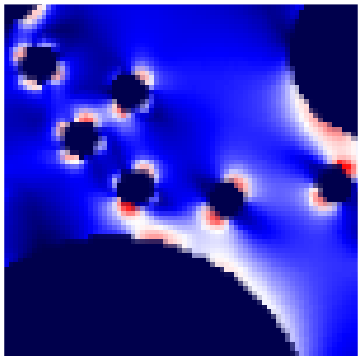
some translational invariance

Multiscale StressNet

Input image (multi-channel)



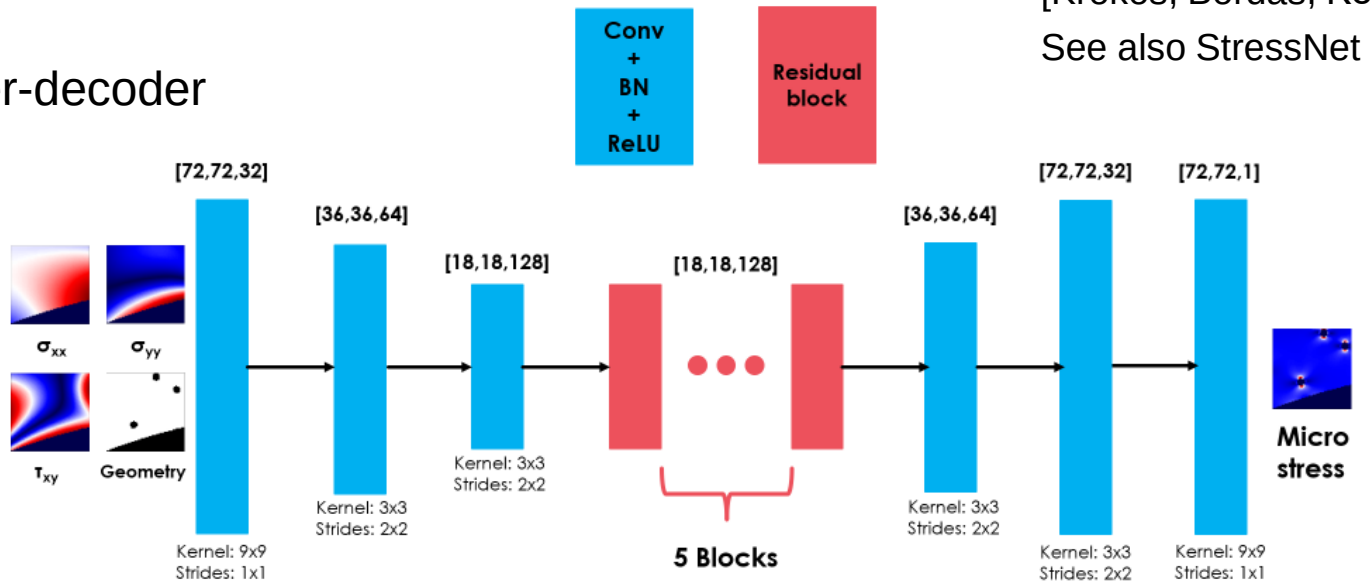
Output image



[Krokos, Bordas, Kerfriden '21]

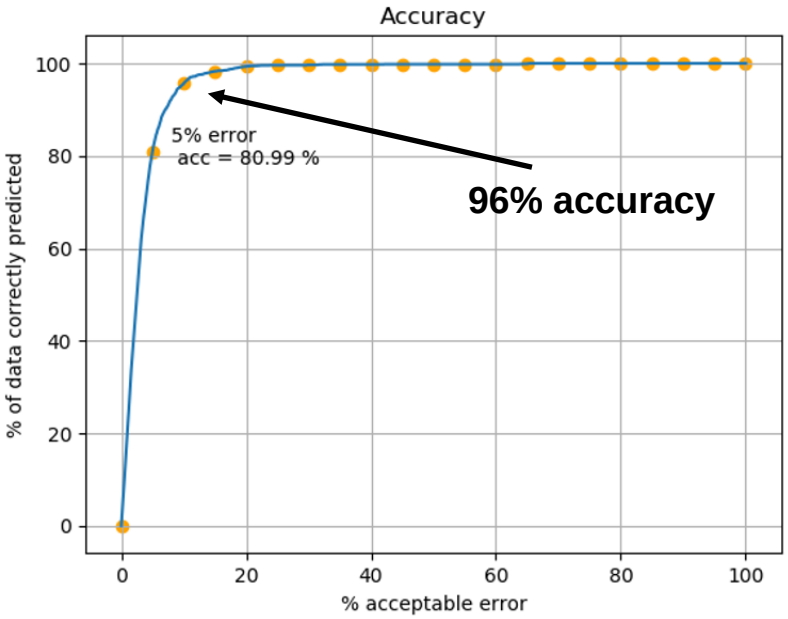
See also StressNet [Nie et al.'19]

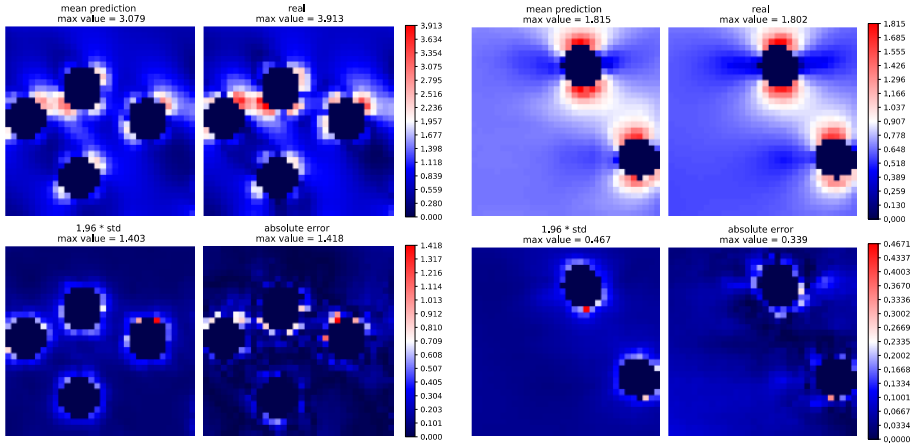
Encoder-decoder

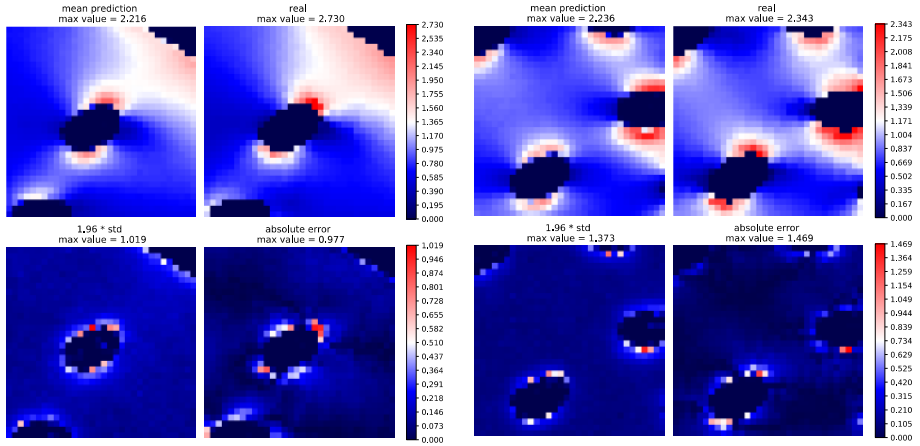


Unets are also very popular image-to-image architectures

Performances





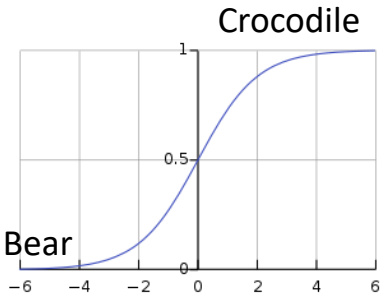


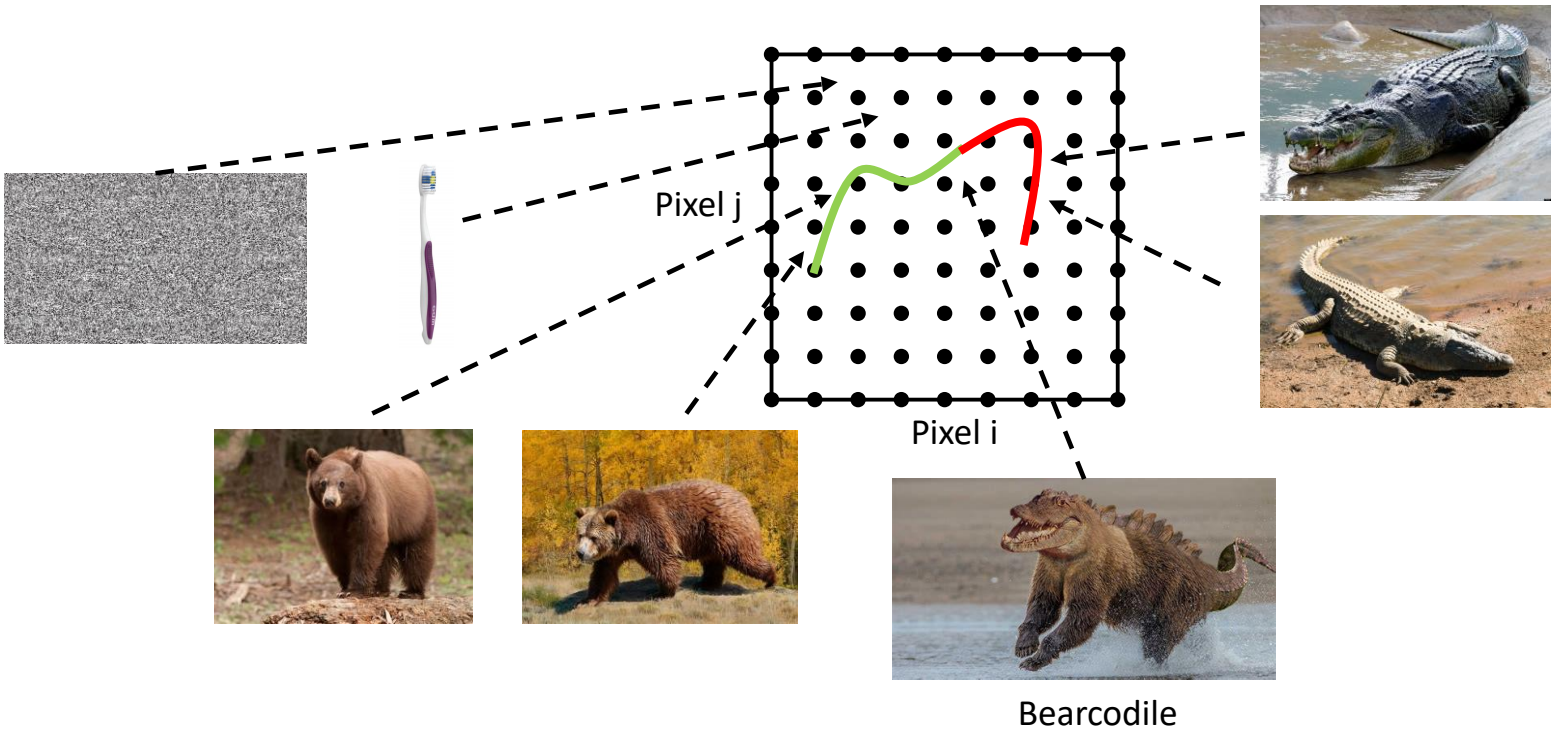
Drops to 70% accuracy when applied to unseen defect morphology (ellipses instead of circles)

- ⟶ Price to pay for learning through examples in huge dimensions
- ⟶ Can we do better? *more later*

[Krokos, Kerfriden et al. 2020]

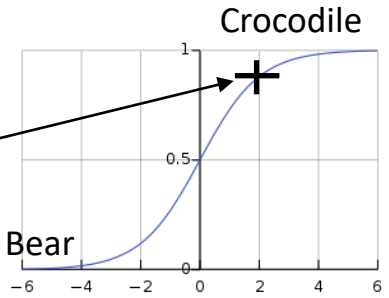
Poor extrapolation capabilities (1/2)

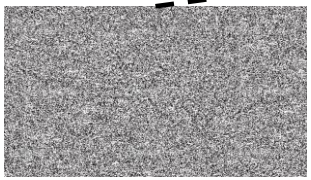


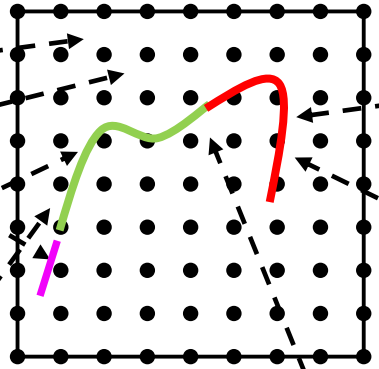


Poor extrapolation capabilities (2/2)

Not recognised as a bear because no training bear examples in the snow!



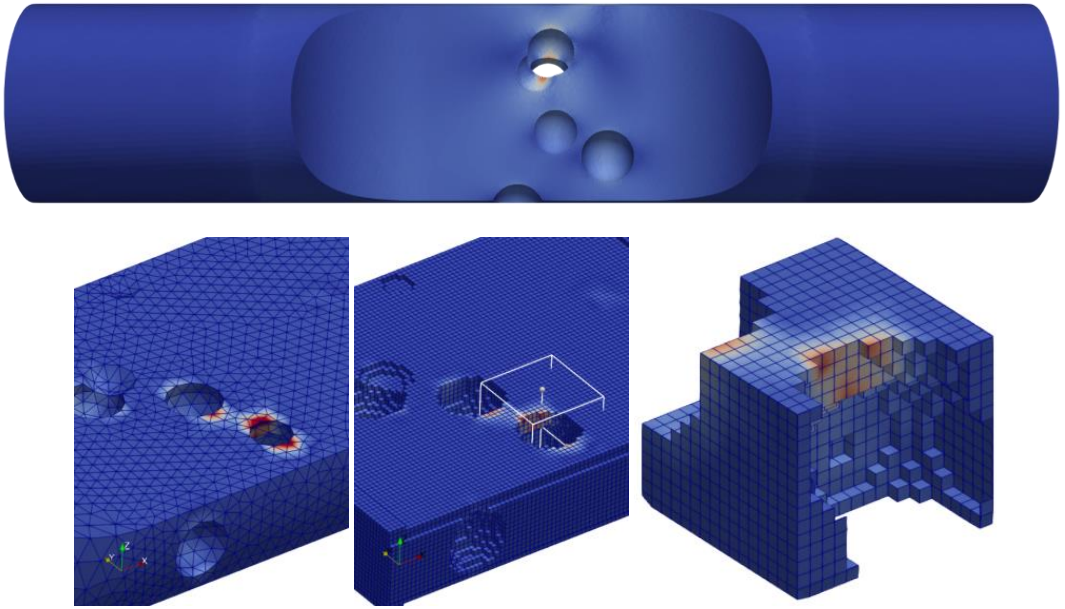




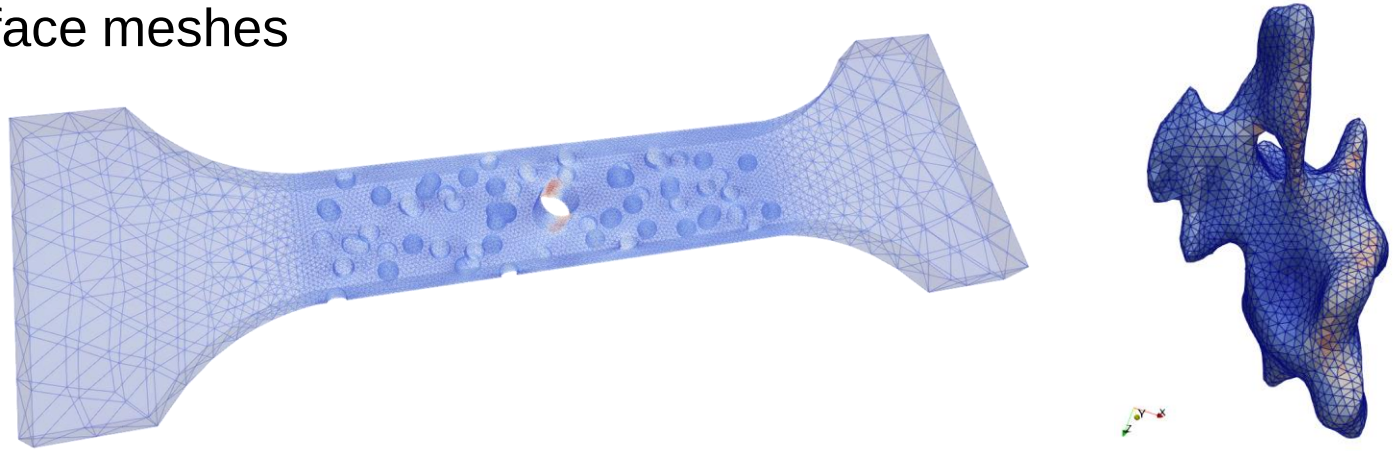
Bearcodile

3D stress surrogate models

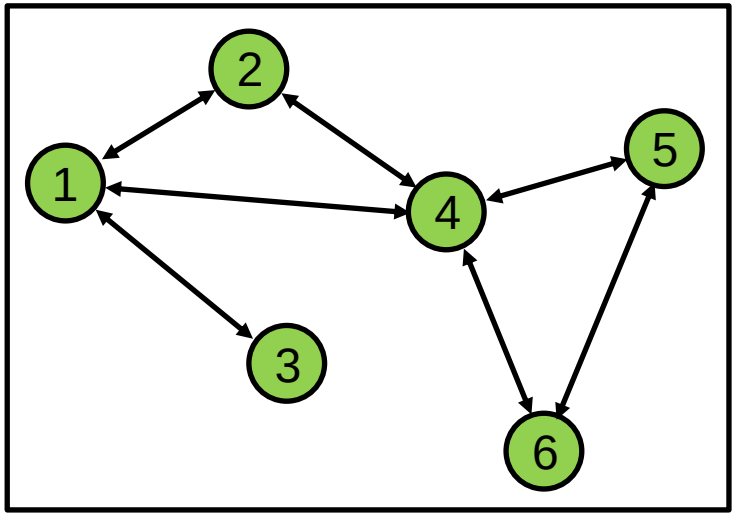
- Can be done but 3D convolutions are expensive and memory intensive

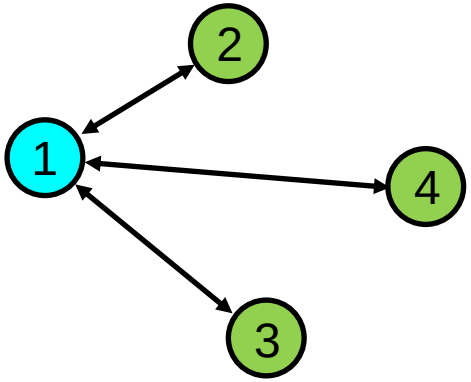


- Better to work on sparser representation of the geometry, for instance surface meshes



Message-passing Graph Neural Nets





Prepare edge messages, using a unique FNC

$$\hat{\mathbf{h}}^{(14)} = \begin{pmatrix} \mathbf{h}^{(v^{(1)})} \\ \mathbf{h}^{(v^{(4)})} \end{pmatrix} \quad \mathbf{m}^{(14)}$$

$$\hat{\mathbf{h}}^{(13)} = \begin{pmatrix} \mathbf{h}^{(v^{(1)})} \\ \mathbf{h}^{(v^{(3)})} \end{pmatrix} \quad \mathbf{m}^{(13)}$$

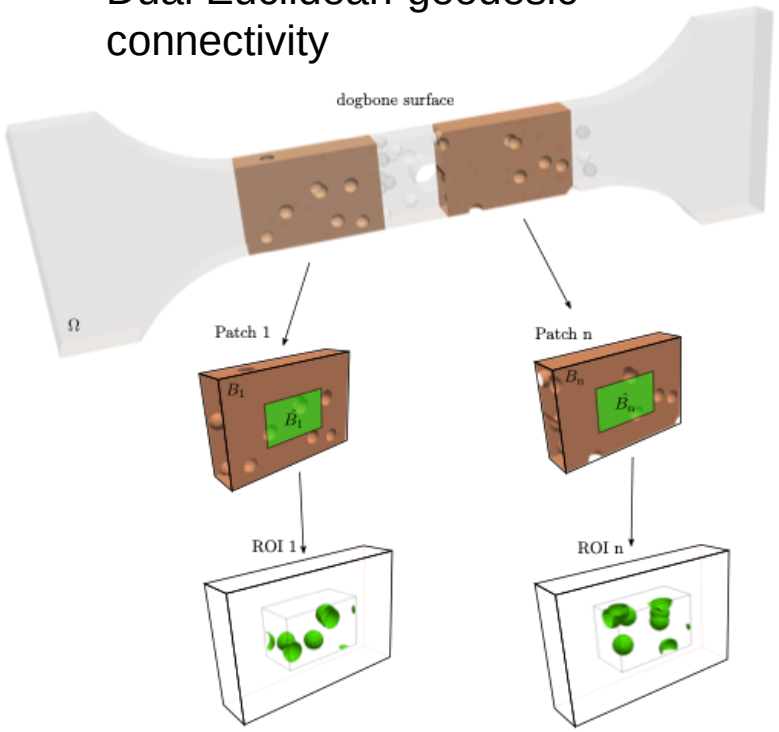
$$\hat{\mathbf{h}}^{(12)} = \begin{pmatrix} \mathbf{h}^{(v^{(1)})} \\ \mathbf{h}^{(v^{(2)})} \end{pmatrix} \quad \mathbf{m}^{(12)}$$

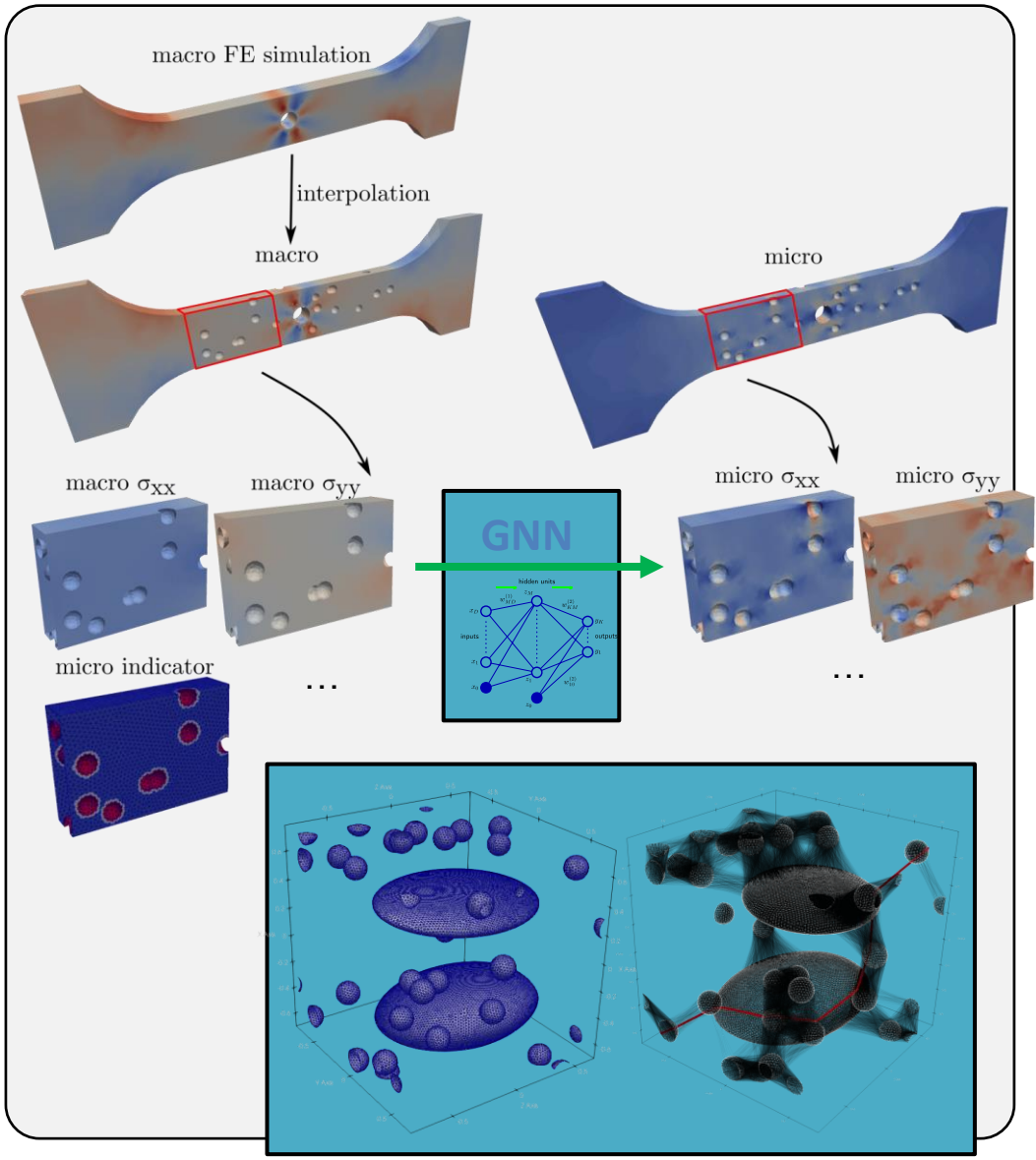
Aggregate messages in a symmetry-preserving manner

$$\mathbf{h}'^{(v^{(1)})} = \phi \left(\bigoplus_{e \in \mathcal{N}_e(v^{(1)})} \mathbf{m}^{(e)}, \mathbf{h}^{(v^{(1)})} \right)$$

Geometric MS StressNet: principle

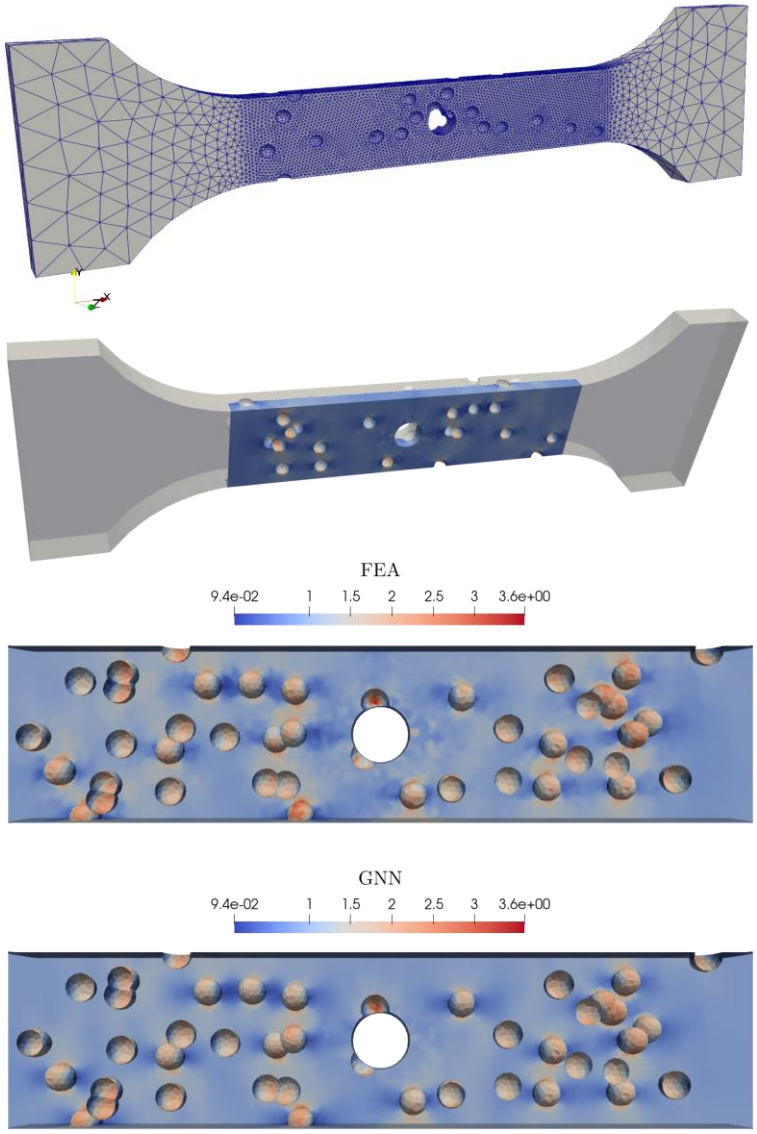
- Procedure
  - ❑ compute 3D macro & micro stress fields using FEA
  - ❑ Project onto surface mesh
  - ❑ decompose into patches
  - ❑ Use GNN to predict micro QoI
- Dual Euclidean-geodesic connectivity

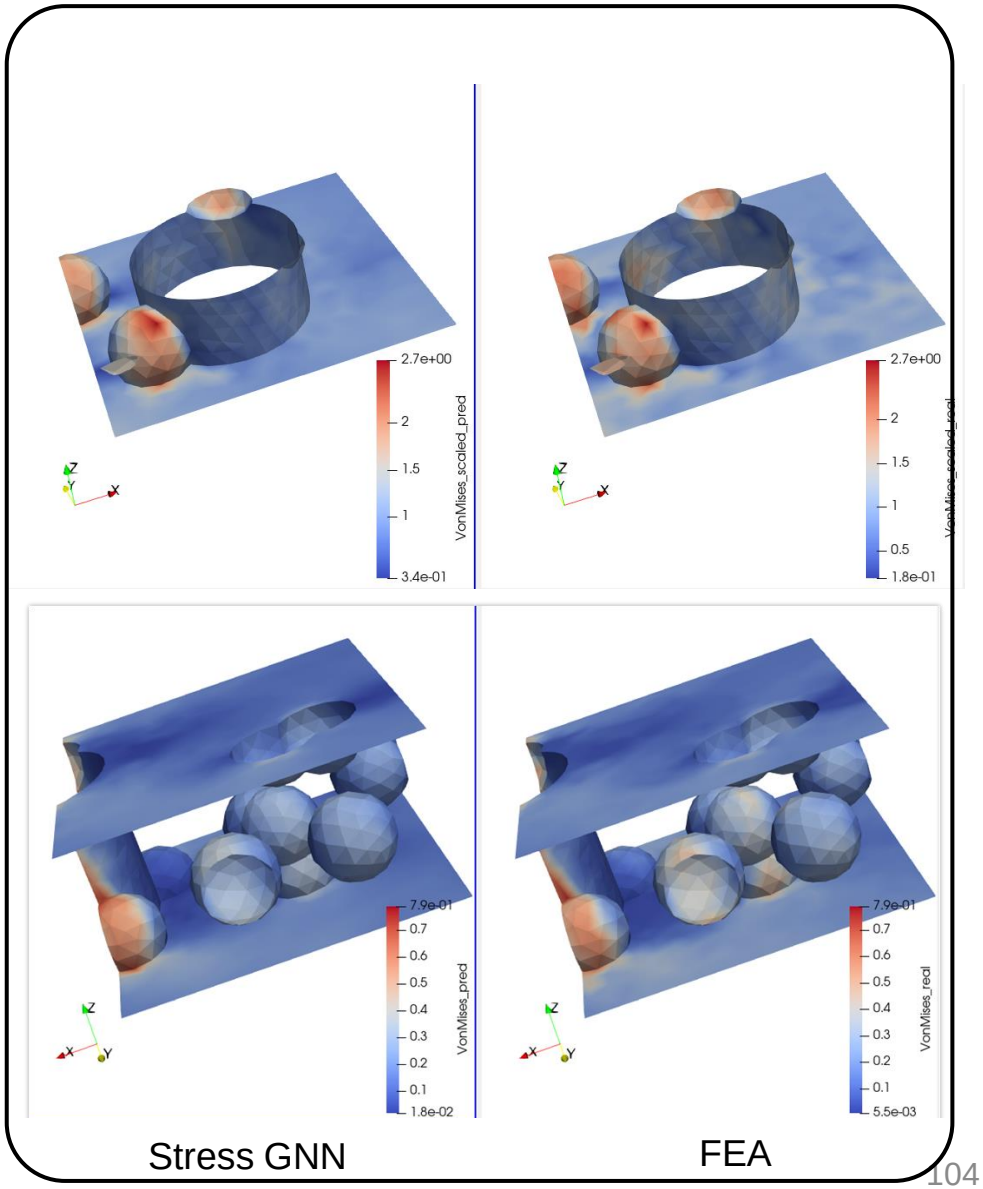




Examples of GNN predictions

Training & validation examples

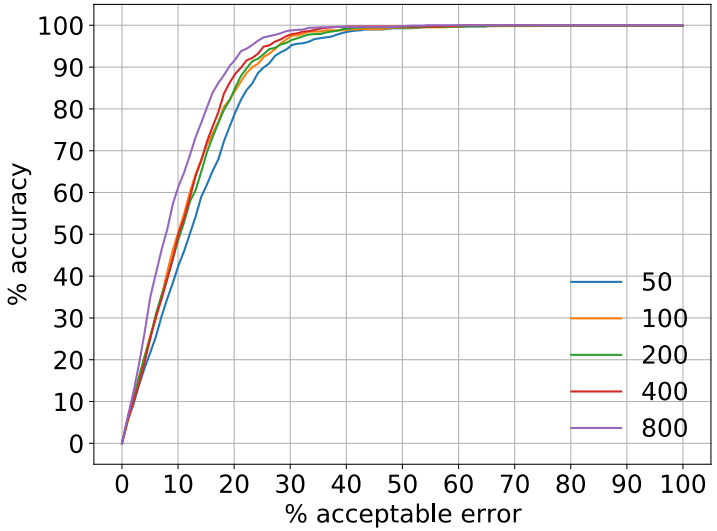


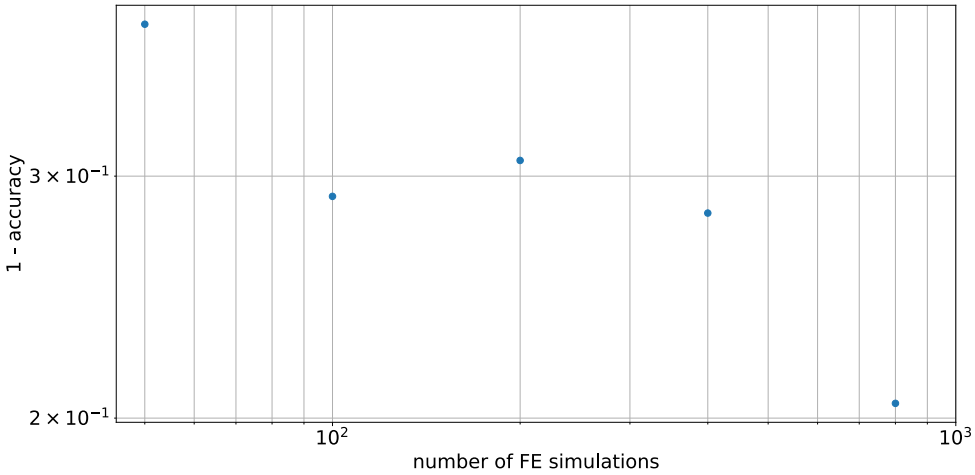


Stress GNN FEA

Accuracy for small datasets

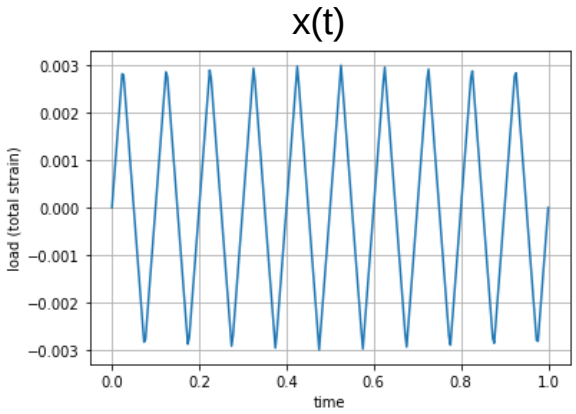
- Extract 10 patches from every FE simulation
- Train GNN with 50, 100, 200, 400, 800 FE simulations, 100 FE simulation as test



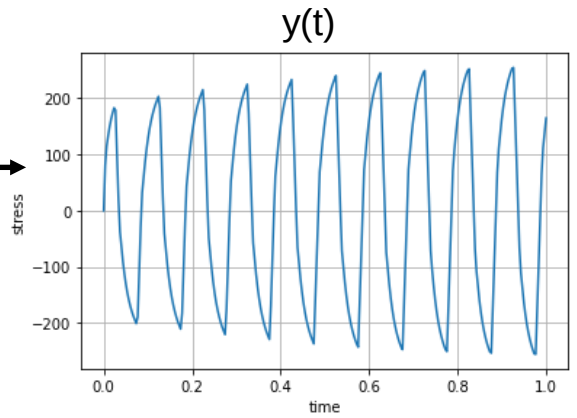


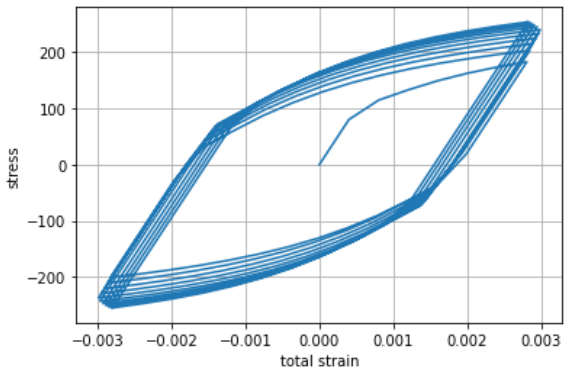
Vasilis Krokos

Deep learning for material laws

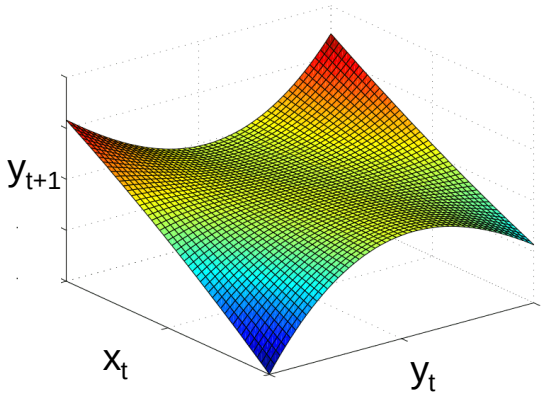


?

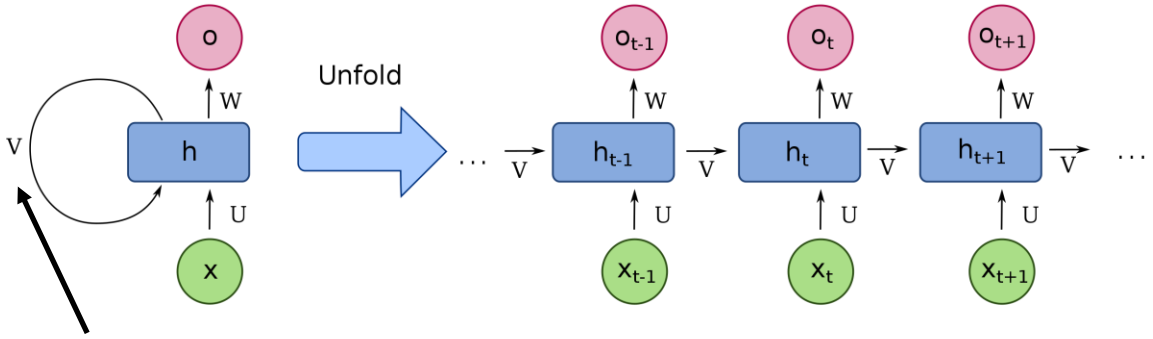




Autoregressive approach



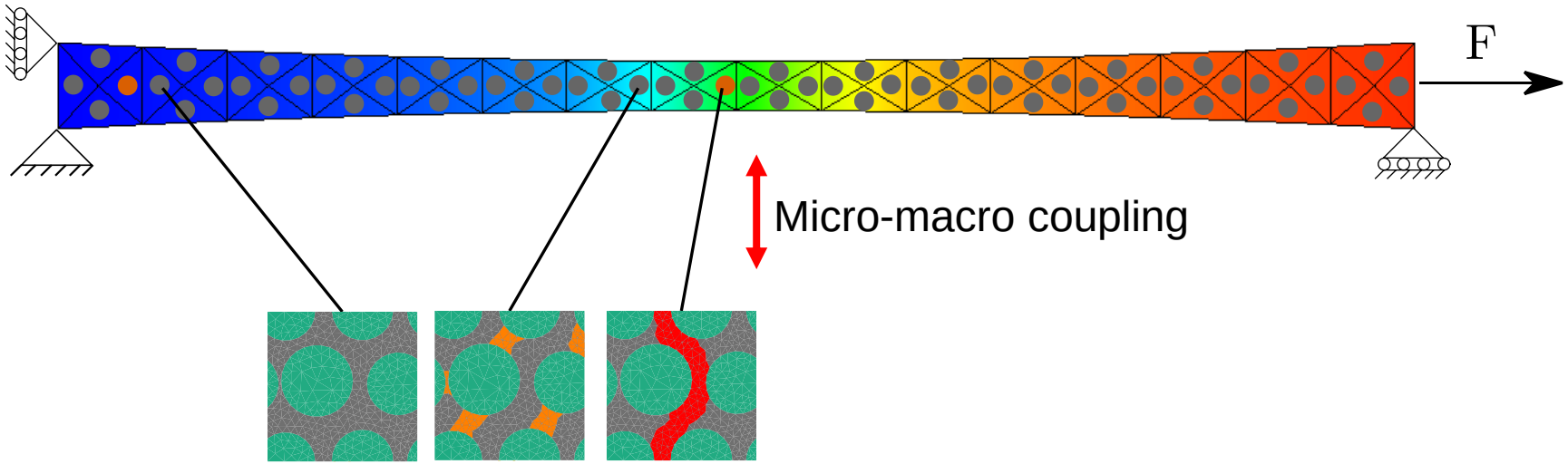
Recurrent Neural Network



Learnt hidden states captures arbitrarily long-time dependencies

→ In both cases, weights are shared across time

Nested material modelling

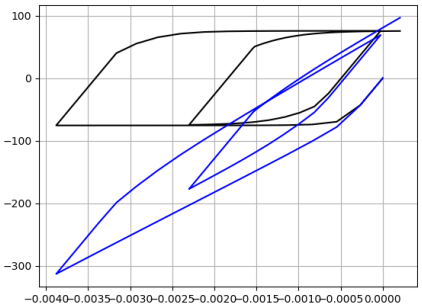


FE^2 [Feyel, Chaboche ‘98]: constitutive law defined through a microscale finite element model that needs to be solved at every quadrature point of the structure-scale finite element model.

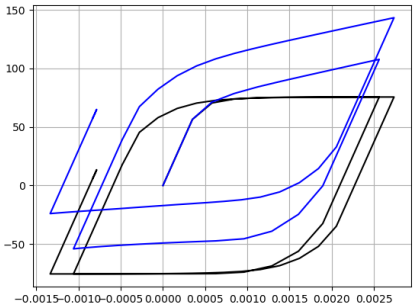
[Goury, kerfriden *et al.* ’16]
[Rocha, Kerfriden *et al.* ‘20, ‘21]

Learning plasticity from examples

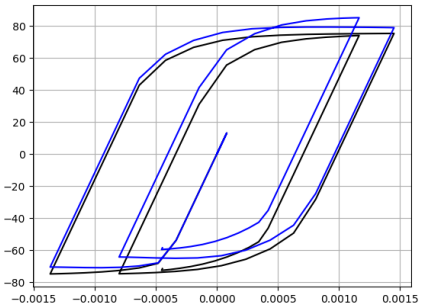
1 iteration



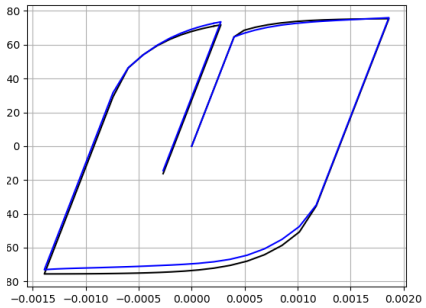
10 iterations



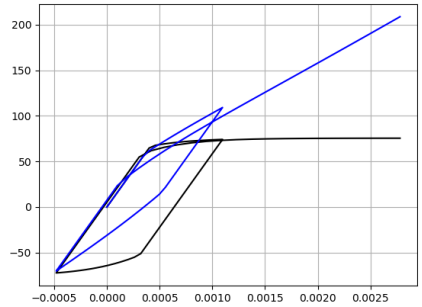
50 iterations

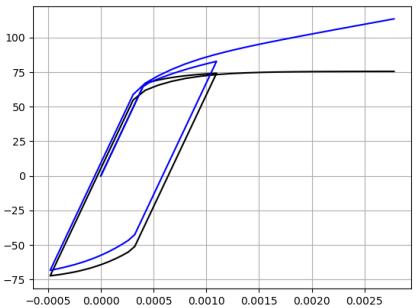


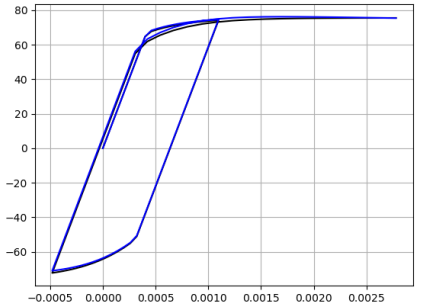
200 iterations

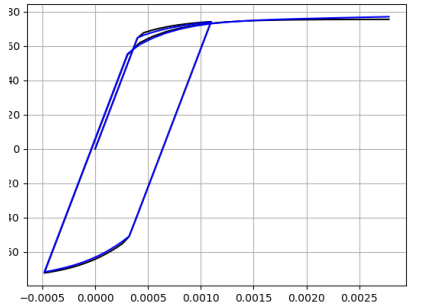


Random training examples









Test case

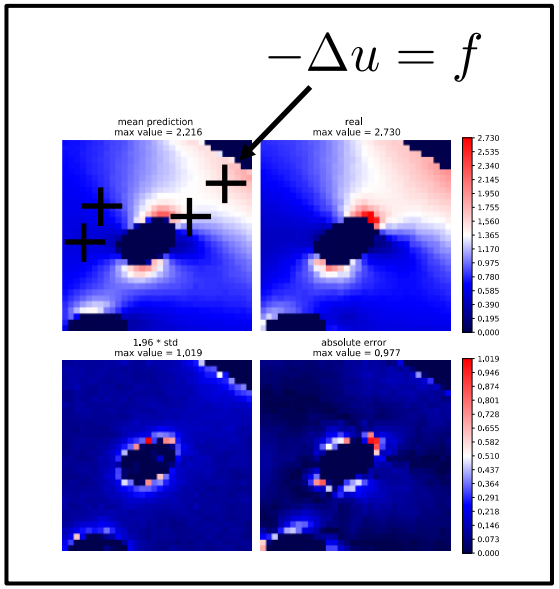
Improving extrapolation ability

- Online information:
  - Active learning: Generate new data whenever predictions are poor

    → Requires error estimates (*e.g.* probabilistic models) and means to perform new simulations online

  - Online corrections via physics constraints (Reduced Order Models, physics-informed Neural Nets)

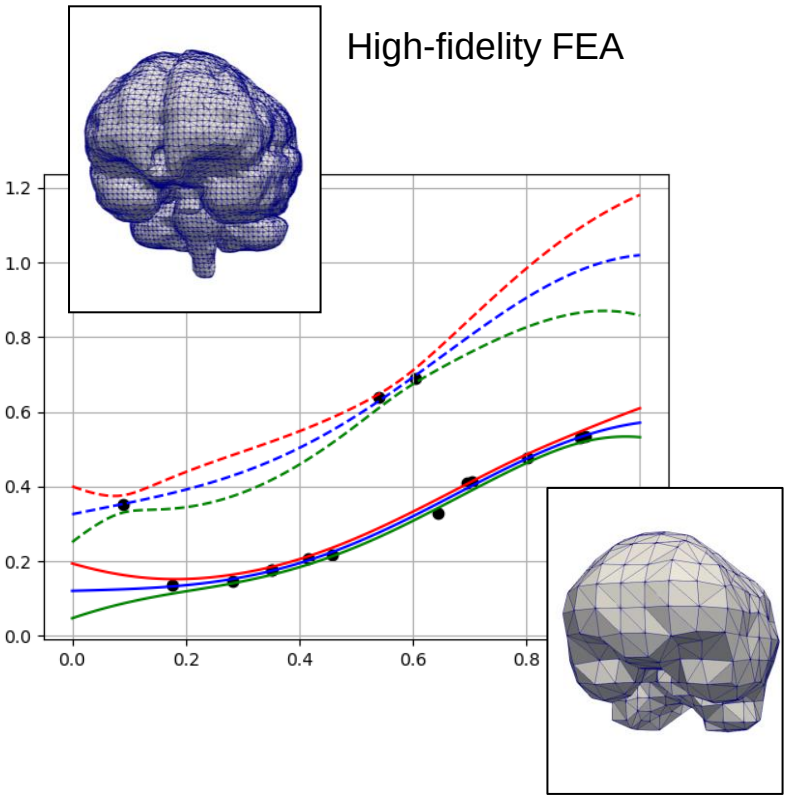
  - Offline/online multi-fidelity modelling : learn corrections to cheap models

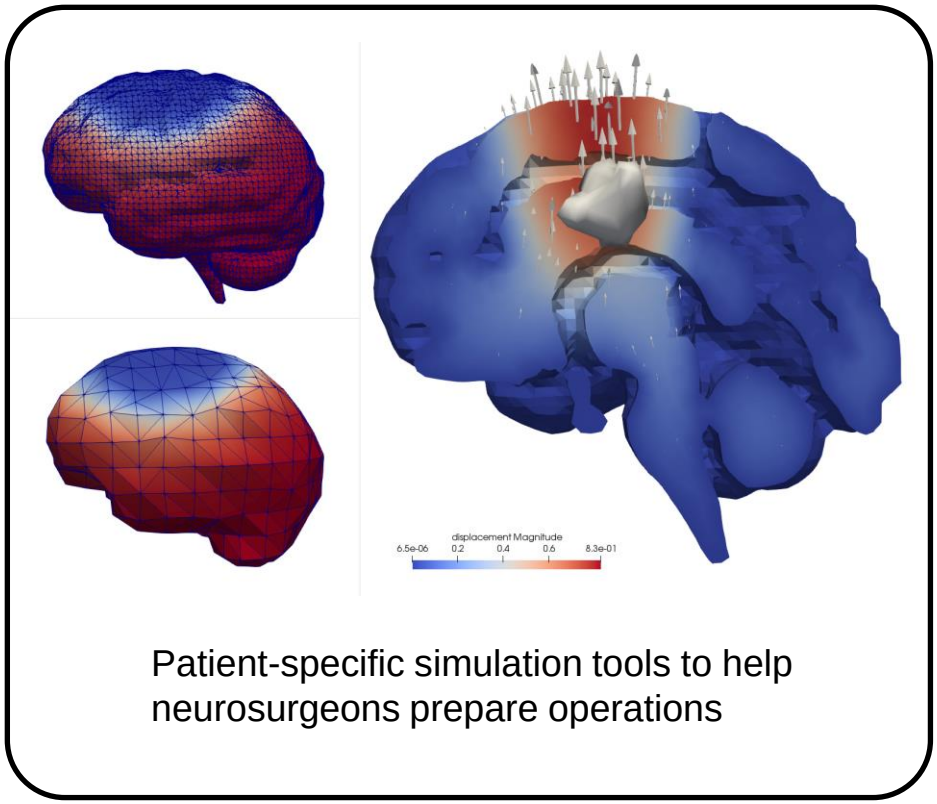


- *Inflated* offline data
  - Unsupervised learning (*e.g* auto-encoding & dimensionality reduction)

  - Transfer learning (supervised)

Multi-fidelity meta-modelling



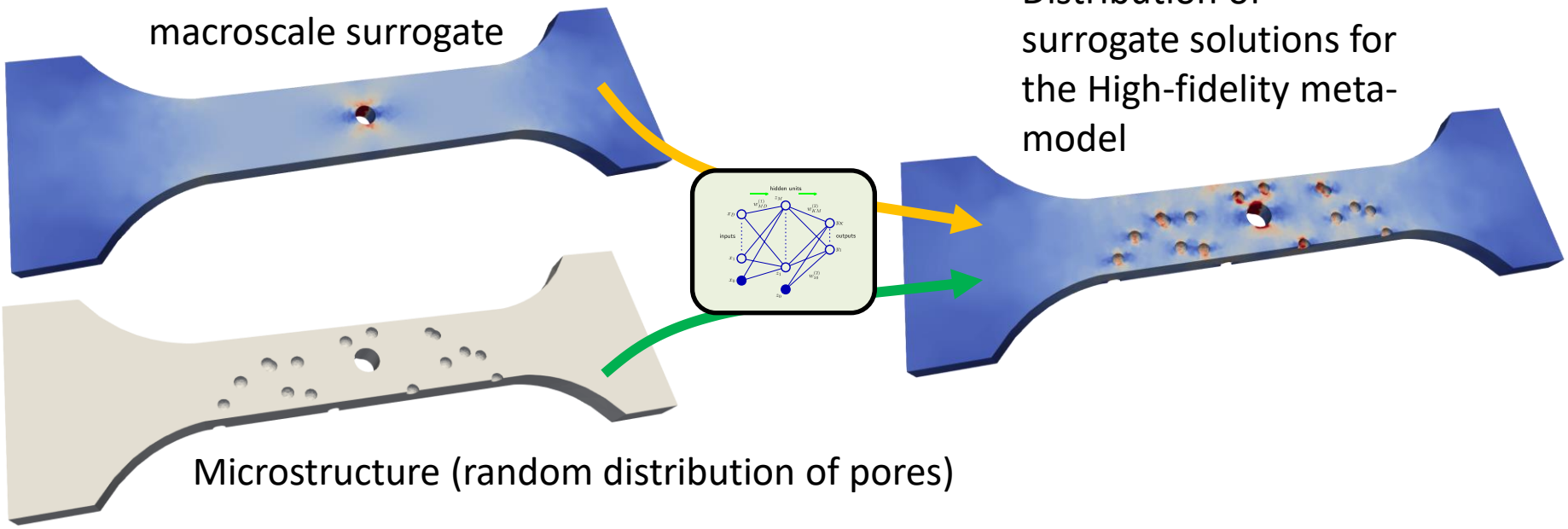


Patient-specific simulation tools to help neurosurgeons prepare operations

- Learn multi-ouput model (co-Kriging, MO neural networks), statistical strength being obtained from low-fidelity models that are evaluated more often
- Coarse model may be systematically evaluated online -> AI acts as a “correction” of the coarse model

Example: Multi-fidelity Graph Neural Nets + online constraints enforced via hierarchical Bayes

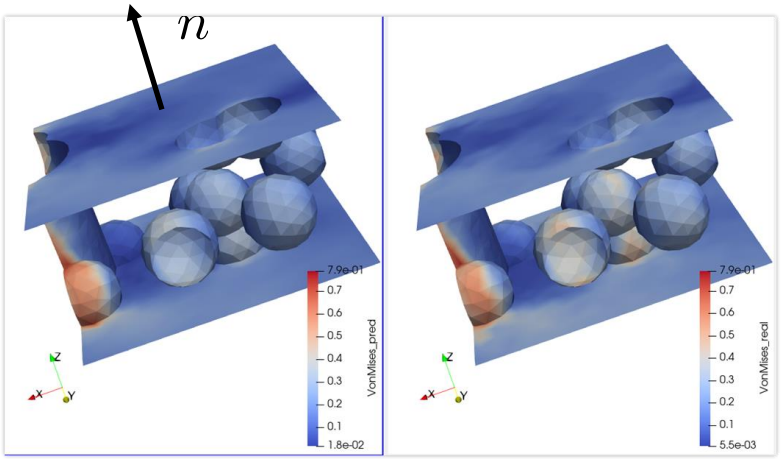
- Bayesian Graph NNet for stress field



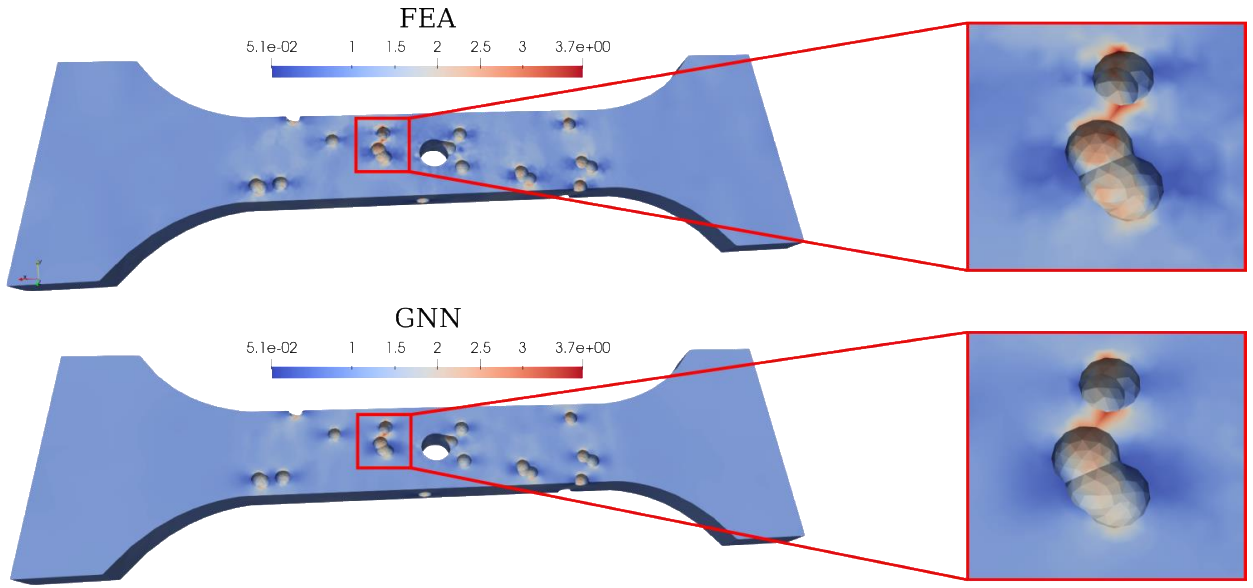
- Select those realisations that satisfy equilibrium (using Bayes)

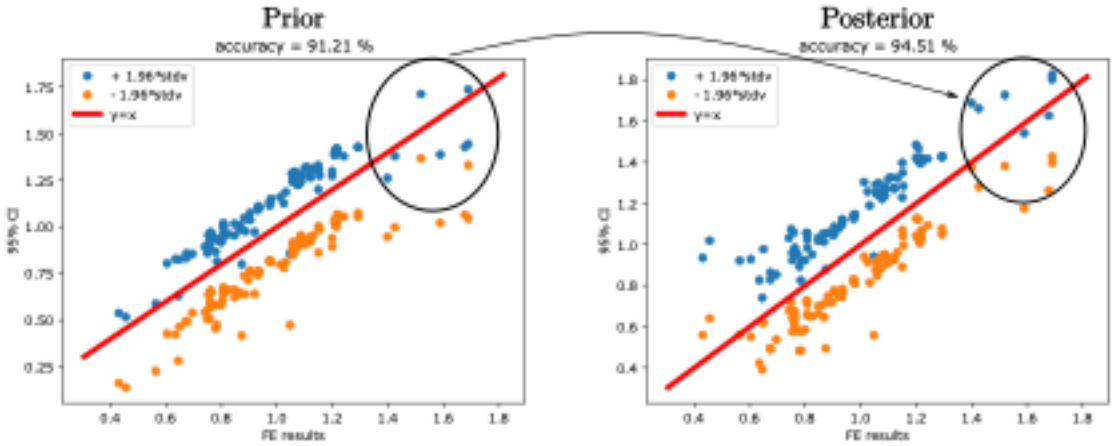
$$\sigma \cdot n = 0 + \epsilon$$

$$\epsilon \sim \mathcal{N}(0, \sigma_d^2 \mathbf{I})$$



Example: Multi-fidelity Graph Neural Nets + online constraints enforced via hierarchical Bayes





(a) 95% CIs before the stress update

(b) 95% CIs after the stress update

Outlines

1. Designs of experiments (DoE)

2. Polynomial surrogate modelling
   a. Tensor-product polynomial approximation spaces
   b. Sparse grids interpolation
   c. Sparse polynomial regression

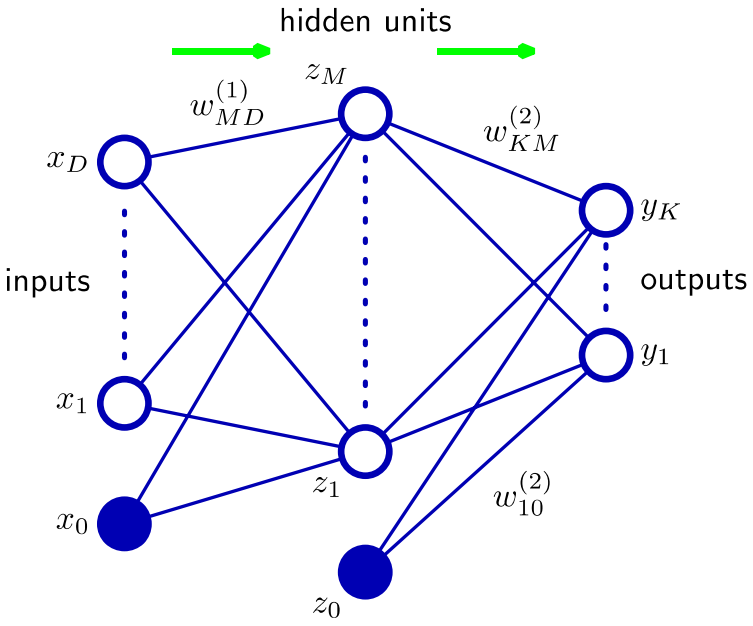
3. Control of (polynomial) surrogate models
   a. Regularisation methods
   b. Model selection

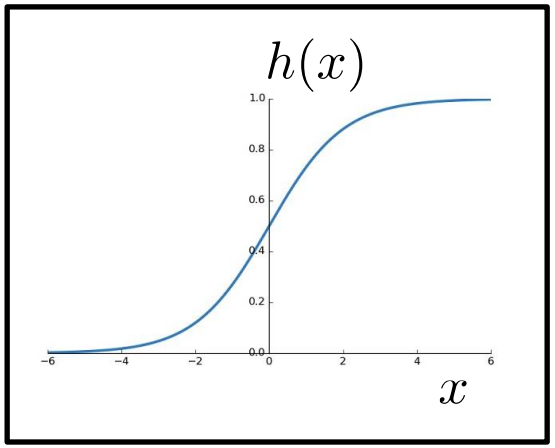
4. Non-parametric surrogate modelling, Gaussian Processes

5. Beyond response surface methodologies using Deep Learning

6. Physics-Informed Neural Nets : surrogate models without numerical solver

Fully connected Network





$$y_k(\mathbf{x}, \mathbf{w}) = \sigma\left(\sum_{j=1}^{M} w_{kj}^{(2)} h\left(\sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)}\right) + w_{k0}^{(2)}\right)$$

For regression, output activation is the identity function

Trained by minimising MSE over training pairs $\{(\mathbf{y}_i, \mathbf{x}_i)\}_i$ using gradient descent (gradient computed using automatic differentiation)

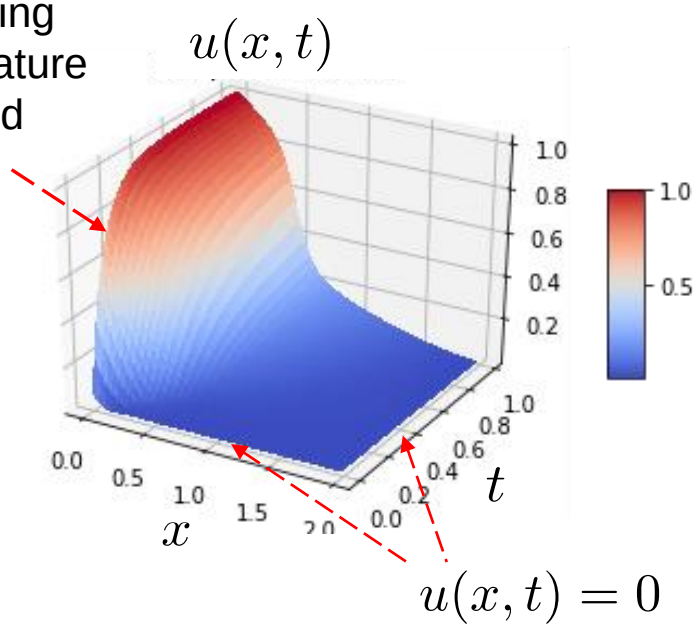
Inverse transient heat diffusion pb

- Transient heat equation in 1D

$$\alpha \frac{\partial u}{\partial t} - \frac{\partial}{\partial x}\left(k \frac{\partial u}{\partial x}\right) = f$$

+ boundary and initial conditions

Increasing temperature enforced



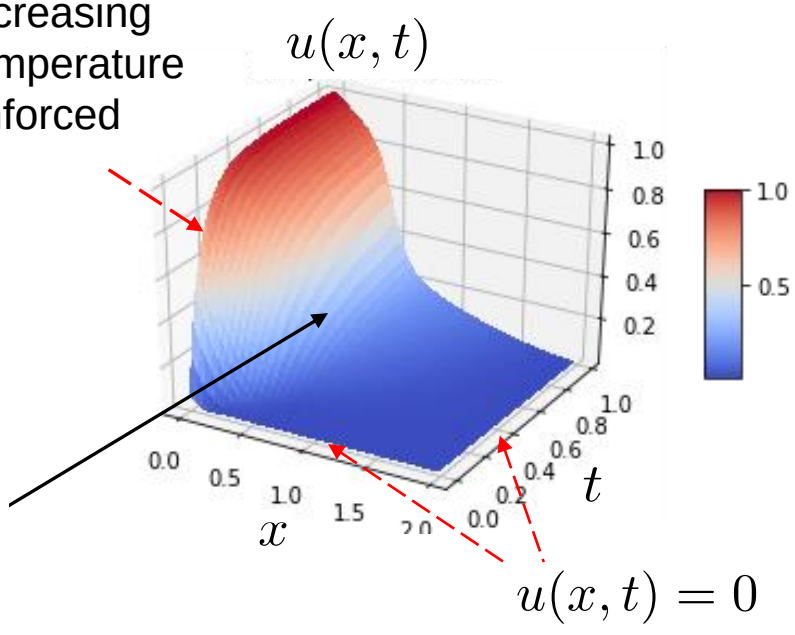
Inverse transient heat diffusion pb

- Transient heat equation in 1D

$$\alpha \frac{\partial u}{\partial t} - \frac{\partial}{\partial x}\left(k \frac{\partial u}{\partial x}\right) = f$$

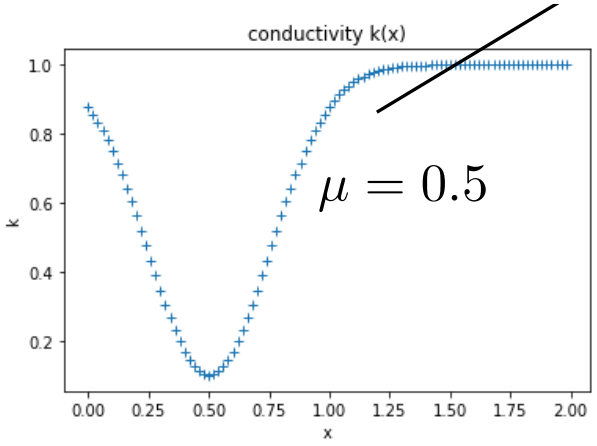
+ boundary and initial conditions

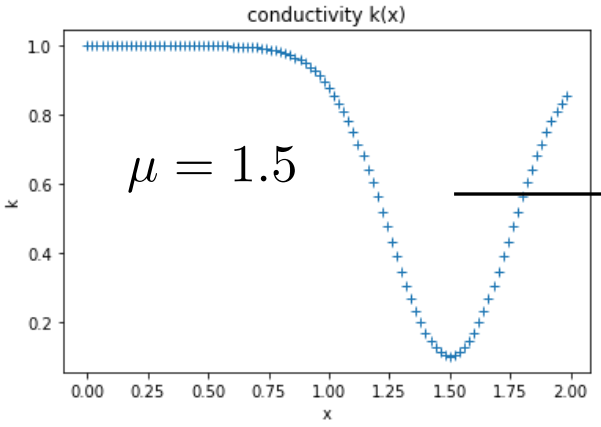
Increasing temperature enforced

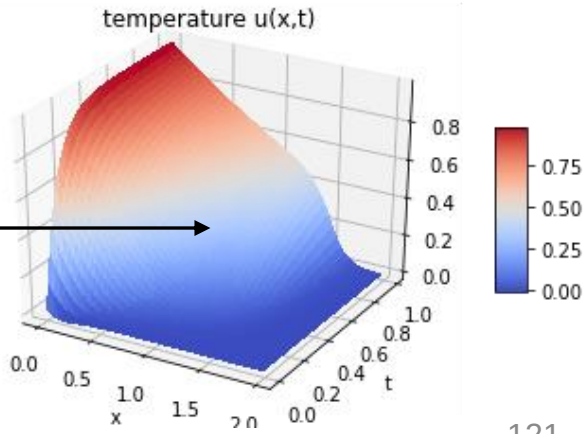


- Prototype Meta-modelling problem

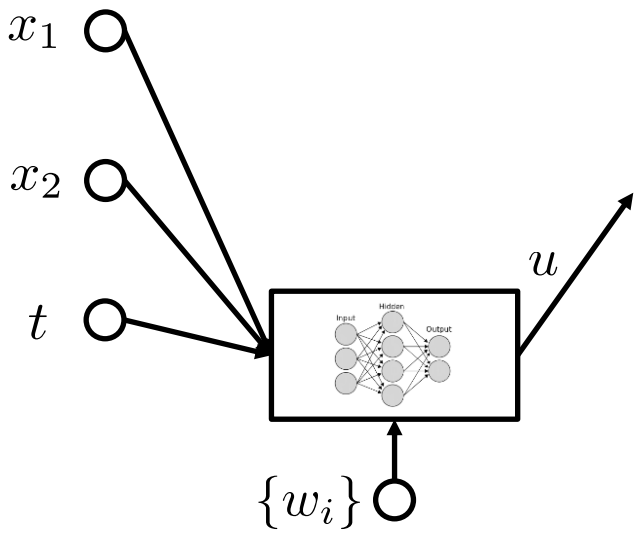
$$k(x;\mu) = 1 - \eta \exp\left(\frac{-(x-\mu)^2}{2l^2}\right)$$





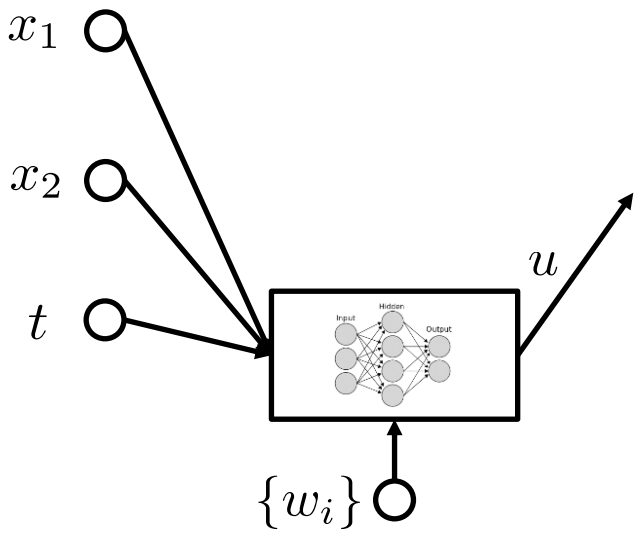


NNets as PDE solver



But we have no data for u …

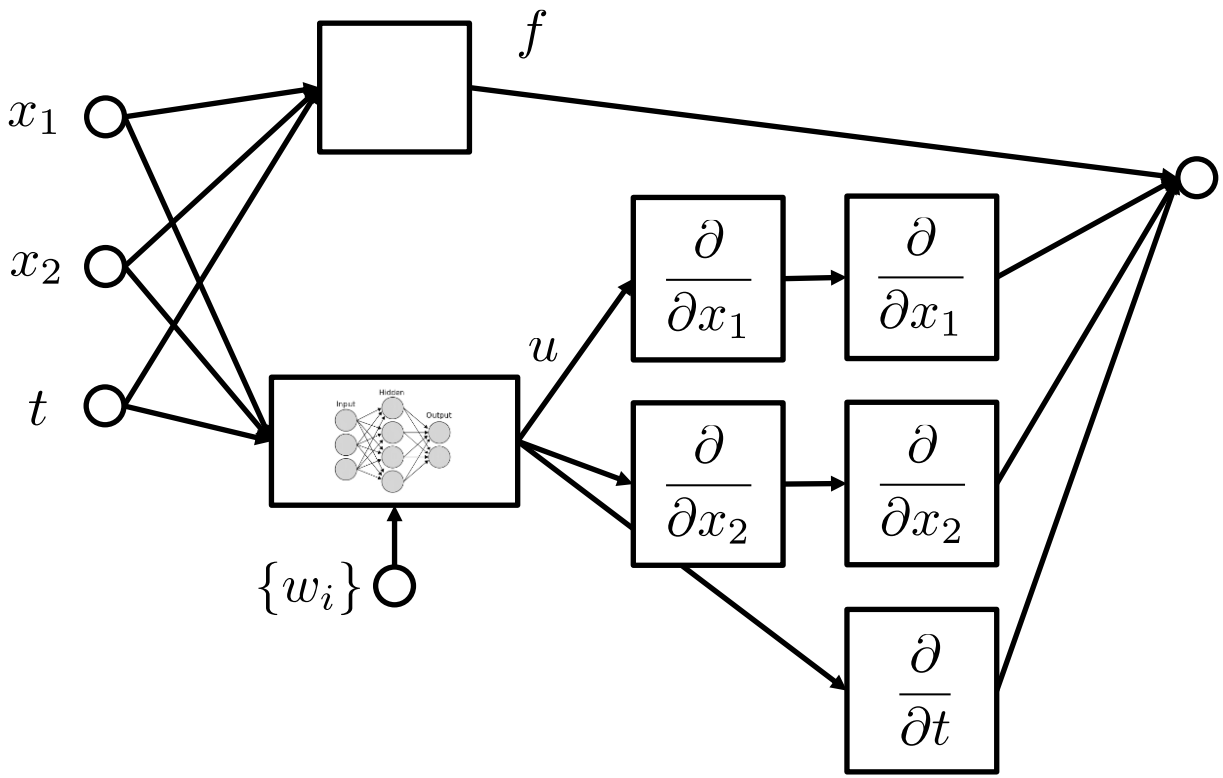
NNets as PDE solver



But we ~~have no~~ data for u …

We do, we have info about its spatial and time derivative!

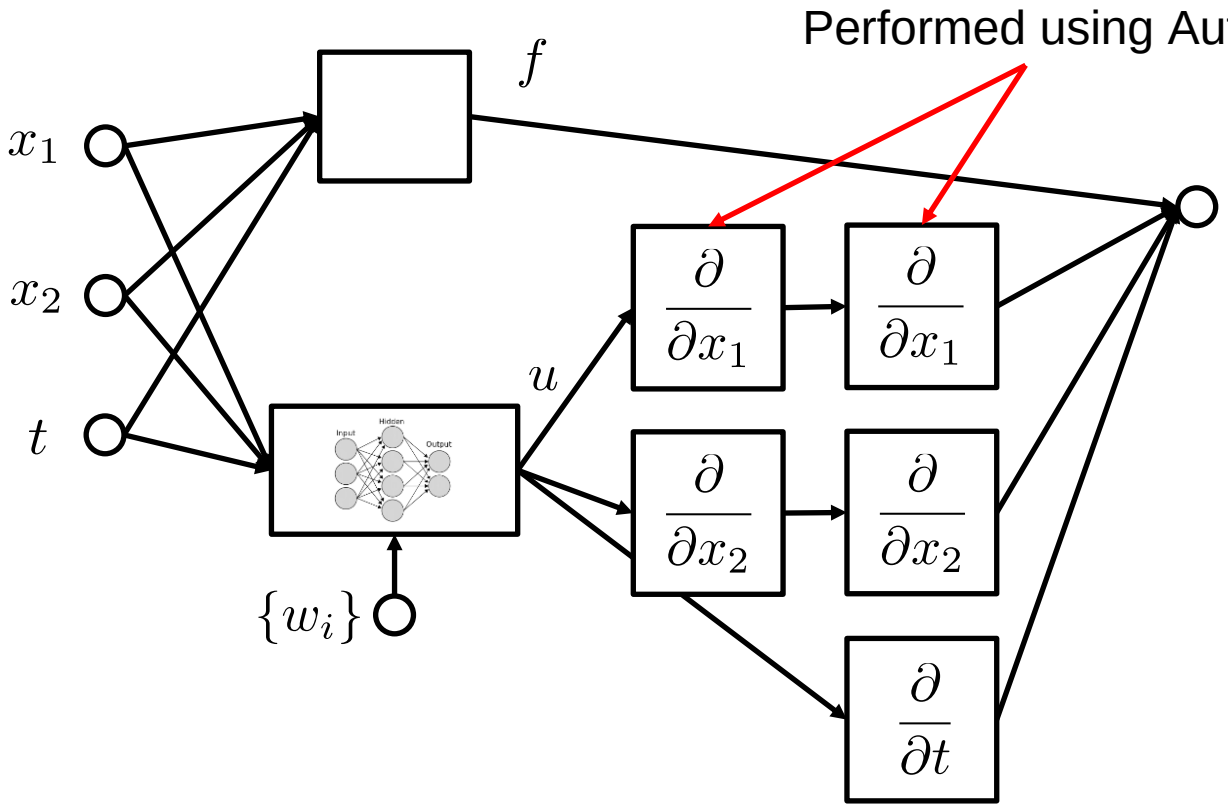
NNets as PDE solver



$$r_f = f - \alpha \frac{\partial u}{\partial t} + k\Delta u$$

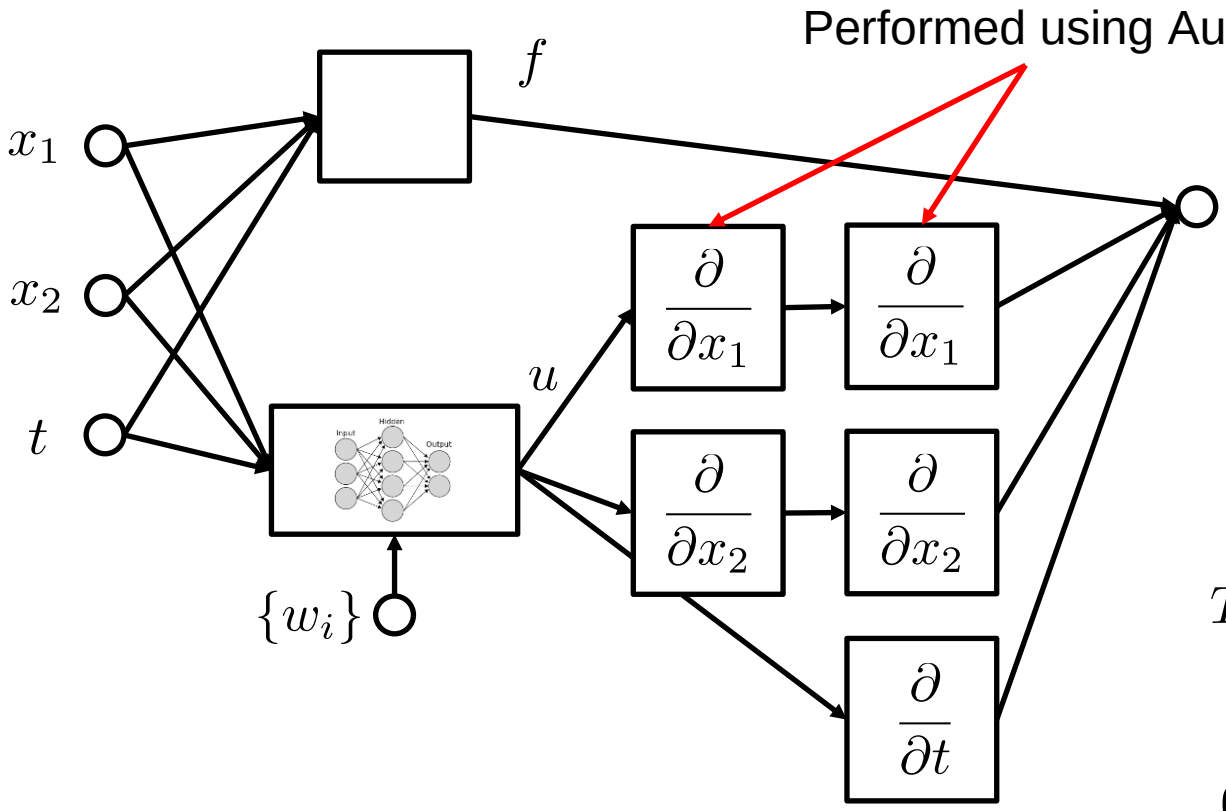
Case when conductivity is a constant

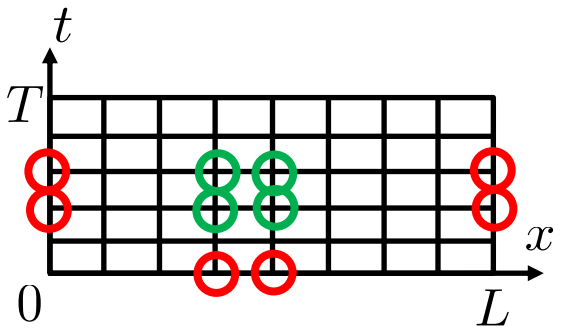
NNets as PDE solver



Minimise residual!
Called Sobolev training (we know derivatives of the output)

NNets as PDE solver



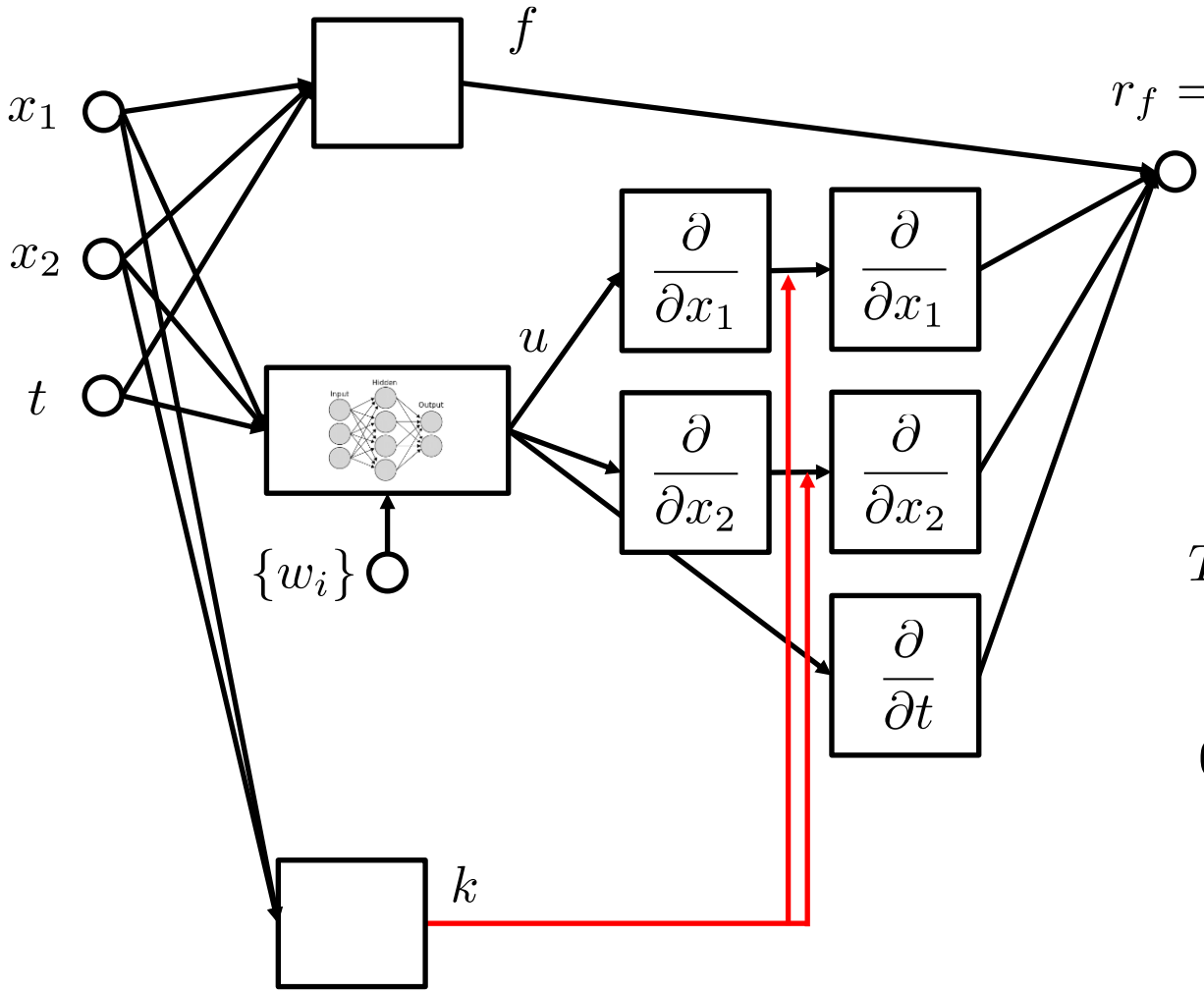


Residual at bulk colloc. points

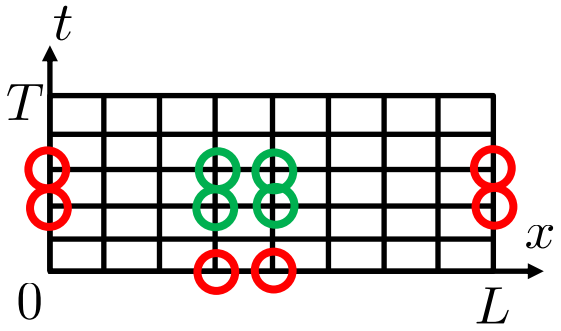
BC & IC residual at colloc. points

$$J(\{w_i\}) = \sum_{\left(x^{(k)},t^{(k)}\right)\in\mathcal{S}_f} r_{f|x^{(k)},t^{(k)}}^2 + \sum_{\left(x^{(l)},t^{(l)}\right)\in\mathcal{S}_d} \|u_d(x^{(l)},t^{(l)}) - u_{|x^{(l)},t^{(l)}}\|_2^2$$

NNets as PDE solver

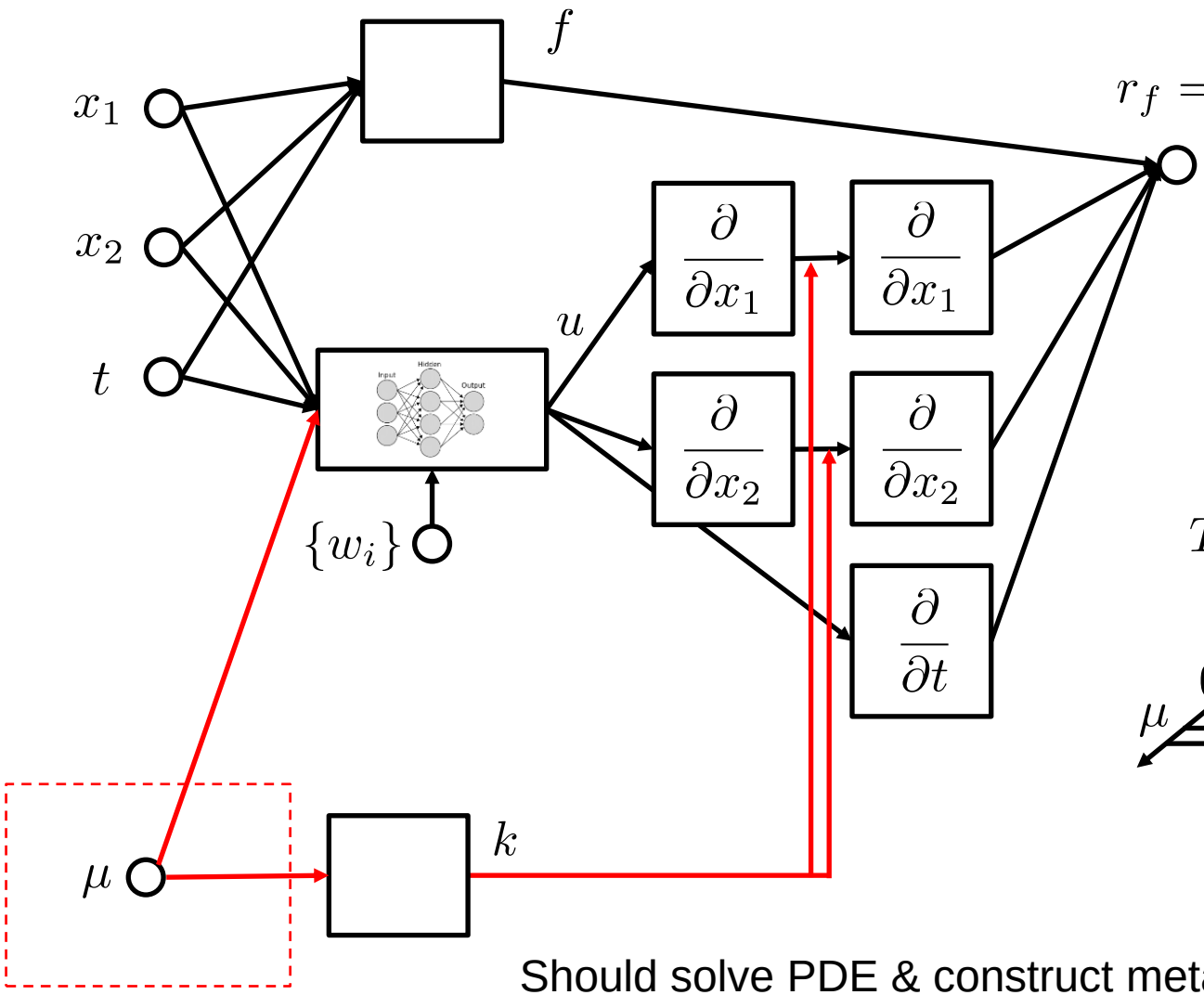


$$r_f = f - \alpha \frac{\partial u}{\partial t} + \nabla \cdot (k \nabla u)$$

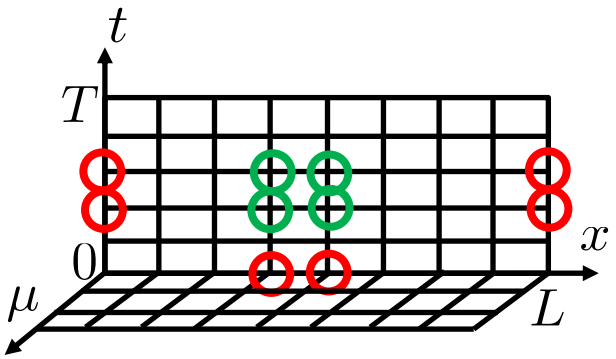


Case when conductivity is a known function of space

NNets as Meta-Modelling solvers



$$r_f = f - \alpha \frac{\partial u}{\partial t} + \nabla \cdot (k \nabla u)$$



Called **conditioning**

Should solve PDE & construct meta-model at the same time!

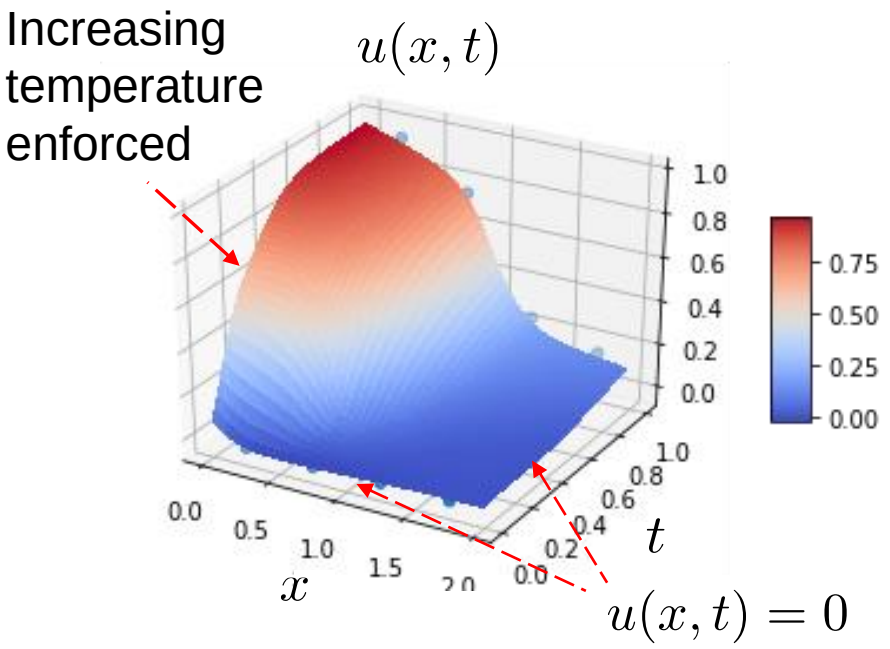
How do you train such a beast?

Inverse transient heat diffusion pb

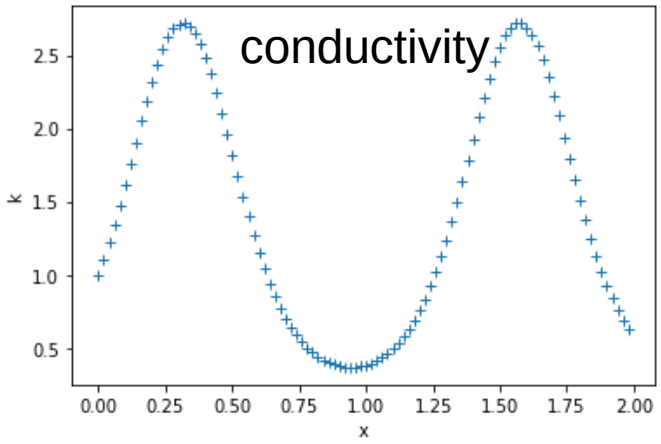
- Transient heat equation in 1D

$$\alpha \frac{\partial u}{\partial t} - \frac{\partial}{\partial x}\left(k \frac{\partial u}{\partial x}\right) = f$$

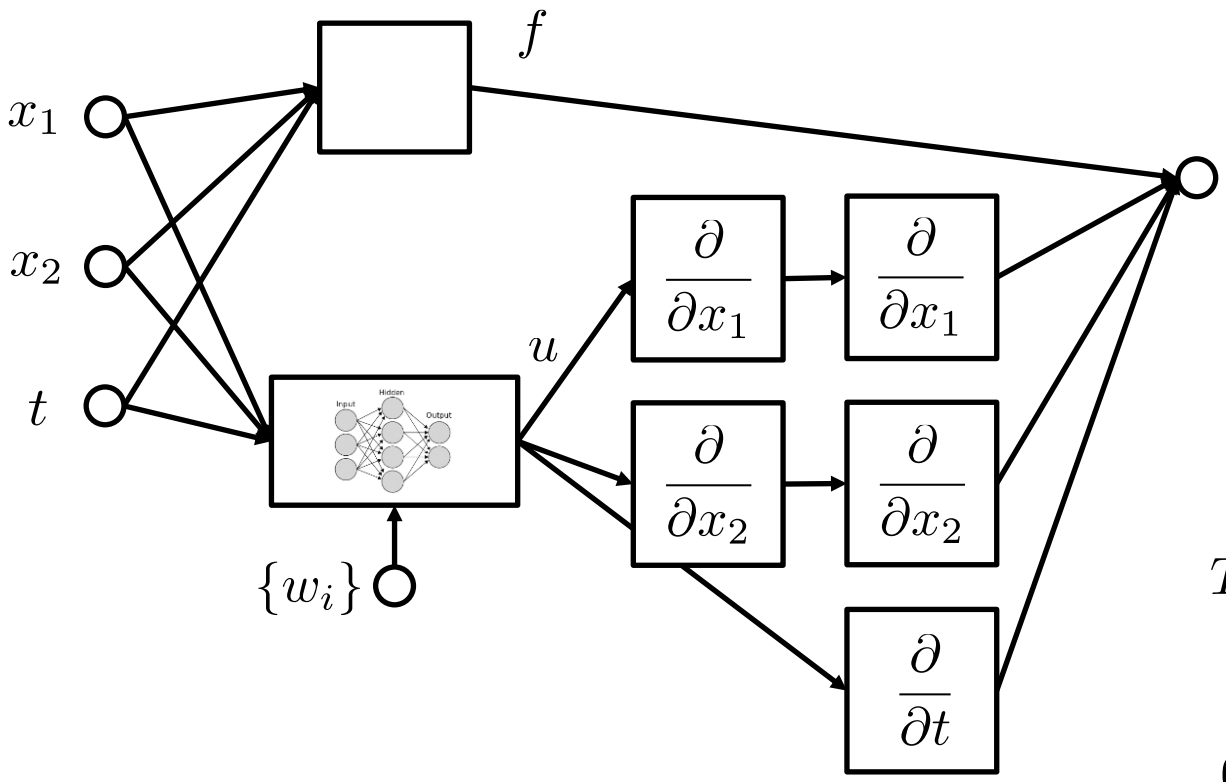
+ boundary and initial conditions



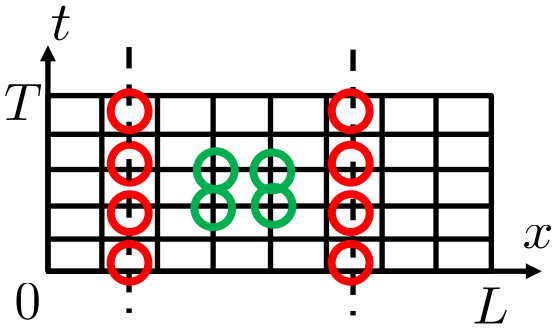
- Prototype inverse problem
  - find conductivity field $k(x)$ given temperature history at sample locations
  - Find boundary and initial conditions given that data



Inverse problems: IC & BC unknown



$$r_f = f - \alpha \frac{\partial u}{\partial t} + k\Delta u$$

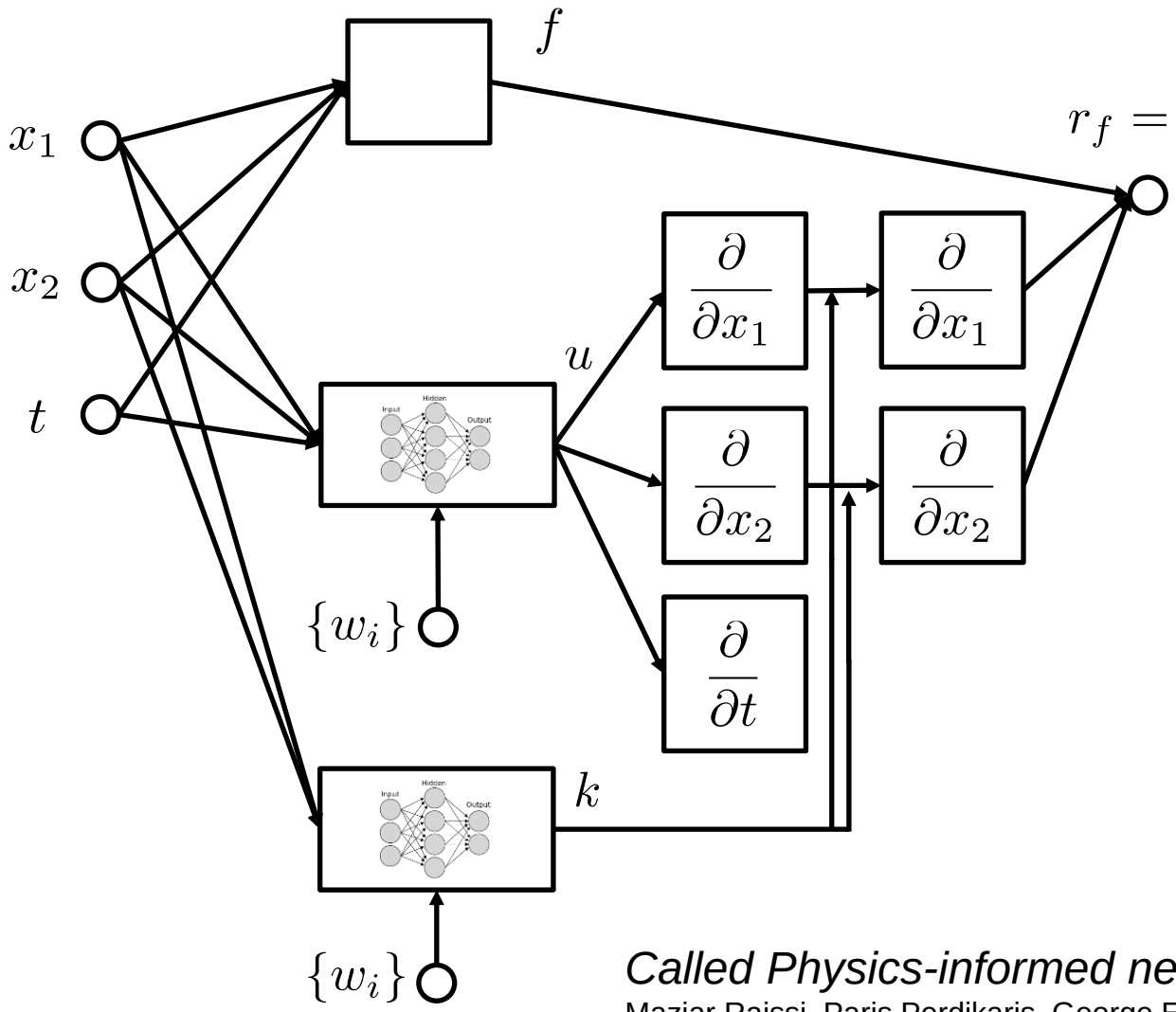


Residual at bulk colloc. points

Data fit at sensor locations

$$J(\{w_i\}) = \sum_{\left(x^{(k)},t^{(k)}\right)\in\mathcal{S}_f} r_{f\,|x^{(k)},t^{(k)}}^2 + \sum_{\left(x^{(l)},t^{(l)}\right)\in\mathcal{S}_{data}} \|U_{data}^{(l)} - u_{|x^{(l)},t^{(l)}}\|_2^2$$

Inverse problems : conductivity field



$$r_f = f - \alpha \frac{\partial u}{\partial t} + \nabla \cdot (k \nabla u)$$

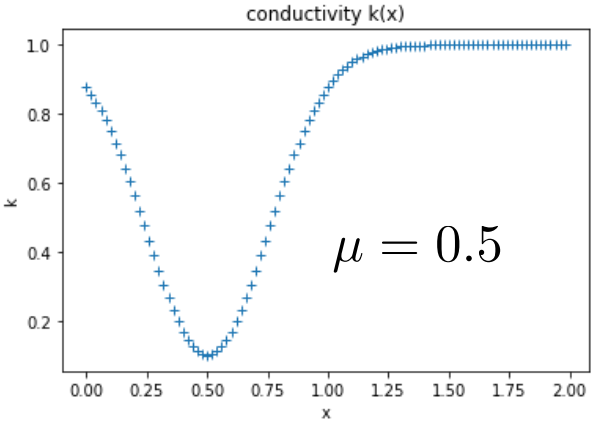
Called Physics-informed neural network (PINN)

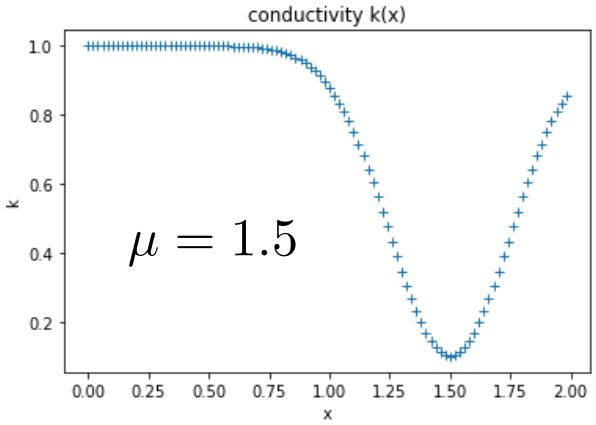
Maziar Raissi, Paris Perdikaris, George Em. Karniadakis, Physics Informed Deep Learning (Part II): Data-driven Discovery of Nonlinear Partial Differential Equations, 2017

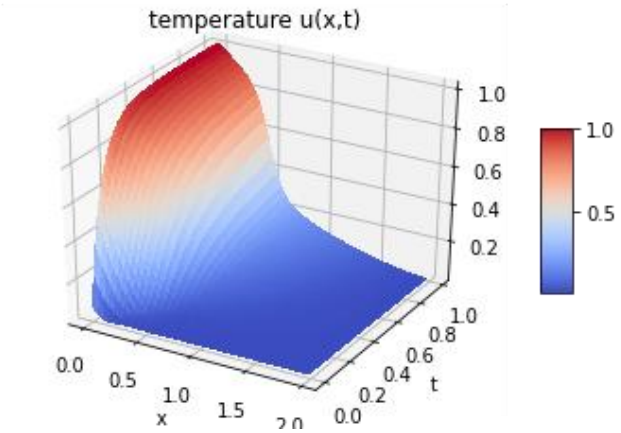
Exercise Conditioned PINNs

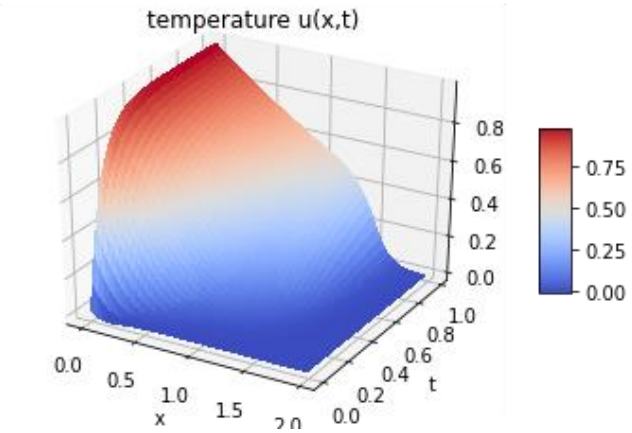
$$k(x;\mu) = 1 - \eta \exp\left(\frac{-(x-\mu)^2}{2l^2}\right)$$

$u(x,t,\mu)$?









Conclusions/comments about PINN

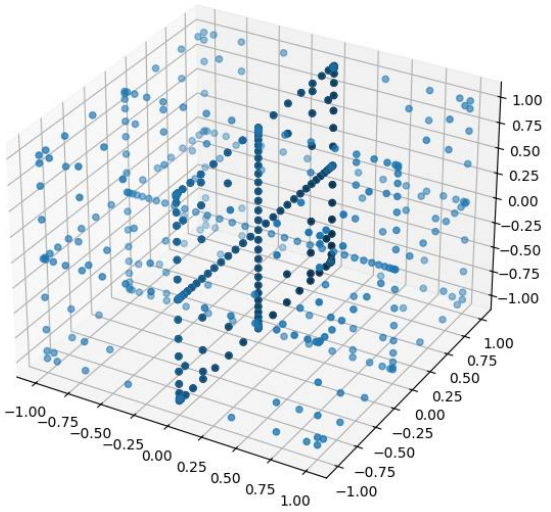
- One tool to do-it-all: solve PDE & find unknown coefficient
- Uses open-source software PyTorch, accessible in one command line and usable distantly (Jupyter notebook on Google Collab)
- Slow. Even a linear PDE turns into a nonconvex optimization problem owing to the nonlinearity of activation functions
- New numerical method for solving PDEs. We do not yet know how to choose hyperparameters optimally (*e.g.* depth of MLP, collocation points, regularisation)

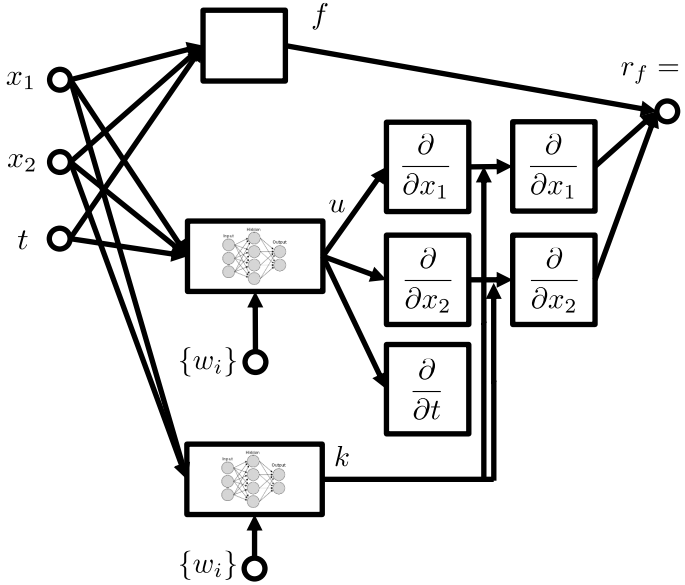
Conclusion: ML for meta-modelling

- Replace solvers for parametrised PDEs by regression or interpolation model that are inexpensive to evaluate online (+ no licence, more portable),
- Design of experiments (set of model evaluation) should explore entire parameter spaces, restricting usage to small parametric dimensions (1 to 6), unless specific structures exist in the data (tensorial structure, main effects)
- Like any ML task, tailored functional restrictions or regularisation via the loss function improves generalisation capabilities up to a point when the model underfits (bias/variance tradeoff).
- Most used algorithms : (sparse) polynomial regression (polynomial chaos) with random, quasi-random or latin hypercube sampling, Gaussian process surrogates (Kriging), sparse grid polynomial interpolation/regression
- Deep learning may be deployed to build Meta-Model in large dimensions when the data points live in hidden manifolds that modern convolutional or recurrent architectures may help uncover

Thank you for your attention

pierre.kerfriden@minesparis.psl.eu





$$r_f = f - \alpha \frac{\partial u}{\partial t} + \nabla \cdot (k \nabla u)$$