IgANets: Physics-machine learning embedded into Isogeometric Analysis

Matthias Möller

Department of Applied Mathematics, TU Delft

HOFEIM 2023

Joint work with Deepesh Toshniwal & Frank van Ruiten (TU Delft),
Casper van Leeuwen & Paul Melis (SURF), and Jaewook Lee (TU Vienna)

The future of engineering (?!)

Siemens blog: *Virtual Reality in Engineering - Are You Ready?* – 7 July 2021
https://blogs.sw.siemens.com/teamcenter/virtual-reality-in-engineering-are-you-ready/

Interactive Design-through-Analysis

Vision: unified computational framework for

- **rapid prototyping** (design exploration & optimization phase) and
- **thorough analysis** (design analysis & fine-tuning phase)

of engineering designs

Ingredients

- physics-informed machine learning for rapid prototyping
- isogeometric analysis for accurate analysis

A demo says more than 1,000 words...

The big picture

Front-ends

IgANet-frontend
by SURF

gustaf
by TU Vienna

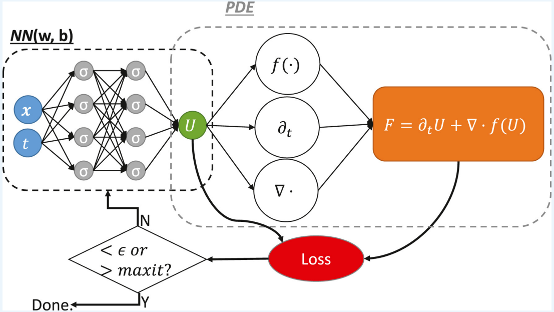
WebSockets protocol for interactive Design-through-Analysis

Back-ends

IgANet

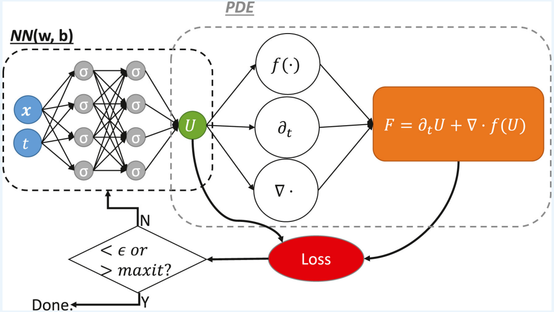
Physics-informed machine learning

PINN (Raissi et al. 2018): *learns the (initial-)boundary-value problem*



Physics-informed machine learning

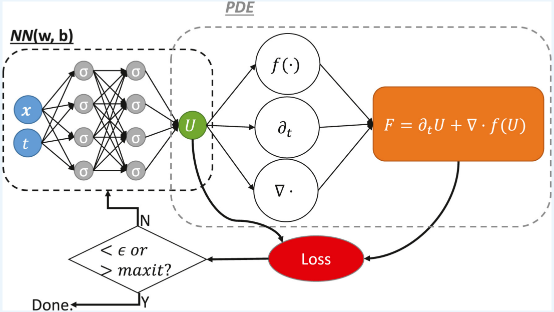
PINN (Raissi et al. 2018): *learns the (initial-)boundary-value problem*



- easy to implement for 'any' PDE because AD magic does it for you
- combined un-/supervised learning
- poor extrapolation/generalization
- point-based approach requires re-evaluation of NN at every point
- rudimentary convergence theory

Physics-informed machine learning

PINN (Raissi et al. 2018): *learns the (initial-)boundary-value problem*



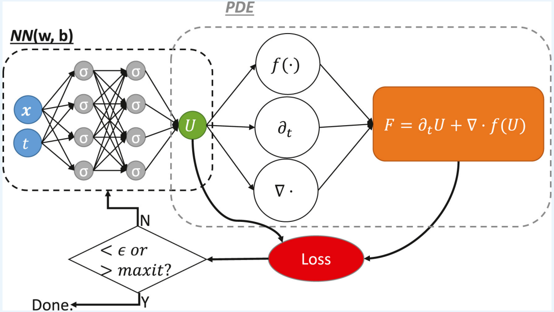
- easy to implement for 'any' PDE because AD magic does it for you
- combined un-/supervised learning
- poor extrapolation/generalization
- point-based approach requires re-evaluation of NN at every point
- rudimentary convergence theory

DeepONet (Lu et al. 2019): *learns the differential operator*

$$G_\theta(u)(y) = \sum_{k=1}^{q} \underbrace{b_k(u(x_1), u(x_2), \ldots, u(x_m))}_{\text{branch}} \underbrace{t_k(y)}_{\text{trunk}}$$

Physics-informed machine learning

PINN (Raissi et al. 2018): *learns the (initial-)boundary-value problem*



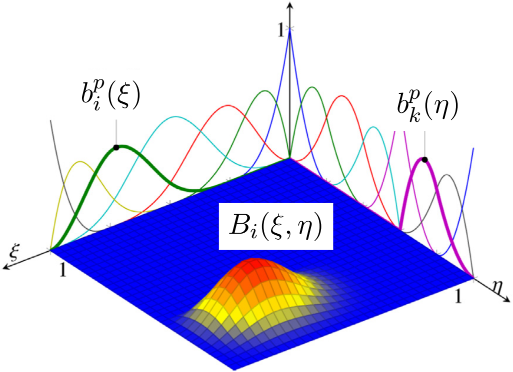
- easy to implement for 'any' PDE because AD magic does it for you
- combined un-/supervised learning
- poor extrapolation/generalization
- point-based approach requires re-evaluation of NN at every point
- rudimentary convergence theory

DeepONet (Lu et al. 2019): *learns the differential operator*

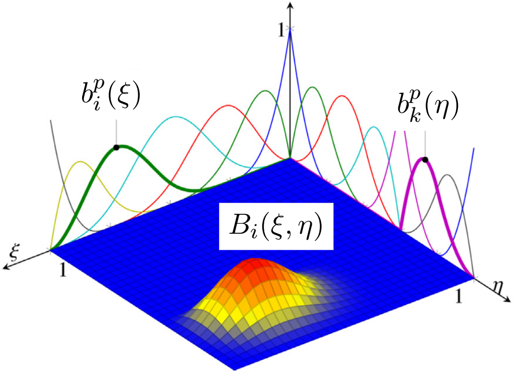
$$G_\theta(u)(y) = \sum_{k=1}^{q} \underbrace{b_k(u(x_1), u(x_2), \ldots, u(x_m))}_{\text{branch}} \underbrace{t_k(y)}_{\text{trunk}}$$

Don't we know good bases?

Tensor-product B-spline basis functions



Tensor-product B-spline basis functions

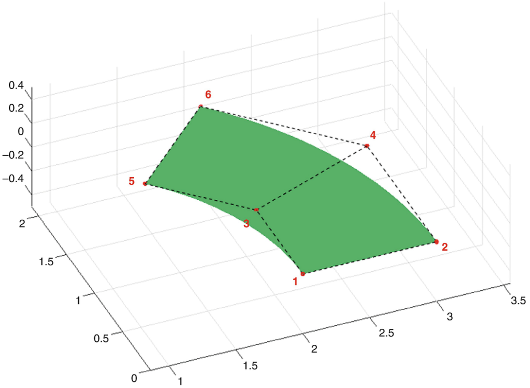


Properties: compact support, positive function values, partition of unity $\sum_{i=1}^{n} B_i(\xi,\eta) \equiv 1$, C^{p-1} continuity, derivatives of B-splines are combinations of lower-order B-splines, ...

Isogeometric Analysis

Geometry: bijective mapping from the unit square to the physical domain $\Omega_h \subset \mathbb{R}^d$

$$\mathbf{x}_h(\xi,\eta) = \sum_{i=1}^{n} B_i(\xi,\eta) \cdot \mathbf{x}_i \qquad \forall (\xi,\eta) \in [0,1]^2 =: \hat{\Omega}$$

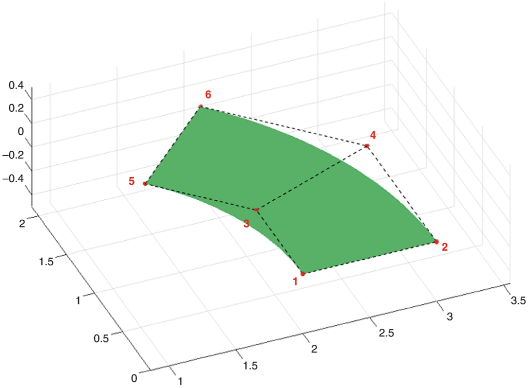


- the shape of Ω_h is fully specified by the set of **control points** $\mathbf{x}_i \in \mathbb{R}^d$

Isogeometric Analysis

Geometry: bijective mapping from the unit square to the physical domain $\Omega_h \subset \mathbb{R}^d$

$$\mathbf{x}_h(\xi,\eta) = \sum_{i=1}^{n} B_i(\xi,\eta) \cdot \mathbf{x}_i \qquad \forall (\xi,\eta) \in [0,1]^2 =: \hat{\Omega}$$

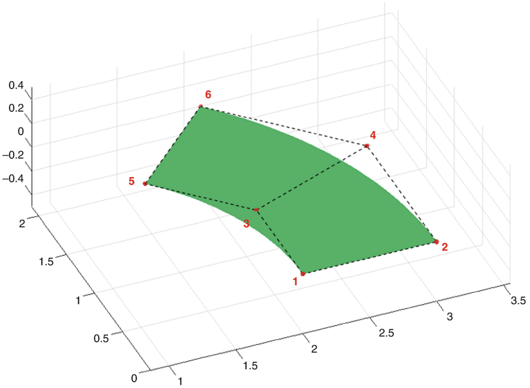


- the shape of Ω_h is fully specified by the set of **control points** $\mathbf{x}_i \in \mathbb{R}^d$
- interior control points must be chosen such that 'grid lines' do not fold as this violates the bijectivity of $\mathbf{x}_h : \hat{\Omega} \to \Omega_h$

Isogeometric Analysis

Geometry: bijective mapping from the unit square to the physical domain $\Omega_h \subset \mathbb{R}^d$

$$\mathbf{x}_h(\xi,\eta) = \sum_{i=1}^{n} B_i(\xi,\eta) \cdot \mathbf{x}_i \qquad \forall(\xi,\eta) \in [0,1]^2 =: \hat{\Omega}$$



- the shape of Ω_h is fully specified by the set of **control points** $\mathbf{x}_i \in \mathbb{R}^d$
- interior control points must be chosen such that 'grid lines' do not fold as this violates the bijectivity of $\mathbf{x}_h : \hat{\Omega} \to \Omega_h$
- refinement in h (knot insertion) and p (order elevation) preserves the shape of Ω_h and can be used to generate finer computational 'grids' for the analysis

Isogeometric Analysis

Model problem: Poisson's equation

$$-\Delta u_h = f_h \quad \text{in} \quad \Omega_h, \qquad u_h = g_h \quad \text{on} \quad \partial\Omega_h$$

with

$$\text{(geometry)} \qquad \mathbf{x}_h(\xi,\eta) = \sum_{i=1}^{n} B_i(\xi,\eta)\cdot \mathbf{x}_i \qquad \forall(\xi,\eta)\in[0,1]^2$$

$$\text{(solution)} \qquad u_h \circ \mathbf{x}_h(\xi,\eta) = \sum_{i=1}^{n} B_i(\xi,\eta)\cdot u_i \qquad \forall(\xi,\eta)\in[0,1]^2$$

$$\text{(r.h.s vector)} \qquad f_h \circ \mathbf{x}_h(\xi,\eta) = \sum_{i=1}^{n} B_i(\xi,\eta)\cdot f_i \qquad \forall(\xi,\eta)\in[0,1]^2$$

$$\text{(boundary conditions)} \qquad g_h \circ \mathbf{x}_h(\xi,\eta) = \sum_{i=1}^{n} B_i(\xi,\eta)\cdot g_i \qquad \forall(\xi,\eta)\in\partial[0,1]^2$$

Isogeometric Analysis

Abstract representation

Given $\mathbf{x}_i$ (geometry), f_i (r.h.s. vector), and g_i (boundary conditions), **compute**

$$\begin{bmatrix} u_1 \\ \vdots \\ u_n \end{bmatrix} = A^{-1}\left(\begin{bmatrix} \mathbf{x}_1 \\ \vdots \\ \mathbf{x}_n \end{bmatrix}, \begin{bmatrix} g_1 \\ \vdots \\ g_n \end{bmatrix} \right) \cdot b\left(\begin{bmatrix} \mathbf{x}_1 \\ \vdots \\ \mathbf{x}_n \end{bmatrix}, \begin{bmatrix} f_1 \\ \vdots \\ f_n \end{bmatrix}, \begin{bmatrix} g_1 \\ \vdots \\ g_n \end{bmatrix} \right)$$

Any point of the solution can afterwards be obtained by a simple **function evaluation**

$$(\xi,\eta) \in [0,1]^2 \quad \mapsto \quad u_h \circ \mathbf{x}_h(\xi,\eta) = [B_1(\xi,\eta),\dots,B_n(\xi,\eta)] \cdot \begin{bmatrix} u_1 \\ \vdots \\ u_n \end{bmatrix}$$

Isogeometric Analysis

Abstract representation

Given $\mathbf{x}_i$ (geometry), f_i (r.h.s. vector), and g_i (boundary conditions), **compute**

$$\begin{bmatrix} u_1 \\ \vdots \\ u_n \end{bmatrix} = A^{-1}\left(\begin{bmatrix} \mathbf{x}_1 \\ \vdots \\ \mathbf{x}_n \end{bmatrix}, \begin{bmatrix} g_1 \\ \vdots \\ g_n \end{bmatrix}\right) \cdot b\left(\begin{bmatrix} \mathbf{x}_1 \\ \vdots \\ \mathbf{x}_n \end{bmatrix}, \begin{bmatrix} f_1 \\ \vdots \\ f_n \end{bmatrix}, \begin{bmatrix} g_1 \\ \vdots \\ g_n \end{bmatrix}\right)$$

Any point of the solution can afterwards be obtained by a simple **function evaluation**

$$(\xi,\eta) \in [0,1]^2 \quad \mapsto \quad u_h \circ \mathbf{x}_h(\xi,\eta) = [B_1(\xi,\eta),\dots,B_n(\xi,\eta)] \cdot \begin{bmatrix} u_1 \\ \vdots \\ u_n \end{bmatrix}$$

Let us interpret the sets of B-spline coefficients $\{\mathbf{x}_i\}$, $\{f_i\}$, and $\{g_i\}$ as an efficient encoding of our PDE problem that is fed into our IgA machinery as **input**.

The **output** of our IgA machinery are the B-spline coefficients $\{u_i\}$ of the solution.

IgANet: replace **computation**

$$\begin{bmatrix} u_1 \\ \vdots \\ u_n \end{bmatrix} = A^{-1}\left(\begin{bmatrix} \mathbf{x}_1 \\ \vdots \\ \mathbf{x}_n \end{bmatrix}, \begin{bmatrix} g_1 \\ \vdots \\ g_n \end{bmatrix}\right) \cdot b\left(\begin{bmatrix} \mathbf{x}_1 \\ \vdots \\ \mathbf{x}_n \end{bmatrix}, \begin{bmatrix} f_1 \\ \vdots \\ f_n \end{bmatrix}, \begin{bmatrix} g_1 \\ \vdots \\ g_n \end{bmatrix}\right)$$

Isogeometric Analysis + Physics-Informed Machine Learning

IgANet: replace **computation** by **physics-informed machine learning**

$$\begin{bmatrix} u_1 \\ \vdots \\ u_n \end{bmatrix} = \text{IgANet}\left(\begin{bmatrix} \mathbf{x}_1 \\ \vdots \\ \mathbf{x}_n \end{bmatrix}, \begin{bmatrix} f_1 \\ \vdots \\ f_n \end{bmatrix}, \begin{bmatrix} g_1 \\ \vdots \\ g_n \end{bmatrix}; \left(\xi^{(k)}, \eta^{(k)}\right)_{k=1}^{N_\text{samples}} \right)$$

Isogeometric Analysis + Physics-Informed Machine Learning

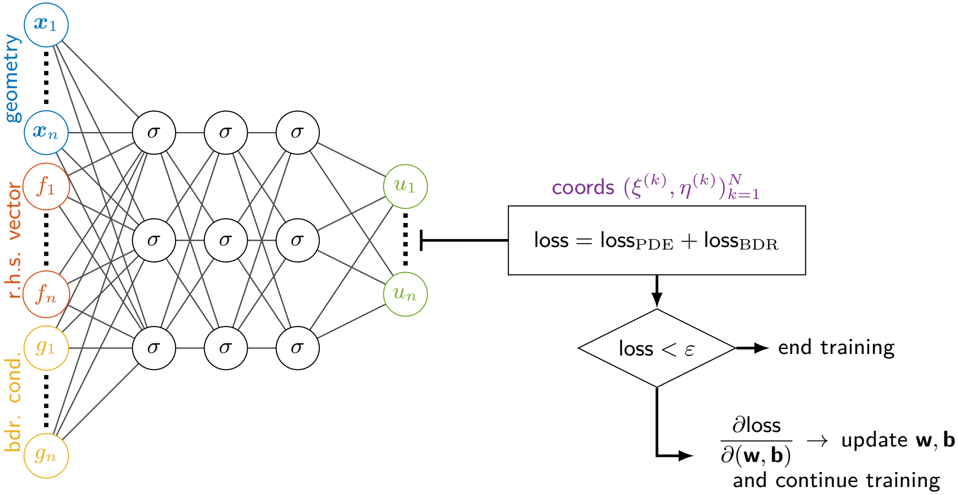
IgANet: replace **computation** by **physics-informed machine learning**

$$\begin{bmatrix} u_1 \\ \vdots \\ u_n \end{bmatrix} = \text{IgANet}\left(\begin{bmatrix} \mathbf{x}_1 \\ \vdots \\ \mathbf{x}_n \end{bmatrix}, \begin{bmatrix} f_1 \\ \vdots \\ f_n \end{bmatrix}, \begin{bmatrix} g_1 \\ \vdots \\ g_n \end{bmatrix}; (\xi^{(k)}, \eta^{(k)})_{k=1}^{N_{\text{samples}}} \right)$$

Compute the solution from the trained neural network as follows

$$u_h(\xi,\eta) = [B_1(\xi,\eta), \dots, B_n(\xi,\eta)] \cdot \begin{bmatrix} u_1 \\ \vdots \\ u_n \end{bmatrix}, \qquad \begin{bmatrix} u_1 \\ \vdots \\ u_n \end{bmatrix} = \text{IgANet}\left(\begin{bmatrix} \mathbf{x}_1 \\ \vdots \\ \mathbf{x}_n \end{bmatrix}, \begin{bmatrix} f_1 \\ \vdots \\ f_n \end{bmatrix}, \begin{bmatrix} g_1 \\ \vdots \\ g_n \end{bmatrix} \right)$$

IgANet architecture



Loss function

Model problem: Poisson's equation with Dirichlet boundary conditions

$$\text{loss}_{\text{PDE}} = \frac{\alpha}{N_\Omega} \sum_{k=1}^{N_\Omega} \left| \Delta \left[u_h \circ \mathbf{x}_h \left(\xi^{(k)}, \eta^{(k)} \right) \right] - f_h \circ \mathbf{x}_h \left(\xi^{(k)}, \eta^{(k)} \right) \right|^2$$

$$\text{loss}_{\text{BDR}} = \frac{\beta}{N_\Gamma} \sum_{k=1}^{N_\Gamma} \left| u_h \circ \mathbf{x}_h \left(\xi^{(k)}, \eta^{(k)} \right) - g_h \circ \mathbf{x}_h \left(\xi^{(k)}, \eta^{(k)} \right) \right|^2$$

Express derivatives with respect to physical space variables using the Jacobian J, the Hessian H and the matrix of squared first derivatives Q (Schillinger *et al.* 2013):

$$\begin{bmatrix} \frac{\partial^2 B}{\partial x^2} \\ \frac{\partial^2 B}{\partial x \partial y} \\ \frac{\partial^2 B}{\partial y^2} \end{bmatrix} = Q^{-\top} \left(\begin{bmatrix} \frac{\partial^2 B}{\partial \xi^2} \\ \frac{\partial^2 B}{\partial \xi \partial \eta} \\ \frac{\partial^2 B}{\partial \eta^2} \end{bmatrix} - H^\top J^{-\top} \begin{bmatrix} \frac{\partial B}{\partial \xi} \\ \frac{\partial B}{\partial \eta} \end{bmatrix} \right)$$

Two-level training strategy

For $[\mathbf{x}_1, \dots, \mathbf{x}_n] \in \mathcal{S}_{\text{geo}}$, $[f_1, \dots, f_n] \in \mathcal{S}_{\text{rhs}}$, $[g_1, \dots, g_n] \in \mathcal{S}_{\text{bcond}}$ **do**

For a batch of randomly sampled $(\xi_k, \eta_k) \in [0,1]^2$ (or the Greville abscissae) **do**

$$\text{Train IgANet} \left(\begin{bmatrix} \mathbf{x}_1 \\ \vdots \\ \mathbf{x}_n \end{bmatrix}, \begin{bmatrix} f_1 \\ \vdots \\ f_n \end{bmatrix}, \begin{bmatrix} g_1 \\ \vdots \\ g_n \end{bmatrix}; (\xi_k, \eta_k)_{k=1}^{N_{\text{samples}}} \right) \mapsto \begin{bmatrix} u_1 \\ \vdots \\ u_n \end{bmatrix}$$

EndFor

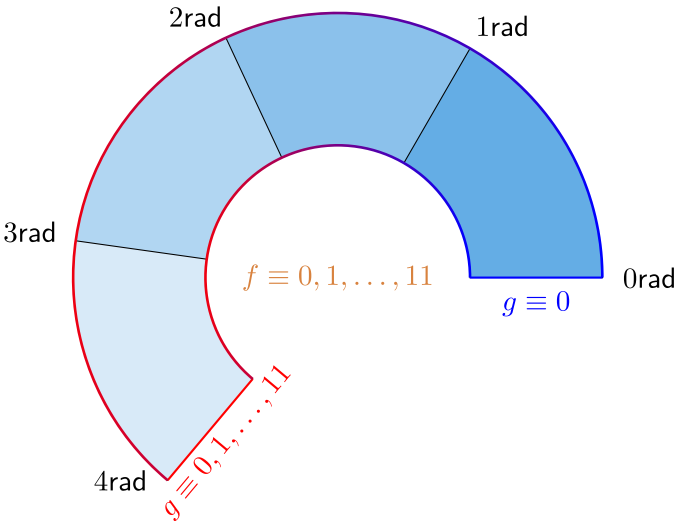
EndFor

Details:

- 7×7 bi-cubic tensor-product B-splines for $\mathbf{x}_h$ and u_h, C^2-continuous
- TensorFlow 2.6, 7-layer neural network with 50 neurons per layer and ReLU activation function (except for output layer), Adam optimizer, 30.000 epochs, training is stopped after 3.000 epochs w/o improvement of the loss value

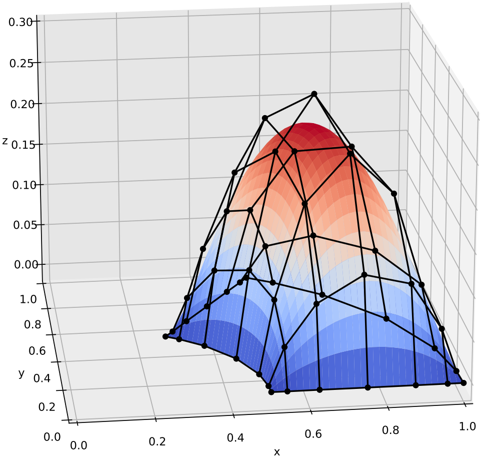
Master thesis work by Frank van Ruiten, TU Delft

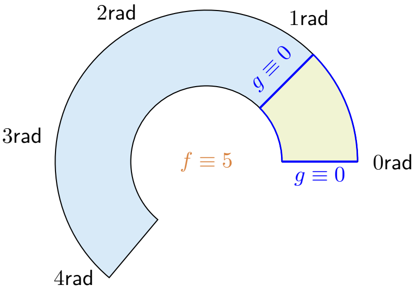
Test case: Poisson's equation on a variable annulus



Master thesis work by Frank van Ruiten, TU Delft

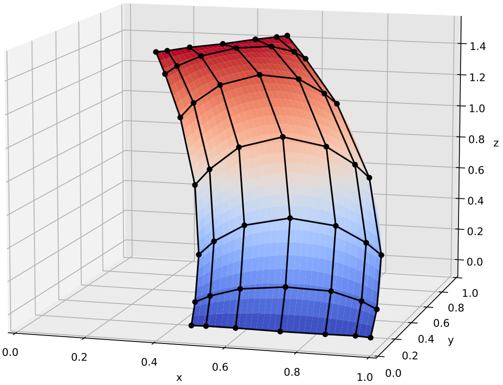
Preliminary results

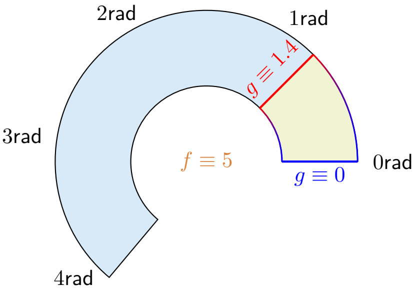




Master thesis work by Frank van Ruiten, TU Delft

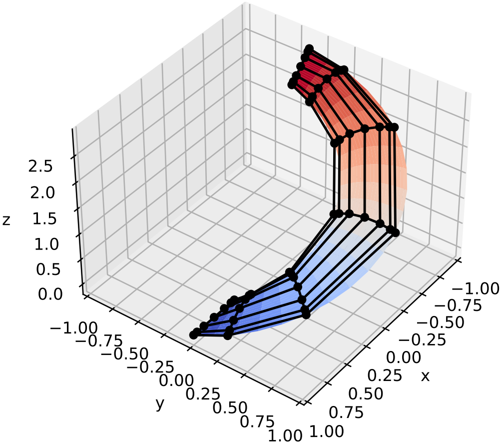
Preliminary results

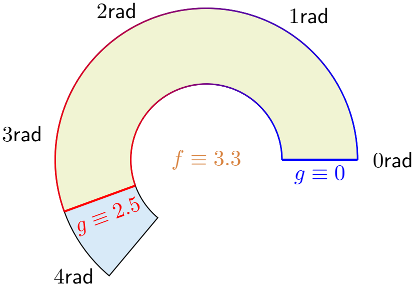




Master thesis work by Frank van Ruiten, TU Delft

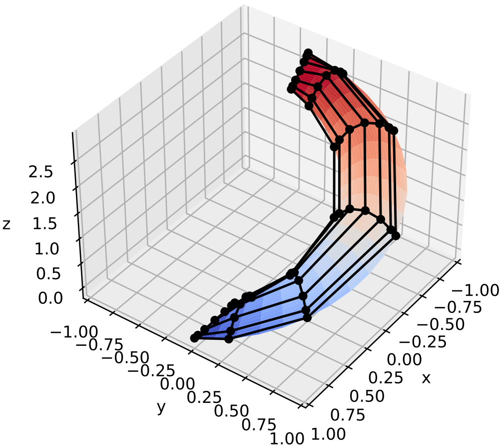
Preliminary results

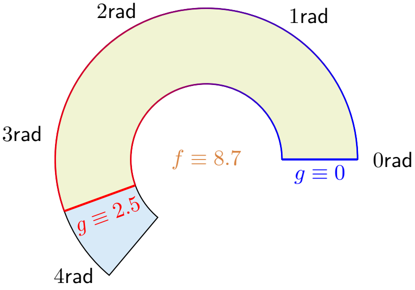




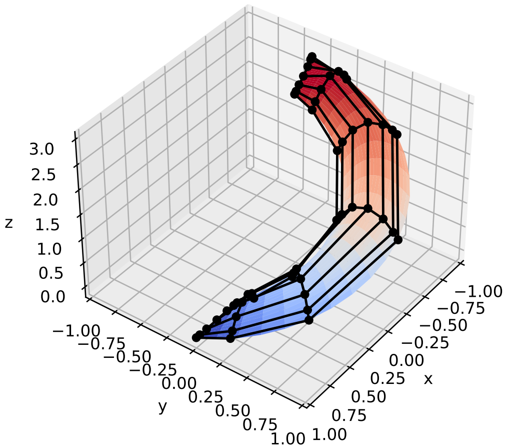
Master thesis work by Frank van Ruiten, TU Delft

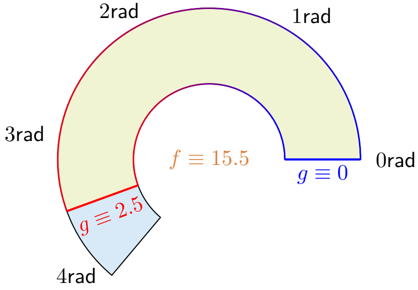
Preliminary results





Preliminary results





Let's have a look under the hood

Computational costs of PINN vs. IgANets, implementation aspects, ...

Computational costs

Working principle of PINNs

$$\mathbf{x} \mapsto u(\mathbf{x}) := \mathsf{NN}(\mathbf{x}; f, g, G) = \sigma_L(\mathbf{W}_L\sigma(\ldots(\sigma_1(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1))) + \mathbf{b}_L)$$

- use AD engine (automated chain rule) to compute derivatives, e.g., $u_x = \mathsf{NN}_x$
- use AD engine on top of AD tree (!!!) to compute gradients w.r.t. weights for training

Computational costs

Working principle of PINNs

$$\mathbf{x} \mapsto u(\mathbf{x}) := \mathsf{NN}(\mathbf{x}; f, g, G) = \sigma_L(\mathbf{W}_L\sigma(\dots(\sigma_1(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1))) + \mathbf{b}_L)$$

- use AD engine (automated chain rule) to compute derivatives, e.g., $u_x = \mathsf{NN}_x$
- use AD engine on top of AD tree (!!!) to compute gradients w.r.t. weights for training

Working principle of IgANets

$$[\mathbf{x}_i, f_i, g_i]_{i=1,\dots,n} \mapsto [u_i]_{i=1,\dots,n} := \mathsf{NN}(\mathbf{x}_i, f_i, g_i, i = 1, \dots, n)$$

- use mathematics to compute derivatives, e.g., $\nabla_{\mathbf{x}} u = \left(\sum_{i=1}^n \nabla_{\boldsymbol{\xi}} B_i(\boldsymbol{\xi}) u_i\right) J_G^{-t}$
- use AD to compute gradients w.r.t. weights for training, i.e. (illustrated in 1D)

$$\frac{\partial(D^r u(\xi))}{\partial w_k} = \sum_{i=1}^n \frac{\partial(D^r b_i^p u_i)}{\partial w_k} = \cancel{\sum_{i=1}^n D^{r+1} b_i^p \frac{\partial \xi}{\partial w_k} u_i} + \sum_{i=1}^n D^r b_i^p \frac{\partial u_i}{\partial w_k}$$

Towards an ML-friendly B-spline evaluation

Major computational task (illustrated in 1D)

Given sampling point $\xi \in [\xi_i, \xi_{i+1})$ compute for $r \geq 0$

$$D^r u(\xi) = \left[D^r b^p_{i-p}(\xi), \ldots, D^r b^p_i(\xi) \right] \cdot \underbrace{[u_{i-p}, \ldots, u_i]}_{\text{network's output}}$$

Textbook derivatives

$$D^r b^p_i(\xi) = p \left(\frac{D^{r-1} b^{p-1}_i(\xi)}{\xi_{i+p} - \xi_i} - \frac{D^{r-1} b^{p-1}_{i+1}(\xi)}{\xi_{i+p-1} - \xi_{i+1}} \right)$$

with (cf. Cox-de-Boor recursion formula)

$$b^p_i(\xi) = \frac{\xi - \xi_i}{\xi_{i+p} - \xi_i} b^{p-1}_i(\xi) + \frac{\xi_{i+p+1} - \xi}{\xi_{i+p+1} - \xi_{i+1}} b^{p-1}_{i+1}(\xi), \quad b^0_i(\xi) = \begin{cases} 1 & \text{if } \xi_i \leq \xi < \xi_{i+1} \\ 0 & \text{otherwise} \end{cases}$$

Towards an ML-friendly B-spline evaluation

Matrix representation of B-splines (Lyche and Mørken 2018)

$$\left[D^r b^p_{i-p}(\xi), \ldots, D^r b^p_i(\xi)\right] = \frac{p!}{(p-r)!}\mathbf{R}_1(\xi)\cdots\mathbf{R}_{p-r}(\xi)D\mathbf{R}_{p-r+1}\cdots D\mathbf{R}_p$$

with $k \times k+1$ matrices $\mathbf{R}_k(\xi)$

$$\mathbf{R}_1(\xi) = \begin{bmatrix} \frac{\xi_{i+1}-\xi}{\xi_{i+1}-\xi_i} & \frac{\xi-\xi_i}{\xi_{i+1}-\xi_i} \end{bmatrix}, \quad \mathbf{R}_2(\xi) = \begin{bmatrix} \frac{\xi_{i+1}-\xi}{\xi_{i+1}-\xi_{i-1}} & \frac{\xi-\xi_{i-1}}{\xi_{i+1}-\xi_{i-1}} & 0 \\ 0 & \frac{\xi_{i+2}-\xi}{\xi_{i+2}-\xi_i} & \frac{\xi-\xi_i}{\xi_{i+2}-\xi_i} \end{bmatrix}, \quad \ldots$$

and

$$D\mathbf{R}_1(\xi) = \begin{bmatrix} \frac{-1}{\xi_{i+1}-\xi_i} & \frac{1}{\xi_{i+1}-\xi_i} \end{bmatrix}, \quad D\mathbf{R}_2(\xi) = \begin{bmatrix} \frac{-1}{\xi_{i+1}-\xi_{i-1}} & \frac{1}{\xi_{i+1}-\xi_{i-1}} & 0 \\ 0 & \frac{-1}{\xi_{i+2}-\xi_i} & \frac{1}{\xi_{i+2}-\xi_i} \end{bmatrix}, \quad \ldots$$

Towards an ML-friendly B-spline evaluation

Matrix representation of B-splines (Lyche and Mørken 2018)

$$\left[D^r b^p_{i-p}(\xi), \ldots, D^r b^p_i(\xi)\right] = \frac{p!}{(p-r)!}\mathbf{R}_1(\xi)\cdots\mathbf{R}_{p-r}(\xi)D\mathbf{R}_{p-r+1}\cdots D\mathbf{R}_p$$

Costs of matrix assembly (arithmetic operations)

$$3p^2 - 3p - r^2 + r \text{ (leading } D\mathbf{R}'s) \quad \text{vs.} \quad 2p^2 - 2p + r^2 - r \text{ (trailing } D\mathbf{R}'s)$$

Costs of matrix-matrix products ($p \geq 3$)

$$(4p^3 - 3p^2 - 7p - 6)/6 \text{ (L2R)} \quad \text{vs.} \quad (4p^4 - 15p^3 + 17p^2 - 6p)/6 \text{ (R2L)}$$

Towards an ML-friendly B-spline evaluation

Matrix representation of B-splines (Lyche and Mørken 2018)

$$\left[D^r b^p_{i-p}(\xi), \ldots, D^r b^p_i(\xi)\right] = \frac{p!}{(p-r)!}\mathbf{R}_1(\xi)\cdots\mathbf{R}_{p-r}(\xi)D\mathbf{R}_{p-r+1}\cdots D\mathbf{R}_p$$

Costs of matrix assembly (arithmetic operations)

$$3p^2 - 3p - r^2 + r \text{ (leading } D\mathbf{R}'s) \quad \text{vs.} \quad 2p^2 - 2p + r^2 - r \text{ (trailing } D\mathbf{R}'s)$$

Costs of matrix-matrix products ($p \geq 3$)

$$(4p^3 - 3p^2 - 7p - 6)/6 \text{ (L2R)} \quad \text{vs.} \quad (4p^4 - 15p^3 + 17p^2 - 6p)/6 \text{ (R2L)}$$

Can we do better?

An ML-friendly B-spline evaluation

Algorithm 2.22 from (Lyche and Mørken 2018) with modifications

1. $\mathbf{b} = 1$
2. For $k = 1, \ldots, p - r$
   1. $\mathbf{t}_1 = (\xi_{i-k+1}, \ldots, \xi_i)$
   2. $\mathbf{t}_{21} = (\xi_{i+1}, \ldots, \xi_{i+k}) - \mathbf{t}_1$
   3. $\textbf{mask} = (\mathbf{t}_{21} < \text{tol})$
   4. $\mathbf{w} = (\xi - \mathbf{t}_1 - \textbf{mask}) \div (\mathbf{t}_{21} - \textbf{mask})$
   5. $\mathbf{b} = [(1 - \mathbf{w}) \odot \mathbf{b}, 0] + [0, \mathbf{w} \odot \mathbf{b}]$
3. For $k = p - r + 1, \ldots, p$
   1. $\mathbf{t}_1 = (\xi_{i-k+1}, \ldots, \xi_i)$
   2. $\mathbf{t}_{21} = (\xi_{i+1}, \ldots, \xi_{i+k}) - \mathbf{t}_1$
   3. $\textbf{mask} = (\mathbf{t}_{21} < \text{tol})$
   4. $\mathbf{w} = (1 - \textbf{mask}) \div (\mathbf{t}_{21} - \textbf{mask})$
   5. $\mathbf{b} = [-\mathbf{w} \odot \mathbf{b}, 0] + [0, \mathbf{w} \odot \mathbf{b}]$

where $\div$ and $\odot$ denote the element-wise division and multiplication of vectors, respectively.

An ML-friendly B-spline evaluation

Algorithm 2.22 from (Lyche and Mørken 2018) with modifications

1. $\mathbf{b} = 1$
2. For $k = 1, \dots, p - r$
   1. $\mathbf{t}_1 = (\xi_{i-k+1}, \dots, \xi_i)$
   2. $\mathbf{t}_{21} = (\xi_{i+1}, \dots, \xi_{i+k}) - \mathbf{t}_1$
   3. $\textbf{mask} = (\mathbf{t}_{21} < \text{tol})$
   4. $\mathbf{w} = (\xi - \mathbf{t}_1 - \textbf{mask}) \div (\mathbf{t}_{21} - \textbf{mask})$
   5. $\mathbf{b} = [(1 - \mathbf{w}) \odot \mathbf{b}, 0] + [0, \mathbf{w} \odot \mathbf{b}]$
3. For $k = p - r + 1, \dots, p$
   1. $\mathbf{t}_1 = (\xi_{i-k+1}, \dots, \xi_i)$
   2. $\mathbf{t}_{21} = (\xi_{i+1}, \dots, \xi_{i+k}) - \mathbf{t}_1$
   3. $\textbf{mask} = (\mathbf{t}_{21} < \text{tol})$
   4. $\mathbf{w} = (1 - \textbf{mask}) \div (\mathbf{t}_{21} - \textbf{mask})$
   5. $\mathbf{b} = [-\mathbf{w} \odot \mathbf{b}, 0] + [0, \mathbf{w} \odot \mathbf{b}]$

where $\div$ and $\odot$ denote the element-wise division and multiplication of vectors, respectively.

Costs: $5(p^2 + p)$ arithmetic operations $+ \ 2p - 1$ for $\mathbf{b} \cdot \mathbf{u}$

An ML-friendly *multi-variate* B-spline evaluation

Task: Given pre-evaluated vectors of univariate B-spline basis functions $\mathbf{b}^d$ compute

$$u(\xi, \eta, \zeta) = [\mathbf{b}_1(\xi) \otimes \mathbf{b}_2(\eta) \otimes \mathbf{b}_3(\zeta)] \cdot \mathbf{u}$$

but sub-matrix of coefficients $\mathbf{u} := u[\mathbf{i} - \mathbf{p} : \mathbf{i}]$ is not contiguous in memory

An ML-friendly *multi-variate* B-spline evaluation

Task: Given pre-evaluated vectors of univariate B-spline basis functions $\mathbf{b}^d$ compute

$$u(\xi, \eta, \zeta) = [\mathbf{b}_1(\xi) \otimes \mathbf{b}_2(\eta) \otimes \mathbf{b}_3(\zeta)] \cdot \mathbf{u}$$

but sub-matrix of coefficients $\mathbf{u} := u[\mathbf{i} - \mathbf{p} : \mathbf{i}]$ is not contiguous in memory

Since $(\mathbf{b}_1 \otimes \mathbf{b}_2 \otimes \mathbf{b}_3) \cdot \mathbf{u} = (\mathbf{I} \otimes \mathbf{I} \otimes \mathbf{b}_3) \cdot (\mathbf{I} \otimes \mathbf{b}_2 \otimes \mathbf{I}) \cdot (\mathbf{b}_1 \otimes \mathbf{I} \otimes \mathbf{I}) \cdot \mathbf{u}$ we can use

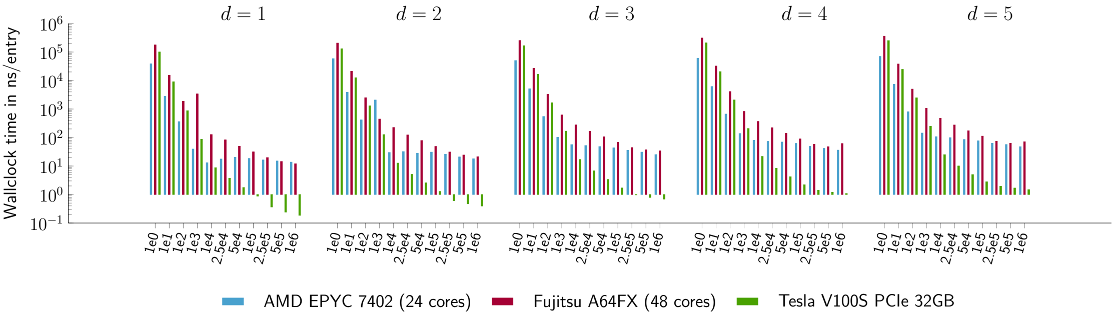
Algorithm 993 from (Fackler 2019) with modifications

For $d = 1, 2$

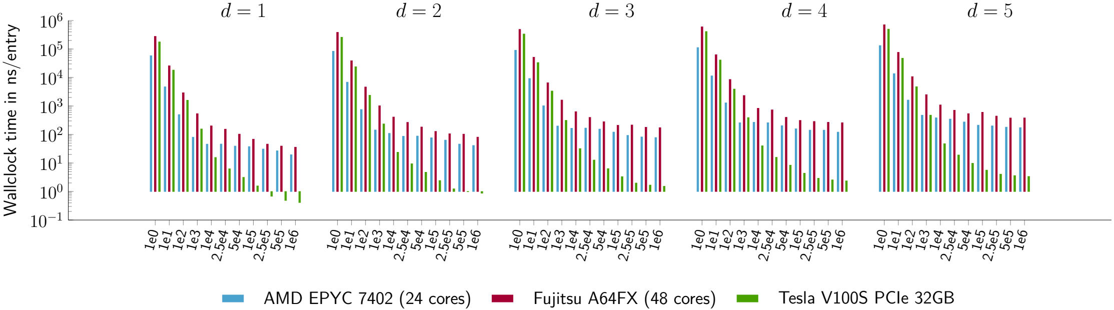
1. $\mathbf{u} = \text{reshape}(\mathbf{u}, [\cdot], n_d)$
2. $\mathbf{u} = \mathbf{b}_d \cdot \mathbf{u}^\top$

Output: $\mathbf{u} = u(\xi, \eta, \zeta)$

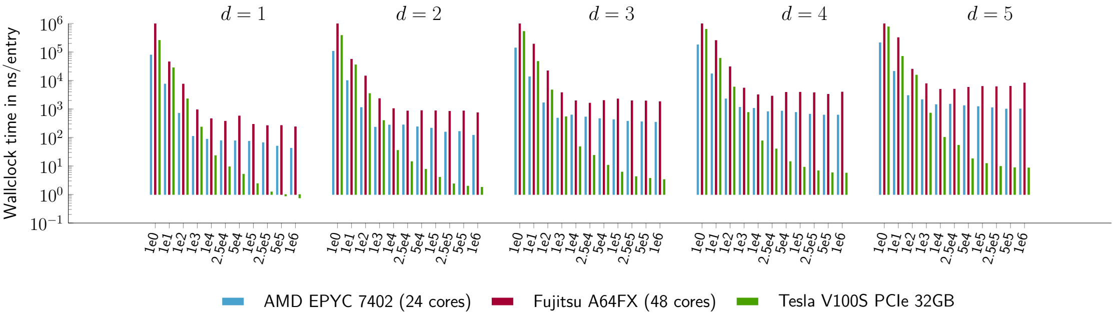
Performance evaluation - univariate B-splines



Performance evaluation - bivariate B-splines



Performance evaluation - trivariate B-splines

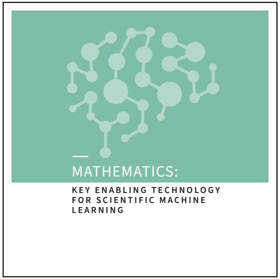


Conclusion and outlook

IgANets combine isogeometric analysis with physics-informed machine learning to enable **interactive design-through-analysis** workflows

WIP

- interactive DTA workflow (/w SURF)
- use of IgA and IgANets in concert
- transfer learning upon basis refinement



Short paper: Möller, Toshniwal, van Ruiten: *Physics-informed machine learning embedded into isogeometric analysis*, 2021. ☞

What's next

1. Journal paper and code release (including Python API) in preparation
2. CISM-ECCOMAS Summer School *Scientific Machine Learning in Design Optimization*

IgANets: Physics-machine learning embedded into Isogeometric Analysis

Matthias Möller

Department of Applied Mathematics, TU Delft

HOFEIM 2023

Joint work with Deepesh Toshniwal & Frank van Ruiten (TU Delft),
Casper van Leeuwen & Paul Melis (SURF), and Jaewook Lee (TU Vienna)

Thank you very much!