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
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

<table>
<thead>
<tr>
<th>Front-ends</th>
<th>Back-ends</th>
</tr>
</thead>
<tbody>
<tr>
<td>IgANet-frontend</td>
<td>IgANet</td>
</tr>
<tr>
<td>by SURF</td>
<td>by TU Vienna</td>
</tr>
<tr>
<td><img src="image" alt="gustaf logo" /></td>
<td><img src="image" alt="Sigma logo" /></td>
</tr>
</tbody>
</table>

**WebSockets protocol for interactive Design-through-Analysis**
Physics-informed machine learning

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

\[
F = \partial_t U + \nabla \cdot f(U)
\]
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

\[
F = \partial_t U + \nabla \cdot f(U)
\]
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

![PINN Diagram](image)

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

\[
G_\theta(u)(y) = \sum_{k=1}^{q} b_k(u(x_1), u(x_2), \ldots, u(x_m)) t_k(y)
\]

- branch
- 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*

\[
G_\theta(u)(y) = \sum_{k=1}^{q} b_k(u(x_1), u(x_2), \ldots, u(x_m)) t_k(y)
\]

Don’t we know good bases?
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 $\Omega_h \subset \mathbb{R}^d$

$$x_h(\xi, \eta) = \sum_{i=1}^{n} B_i(\xi, \eta) \cdot x_i \quad \forall (\xi, \eta) \in [0, 1]^2 =: \hat{\Omega}$$

- the shape of $\Omega_h$ is fully specified by the set of control points $x_i \in \mathbb{R}^d$
Isogeometric Analysis

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

$$x_h(\xi, \eta) = \sum_{i=1}^{n} B_i(\xi, \eta) \cdot x_i \quad \forall (\xi, \eta) \in [0, 1]^2 =: \hat{\Omega}$$

- the shape of $\Omega_h$ is fully specified by the set of **control points** $x_i \in \mathbb{R}^d$
- interior control points must be chosen such that ‘grid lines’ do not fold as this violates the bijectivity of $x_h : \hat{\Omega} \rightarrow \Omega_h$
Isogeometric Analysis

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

$$x_h(\xi, \eta) = \sum_{i=1}^{n} B_i(\xi, \eta) \cdot x_i \quad \forall (\xi, \eta) \in [0, 1]^2 =: \hat{\Omega}$$

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

**Model problem**: Poisson’s equation

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

with

(geometry) \quad x_h(\xi, \eta) = \sum_{i=1}^{n} B_i(\xi, \eta) \cdot x_i \quad \forall (\xi, \eta) \in [0, 1]^2

(solution) \quad u_h \circ x_h(\xi, \eta) = \sum_{i=1}^{n} B_i(\xi, \eta) \cdot u_i \quad \forall (\xi, \eta) \in [0, 1]^2

(r.h.s vector) \quad f_h \circ x_h(\xi, \eta) = \sum_{i=1}^{n} B_i(\xi, \eta) \cdot f_i \quad \forall (\xi, \eta) \in [0, 1]^2

(boundary conditions) \quad g_h \circ x_h(\xi, \eta) = \sum_{i=1}^{n} B_i(\xi, \eta) \cdot g_i \quad \forall (\xi, \eta) \in \partial[0, 1]^2
Isogeometric Analysis

**Abstract representation**
Given $x_i$ (geometry), $f_i$ (r.h.s. vector), and $g_i$ (boundary conditions), compute

$$
\begin{bmatrix}
    u_1 \\
    \vdots \\
    u_n
\end{bmatrix} = A^{-1} \left( \begin{bmatrix}
    x_1 \\
    \vdots \\
    x_n
\end{bmatrix}, \begin{bmatrix}
    g_1 \\
    \vdots \\
    g_n
\end{bmatrix} \right) \cdot b \left( \begin{bmatrix}
    x_1 \\
    \vdots \\
    x_n
\end{bmatrix}, \begin{bmatrix}
    f_1 \\
    \vdots \\
    f_n
\end{bmatrix}, \begin{bmatrix}
    g_1 \\
    \vdots \\
    g_n
\end{bmatrix} \right)
$$

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

$$(\xi, \eta) \in [0, 1]^2 \quad \mapsto \quad u_h \circ x_h(\xi, \eta) = [B_1(\xi, \eta), \ldots, B_n(\xi, \eta)] \cdot \begin{bmatrix}
    u_1 \\
    \vdots \\
    u_n
\end{bmatrix}$$
Isogeometric Analysis

Abstract representation
Given $x_i$ (geometry), $f_i$ (r.h.s. vector), and $g_i$ (boundary conditions), compute

$$
\begin{bmatrix}
u_1 \\
\vdots \\
u_n
\end{bmatrix} = A^{-1} \left( \begin{bmatrix} x_1 \\
\vdots \\
x_n \end{bmatrix}, \begin{bmatrix} g_1 \\
\vdots \\
g_n \end{bmatrix} \right) \cdot b \left( \begin{bmatrix} x_1 \\
\vdots \\
x_n \end{bmatrix}, \begin{bmatrix} f_1 \\
\vdots \\
f_n \end{bmatrix}, \begin{bmatrix} g_1 \\
\vdots \\
g_n \end{bmatrix} \right)
$$

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

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

Let us interpret the sets of B-spline coefficients $\{x_i\}$, $\{f_i\}$, 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.
Isogeometric Analysis + Physics-Informed Machine Learning

IgANet: replace *computation*

\[
\begin{bmatrix}
  u_1 \\
  \vdots \\
  u_n
\end{bmatrix} = A^{-1} \left( \begin{bmatrix}
  x_1 \\
  \vdots \\
  x_n
\end{bmatrix}, \begin{bmatrix}
  g_1 \\
  \vdots \\
  g_n
\end{bmatrix} \right) \cdot b \left( \begin{bmatrix}
  x_1 \\
  \vdots \\
  x_n
\end{bmatrix}, \begin{bmatrix}
  f_1 \\
  \vdots \\
  f_n
\end{bmatrix}, \begin{bmatrix}
  g_1 \\
  \vdots \\
  g_n
\end{bmatrix} \right)
\]
IgANet: replace computation by physics-informed machine learning

\[
\begin{bmatrix}
  u_1 \\
  \vdots \\
  u_n
\end{bmatrix} = \text{IgANet} \left( \begin{bmatrix}
  x_1 \\
  \vdots \\
  x_n
\end{bmatrix}, \begin{bmatrix}
  f_1 \\
  \vdots \\
  f_n
\end{bmatrix}, \begin{bmatrix}
  g_1 \\
  \vdots \\
  g_n
\end{bmatrix}; \left( \xi^{(k)}, \eta^{(k)} \right)_{k=1}^{N_{\text{samples}}} \right)
\]
Isogeometric Analysis + Physics-Informed Machine Learning

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

\[
\begin{bmatrix}
u_1 \\
\vdots \\
u_n
\end{bmatrix} = \text{IgANet}\left(\begin{bmatrix}
x_1 \\
\vdots \\
x_n
\end{bmatrix}, \begin{bmatrix}
f_1 \\
\vdots \\
f_n
\end{bmatrix}, \begin{bmatrix}
g_1 \\
\vdots \\
g_n
\end{bmatrix}; (\xi^{(k)}, \eta^{(k)})_{k=1}^{N_{\text{samples}}} \right)
\]

Compute the solution from the trained neural network as follows

\[
u_h(\xi, \eta) = [B_1(\xi, \eta), \ldots, B_n(\xi, \eta)] \cdot \begin{bmatrix}
u_1 \\
\vdots \\
u_n
\end{bmatrix} = \text{IgANet}\left(\begin{bmatrix}
x_1 \\
\vdots \\
x_n
\end{bmatrix}, \begin{bmatrix}
f_1 \\
\vdots \\
f_n
\end{bmatrix}, \begin{bmatrix}
g_1 \\
\vdots \\
g_n
\end{bmatrix} \right)
\]
IgANet architecture

\[ \text{loss} = \text{loss}_\text{PDE} + \text{loss}_\text{BDR} \]

\[ \frac{\partial \text{loss}}{\partial (\mathbf{w}, \mathbf{b})} \rightarrow \text{update } \mathbf{w}, \mathbf{b} \]

and continue training

\[ \text{loss} < \varepsilon \rightarrow \text{end training} \]
Loss function

**Model problem**: Poisson’s equation with Dirichlet boundary conditions

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

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

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):

\[
\begin{pmatrix}
\frac{\partial^2 B}{\partial x^2} \\
\frac{\partial^2 B}{\partial x \partial y} \\
\frac{\partial^2 B}{\partial y^2}
\end{pmatrix}
= Q^{-\top} \begin{pmatrix}
\frac{\partial^2 B}{\partial \xi^2} \\
\frac{\partial^2 B}{\partial \xi \partial \eta} \\
\frac{\partial^2 B}{\partial \eta^2}
\end{pmatrix} - H^\top J^{-\top} \begin{pmatrix}
\frac{\partial B}{\partial \xi} \\
\frac{\partial B}{\partial \eta}
\end{pmatrix}
\]
Two-level training strategy

For \([x_1, \ldots, x_n] \in S_{\text{geo}}, [f_1, \ldots, f_n] \in S_{\text{rhs}}, [g_1, \ldots, g_n] \in S_{\text{bcond}}\) do

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

Train IgANet:

\[
\begin{pmatrix}
[x_1] & [f_1] & [g_1] \\
\vdots & \vdots & \vdots \\
[x_n] & [f_n] & [g_n]
\end{pmatrix}
\rightarrow
\begin{pmatrix}
u_1 \\
\vdots \\
u_n
\end{pmatrix}
\]

EndFor

EndFor

Details:

- 7 × 7 bi-cubic tensor-product B-splines for \(x_h\) 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
Test case: Poisson’s equation on a variable annulus

\[ f \equiv 0, 1, \ldots, 11 \]
\[ g \equiv 0 \]
Preliminary results

Master thesis work by Frank van Ruiten, TU Delft
Preliminary results

\[ g \equiv 0, 4 \text{ rad} \]

Master thesis work by Frank van Ruiten, TU Delft
Preliminary results

\[ g \equiv 0 \]

\[ g \equiv 2.5 \]

\[ f \equiv 3.3 \]

\[ g \equiv 0 \]

\[ 0 \text{rad} \]

\[ 1 \text{rad} \]

\[ 2 \text{rad} \]

\[ 3 \text{rad} \]

\[ 4 \text{rad} \]
Preliminary results

Master thesis work by Frank van Ruiten, TU Delft
Preliminary results

Master thesis work by Frank van Ruiten, TU Delft
Let’s have a look under the hood

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

Working principle of PINNs

\[ x \mapsto u(x) := \text{NN}(x; f, g, G) = \sigma_L(W_L\sigma(\ldots(\sigma_1(W_1x + b_1))) + b_L) \]

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

Working principle of PINNs

\[
\begin{align*}
  x & \mapsto u(x) := \text{NN}(x; f, g, G) = \sigma_L(\mathbf{W}_L \sigma(\ldots(\sigma_1(\mathbf{W}_1 x + b_1)))) + b_L \\
  \text{• use AD engine (automated chain rule) to compute derivatives, e.g., } u_x &= \text{NN}_x \\
  \text{• use AD engine on top of AD tree (!!!) to compute gradients w.r.t. weights for training}
\end{align*}
\]

Working principle of IgANets

\[
\begin{align*}
  [x_i, f_i, g_i]_{i=1,\ldots,n} & \mapsto [u_i]_{i=1,\ldots,n} := \text{NN}(x_i, f_i, g_i, i = 1, \ldots, n) \\
  \text{• use mathematics to compute derivatives, e.g., } \nabla_x u &= (\sum_{i=1}^{n} \nabla \xi B_i(\xi) u_i) J_G^{-t} \\
  \text{• use AD to compute gradients w.r.t. weights for training, i.e. (illustrated in 1D)}
\end{align*}
\]

\[
\begin{align*}
  \frac{\partial(D^r u(\xi))}{\partial w_k} &= \sum_{i=1}^{n} \frac{\partial(D^r b_i^p u_i)}{\partial w_k} = \sum_{i=1}^{n} D^{r+1} b_i^p \frac{\partial \xi}{\partial w_k} u_i + \sum_{i=1}^{n} D^r b_i^p \frac{\partial u_i}{\partial w_k}
\end{align*}
\]
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 \( r \geq 0 \)

\[
D^r u(\xi) = \left[ D^r b_{i-p}^p(\xi), \ldots, D^r b_i^p(\xi) \right] \cdot [u_{i-p}, \ldots, u_i]
\]

Textbook derivatives

\[
D^r b_i^p(\xi) = p \left( \frac{D^{r-1}b_i^{p-1}(\xi)}{\xi_{i+p} - \xi_i} - \frac{D^{r-1}b_{i+1}^{p-1}(\xi)}{\xi_{i+p+1} - \xi_{i+1}} \right)
\]

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

\[
b_i^p(\xi) = \frac{\xi - \xi_i}{\xi_{i+p} - \xi_i} b_i^{p-1}(\xi) + \frac{\xi_{i+p+1} - \xi}{\xi_{i+p+1} - \xi_{i+1}} b_{i+1}^{p-1}(\xi), \quad b_i^0(\xi) = \begin{cases} 1 & \text{if } \xi_i \leq \xi < \xi_{i+1} \\ 0 & \text{otherwise} \end{cases}
\]
Towards an ML-friendly B-spline evaluation

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

\[
\begin{bmatrix}
D_r b_{i-p}^p(\xi), \ldots, D_r b_i^p(\xi)
\end{bmatrix} = \frac{p!}{(p-r)!} R_1(\xi) \cdots R_{p-r}(\xi) D R_{p-r+1} \cdots D R_p
\]

with \( k \times k + 1 \) matrices \( R_k(\xi) \)

\[
R_1(\xi) = \begin{bmatrix}
\frac{\xi_{i+1}-\xi_i}{\xi_{i+1}-\xi_i} & \frac{\xi_i-\xi}{\xi_{i+1}-\xi_i}
\end{bmatrix},
R_2(\xi) = \begin{bmatrix}
\frac{\xi_{i+1}-\xi_i}{\xi_{i+1}-\xi_{i-1}} & \frac{\xi_i-\xi_{i-1}}{\xi_{i+1}-\xi_{i-1}} & 0 \\
0 & \frac{\xi_{i+2}-\xi_i}{\xi_{i+2}-\xi_i} & \frac{\xi_i-\xi}{\xi_{i+2}-\xi_i}
\end{bmatrix}, \ldots
\]

and

\[
D R_1(\xi) = \begin{bmatrix}
-1 & 1 \\
\frac{1}{\xi_{i+1}-\xi_i} & \frac{1}{\xi_{i+1}-\xi_i}
\end{bmatrix},
D R_2(\xi) = \begin{bmatrix}
\frac{-1}{\xi_{i+1}-\xi_{i-1}} & \frac{1}{\xi_{i+1}-\xi_{i-1}} & 0 \\
0 & \frac{-1}{\xi_{i+2}-\xi_i} & \frac{1}{\xi_{i+2}-\xi_i}
\end{bmatrix}, \ldots
\]
Towards an ML-friendly B-spline evaluation

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

\[
\left[ D^r b_{i-p}^p(\xi), \ldots, D^r b_i^p(\xi) \right] = \frac{p!}{(p-r)!} R_1(\xi) \cdots R_{p-r}(\xi) D R_{p-r+1} \cdots D R_p
\]

Costs of matrix assembly (arithmetic operations)

\[3p^2 - 3p - r^2 + r \text{ (leading } DR\text{'s)}\] vs. \[2p^2 - 2p + r^2 - r \text{ (trailing } DR\text{'s)}\]

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

\[\frac{(4p^3 - 3p^2 - 7p - 6)}{6} \text{ (L2R)}\] vs. \[\frac{(4p^4 - 15p^3 + 17p^2 - 6p)}{6} \text{ (R2L)}\]
Towards an ML-friendly B-spline evaluation

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

\[
\left[ D^r b^p_{i-p}(\xi), \ldots, D^r b^p_i(\xi) \right] = \frac{p!}{(p-r)!} R_1(\xi) \cdots R_{p-r}(\xi) DR_{p-r+1} \cdots DR_p
\]

Costs of matrix assembly (arithmetic operations)

\[3p^2 - 3p - r^2 + r\] (leading DR’s) vs. \[2p^2 - 2p + r^2 - r\] (trailing DR’s)

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

\[(4p^3 - 3p^2 - 7p - 6)/6\] (L2R) vs. \[(4p^4 - 15p^3 + 17p^2 - 6p)/6\] (R2L)

Can we do better?
An ML-friendly B-spline evaluation

**Algorithm 2.22** from (Lyche and Mørken 2018) with modifications

1. \( b = 1 \)
2. For \( k = 1, \ldots, p - r \)
   1. \( t_1 = (\xi_{i-k+1}, \ldots, \xi_i) \)
   2. \( t_{21} = (\xi_{i+1}, \ldots, \xi_{i+k}) - t_1 \)
   3. \( \text{mask} = (t_{21} < \text{tol}) \)
   4. \( w = (\xi - t_1 - \text{mask}) \div (t_{21} - \text{mask}) \)
   5. \( b = [(1 - w) \odot b, 0] + [0, w \odot b] \)
3. For \( k = p - r + 1, \ldots, p \)
   1. \( t_1 = (\xi_{i-k+1}, \ldots, \xi_i) \)
   2. \( t_{21} = (\xi_{i+1}, \ldots, \xi_{i+k}) - t_1 \)
   3. \( \text{mask} = (t_{21} < \text{tol}) \)
   4. \( w = (1 - \text{mask}) \div (t_{21} - \text{mask}) \)
   5. \( b = [-w \odot b, 0] + [0, w \odot b] \)

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. \( b = 1 \)
2. For \( k = 1, \ldots, p - r \)
   1. \( t_1 = (\xi_{i-k+1}, \ldots, \xi_i) \)
   2. \( t_{21} = (\xi_{i+1}, \ldots, \xi_{i+k}) - t_1 \)
   3. \( \text{mask} = (t_{21} < \text{tol}) \)
   4. \( w = (\xi - t_1 - \text{mask}) \div (t_{21} - \text{mask}) \)
   5. \( b = [(1 - w) \odot b, 0] + [0, w \odot b] \)
3. For \( k = p - r + 1, \ldots, p \)
   1. \( t_1 = (\xi_{i-k+1}, \ldots, \xi_i) \)
   2. \( t_{21} = (\xi_{i+1}, \ldots, \xi_{i+k}) - t_1 \)
   3. \( \text{mask} = (t_{21} < \text{tol}) \)
   4. \( w = (1 - \text{mask}) \div (t_{21} - \text{mask}) \)
   5. \( b = [-w \odot b, 0] + [0, w \odot b] \)

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

**Costs:** \( 5(p^2 + p) \) arithmetic operations + \( 2p - 1 \) for \( b \cdot u \)
An ML-friendly *multi-variate* B-spline evaluation

**Task:** Given pre-evaluated vectors of univariate B-spline basis functions $b^d$ compute

$$ u(\xi, \eta, \zeta) = [b_1(\xi) \otimes b_2(\eta) \otimes b_3(\zeta)] \cdot u $$

but sub-matrix of coefficients $u := u[i - p : i]$ is not contiguous in memory
An ML-friendly *multi-variate* B-spline evaluation

**Task:** Given pre-evaluated vectors of univariate B-spline basis functions $b^d$ compute

$$u(\xi, \eta, \zeta) = \left[ b_1(\xi) \otimes b_2(\eta) \otimes b_3(\zeta) \right] \cdot u$$

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

Since $(b_1 \otimes b_2 \otimes b_3) \cdot u = (I \otimes I \otimes b_3) \cdot (I \otimes b_2 \otimes I) \cdot (b_1 \otimes I \otimes I) \cdot u$ we can use

**Algorithm 993** from (Fackler 2019) with modifications

For $d = 1, 2$

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

Output: $u = u(\xi, \eta, \zeta)$
Performance evaluation - univariate B-splines

Wallclock time in ns/entry

$d = 1$

$d = 2$

$d = 3$

$d = 4$

$d = 5$

- AMD EPYC 7402 (24 cores)
- Fujitsu A64FX (48 cores)
- Tesla V100S PCIe 32GB
Performance evaluation - bivariate B-splines

Wallclock time in ns/entry

$d = 1$  $d = 2$  $d = 3$  $d = 4$  $d = 5$

- AMD EPYC 7402 (24 cores)
- Fujitsu A64FX (48 cores)
- Tesla V100S PCIe 32GB
Performance evaluation - trivariate B-splines

![Graph showing performance evaluation for different dimensions](image)

- Wallclock time in ns/entry
- d = 1
- d = 2
- d = 3
- d = 4
- d = 5

- AMD EPYC 7402 (24 cores)
- Fujitsu A64FX (48 cores)
- Tesla V100S PCIe 32GB
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!