Non-intrusive Reduced Order Models for CFD

Jan S Hesthaven
EPFL, Lausanne, CH
Jan.Hesthaven@epfl.ch

w/ S. Ubbiali (CSCS, CH), J. Yu (Beihang Univ, PRC), B. Stamm (Aachen, D)
Parametrized problems

The parameters can describe

- Materials
- Sources
- Geometries
- Parameterized uncertainty
- Time
- etc
Parametrized problems

The parameters can describe

- Materials
- Sources
- Geometries
- Parameterized uncertainty
- Time
- etc

An example

\[-\nabla^2 u(x, \mu) = f(x, \mu)\]
\[u(x, \mu) = g(x, \mu)\]

\[x \in \Omega(\mu)\]
\[x \in \partial \Omega(\mu)\]
Model order reduction

Assume we are interested in

\[-\nabla^2 u(x, \mu) = f(x, \mu) \quad x \in \Omega\]

and wish to solve it accurately for many values of the parameter \( \mu \).
Model order reduction

Assume we are interested in

$$-\nabla^2 u(x, \mu) = f(x, \mu) \quad x \in \Omega$$

and wish to solve it accurately for many values of the parameter $\mu$

We can use our favorite numerical method

$$A_h u_h(x, \mu) = f_h(x, \mu) \quad \text{dim}(u_h) = N \gg 1$$

For many parameter values, this is expensive - and slow!
Reduced order models

An **accurate** way to evaluate the solution at new parameter values *at reduced complexity*.
Reduced order models

An accurate way to evaluate the solution at new parameter values at reduced complexity.

\[
\mathcal{L}_h(u_h(\mu); \mu) = 0
\]

input: parameter value \( \mu \in D \)

output: \( s_h(\mu) = l(u_h(\mu); \mu) \)
Model order reduction

Assume we (somehow) know

\[ u_h(x, \mu) \simeq u_{RB}(x, \mu) = V a(\mu) \quad V^T V = I \]

\[ \text{dim}(a) = N \quad \text{dim}(V) = \mathcal{N} \times N \]
Model order reduction

Assume we (somehow) know

\[ u_h(x, \mu) \simeq u_{RB}(x, \mu) = V a(\mu) \quad V^T V = I \]

\[ \dim(a) = N \quad \dim(V) = N \times N \]

Then we can recover a solution for a new parameter as little cost

\[ (V^T A_h V) V^T u_h(\mu) = V^T f_h(\mu) \]
Model order reduction

Assume we (somehow) know

\[ u_h(x, \mu) \simeq u_{RB}(x, \mu) = V a(\mu) \quad V^T V = I \]

\[ \text{dim}(a) = N \quad \text{dim}(V) = \mathcal{N} \times N \]

Then we can recover a solution for a new parameter as little cost

\[ (V^T A_h V) V^T u_h(\mu) = V^T f_h(\mu) \]

\[ N \times N \quad N \quad N \]
Model order reduction

Assume we (somehow) know

\[ u_h(x, \mu) \simeq u_{RB}(x, \mu) = V a(\mu) \quad \quad V^T V = I \]

\[ \text{dim}(a) = N \quad \quad \text{dim}(V) = N \times N \]

Then we can recover a solution for a new parameter as little cost

\[
\begin{align*}
(V^T A_h V) V^T u_h(\mu) &= V^T f_h(\mu) \\
N \times N &\quad \quad N &\quad \quad N
\end{align*}
\]

.. if this behaves!
When can we expect this to work?

For this to be successful there must be some structure to the solution under parameter variation
When can we expect this to work?

For this to be successful there must be some structure to the solution under parameter variation.

**Assumption:** The solution varies smoothly on a low-dimensional manifold under parameter variation.

Choosing the samples well, we should be able to derive good approximations for all parameters.
Check the assumption

Two dimensional parameterization with polar angle and frequency

\((k, \theta) \in [1, 25] \times [0, \pi], \phi \text{ is fixed.}\)

Geometry:

With 200 basis functions you can reach a precision of 1e-7!
When is valuable?

Examples in many application domains

- Optimization/inversion/control problems
- Simulation based data bases
- Uncertainty quantification
- Sub-scale models in multi-scale modeling
- In-situ/deployed modeling
When is valuable?

Examples in many applications:

- Flow simulation
- Radar based
- Eddy current
- Quenching
- Model reduction
- Probes
We consider physical systems of the form

\[ \begin{align*}
\mathcal{L}(x, \mu) u(x, \mu) &= f(x, \mu) \\
\nu(x, \mu) &= g(x, \mu)
\end{align*} \]

where the solutions are implicitly parameterized by

\[ \mu \in D \subseteq \mathbb{R}^M \]
A second look

We consider physical systems of the form

\[ \mathcal{L}(x, \mu)u(x, \mu) = f(x, \mu) \quad x \in \Omega \]
\[ u(x, \mu) = g(x, \mu) \quad x \in \partial\Omega \]

where the solutions are implicitly parameterized by

\[ \mu \in \mathcal{D} \subset \mathbb{R}^M \]

- How do we find the basis.
- How do we ensure accuracy under parameter variation?
- What about speed?
The truth

Let us define:

The *exact solution*: Find \( u(\mu) \in X \) such that

\[
a(u, \mu, v) = f(\mu, v), \quad \forall v \in X
\]
The truth

Let us define:

The **exact solution**: Find $u(\mu) \in X$ such that

$$ a(u, \mu, v) = f(\mu, v), \; \forall v \in X $$

The **truth solution**: Find $u_h(\mu) \in X_h$ such that

$$ a_h(u_h, \mu, v_h) = f_h(\mu, v_h), \; \forall v_h \in X_h \quad \text{dim}(X_h) = \mathcal{N} $$
The truth

Let us define:

The **exact solution**: Find \( u(\mu) \in X \) such that
\[
a(u, \mu, v) = f(\mu, v), \quad \forall v \in X
\]

The **truth solution**: Find \( u_h(\mu) \in X_h \) such that
\[
a_h(u_h, \mu, v_h) = f_h(\mu, v_h), \quad \forall v_h \in X_h \quad \text{dim}(X_h) = N
\]

The **RB solution**: Find \( u_{RB}(\mu) \in X_N \) such that
\[
a_h(u_{RB}, \mu, v_N) = f_h(\mu, v_N), \quad \forall v_N \in X_N \quad \text{dim}(X_N) = N
\]
Let us define:

The **exact solution**: Find \( u(\mu) \in X \) such that

\[
a(u, \mu, v) = f(\mu, v), \quad \forall v \in X
\]

The **truth solution**: Find \( u_h(\mu) \in X_h \) such that

\[
a_h(u_h, \mu, v_h) = f_h(\mu, v_h), \quad \forall v_h \in X_h \quad \text{dim}(X_h) = N
\]

The **RB solution**: Find \( u_{RB}(\mu) \in X_N \) such that

\[
a_h(u_{RB}, \mu, v_N) = f_h(\mu, v_N), \quad \forall v_N \in X_N \quad \text{dim}(X_N) = N
\]

We always assume that \( N \gg N \)
The truth and errors

Solving for the truth is expensive - but we need to be able to trust the RB solution

$$\|u(\mu) - u_{RB}(\mu)\| \leq \|u(\mu) - u_h(\mu)\| + \|u_h(\mu) - u_{RB}(\mu)\|$$
The truth and errors

Solving for the truth is expensive - but we need to be able to trust the RB solution

\[ \|u(\mu) - u_{RB}(\mu)\| \leq \|u(\mu) - u_h(\mu)\| + \|u_h(\mu) - u_{RB}(\mu)\| \]

We assume that

\[ \|u(\mu) - u_h(\mu)\| \leq \varepsilon \]

This is your favorite solver and it is assumed it can be as accurate as you desire.
The truth and errors

Solving for the truth is expensive - but we need to be able to trust the RB solution

\[ \| u(\mu) - u_{RB}(\mu) \| \leq \| u(\mu) - u_h(\mu) \| + \| u_h(\mu) - u_{RB}(\mu) \| \]

We assume that

\[ \| u(\mu) - u_h(\mu) \| \leq \varepsilon \]

This is your favorite solver and it is assumed it can be as accurate as you desire.

Bounding we achieve two things

- Ability to build a basis at minimal cost
- Certify the quality of the model
Constructing the basis

We use the error estimator to construct the reduced basis in a **greedy** approach.

1. Define a (fine) training set in parameter space $\Pi_{\text{train}}$.
2. Choose a member randomly and solve truth.
3. Define $u_{RB} = u_h(\mu_1)$
   a. Find $\mu_{i+1} = \arg \sup_{\mu \in \Pi_{\text{train}}} \varepsilon_N(\mu)$
   b. Compute $u_h(\mu_{i+1})$
   c. Orthonormalize wrt $u_{RB}$
   d. Add new solution basis
4. Continue until $\sup_{\mu \in \Pi_{\text{train}}} \varepsilon_N \leq \varepsilon$

Resulting in

$$u_{RM}(\mu) = \sum_{i=1}^{N} u_{N}^i(\mu) \xi_i$$
Constructing the basis

We use the error estimator to construct the reduced basis in a greedy approach.

1. Define a (fine)training set in $\mathbb{P}$.
2. Choose a member randomly $\mu_1$.
3. Define $u_{RB} = u_h(\mu_1)$
   a. Find $\mu_{i+1} = \arg \sup_{\mu \in \Pi_{train}} u_h(\mu)$
   b. Compute $u_h(\mu_{i+1})$
   c. Orthonormalize wrt $u_R$.
   d. Add new solution basis.
4. Continue until $\sup_{\mu \in \Pi_{train}} \varepsilon_N \leq \varepsilon$.

Resulting in

$$u_{RM}(\mu) = \sum_{i=1}^{N} u_N^i(\mu) \xi_i$$
What about speed?

This relies on the affine assumption

\[ a(u, \mu, v) = \sum_{k=1}^{Q_A} \Theta_k(\mu) a_k(u, v) \]

\[ f(\mu, v) = \sum_{k=1}^{Q_f} \Theta^f_k(\mu) f_k(v) \]
What about speed?

This relies on the affine assumption

\begin{align*}
  a(u, \mu, v) &= \sum_{k=1}^{Q_a} \Theta_k(\mu) a_k(u, v) \\
  f(\mu, v) &= \sum_{k=1}^{Q_f} \Theta_k^f(\mu) f_k(v)
\end{align*}

This pushes majority of work off-line, e.g.

\[
  \sum_{i=1}^{N} \left[ \sum_{k=1}^{Q_a} \Theta_k(\mu) a_k(\xi_i, \xi_j) \right] u_N^j(\mu) = \sum_{k=1}^{Q_f} \Theta_k^f(\mu) f_k(\xi_j), \quad j \in [1, \ldots, N]
\]

All operations are now independent of \( N \) and depends solely on \( N \) and \( Q \)

If not on this form — approximate - EIM
Multiple scattering problem

Vertical position of middle cavity uniformly distributed within [-1,1]
Multiple scattering problem

$k = 3, \phi^i = 0, \theta^i = 0, 90$

$\phi^o = 0, \theta^o = 0 - 180$

Vertical position of middle cavity uniformly distributed within [-1, 1]
Larger scale

(a) Shuttle model with clamping locations.  (b) First mode.  (c) Fifth mode.
Larger scale

Figure 7: A local lateral shock is applied at initial time $t=0$s.
... and many more applications

- Elliptic problems (Machiels, Maday et al 2000)
- Parabolic problems (Grepl et al, 2005)
- Steady Navier-Stokes (Veroy et al, 2005)
- Stokes problems (Rozza et al, 2007)
- Multi-scale problems (Boyaval et al, 2008)
- Stochastic problems (Knezevic et al 2009)
- Electromagnetics (Chen et al, 2010)
- Boussinesq problems (Knezevic et al, 2011)
- RB static condensation (Patera et al, 2011)
- Integral equations (H et al, 2011)
- Fluid-structure problems (Rozza et al 2013)
- etc
... and many more applications

- Elliptic problems (Machiels, Maday et al 2000)
- Parabolic problems (Grepl et al, 2005)
- Steady Navier-Stokes (Veroy et al, 2005)
- Stokes problems (Rozza et al, 2007)
- Multi-scale problems (Boyaval et al, 2008)
- Stochastic problems (Knezevic et al 2009)
- Electromagnetics (Chen et al, 2010)
- Boussinesq problems (Knezevic et al, 2011)
- RB static condensation (Patera et al, 2011)
- Integral equations (H et al, 2011)
- Fluid-structure problems (Rozza et al 2013)
- etc

Generally, this works well for stationary and linear problems
If you want to know more about reduced basis methods

It's free!

Look at arXiv.org or infosciences.epfl.ch
If you want to know more about reduced basis methods

Its free!

Look at arXiv.org or infosciences.epfl.ch

Nonlinear problems are more challenging.

The standard approach is intrusive
Reduced order models revisited

Let us again consider the generic problem

\[ L(\mu)u(x, \mu) = f(x, \mu) \quad x \in \Omega(\mu) \]
Reduced order models revisited

Let us again consider the generic problem

\[ L(\mu)u(x, \mu) = f(x, \mu) \quad x \in \Omega(\mu) \]

A reduced model is easily constructed as

\[ u = V\alpha \quad \tilde{L}(\mu)\alpha(\mu) = V^T f(x, \mu) \quad \tilde{L}(\mu) = V^T L(\mu)V \]
Reduced order models revisited

Let us again consider the generic problem

$$L(\mu)u(x, \mu) = f(x, \mu) \quad x \in \Omega(\mu)$$

A reduced model is easily constructed as

$$u = V\alpha \quad \tilde{L}(\mu)\alpha(\mu) = V^T f(x, \mu) \quad \tilde{L}(\mu) = V^T L(\mu)V$$

However, for a more general/nonlinear case it becomes

$$\tilde{L}(\alpha, \mu) = V^T L(V\alpha, \mu)V$$

This is much harder to solve while maintaining the potential for acceleration

- EIM etc helps but at potentially high cost
Observe

We seek a linear approximation to a smooth solution manifold.

It is - basically - a regression problem.
Consider a neural network

\[ h_{i,j} = f \left( \sum_j w_{ij} h_{i-1,j} \right) \]

This attempts to use supervised learning to create a map between parameters and coefficients.
Neural Networks 101

The neurons are defined as

\[ y_j \in \mathbb{R} \text{ output of neuron } i \]
\[ \theta_j \in \mathbb{R} \text{ bias of neuron } j \]
\[ w_{i,j} \in \mathbb{R} \text{ synaptic weight between neurons } i \text{ and } j \]
\{s_1, \ldots, s_m\} sending neurons
\{r_1, \ldots, r_n\} receiving neurons

I. Accumulate weighted inputs from sending neurons:

\[ n_j = \sum_{l=1}^{m} w_{s_l,j} \cdot y_{s_l} \]

II. Thresholding: \( \bar{n}_j = n_j - \theta_j \)

III. Output given by activation function: \( y_j = f_{\text{act}}(\bar{n}_j) \)

IV. Output sent to all receiving neurons

It is nothing but a (very) complex nonlinear fitting
Reduced order models using NN

.. but why could one think why would that work?
Reduced order models using NN

.. but why could one think why would that work?

- An NN is - essentially - a highly nonlinear approximation - assuming a smooth variation of the coefficients with parameter variation it should be manageable.
Reduced order models using NN

.. but why could one think why would that work?

- An NN is - essentially - a highly nonlinear approximation - assuming a smooth variation of the coefficients with parameter variation it should be manageable.

- We can solve the forward problem as needed offline - needed to train the NN. A greedy approach is a suitable approach.
Reduced order models using NN

.. but why could one think why would that work?

- An NN is - essentially - a highly nonlinear approximation - assuming a smooth variation of the coefficients with parameter variation it should be manageable.

- We can solve the forward problem as needed offline - needed to train the NN. A greedy approach is a suitable approach.

- Once the weights are computed/learned, evaluation is fast - as needed in online phase, i.e., a natural offline-online split.
Predicting 3D from 2D

Let us first consider a data-base type query

- We have a (small) collection of 3D results
- We would like to recover solutions for other parameters
Predicting 3D from 2D

Let us first consider a database type query

- We have a (small) collection of 3D results
- We would like to recover solutions for other parameters

Approach

- Identify a descriptive 2D problem and meta parameters
- Train NN for 2D->3D connection on basis
- Solve 2D problem to recover 3D solution
Predicting 2D from 1D

Let us first consider a nozzle flow

$\text{Re}=500, \frac{pe}{po}=0.89$
Predicting 2D from 1D

Let us first consider a nozzle flow

Re=500, pe/po=0.89

One-dimensional model

(a) Viscous case (Re = 500, \( h_f = 0.16, p_e/p_0 = 0.89 \))

(b) Inviscid case (\( h_f = 0.16, p_e/p_0 = 0.89 \))
Predicting 2D from 1D

Parameters are Reynolds number, throat cross section and pressure ratio

<table>
<thead>
<tr>
<th></th>
<th>$Re$</th>
<th>$h_t$</th>
<th>$p_e/p_0$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Range</td>
<td>[500, 2000]</td>
<td>[0.16, 0.26]</td>
<td>[0.25, 0.89]</td>
</tr>
<tr>
<td>Training set</td>
<td>8, uniform</td>
<td>10, uniform</td>
<td>12, uniform</td>
</tr>
<tr>
<td>Test set</td>
<td>3, uniform</td>
<td>4, uniform</td>
<td>5, uniform</td>
</tr>
<tr>
<td>Validation set</td>
<td>3, random</td>
<td>4, random</td>
<td>5, random</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>1D viscous nozzle flow</th>
<th>1D inviscid nozzle flow</th>
<th>2D viscous nozzle flow</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\varepsilon_{POD}^2$</td>
<td>$10^{-5}$</td>
<td>$10^{-5}$</td>
<td>$10^{-5}$</td>
</tr>
<tr>
<td>$M_{mo}$</td>
<td>28</td>
<td>29</td>
<td>14</td>
</tr>
</tbody>
</table>
Predicting 2D from 1D
Predicting 2D from 1D

Input: \((\mathbf{A}_{\text{mod}}^p)\)

Input: \((\mathbf{A}_{\text{inv-\text{Re}}}^p)\)

Input: \((\text{Re}, h, p/p_0)\)

(a) \(Re = 1250, h_t = 0.124, p_e/p_0 = 0.826\)

(b) \(Re = 1750, h_t = 0.0863, p_e/p_0 = 0.314\)

(c) \(Re = 1750, h_t = 0.0988, p_e/p_0 = 0.698\)
Predicting 2D from 1D

Let us now consider a case which cannot be captured in 1D — a nozzle flow with a non-parallel inflow.

Training is in the same way - but we add the angle as a meta parameter in the training.

<table>
<thead>
<tr>
<th>Range</th>
<th>$Re$</th>
<th>$\alpha$</th>
<th>$\frac{p_e}{p_0}$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>[500, 2000]</td>
<td>[0°, 45°]</td>
<td>[0.25, 0.89]</td>
</tr>
<tr>
<td>Training set</td>
<td>8, uniform</td>
<td>10, uniform</td>
<td>12, uniform</td>
</tr>
<tr>
<td>Test set</td>
<td>3, uniform</td>
<td>4, uniform</td>
<td>5, uniform</td>
</tr>
<tr>
<td>Validation set</td>
<td>3, random</td>
<td>4, random</td>
<td>5, random</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$\alpha$</th>
<th>L^2 error</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.625°</td>
<td>0.17</td>
</tr>
<tr>
<td>16.875°</td>
<td>0.16</td>
</tr>
<tr>
<td>28.125°</td>
<td>0.15</td>
</tr>
<tr>
<td>39.375°</td>
<td>0.14</td>
</tr>
</tbody>
</table>
Predicting 2D from 1D

Mach number contours for the non-parallel nozzle flow ($Re = 1750, \alpha = 5.625^\circ, p_e/p_0 = 0.826$).
Predicting 2D from 1D

Mach number contours for the non-parallel nozzle flow (Re = 1750, α = 5.625°, $p_e/p_0 = 0.826$).

Mach number contours for the non-parallel nozzle flow (Re = 1750, α = 39.375°, $p_e/p_0 = 0.314$).
Predicting 3D from 2D

Let us now consider 2D->3D

<table>
<thead>
<tr>
<th>CFD</th>
<th>M6</th>
</tr>
</thead>
</table>

| NACA 0012    |             |

<table>
<thead>
<tr>
<th>Ma</th>
<th>( \alpha )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Range</td>
<td>[0.2, 0.85]</td>
</tr>
<tr>
<td>Training set</td>
<td>13, uniform</td>
</tr>
<tr>
<td>Test set</td>
<td>7, uniform</td>
</tr>
<tr>
<td>Validation set</td>
<td>7, random</td>
</tr>
</tbody>
</table>
Predicting 3D from 2D
Predicting 3D from 2D

Fig. 14  Comparison of the pressure on the wall at different sections (Section 1: 20%, Section 2: 50%, Section 3: 80%). The sections are specified with respect to the wing span, originating from the symmetry plane. $Ma = 0.804, \alpha = 2.84^\circ$. 
Predicting 3D from 2D

Hypersonic reentry vehicle - HYFLEX

MACH number range: 2 to 6
Predicting 3D from 2D

<table>
<thead>
<tr>
<th></th>
<th>Pressure</th>
<th>Mach</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2D</td>
<td>3D</td>
</tr>
<tr>
<td>$\varepsilon^2_{POD}$</td>
<td>$2.75 \times 10^{-5}$</td>
<td>$3.55 \times 10^{-5}$</td>
</tr>
<tr>
<td>$M_{mo}$</td>
<td>30</td>
<td>15</td>
</tr>
</tbody>
</table>
Predicting 3D from 2D

<table>
<thead>
<tr>
<th></th>
<th>Pressure</th>
<th>Mach</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2D</td>
<td>3D</td>
</tr>
<tr>
<td>$\varepsilon_{POD}^2$</td>
<td>$2.75 \times 10^{-5}$</td>
<td>$3.55 \times 10^{-5}$</td>
</tr>
<tr>
<td>$M_{mo}$</td>
<td>30</td>
<td>15</td>
</tr>
<tr>
<td></td>
<td>2D</td>
<td>3D</td>
</tr>
<tr>
<td></td>
<td>$3.90 \times 10^{-5}$</td>
<td>$1.56 \times 10^{-5}$</td>
</tr>
</tbody>
</table>

(a) $Ma = 2.09$

(b) $Ma = 3.91$

(c) $Ma = 5.91$
Predicting 3D from 2D

Mach: 0.354 1.412 2.469 3.527 4.585 5.642 6.700

Fig. 22  Comparison of the Mach number on the symmetry plane ($Ma = 3.91$).
Predicting 3D from 2D

Fig. 22  Comparison of the Mach number on the symmetry plane ($Ma = 3.91$).

Fig. 23  Comparison of the Mach number on the symmetry plane ($Ma = 5.91$).
Reduced order models

The neural net does a very good job at interpolating between data - a purely data driven approach

Let us consider using the neural net for regression directly on the reduced model — a **non-intrusive model**
Reduced order models

The neural net does a very good job at interpolating between data - a purely data driven approach

Let us consider using the neural net for regression directly on the reduced model — a non-intrusive model

- Build the reduced model with snapshots as usual
- Use snapshots to train NN as map
- In the online phase, evaluate NN to recover coefficients for RBM solution directly rather than solving reduced system
1D Poisson equation

Let’s start in 1D

\[
\begin{align*}
- u''(x; \mu) &= f(x; \mu) \quad \text{in } \Omega = (-1, 1), \\
  u(-1; \mu) &= u(1; \mu) = 0,
\end{align*}
\]

\[
f(x; \mu) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x - \mu)^2}{2\sigma^2}\right),
\]
1D Poisson equation

Let’s start in 1D

\[
\begin{aligned}
&-u''(x; \mu) = f(x; \mu) \quad \text{in } \Omega = (-1, 1), \\
u(-1; \mu) = u(1; \mu) = 0, \\
f(x; \mu) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x - \mu)^2}{2\sigma^2}\right),
\end{aligned}
\]
1D Poisson equation
1D Poisson equation

Let's make it a little more interesting

\[
\begin{aligned}
- \left( (1 + u(\mu)^2) u'(\mu) \right)' &= s(x; \mu) \quad \text{in } \Omega = [-\pi, \pi], \\
\left. u(\mu) \right|_{x=-\pi} &= \mu_2 \sin(2 - \mu_1 \pi), \\
\left. u(\mu) \right|_{x=\pi} &= \mu_2 \sin(2 + \mu_1 \pi),
\end{aligned}
\]
How confident can we be?

Looks as if it works - but what do we know?

Consider the error

$$\| u - u_{NN} \|_{L^2(\Omega)} \leq \| u - u_h \|_{L^2(\Omega)} + \| u_h - u_h^\nabla \|_{L^2(\Omega)} + \| u_h^\nabla - u_{NN} \|_{L^2(\Omega)}$$
How confident can we be?

Looks as if it works - but what do we know?

Consider the error

\[
\| u - u_{NN}^L \|_{L^2(\Omega)} \leq \| u - u_h \|_{L^2(\Omega)} + \| u_h - u_h^V \|_{L^2(\Omega)} + \| u_h^V - u_{NN}^L \|_{L^2(\Omega)}
\]

- **Finite element error** - \( \| u - u_h \|_{L^2(\Omega)} \leq \| u - u_h \|_V \leq \delta_{HF} \),

- **Reduced basis projection error** - \( \| u_h - u_h^V \|_{L^2(\Omega)} = \sum_{j=L+1}^{R} \sigma_j^2 \approx \beta e^{-\alpha L} \),

- **Error from neural network** - not so clear!
How confident can we be?

Looks as if it works - but what do we know?

Consider the error

\[ \| u - u_{NN}^L \|_{L^2(\Omega)} \leq \| u - u_h \|_{L^2(\Omega)} + \| u_h - u_N^V \|_{L^2(\Omega)} + \| u_N^V - u_{NN}^L \|_{L^2(\Omega)} \]

- **Finite element error** - \( \| u - u_h \|_{L^2(\Omega)} \leq \| u - u_h \|_V \leq \delta_{HF} \),

- **Reduced basis projection error** - \( \| u_h - u_N^V \|_{L^2(\Omega)} = \sum_{j=L+1}^{R} \sigma_j^2 \approx \beta e^{-\alpha L} \),

- **Error from neural network** - not so clear!

\[ \| u - u_{NN}^L \|_{L^2(\Omega)} \leq \delta_{HF} + \beta e^{-\alpha L} + \gamma \sqrt{L} \delta_{NN}, \]
How confident can we be?

... but we have the generic results (Cybenko ’88,’89)

- an NN with one hidden layer and differentiable activation function can approximate any continuous function

- an NN with at least 2 hidden layers and differentiable activation function can approximate any function
How confident can we be?

... but we have the generic results (Cybenko ’88,’89)

- an NN with one hidden layer and differentiable activation function can approximate any continuous function

- an NN with at least 2 hidden layers and differentiable activation function can approximate any function

This result is, however, not constructive, nor does it have an error estimate associated with it.
How confident can we be?

We shall use overfitting to guide the choice

Error estimated on small test set. This controls the NN error
How confident can we be?

We shall use overfitting to guide the choice

Error estimated on small test set. This controls the NN error
How confident can we be?

We shall use overfitting to guide the choice.

Error estimated on small test set. This controls the NN error.
How confident can we be?

We shall use overfitting to guide the choice.

Error estimated on small test set. This controls the NN error.
How confident can we be?

We shall use overfitting to guide the choice.

Error estimated on small test set. This controls the NN error.
How confident can we be?

We shall use overfitting to guide the choice.

Error estimated on small test set. This controls the NN error.
How confident can we be?

We shall use overfitting to guide the choice

Error estimated on small test set. This controls the NN error
How confident can we be?

We shall use overfitting to guide the choice.

Error estimated on small test set. This controls the NN error.
How confident can we be?

We shall use overfitting to guide the choice

Error estimated on small test set. This controls the NN error.
2D Poisson equation

Let’s consider 2D

\[
\begin{align*}
-\Delta u(x) &= f(x) \quad \text{in } \Omega(\mu), \\
u(\sigma) &= \sin(\sigma_x) \cos(\sigma_y) \quad \text{on } \partial\Omega(\mu),
\end{align*}
\]
2D Poisson equation

Reduced model error

NN model
2D Poisson equation

![Graph showing online run time and error comparison between POD-G and POD-NN models over samples. The graph displays the run time on a logarithmic scale against sample numbers. The error is represented by a color gradient. The graph includes markers for different datasets, with POD-G and POD-NN models highlighted.](image-url)
Alternatives

We basically have a non-intrusive reduced order model

Why not simply do interpolation?
Alternatives

We basically have a non-intrusive reduced order model

Why not simply do interpolation?

\[
\begin{cases}
-\nabla \cdot (\exp(\tilde{u}(\mu)) \ \nabla \tilde{u}(\mu)) = \tilde{s}(\tilde{x}, \tilde{y}) & \text{in } \tilde{\Omega}(\mu) \\
\tilde{u}(\mu) = \tilde{\sigma}_x \sin(\pi \tilde{\sigma}_x) \cos(\pi \tilde{\sigma}_y) & \text{on } \partial \tilde{\Omega}(\mu)
\end{cases}
\]
Alternatives

We basically have a non-intrusive reduced order model

Why not simply do interpolation?

\[
\begin{aligned}
-\nabla \cdot \left( \exp(\tilde{u}(\mu)) \, \nabla \tilde{u}(\mu) \right) &= \tilde{s}(\tilde{x}, \tilde{y}) \quad \text{in } \tilde{\Omega}(\mu) \\
\tilde{u}(\mu) &= \tilde{\sigma}_x \sin(\pi \tilde{\sigma}_x) \cos(\pi \tilde{\sigma}_y) \quad \text{on } \partial\tilde{\Omega}(\mu)
\end{aligned}
\]

It is too slow, does not scale and requires a regular grid
2D Navier-Stokes equation

Let us finally consider the 2D Navier Stokes equations

\[
\partial_t \mathbf{v} + (\mathbf{v} \cdot \nabla) \mathbf{v} = -\nabla p + \nu \Delta \mathbf{v}
\]

\[
\nabla \cdot \mathbf{v} = 0
\]

\[\mu_1 \quad \mu_2 \quad \mu_3 \quad \mu_4\]

\[\bar{\Gamma}_1 \quad \bar{\Gamma}_2 \quad \bar{\Gamma}_3 \quad \bar{\Gamma}_4\]

\[\bar{\nu}_x = 0, \bar{\nu}_y = 0\]

\[\bar{\nu}_x = 1, \bar{\nu}_y = 0\]

\[\text{Re}=400\]
2D Navier-Stokes equation

A qualitative comparison

\[ \mu = (1.12, 1.70, 1.08), \text{FEM} \]
\[ \mu = (1.90, 1.50, 1.60), \text{FEM} \]
\[ \mu = (1.78, 1.99, 2.29), \text{FEM} \]
\[ \mu = (1.12, 1.70, 1.08), \text{POD-NN} \]
\[ \mu = (1.90, 1.50, 1.60), \text{POD-NN} \]
\[ \mu = (1.78, 1.99, 2.29), \text{POD-NN} \]
A bit more quantitative

2D Navier-Stokes equation
2D Navier-Stokes equation

A bit more quantitative
2D Navier-Stokes equation

Offline cost ~
- POD: 1.2 hours
- NN: 3.1 hours

Online cost ~
- POD/NN: > 1000
Reduced order models for parameterized PDE’s

- For linear, stationary problems, all is well
- For nonlinear problems, much is open

Here we suggest that neural networks offers a natural component and yields great flexibility.
A brief summary

Reduced order models for parameterized PDE’s

- For linear, stationary problems, all is well
- For nonlinear problems, much is open

Here we suggest that neural networks offers a natural component and yields great flexibility.

Many open questions

- Problem dependent NN structure
- How to minimize cost of training
- How to certify results
Thank you