RICE ENGINEERING
Computational and
Applied Mathematics

Data-driven enhancements of numerical methods

Deep Ray
Email: deep.ray@rice.edu
Website: deepray.github.io

Colloquium Talk, Michigan Tech
March 2, 2020

Outline

- Problems of interest
- Neural networks: a brief overview
- Deep learning-based enhancements: simulation and theory
- Conclusion

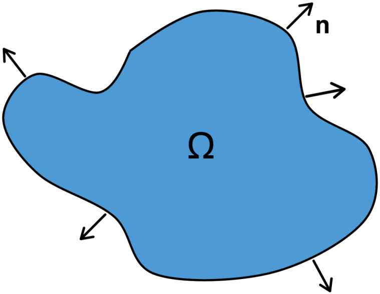
Problems of interest

1. Handling discontinuities

Conservation laws

Consider a quantity $\mathbf{u}$ in a domain Ω. Let $\mathbf{f}$ be the flux across $\partial\Omega$.

$$\frac{\mathrm{d}}{\mathrm{d}t}\int_{\Omega}\mathbf{u}\,\mathrm{d}x = -\int_{\partial\Omega}\mathbf{f}\cdot\mathbf{n}\,\mathrm{d}s$$

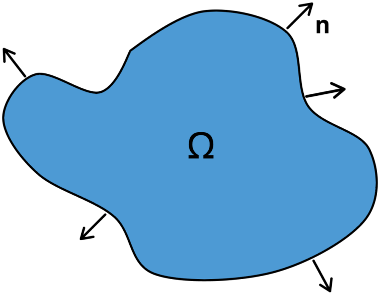


Conservation laws

Consider a quantity $\mathbf{u}$ in a domain Ω. Let $\mathbf{f}$ be the flux across $\partial\Omega$.

$$\frac{\partial \mathbf{u}}{\partial t} + \nabla \cdot \mathbf{f}(\mathbf{u}) = 0$$

(divergence theroem)



Conservation laws

Consider a quantity $\mathbf{u}$ in a domain Ω. Let $\mathbf{f}$ be the flux across $\partial\Omega$.

$$\frac{\partial \mathbf{u}}{\partial t} + \nabla \cdot \mathbf{f}(\mathbf{u}) = 0$$

Examples:

- Shallow water equations [atmospheric and oceanic modelling]
- Euler equations [aerospace]
- MHD equations [astrophysics]

Conservation laws

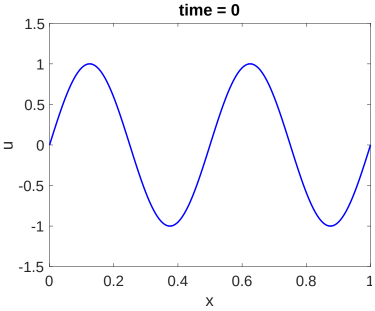
Consider a quantity $\mathbf{u}$ in a domain Ω. Let $\mathbf{f}$ be the flux across $\partial\Omega$.

$$\frac{\partial \mathbf{u}}{\partial t} + \nabla \cdot \mathbf{f}(\mathbf{u}) = 0$$

Discontinuities in finite time

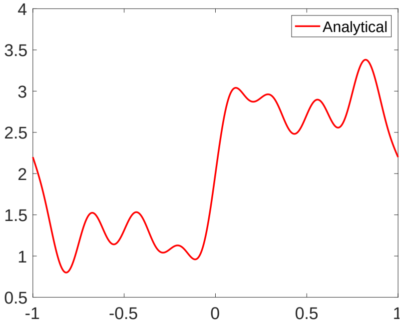
$$\frac{\partial u}{\partial t} + \frac{\partial}{\partial x}\left(\frac{u^2}{2}\right) = 0$$

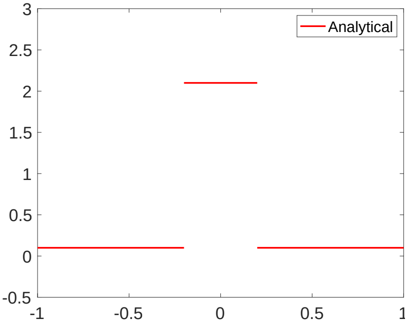
(Burgers' equation)



Spurious oscillations

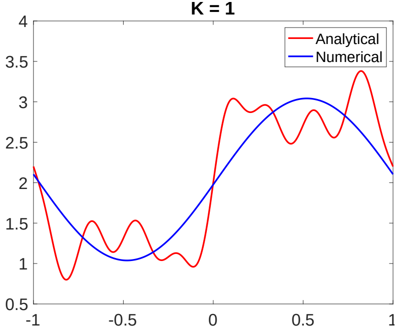
$$f(x) \approx \sum_{k=1}^{K} f_k \phi_k(x)$$

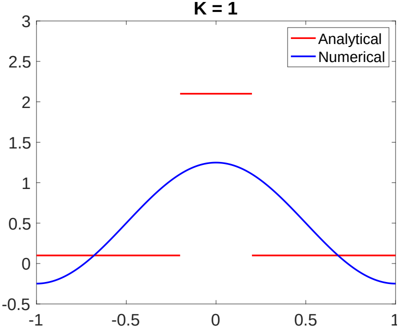




Spurious oscillations

$$f(x) \approx \sum_{k=1}^{K} f_k \phi_k(x)$$





Spurious oscillations

$$f(x) \approx \sum_{k=1}^{K} f_k \phi_k(x)$$

Challenge: Most detection/control methods have problem-dependent parameters.

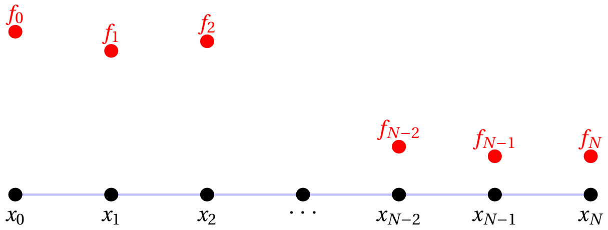
Problems of interest

2. Interpolating non-smooth functions

Interpolating functions

Let $f : [a, b] \mapsto \mathbb{R}$. Consider the uniform partition

$$a = x_0 < x_1 < \ldots < x_{N-1} < x_N = b, \quad h = \frac{b - a}{N}$$



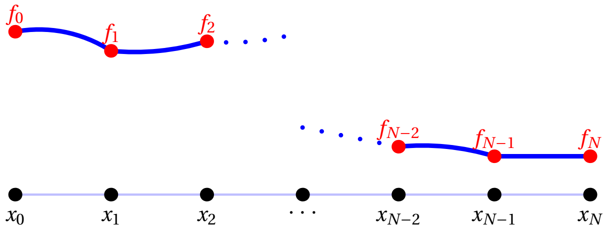
Interpolating functions

Let $f : [a, b] \mapsto \mathbb{R}$. Consider the uniform partition

$$a = x_0 < x_1 < ... < x_{N-1} < x_N = b, \quad h = \frac{b-a}{N}$$

Find an interpolation operator $\mathcal{I}_h$ such that

$$\mathcal{I}_h f(x_i) = f(x_i) \ \forall \ i = 0, ..., N \quad \text{and} \quad \|\mathcal{I}_h f - f\|_\infty = \mathcal{O}(h^p).$$

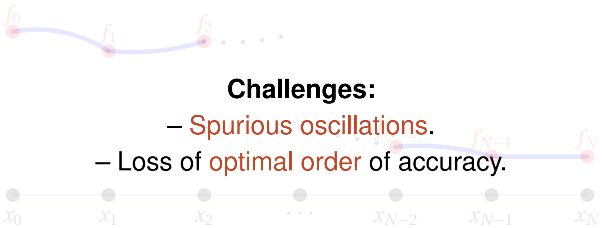


Let $f : [a, b] \mapsto \mathbb{R}$. Consider the uniform partition

$$a = x_0 < x_1 < ... < x_{N-1} < x_N = b, \quad h = \frac{b-a}{N}$$

Find an interpolation operator $\mathcal{I}_h$ such that

$$\mathcal{I}_h f(x_i) = f(x_i) \ \forall \ i = 0, ..., N \quad \text{and} \quad \|\mathcal{I}_h f - f\|_\infty = \mathcal{O}(h^p).$$



Challenges:

– Spurious oscillations.

– Loss of optimal order of accuracy.

Problems of interest

3. Uncertainty quantification

Uncertainty in realistic problems

Solution $\mathbf{u}$ $\longleftarrow$ Model $\mathcal{M}$ (ODE, PDE, ...)

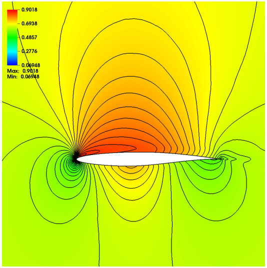
Evaluate observable $\mathcal{L}(\mathbf{u})$ (lift, drag, wave height, ...)

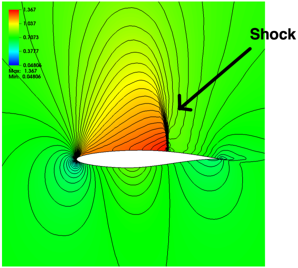
Uncertainty in realistic problems

Solution $\mathbf{u}$ $\longleftarrow$ Model $\mathcal{M}$ (ODE, PDE, ...)

Evaluate observable $\mathcal{L}(\mathbf{u})$ (lift, drag, wave height, ...)

- Uncertainty in measuring input (initial and boundary conditions, ...).



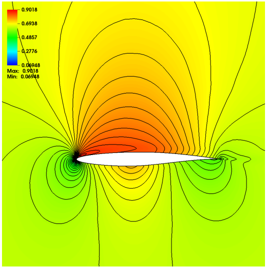


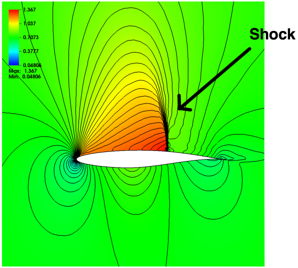
Uncertainty in realistic problems

Solution $\mathbf{u}$ $\longleftarrow$ Model $\mathcal{M}$ (ODE, PDE, ...)

Evaluate observable $\mathcal{L}(\mathbf{u})$ (lift, drag, wave height, ...)

- Uncertainty in measuring input (initial and boundary conditions, ...).





- Uncertainty Quantification (UQ):

  Given input uncertainties $\longrightarrow$ measure uncertainty in u or $\mathcal{L}$.

Uncertainty in realistic problems

$$\text{Solution } \mathbf{u}(z) \quad \longleftarrow \quad \text{Model } \mathcal{M}(z) \quad \text{(ODE, PDE, ...)}$$

$$\text{Evaluate observable } \mathcal{L}(z, \mathbf{u}(z)) \quad \text{(lift, drag, wave height, ...)}$$

- Parametrize uncertainty by $z \in \mathcal{Z}$.

Uncertainty in realistic problems

Solution $\mathbf{u}(z)$ $\longleftarrow$ Model $\mathcal{M}(z)$ (ODE, PDE, ...)

Evaluate observable $\mathcal{L}(z, \mathbf{u}(z))$ (lift, drag, wave height, ...)

- Parametrize uncertainty by $z \in \mathcal{Z}$.
- Assume z is sampled using $\mu \in \text{Prob}(\mathcal{Z})$.
- Compute statistics of $\mathcal{L}$.

Uncertainty in realistic problems

$$\text{Solution } \mathbf{u}(z) \quad \longleftarrow \quad \text{Model } \mathcal{M}(z) \quad \text{(ODE, PDE, ...)}$$

$$\text{Evaluate observable } \mathcal{L}(z, \mathbf{u}(z)) \quad \text{(lift, drag, wave height, ...)}$$

- Parametrize uncertainty by $z \in \mathcal{Z}$.
- Assume z is sampled using $\mu \in \text{Prob}(\mathcal{Z})$.
- Compute statistics of $\mathcal{L}$.

How ?

Monte Carlo

Algorithm:

- Draw M samples $\{z_i\}_{1 \leq i \leq M}$.
- Solve for $\mathbf{u}(z_i)$ and evaluate $\mathcal{L}(z_i)$.
- Approximate moments of $\mathcal{L}$:

$$\int_{\mathcal{Z}} (\mathcal{L}(z))^p \mathrm{d}\mu(z) \approx \frac{1}{M} \sum_{i=1}^{M} (\mathcal{L}(z_i))^p, \quad p = 1, 2, ...$$

Monte Carlo

Algorithm:

- Draw M samples $\{z_i\}_{1 \leq i \leq M}$.
- Solve for $\mathbf{u}(z_i)$ and evaluate $\mathcal{L}(z_i)$.
- Approximate moments of $\mathcal{L}$:

$$\int_{\mathcal{Z}} (\mathcal{L}(z))^p \mathrm{d}\mu(z) \approx \frac{1}{M} \sum_{i=1}^{M} (\mathcal{L}(z_i))^p, \quad p = 1, 2, \dots$$

Classical Monte Carlo:

- Choose M i.i.d samples.
- Error $= \mathcal{O}(M)^{-\frac{1}{2}}$.

Quasi-Monte Carlo:

- Choose from low discrepancy sequence (Sobol, Halton, etc).
- Error $= \mathcal{O}\left(\frac{(\log(M))^d}{M}\right)$.

Monte Carlo

Algorithm:

- Draw M samples $\{z_i\}_{1 \le i \le M}$.
- Solve for $\mathbf{u}(z_i)$ and evaluate $\mathcal{L}(z_i)$.
- Approximate moments of $\mathcal{L}$:

$$\int_{\mathcal{Z}} (\mathcal{L}(z))^p \mathrm{d}\mu(z) \;\approx\; \frac{1}{M}\sum_{i=1}^{M} (\mathcal{L}(z_i))^p, \quad p = 1, 2, \ldots$$

Classical Monte Carlo:

- Choose M i.i.d samples.
- Error $= \mathcal{O}(M)^{-\frac{1}{2}}$.

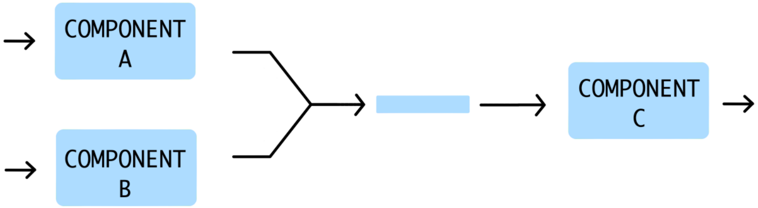
Quasi-Monte Carlo:

- Choose from low discrepancy sequence (Sobol, Halton, etc).
- Error $= \mathcal{O}\left(\frac{(\log(M))^d}{M}\right)$.

Challenge: For realistic problems, it is expensive to generate samples.

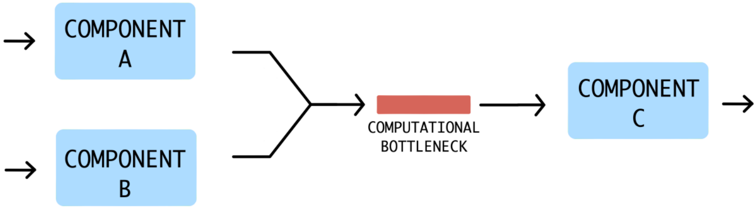
Common objective

- Handling discontinuities.
- Interpolating non-smooth functions.
- Uncertainty quantification.



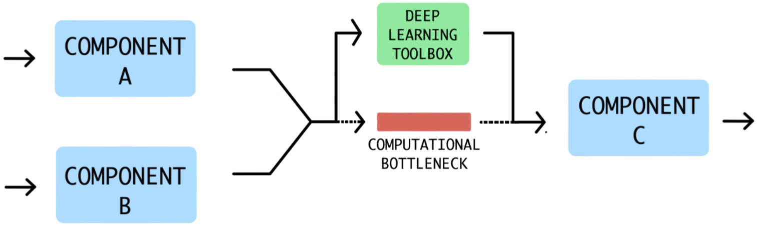
Common objective

- Handling discontinuities.
- Interpolating non-smooth functions.
- Uncertainty quantification.



Common objective

- Handling discontinuities.
- Interpolating non-smooth functions.
- Uncertainty quantification.



Neural networks

A brief overview

Neural networks

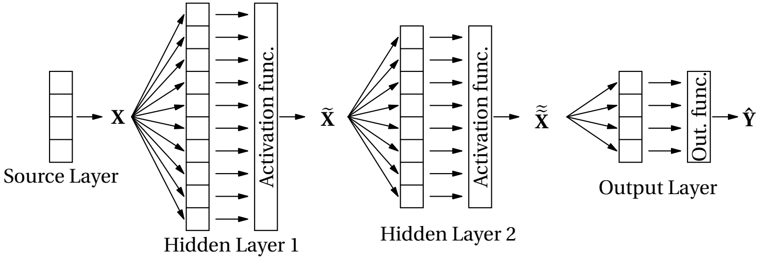
Aim: Approximate a function

$$\mathbf{F}: \mathbf{X} \mapsto \mathbf{Y}, \quad \mathbf{X} \in \mathbb{R}^n,\ \mathbf{Y} \in \mathbb{R}^m \qquad \text{given} \qquad \mathbb{T} = \{(\mathbf{X}_i, \mathbf{Y}_i)\}_i.$$

Aim: Approximate a function

$$\mathbf{F} : \mathbf{X} \mapsto \mathbf{Y}, \quad \mathbf{X} \in \mathbb{R}^n, \ \mathbf{Y} \in \mathbb{R}^m \qquad \text{given} \qquad \mathbb{T} = \{(\mathbf{X}_i, \mathbf{Y}_i)\}_i.$$

- Train a multilayer perceptron (MLP):

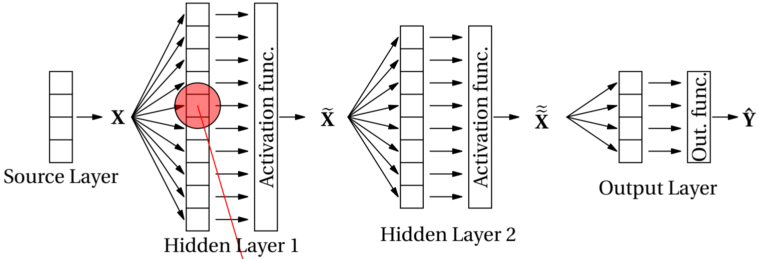


Neural networks

Aim: Approximate a function

$$\mathbf{F} : \mathbf{X} \mapsto \mathbf{Y}, \quad \mathbf{X} \in \mathbb{R}^n, \ \mathbf{Y} \in \mathbb{R}^m \qquad \text{given} \qquad \mathbb{T} = \{(\mathbf{X}_i, \mathbf{Y}_i)\}_i.$$

- Train a multilayer perceptron (MLP):

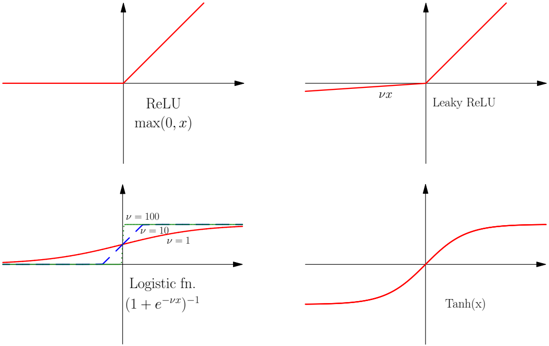


$$z = \mathbf{w}\mathbf{X} + b \longrightarrow \mathscr{A}(z)$$

Aim: Approximate a function

$$\mathbf{F} : \mathbf{X} \mapsto \mathbf{Y}, \quad \mathbf{X} \in \mathbb{R}^n,\ \mathbf{Y} \in \mathbb{R}^m \qquad \text{given} \qquad \mathbb{T} = \{(\mathbf{X}_i, \mathbf{Y}_i)\}_i.$$

- Train a multilayer perceptron (MLP):



Neural networks

Aim: Approximate a function

$$\mathbf{F} : \mathbf{X} \mapsto \mathbf{Y}, \quad \mathbf{X} \in \mathbb{R}^n, \; \mathbf{Y} \in \mathbb{R}^m \qquad \text{given} \qquad \mathbb{T} = \{(\mathbf{X}_i, \mathbf{Y}_i)\}_i.$$

- Train a multilayer perceptron (MLP):

$$\hat{\mathbf{Y}} = \mathcal{O} \circ H^L \circ \mathcal{A} \circ H^{L-1} \circ ... \circ \mathcal{A} \circ H^1(\mathbf{X}), \quad H^l(\tilde{\mathbf{X}}) = \mathbf{W}^l \tilde{\mathbf{X}} + \mathbf{b}^l.$$

- Loss function:

$$J(\theta) := \sum_{\mathbb{T}} |\mathbf{Y}_i - \hat{\mathbf{Y}}_i|^p, \quad \theta = \{\mathbf{W}^l, \mathbf{b}^l\}_{1 \le l \le L}.$$

Neural networks

Types of networks:

- Classification
- Regression

Hyperparameters:

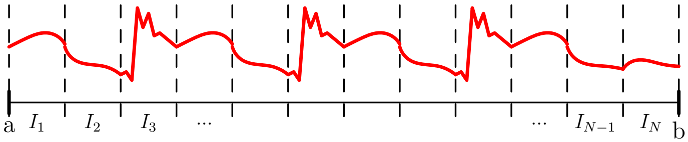
- Network size – depth and width
- Activation function
- Loss function
- Regularization technique – to avoid overfitting
- Training and validation datasets
- Optimizer: Stochastic gradient descent, AdaGrad, ADAM, etc.

Deep learning-based enhancements

1. Handling discontinuities

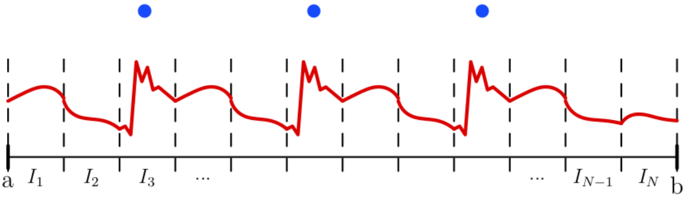
Troubled-cell detection

Piecewise polynomial solution on *N* cells.



Piecewise polynomial solution on N cells.

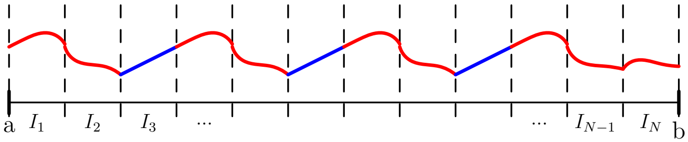
1. Find troubled-cells $\longrightarrow$ cells with discontinuities.



Troubled-cell detection

Piecewise polynomial solution on N cells.

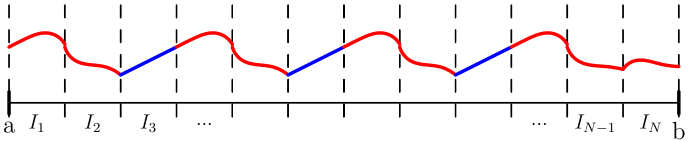
1. Find troubled-cells $\longrightarrow$ cells with discontinuities.
2. Limit solution in flagged cells.



Troubled-cell detection

Piecewise polynomial solution on *N* cells.

1. Find troubled-cells $\longrightarrow$ cells with discontinuities.
2. Limit solution in flagged cells.

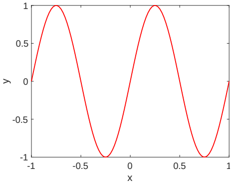


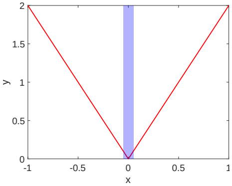
Issues:

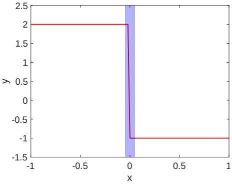
- Existing detectors have problem-dependent parameters.
- If insufficient cells flagged $\longrightarrow$ re-appearance of oscillations.
- If excessive cells flagged
  - Unnecessary computational cost.
  - Loss of accuracy in smooth regions.

An MLP-based detector [R and Hesthaven, 2018]

- Input: local solution in the cell.
- 5 hidden layers with leaky ReLU activation.
- Output $\hat{\mathbf{Y}} \in [0, 1]$. Troubled-cell if $\hat{\mathbf{Y}} > 0.5$.
- Cost functional: L2 regularized cross-entropy.
- Training set generated using:



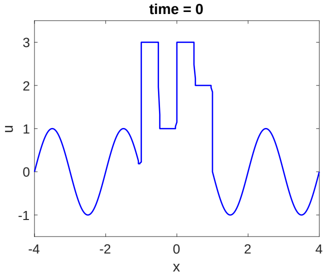


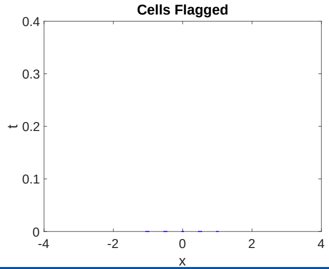


MLP trained only once (offline) and used for all systems.

Burgers equation: $u_t + (u^2/2)_x = 0$

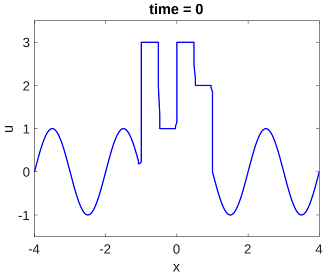
$N = 200$
$T_f = 0.4$

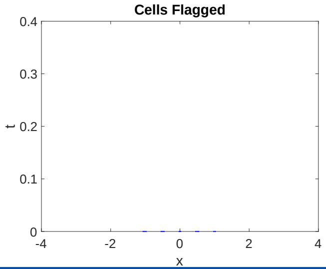




Burgers equation: $u_t + (u^2/2)_x = 0$

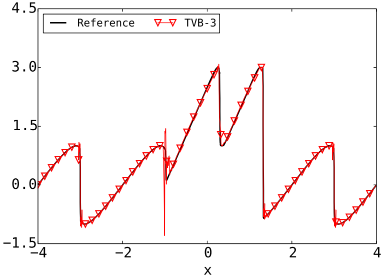
$N = 200$
$T_f = 0.4$

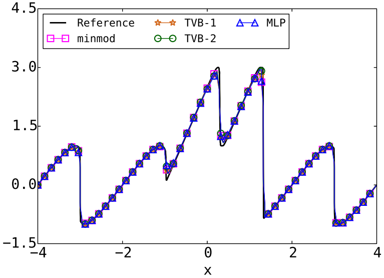




Burgers equation: $u_t + (u^2/2)_x = 0$

$$\text{TVB-1} \longrightarrow M = 10, \qquad \text{TVB-2} \longrightarrow M = 100, \qquad \text{TVB-3} \longrightarrow M = 1000$$





Euler equations (2D)

$$\frac{\partial \mathbf{u}}{\partial t} + \frac{\partial \mathbf{f}_1}{\partial x} + \frac{\partial \mathbf{f}_2}{\partial y} = 0$$

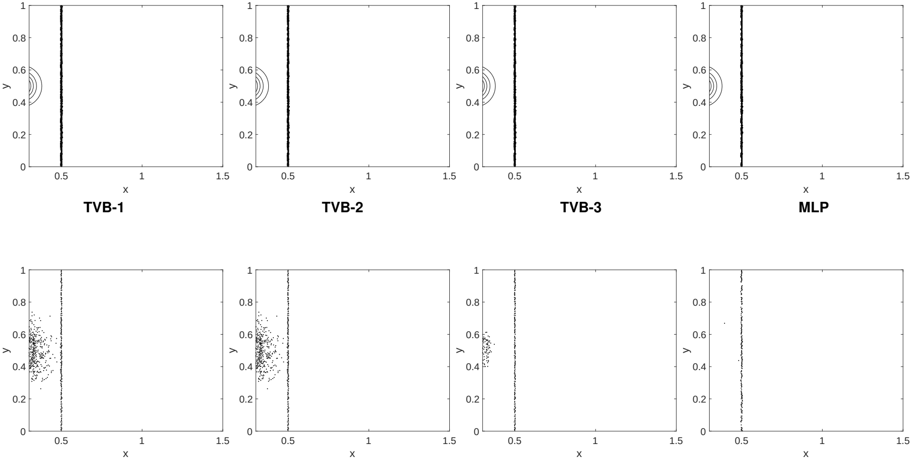
Vector of conserved variables $\mathbf{u} = \begin{bmatrix} \rho & \rho v_1 & \rho v_2 & E \end{bmatrix}^\top$

$$E = \rho \left(\frac{v_1^2 + v_2^2}{2} + e \right), \qquad e = \frac{p}{(\gamma - 1)\rho}, \qquad \gamma = 1.4$$

$$\begin{aligned} \rho &\longrightarrow \text{fluid density} \\ (v_1, v_2) &\longrightarrow \text{velocity} \\ p &\longrightarrow \text{pressure} \\ E &\longrightarrow \text{total energy} \\ e &\longrightarrow \text{internal energy} \end{aligned}$$

Shock-vortex [R and Hesthaven, 2019]

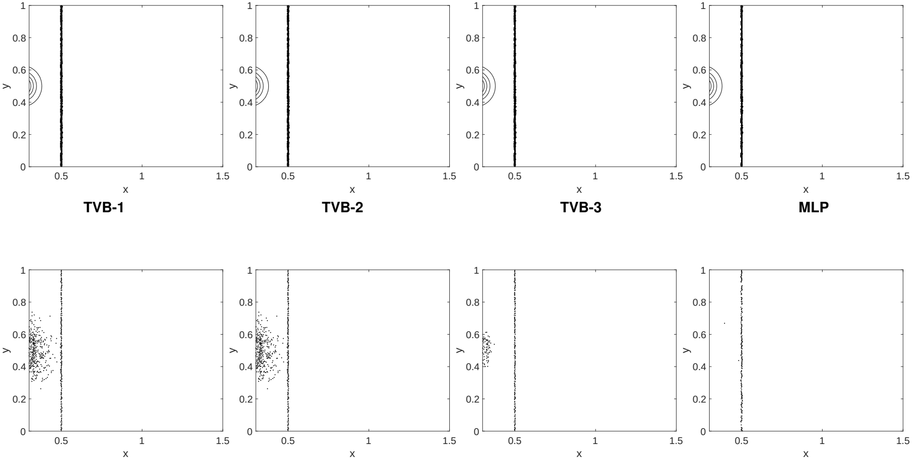
Solution



Flagged cells

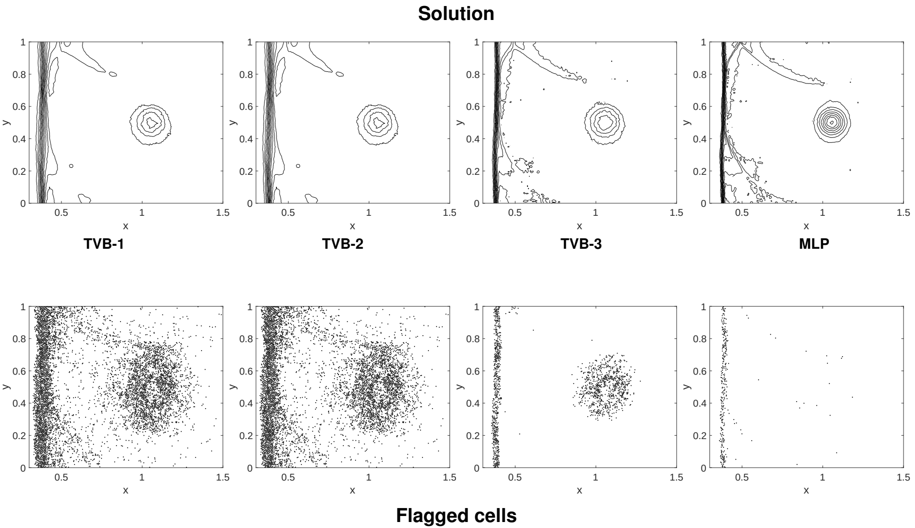
Shock-vortex [R and Hesthaven, 2019]

Solution



Flagged cells

Shock-vortex [R and Hesthaven, 2019]



Deep learning-based enhancements

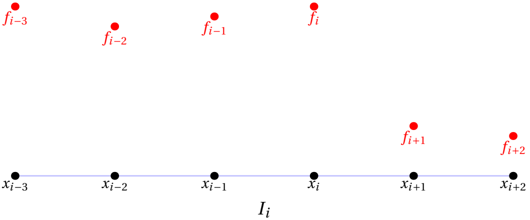
2. Interpolating non-smooth functions

Choosing an interpolation polynomial

Using point values of f, find an interpolation in $I_i = [x_{i-1}, x_i]$ such that

- f_{i-1} and f_i are used.
- The stencil is compact.

Example: cubic interpolation has three candidates:

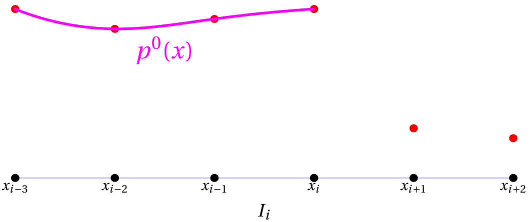


Choosing an interpolation polynomial

Using point values of f, find an interpolation in $I_i = [x_{i-1}, x_i]$ such that

- f_{i-1} and f_i are used.
- The stencil is compact.

Example: cubic interpolation has three candidates:

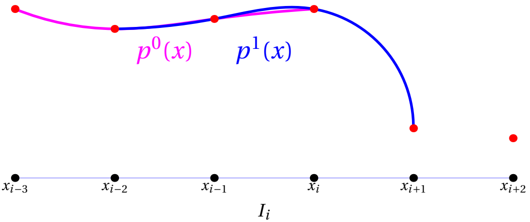


Choosing an interpolation polynomial

Using point values of f, find an interpolation in $I_i = [x_{i-1}, x_i]$ such that

- f_{i-1} and f_i are used.
- The stencil is compact.

Example: cubic interpolation has three candidates:

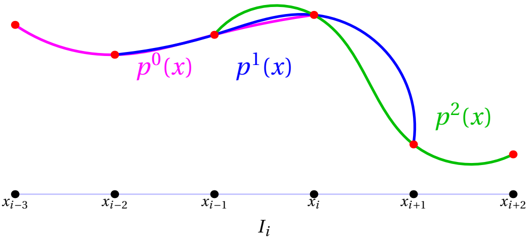


Choosing an interpolation polynomial

Using point values of f, find an interpolation in $I_i = [x_{i-1}, x_i]$ such that

- f_{i-1} and f_i are used.
- The stencil is compact.

Example: cubic interpolation has three candidates:

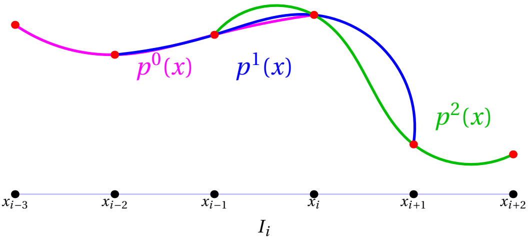


Choosing an interpolation polynomial

Using point values of f, find an interpolation in $I_i = [x_{i-1}, x_i]$ such that

- f_{i-1} and f_i are used.
- The stencil is compact.

Example: cubic interpolation has three candidates:



Which polynomial to choose?

Interpolation of non-smooth functions

Essentially non-oscillatory (ENO) method [Harten et al., 1987]:

- Adaptively chooses stencil of size p.
- Suppresses spurious oscillations.

Interpolation of non-smooth functions

Essentially non-oscillatory (ENO) method [Harten et al., 1987]:

- Adaptively chooses stencil of size p.
- Suppresses spurious oscillations.

Multilayer perceptrons:

- Universal approximators [Cybenko, Funahashi, Hornik, Barron, Pinkus, etc].
- Approximation with deep ReLU network [Yarotsky, 2018].

ENO and MLP

Essentially non-oscillatory (ENO) method [Harten et al., 1987]:

- Adaptively chooses stencil of size k.
- Suppresses spurious oscillations.

$\Updownarrow$?

MLPs:

- Universal approximators [Cybenko, Funahashi, Hornik, Barron, Pinkus, etc].
- Approximation with deep ReLU network [Yarotsky, 2018].

Theorem 1 (De Ryck, Mishra and R, 2019)

The ENO interpolator of order k can be cast as a ReLU network with

$$k + \left\lceil \log_2 \binom{k-1}{\lfloor \frac{k-2}{2} \rfloor} \right\rceil$$

hidden layers that takes input $\mathbf{X} = (f_{i+j})_{j=-k+1}^{k-2} \in \mathbb{R}^{2k-2}$.

- The network architecture is explicit and independent of f.
- The network is not unique.
- Smaller networks have been constructed for $k = 3, 4$.

Theorem 2 (De Ryck, Mishra and R, 2019)

A modified linear ENO interpolator can be cast as a ReLU neural network consisting of 5 hidden layers.
Let f be globally continuous, with $|f''|$ bounded in $\in \mathbb{R} \setminus \{z\}$ and a discontinuity in f' at z. Then

$$\|f - \mathcal{I}_h f\|_\infty \leq Ch^2 \sup_{\mathbb{R}\setminus\{z\}} |f''|,$$

for all $h > 0$, with $C > 0$ independent of f.

- Standard interpolation methods give at most first-order accuracy!

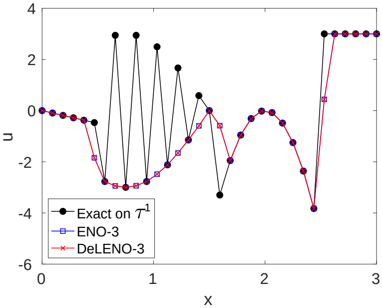
Training data:

- Components of **X** chosen from $\mathcal{U}(-1, 1)$.
- Samples from $\sin(n\pi x)$.

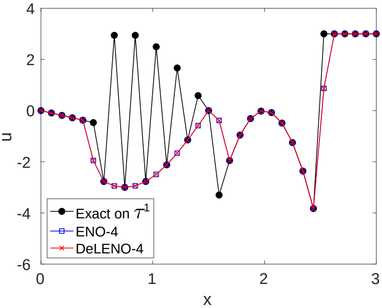
k	Hidden layer width	$\mathbb{T}_{acc}$	$\mathbb{V}_{acc}$
3	4	99.36/%	99.32/%
4	10,6,4	99.22/%	99.14/%

Trained networks called deep learning ENO (DeLENO).

$$f(x) = \begin{cases} -x & \text{if } x < 0.5, \\ 3\sin(10\pi x) & \text{if } 0.5 < x < 1.5, \\ 20(x-2)^2 & \text{if } 1.5 < x < 2.5, \\ 3 & \text{if } 2.5 < x. \end{cases}$$



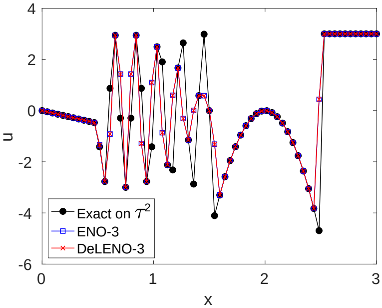
N=16, k=3



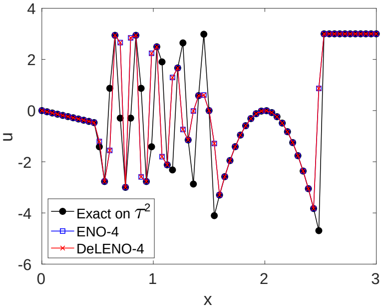
N=16, k=4

1D interpolation

$$f(x) = \begin{cases} -x & \text{if } x < 0.5, \\ 3\sin(10\pi x) & \text{if } 0.5 < x < 1.5, \\ 20(x-2)^2 & \text{if } 1.5 < x < 2.5, \\ 3 & \text{if } 2.5 < x. \end{cases}$$

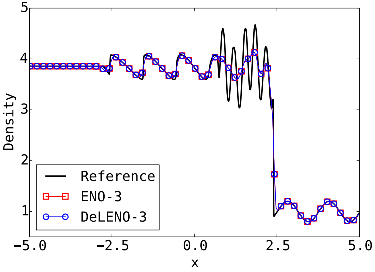


N=32, k=3

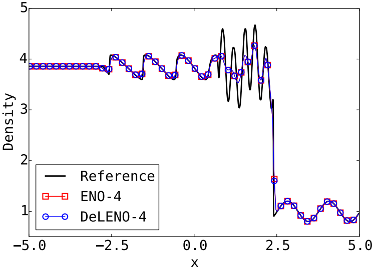


N=32, k=4

1D Euler equations: Shock-Entropy problem, N=200



k=3



k=4

Image compression (705×929)

Original

ENO-4

DeLENO-4

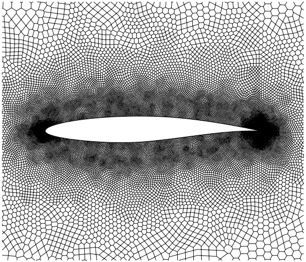
Scheme	Rel. L^1	Rel. L^2	Rel. L^∞	c_r
ENO-4	5.422e-2	8.485e-2	5.581e-1	0.996
DeLENO-4	5.422e-2	8.492e-2	5.581e-1	0.996

Deep learning-based enhancements

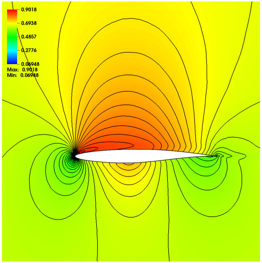
3. Uncertainty quantification

High cost of sample generation

- Need to measure the uncertainty in observable $\mathcal{L}$.
- 2D Euler simulation past RAE airfoil: single run costs 7 hours (wall clock) on a cluster!
- Cost is much higher in 3D.



Mesh



Flow (Mach Number)

High cost of sample generation

Goal: approximate the push-forward measure $\hat{\mu} \in \mathrm{Prob}(\mathbb{R})$

$$\int_{\mathbb{R}} \phi(z) \mathrm{d}\hat{\mu}(z) = \int_{\mathcal{Z}} \phi(\mathcal{L}(z)) \mathrm{d}\mu(z).$$

Two steps procedure:

1. Replace $\mathcal{L}$ with a network surrogate $\mathcal{L}^*$.
2. Approximate $\hat{\mu}$ using Monte Carlo.

QMC algorithm

- Select: J_{qmc} sample points $\{z_i\}_{i=1}^{J_{qmc}} \subset \mathcal{Z}$ (Sobol, Halton, ...).
- For each z_i:
  - Compute: solution $\mathbf{u}(z_i)$ (numerically).
  - Evaluate: $\mathcal{L}(z_i) = \mathcal{L}(\mathbf{u}(z_i))$.
- Approximate: statistics of $\mathcal{L}$ using the push forward measure

$$\hat{\mu}_{qmc} := \frac{1}{J_{qmc}} \sum_{i=1}^{J_{qmc}} \delta_{\mathcal{L}(z_i)}$$

$$\implies \quad \int_{\mathbb{R}} \phi(z) \mathrm{d}\hat{\mu}_{qmc}(z) = \frac{1}{J_{qmc}} \sum_{i=1}^{J_{qmc}} \phi(\mathcal{L}(z_i)).$$

DL-QMC algorithm

- Select: J_{dl} sample points $\{z_i\}_{i=1}^{J_{dl}} \subset \mathcal{Z}$ (Sobol, Halton, ...).
- Choose: the training set $\mathbb{T} = \{z_i\}_{i=1}^{N}$ for $N \ll J_{dl}$.
- For each $z_i \in \mathbb{T}$:
  - Compute: solution $\mathbf{u}(z_i)$ (numerically).
  - Evaluate: $\mathcal{L}(z_i) = \mathcal{L}(\mathbf{u}(z_i))$.
- Train: the neural network $\mathcal{L}^*$.
- Approximate: statistics of $\mathcal{L}^*$ using the push forward measure

$$\hat{\mu}^* := \frac{1}{J_{dl}} \left(\sum_{i=1}^{N} \delta_{\mathcal{L}^*(z_i)} + \sum_{i=N+1}^{J_{dl}} \delta_{\mathcal{L}^*(z_i)} \right).$$

Analysis of DL-QMC

- Define generalization error of the network

$$\varepsilon_G := \int_{\mathcal{Z}} |\mathcal{L}(z) - \mathcal{L}^*(z)| \mathrm{d}\mu(z).$$

- Wasserstein metric to estimate error $W_1(\hat{\mu}, \hat{\mu}^*)$.
- Speedup ($\mathcal{S}$) over QMC: For $\epsilon > 0$ if $W_1(\hat{\mu}, \hat{\mu}_{qmc}) = \mathcal{O}(\epsilon)$ and $W_1(\hat{\mu}, \hat{\mu}^*) = \mathcal{O}(\epsilon)$

$$\mathcal{S} = \frac{\text{Cost of baseline QMC method}}{\text{Cost of DL-QMC method}}.$$

Analysis of DL-QMC

Theorem 3 (Lye, Mishra and R, 2019)

For $\epsilon > 0$, if $\varepsilon_G = \mathcal{O}(\epsilon)$ then $W_1(\hat{\mu}, \hat{\mu}^*) = \mathcal{O}(\epsilon)$. Furthermore, if J_{qmc} samples are needed for $W_1(\hat{\mu}, \hat{\mu}_{qmc}) = \mathcal{O}(\epsilon)$, then

$$\frac{1}{\mathcal{S}} = \mathcal{O}\left(\frac{N}{J_{qmc}} + \frac{\mathcal{C}_*}{\mathcal{C}}\right),$$

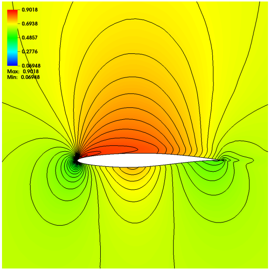
where $\mathcal{C}$ and $\mathcal{C}_*$ are the costs for computing $\mathcal{L}^h(z)$ and $\mathcal{L}^*(z)$ respectively.

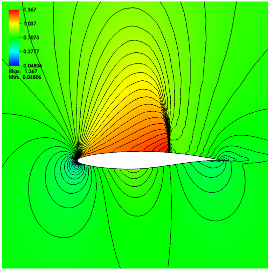
Speedup over the baseline algorithm if:

- Low generalization error: $\varepsilon_G = \mathcal{O}(\epsilon)$.
- Low cost of generating training data: $N \ll J_{qmc}$.
- $\mathcal{C}_* \ll \mathcal{C}$.

Numerical example: RAE2822 airfoil

- Observables: Lift and Drag.
- $\mathcal{Z} = [0, 1]^6$: 2 shape variables + 4 operational parameters.
- High-resolution finite volume solver.
- 128 consecutive Sobol points (for training).
- 1001 Sobol points (for testing).
- Systematic ensemble training over hyperparameters.

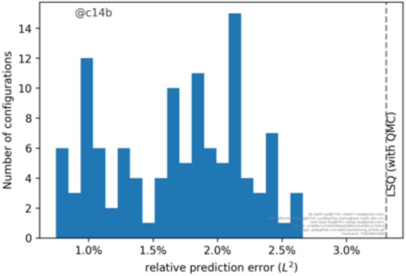




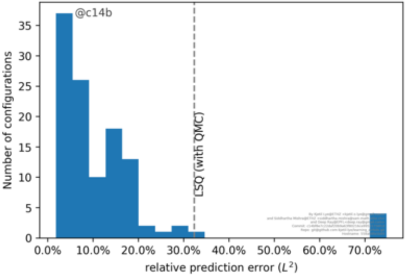
Numerical example: RAE2822 airfoil

Network architecture:

Layer	Width (Number of Neurons)	Number of parameters
Hidden Layer 1	12	84
Hidden Layer 2	12	156
Hidden Layer 3	10	130
Hidden Layer 4	12	132
Hidden Layer 5	10	130
Hidden Layer 6	12	156
Hidden Layer 7	10	130
Hidden Layer 8	10	110
Hidden Layer 9	12	132
Output Layer	1	13
		1149



Lift

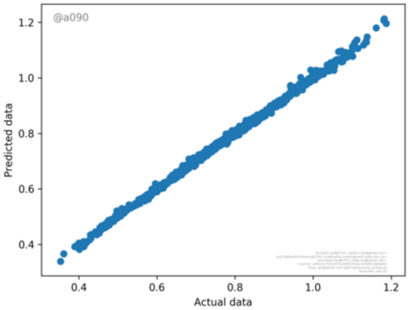


Drag

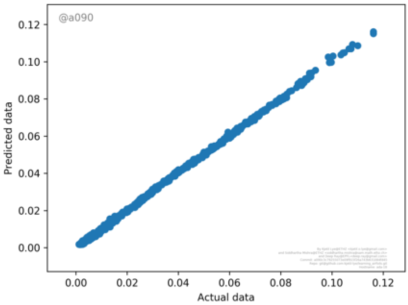
Best performing networks:

$\mathcal{L}$	Optimizer	Loss	L^2 reg.	% Error mean (std)
Lift	ADAM	MSE	7.8e-6	0.786 (0.010)
Drag	ADAM	MAE	7.8e-6	1.847 (0.022)

Predictions

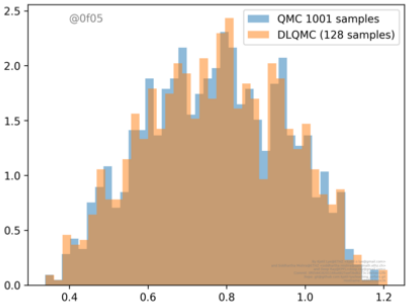


Lift

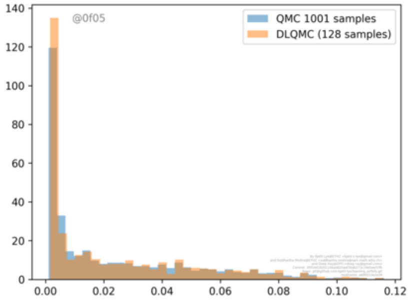


Drag

Predictions



Lift



Drag

Computational times and speedup

	time (sec)
Sample generation	24000
Training (Lift)	700
Evaluation ($\mathcal{L}^*_{lift}$)	9e-6
Training (Drag)	840
Evaluation ($\mathcal{L}^*_{drag}$)	1e-5

Observable	Speedup
Lift	6.64
Drag	8.56

Conclusion

Conclusion

- Demonstrated the bridging of traditional methods and deep learning.
- Neural networks for classification and regression.
- Train once offline for global use (detectors and ENO).
- Theoretically prove why it works and measure improvement.

Conclusion

Extensions:

- Predicting artificial viscosity using deep learning [Discacciati, Hesthaven and R, 2019].
- Constraint-aware neural networks for Riemann solvers [Magiera, R, Hesthaven and Rhode, 2019].
- Reduced order modelling with deep learning [Wang, Hesthaven and R, 2019].

Conclusion

Extensions:

- Predicting artificial viscosity using deep learning [Discacciati, Hesthaven and R, 2019].
- Constraint-aware neural networks for Riemann solvers [Magiera, R, Hesthaven and Rhode, 2019].
- Reduced order modelling with deep learning [Wang, Hesthaven and R, 2019].

Future scope:

- Data-driven hp-adaptation.
- Deep learning for shape optimization.
- Estimating discontinuity alignment (classification + regression).
- Control spurious oscillations in spectral methods.

Other work

Numerical methods for conservation laws:

- High-order entropy stable schemes [Fjordholm and R, 2015].
- Kinetic energy preserving schemes for Euler equations [R, Chandrashekar, Fjordholm and Mishra, 2016].
- Entropy stable schemes for Navier-Stokes equations [R and Chandrashekar, 2017].

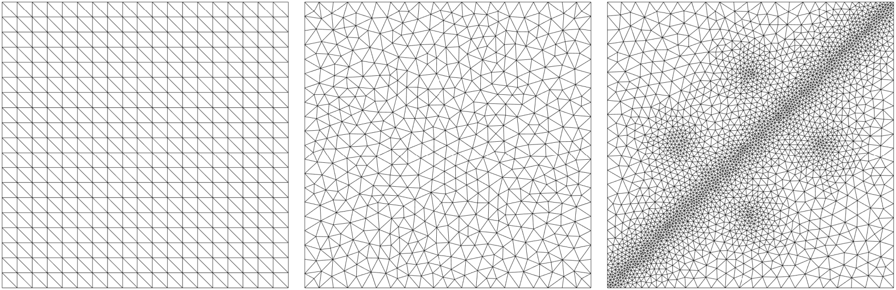
Porous media flow: (ongoing)

- Two-phase flows through rocks in the presence of surfactants.
- Data-driven limiting for two-phase flows.

Thank You

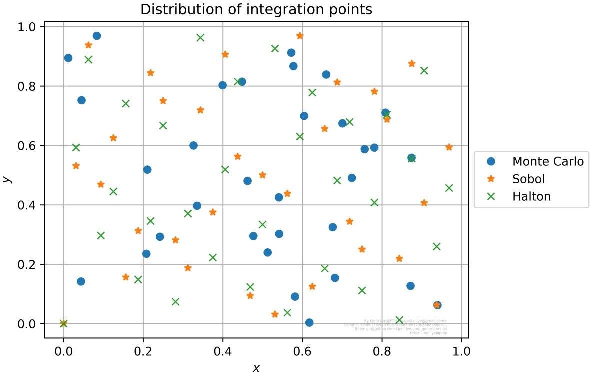
Contact:
deep.ray@rice.edu
deepray.github.io

2D MLP indicator: meshes



Locally quasi-uniform mesh

Low discrepancy sequences



$$\mathcal{D}_N = \max_B \left| \frac{A(B)}{N} - |B| \right|, \quad B \subset [0,1]^d$$

ENO interpolation

Idea: Use Newton's divided differences to construct the stencil hierarchically

Idea: Use Newton's divided differences to construct the stencil hierarchically

Newton's differences: Given values $f_1, f_2, ..., f_N$, we compute

$$\begin{aligned}
\Delta^0[f_i] &= f_i, \quad 1 \leq i \leq N \\
\Delta^1[f_i, f_{i+1}] &= \frac{\Delta^0[f_{i+1}] - \Delta^0[f_i]}{x_{i+1} - x_i}, \quad 1 \leq i \leq (N-1) \\
\Delta^k[f_i, ..., f_{i+k}] &= \frac{\Delta^1[f_{i+1}, ..., u_{i+k}] - \Delta^1[f_i, ..., f_{i+k-1}]}{x_{i+k} - x_i}
\end{aligned}$$

ENO interpolation

Idea: Use Newton's divided differences to construct the stencil hierarchically

Newton's differences: Given values $f_1, f_2, ..., f_N$, we compute

$$
\begin{aligned}
\Delta^0[f_i] &= f_i, \quad 1 \leq i \leq N \\
\Delta^1[f_i, f_{i+1}] &= \frac{\Delta^0[f_{i+1}] - \Delta^0[f_i]}{x_{i+1} - x_i}, \quad 1 \leq i \leq (N-1) \\
\Delta^k[f_i, ..., f_{i+k}] &= \frac{\Delta^1[f_{i+1}, ..., u_{i+k}] - \Delta^1[f_i, ..., f_{i+k-1}]}{x_{i+k} - x_i}
\end{aligned}
$$

$$
\Delta^k[f_i, ..., f_{i+k}] = \begin{cases} \frac{1}{k!}\frac{\mathrm{d}^k f(\xi)}{\mathrm{d}x^k} & \text{if } f \text{ is smooth in } [x_i, x_{i+k}] \\ \mathcal{O}(h^{-k}) & \text{if } f \text{ is discontinuous in } [x_i, x_{i+k}] \end{cases}
$$

ENO interpolation: stencil selection

The ENO algorithm: For a k-th order approx., we have to pick from $\{ \{p^j(x)\}_{j=0}^{k-1} \}$

ENO interpolation: stencil selection

The ENO algorithm: For a k-th order approx., we have to pick from $\{\{p^j(x)\}_{j=0}^{k-1}\}$

1. Set the initial stencil $S = \{x_{i-1}, x_i\}$.

ENO interpolation: stencil selection

The ENO algorithm: For a k-th order approx., we have to pick from $\{p^j(x)\}_{j=0}^{k-1}\}$

1. Set the initial stencil $S = \{x_{i-1}, x_i\}$.
2. Compare $\Delta^2[f_{i-2}, f_{i-1}, f_i]$ and $\Delta^2[f_{i-1}, f_i, f_{i+1}]$

$$S = \begin{cases} \{x_{i-2}, x_{i-1}, x_i\} & \text{if } |\Delta^2[f_{i-2}, f_{i-1}, f_i]| < |\Delta^2[f_{i-1}, f_i, f_{i+1}]| \\ \{x_{i-1}, x_i, x_{i+1}\} & \text{otherwise} \end{cases}$$

Lets say $S = \{x_{i-1}, x_i, x_{i+1}\}$.

ENO interpolation: stencil selection

The ENO algorithm: For a k-th order approx., we have to pick from $\{p^j(x)\}_{j=0}^{k-1}\}$

1. Set the initial stencil $S = \{x_{i-1}, x_i\}$.
2. Compare $\Delta^2[f_{i-2}, f_{i-1}, f_i]$ and $\Delta^2[f_{i-1}, f_i, f_{i+1}]$

$$S = \begin{cases} \{x_{i-2}, x_{i-1}, x_i\} & \text{if } |\Delta^2[f_{i-2}, f_{i-1}, f_i]| < |\Delta^2[f_{i-1}, f_i, f_{i+1}]| \\ \{x_{i-1}, x_i, x_{i+1}\} & \text{otherwise} \end{cases}$$

   Lets say $S = \{x_{i-1}, x_i, x_{i+1}\}$.
3. Compare Δ^3

$$S = \begin{cases} \{x_{i-2}, x_{i-1}, x_i, x_{i+1}\} & \text{if } |\Delta^3[u_{i-2}, u_{i-1}, u_i, u_{i+1}]| < |\Delta^3[u_{i-1}, u_i, u_{i+1}, u_{i+2}]| \\ \{x_{i-1}, x_i, x_{i+1}, x_{i+2}\} & \text{otherwise} \end{cases}$$

ENO interpolation: stencil selection

The ENO algorithm: For a k-th order approx., we have to pick from $\{\{p^j(x)\}_{j=0}^{k-1}\}$

1. Set the initial stencil $S = \{x_{i-1}, x_i\}$.
2. Compare $\Delta^2[f_{i-2}, f_{i-1}, f_i]$ and $\Delta^2[f_{i-1}, f_i, f_{i+1}]$

$$S = \begin{cases} \{x_{i-2}, x_{i-1}, x_i\} & \text{if } |\Delta^2[f_{i-2}, f_{i-1}, f_i]| < |\Delta^2[f_{i-1}, f_i, f_{i+1}]| \\ \{x_{i-1}, x_i, x_{i+1}\} & \text{otherwise} \end{cases}$$

   Lets say $S = \{x_{i-1}, x_i, x_{i+1}\}$.

3. Compare Δ^3

$$S = \begin{cases} \{x_{i-2}, x_{i-1}, x_i, x_{i+1}\} & \text{if } |\Delta^3[u_{i-2}, u_{i-1}, u_i, u_{i+1}]| < |\Delta^3[u_{i-1}, u_i, u_{i+1}, u_{i+2}]| \\ \{x_{i-1}, x_i, x_{i+1}, x_{i+2}\} & \text{otherwise} \end{cases}$$

4. Continue till S has k nodes.

Idea of proof:

- ReLU function:

$$\mathcal{A}(x) = \max(x, 0) = |x|_+.$$

Idea of proof:

- ReLU function:

$$\mathcal{A}(x) = \max(x, 0) = |x|_+.$$

- Identity function:

$$x = |x|_+ - |x|_- = \mathcal{A}(x) - \mathcal{A}(-x).$$

- Max function:

$$\max(x, y) = x + |y - x|_+ = \mathcal{A}(x) - \mathcal{A}(-x) + \mathcal{A}(y - x).$$

- Action of the Heaviside function:

$$H(x) = \begin{cases} -1 & \text{if } x < 0, \\ +1 & \text{if } x >= 0, \end{cases}$$

can be represented using combination of ReLU.

ENO with sub-cell resolution (ENO-SR)

The good: ENO interpolation controls Gibbs oscillations.

The bad: Only first-order accurate if f is piecewise smooth.

ENO with sub-cell resolution (ENO-SR)

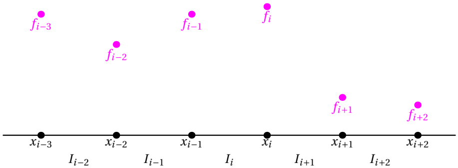
The good: ENO interpolation controls Gibbs oscillations.
The bad: Only first-order accurate if f is piecewise smooth.

Linear ENO-SR algorithm: (adapted from Arándiga et al., 2005)

The good: ENO interpolation controls Gibbs oscillations.
The bad: Only first-order accurate if f is piecewise smooth.

Linear ENO-SR algorithm: (adapted from Arándiga et al., 2005)

1. Use divided-difference $\Delta^2[f]$ to flag cells as good (G) or bad (B).

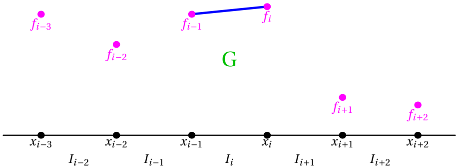


ENO with sub-cell resolution (ENO-SR)

The good: ENO interpolation controls Gibbs oscillations.
The bad: Only first-order accurate if f is piecewise smooth.

Linear ENO-SR algorithm: (adapted from Arándiga et al., 2005)

1. Use divided-difference $\Delta^2[f]$ to flag cells as good (G) or bad (B).
2. If I_i is G, then use linear interpolant.

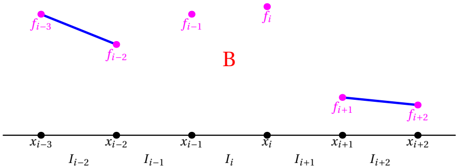


ENO with sub-cell resolution (ENO-SR)

The good: ENO interpolation controls Gibbs oscillations.
The bad: Only first-order accurate if f is piecewise smooth.

Linear ENO-SR algorithm: (adapted from Arándiga et al., 2005)

1. Use divided-difference $\Delta^2[f]$ to flag cells as good (G) or bad (B).
2. If I_i is G, then use linear interpolant.
3. If I_i is B, then use linear interpolant from I_{i-2} and I_{i+2} and extrapolate.

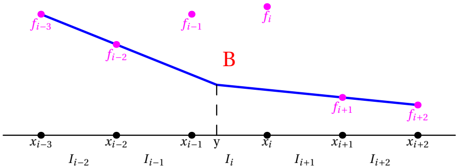


ENO with sub-cell resolution (ENO-SR)

The good: ENO interpolation controls Gibbs oscillations.
The bad: Only first-order accurate if f is piecewise smooth.

Linear ENO-SR algorithm: (adapted from Arándiga et al., 2005)

1. Use divided-difference $\Delta^2[f]$ to flag cells as good (G) or bad (B).
2. If I_i is G, then use linear interpolant.
3. If I_i is B, then use linear interpolant from I_{i-2} and I_{i+2} and extrapolate.
   - If extrapolants intersect at y, then $\tilde{p}_i(x) = p_{i-2}(x)\mathbb{I}_{[x_{i-1},y]} + p_{i+2}(x)\mathbb{I}_{[y,x_i]}$.

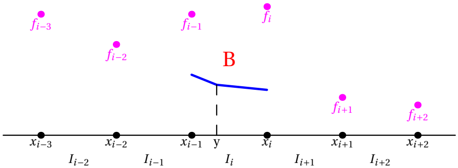


ENO with sub-cell resolution (ENO-SR)

The good: ENO interpolation controls Gibbs oscillations.
The bad: Only first-order accurate if f is piecewise smooth.

Linear ENO-SR algorithm: (adapted from Arándiga et al., 2005)

1. Use divided-difference $\Delta^2[f]$ to flag cells as good (G) or bad (B).
2. If I_i is G, then use linear interpolant.
3. If I_i is B, then use linear interpolant from I_{i-2} and I_{i+2} and extrapolate.
   - If extrapolants intersect at y, then $\tilde{p}_i(x) = p_{i-2}(x)\mathbb{I}_{[x_{i-1},y]} + p_{i+2}(x)\mathbb{I}_{[y,x_i]}$.

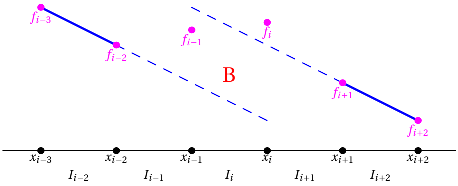


ENO with sub-cell resolution (ENO-SR)

The good: ENO interpolation controls Gibbs oscillations.
The bad: Only first-order accurate if f is piecewise smooth.

Linear ENO-SR algorithm: (adapted from Arándiga et al., 2005)

1. Use divided-difference $\Delta^2[f]$ to flag cells as good (G) or bad (B).
2. If I_i is G, then use linear interpolant.
3. If I_i is B, then use linear interpolant from I_{i-2} and I_{i+2} and extrapolate.
   - If extrapolants intersect at y, then $\tilde{p}_i(x) = p_{i-2}(x)\mathbb{I}_{[x_{i-1},y]} + p_{i+2}(x)\mathbb{I}_{[y,x_i]}$.
   - Otherwise reflag as G and use linear interpolant.

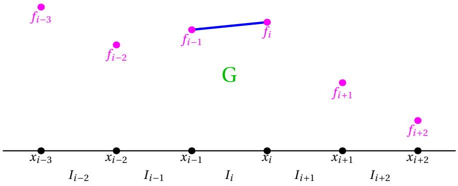


ENO with sub-cell resolution (ENO-SR)

The good: ENO interpolation controls Gibbs oscillations.
The bad: Only first-order accurate if f is piecewise smooth.

Linear ENO-SR algorithm: (adapted from Arándiga et al., 2005)

1. Use divided-difference $\Delta^2[f]$ to flag cells as good (G) or bad (B).
2. If I_i is G, then use linear interpolant.
3. If I_i is B, then use linear interpolant from I_{i-2} and I_{i+2} and extrapolate.
   - If extrapolants intersect at y, then $\tilde{p}_i(x) = p_{i-2}(x)\mathbb{I}_{[x_{i-1},y]} + p_{i+2}(x)\mathbb{I}_{[y,x_i]}$.
   - Otherwise reflag as G and use linear interpolant.



Wasserstein metric

- 1-Wasserstein metric

$$W_1(\mu, \nu) := \inf_{\pi \in \Pi(\mu,\nu)} \int_{\mathbb{R}\times\mathbb{R}} |u - v| d\pi(u, v).$$

- Koksma-Hlwaka inequality + optimal transport results

$$W_1(\hat{\mu}, \hat{\mu}_{qmc}) \sim V\mathcal{D}(\mathcal{J}_{qmc})$$

where V bounded by Hardy-Krause variations of $\mathcal{L}$.

Generalization error

- For N i.i.d. training samples [Lye et al., 2019]

$$\varepsilon_{1,G} \sim \varepsilon_{1,T} + \frac{D(\mathcal{L}, \mathcal{L}^*)}{\sqrt{N}}.$$

- Low discrepancy sequence (Sobol, Halton, etc) [Mishra and Rutsch, 2019]

$$\varepsilon_{1,G} \sim \varepsilon_{1,T} + \frac{\widetilde{D}(\mathcal{L}, \mathcal{L}^*)(\log(N))^d}{N}.$$

  $\widetilde{D}$ depends on the Hardy-Krause variations.

- Observables have lower variation than fields (empirically observed).

HPC

- Euler cluster (ETHZ).
- Parallelized finite- volume solver.
- 16 Intel(R) Xeon(R) Gold 5118 with 2.30GHz processor cores.
- Wall clock time = 1500 X 16 = 24000 secs.
- Training on a Intel(R) Core(TM) i7-8700K CPU with 3.70GHz machine.