Data Driven Stabilization Schemes for Singularly Perturbed Partial Differential Equations

Dr. Sangeeta Yadav

Assistant Professor,
Department of Computer Science and Engineering,
University of Delhi

December 17, 2025

Motivation

Figure: Convection

Figure: Diffusion

Applications of Convection Diffusion Equation

- Water pollution problems
- Oil extraction
- Convective heat transport
- Continuity equation in semiconductors
- Chemotaxis modelling in bacteria
- Chromatography
- Image processing (Perona-Malik equation)

What is the Convection–Diffusion Equation?

- A parabolic partial differential equation

What is the Convection–Diffusion Equation?

- A parabolic partial differential equation
- Combines diffusion and convection (advection)

What is the Convection–Diffusion Equation?

- A parabolic partial differential equation
- Combines diffusion and convection (advection)
- Describes transport of mass, heat, or energy

What is the Convection–Diffusion Equation?

- A parabolic partial differential equation
- Combines diffusion and convection (advection)
- Describes transport of mass, heat, or energy
- Also called advection–diffusion or drift–diffusion equation

Physical Interpretation

- Diffusion: spreading due to concentration gradients

Key Idea

Transport = Diffusion + Convection + Sources/Sinks

Summary

The convection–diffusion equation unifies diffusion, advection, and source effects into a single mathematical framework for transport phenomena.

Physical Interpretation

- Diffusion: spreading due to concentration gradients
- Convection: transport due to bulk motion

Key Idea

Transport = Diffusion + Convection + Sources/Sinks

Summary

The convection–diffusion equation unifies diffusion, advection, and source effects into a single mathematical framework for transport phenomena.

Physical Interpretation

- Diffusion: spreading due to concentration gradients
- Convection: transport due to bulk motion
- Both effects act simultaneously in many systems

Key Idea

Transport = Diffusion + Convection + Sources/Sinks

Summary

The convection–diffusion equation unifies diffusion, advection, and source effects into a single mathematical framework for transport phenomena.

Convection Diffusion Equation

$$\underbrace{-\epsilon \Delta u}_{\text{Diffusion term}} + \underbrace{\mathbf{b}.\nabla u}_{\text{Convection term}} = \underbrace{f}_{\text{Source term}} \quad \text{in } \Omega$$

$$u = u_b \text{ on } \Gamma^D,$$

Variable	Description	Variable	Description
$\Omega \subset \mathbb{R}^n$	Bounded Domain	$x \in \Omega \cup \Gamma$	Spatial point in Domain
$\epsilon > 0$	Diffusion coefficient	$u(x)$	Unknown scalar function
$\mathbf{b} \in W^{1,\infty}(\Omega)^2$	Convection velocity	$u_b \in H^{1/2}(\Gamma^D)$	Dirichlet boundary value
$f \in L^2(\Omega)$	External source term		

Numerical Schemes for Partial Differential Equation

- Finite Difference Method
- Finite Element Method
- Finite Volume Method
- Stabilized Numerical algorithms
- Neural Network-based PDE solvers

Why Finite Element Method (FEM)?

- Handles complex geometries and boundaries

Key Idea

FEM converts the PDE into a weak (variational) problem

Why Finite Element Method (FEM)?

- Handles complex geometries and boundaries
- Naturally incorporates boundary conditions

Key Idea

FEM converts the PDE into a weak (variational) problem

Why Finite Element Method (FEM)?

- Handles complex geometries and boundaries
- Naturally incorporates boundary conditions
- Well-suited for diffusion-dominated problems

Key Idea

FEM converts the PDE into a weak (variational) problem

Why Finite Element Method (FEM)?

- Handles complex geometries and boundaries
- Naturally incorporates boundary conditions
- Well-suited for diffusion-dominated problems
- Extensible to stabilized methods for convection

Key Idea

FEM converts the PDE into a weak (variational) problem

Weak (Variational) Formulation

- Multiply by test function v and integrate:
  $\int_\Omega \epsilon \nabla u \cdot \nabla v dx * \int_\Omega (\mathbf{b} \cdot \nabla u) v dx = \int_\Omega f v dx$

Weak (Variational) Formulation

- Multiply by test function v and integrate:
  $\int_\Omega \epsilon \nabla u \cdot \nabla v dx * \int_\Omega (\mathbf{b} \cdot \nabla u) v dx = \int_\Omega f v dx$
- Dirichlet conditions enforced

Finite Element Discretization

- Discretize domain Ω into elements

Result

$$\mathbf{AU} = \mathbf{F}$$

Finite Element Discretization

- Discretize domain Ω into elements
- Choose finite-dimensional space V_h

Result

$$\mathbf{AU} = \mathbf{F}$$

Finite Element Discretization

- Discretize domain Ω into elements
- Choose finite-dimensional space V_h
- Approximate $u \approx u_h = \sum U_i \phi_i$

Result

$$\mathbf{AU} = \mathbf{F}$$

Finite Element Discretization

- Discretize domain Ω into elements
- Choose finite-dimensional space V_h
- Approximate $u \approx u_h = \sum U_i \phi_i$
- Solve resulting linear system

Result

$$\mathbf{AU} = \mathbf{F}$$

FEniCS Implementation

- High-level Python interface for FEM

FEniCS Implementation

- High-level Python interface for FEM
- Automatic assembly of variational forms

FEniCS Implementation

- High-level Python interface for FEM
- Automatic assembly of variational forms
- Close to mathematical notation

FEniCS: Problem Setup

Define mesh and function space:

FEniCS: Problem Setup

Define mesh and function space:

Python

```
mesh = UnitSquareMesh(50, 50)
V = FunctionSpace(mesh, "CG", 1)
```

Trial and Test Functions

Trial and test function:

Trial and Test Functions

Trial and test function:

Python

```
u = TrialFunction(V)  v = TestFunction(V)
```

Physical Coefficients

Define diffusion and velocity:

Physical Coefficients

Define diffusion and velocity:

Python

```
epsilon = Constant(0.01)
b = Constant(1.0, 0.0)
```

Variational Form in FEniCS

Translate weak form directly:

Variational Form in FEniCS

Translate weak form directly:

Python

```
a = epsilon*dot(grad(u), grad(v))*dx
  + dot(b, grad(u))*v*dx
```

Boundary Conditions

Apply Dirichlet BC:

Boundary Conditions

Apply Dirichlet BC:

Python

```
bc = DirichletBC(V, Constant(0.0), "on_boundary")
```

Solving the System

Compute FEM solution:

Solving the System

Compute FEM solution:

Python

$u = Function(V)$
`solve(`$a == f * v * dx$`, u, bc)`

Final Takeaway

Summary

FEM provides a robust framework for convection–diffusion problems, and FEniCS enables concise, physics-driven implementations close to mathematical formulations.

Singularly Perturbed Differential Equations (SPDE)

$$-\epsilon u''(x) + u'(x) = 1, \text{ for } x \in (0,1),$$
$$u(0) = u(1) = 0$$

Suppose, we set $\epsilon = 0$, the above example will be converted as first-order ODE

$$u'(x) = 1 \text{ for } 0 < x < 1$$

The exact solution will not satisfy both boundary conditions

This problem has no solution in $C^1[0,1]$

We infer that when ϵ is near zero, the solution behaves badly in some way

So these types of differential equations are called singularly perturbed differential equations (SPDE)

Boundary Layer

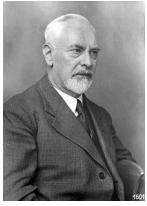
Ludwig Prandtl introduced the terminology boundary layer at the 3rd International Congress of Mathematicians in Heidelberg in 1904. Layers are narrow regions where the solution has a steep gradient.

$$u(x) = \alpha x + \frac{(R - L - \alpha)[\exp^{(-\beta(1-x))} - \exp^{-\beta}]}{1 - \exp^{-\beta}} + L$$

$$\alpha = \frac{f}{b}; \quad \beta = \frac{b}{\epsilon}$$

For $a \in [0, 1)$, $\quad \lim_{x \to a} \lim_{\epsilon \to 0} u(x) = a = \lim_{\epsilon \to 0} \lim_{x \to a} u(x)$

$$1 = \lim_{x \to 1} \lim_{\epsilon \to 0} u(x) \neq \lim_{\epsilon \to 0} \lim_{x \to 1} u(x) = 0$$



Singularly Perturbed Differential Equations (SPDE)

- Solution approaches a discontinuous limit when $\epsilon \to 0$ and $x = 1$
- Due to this boundary layer, the numerical solution shows spurious oscillations.
- Stabilization techniques are used to get rid of these spurious oscillations
- Finding an optimal stabilization parameter is a challenge

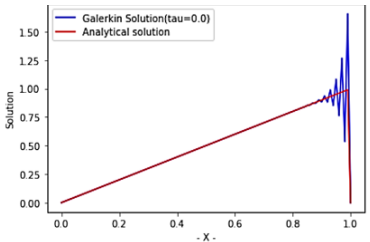
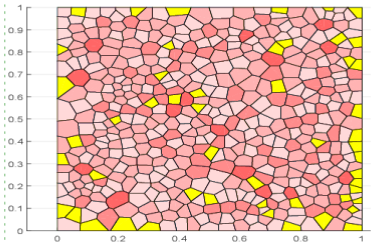
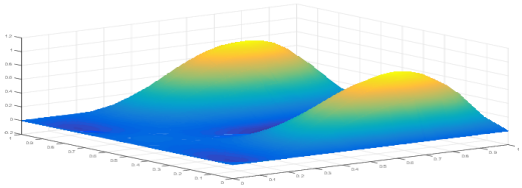


Figure: Oscillations in Galerkin solution

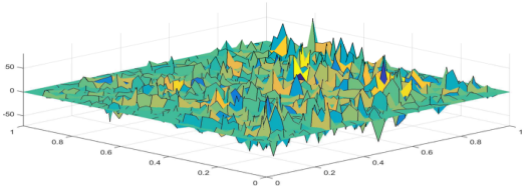
Spurious oscillations in numerical solution from Galerkin FEM technique



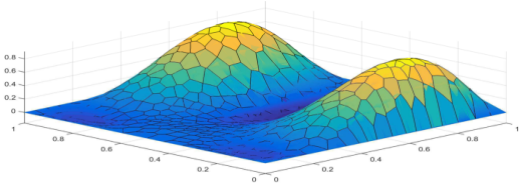
(a) Sample random polygonal mesh



(b) Exact solution

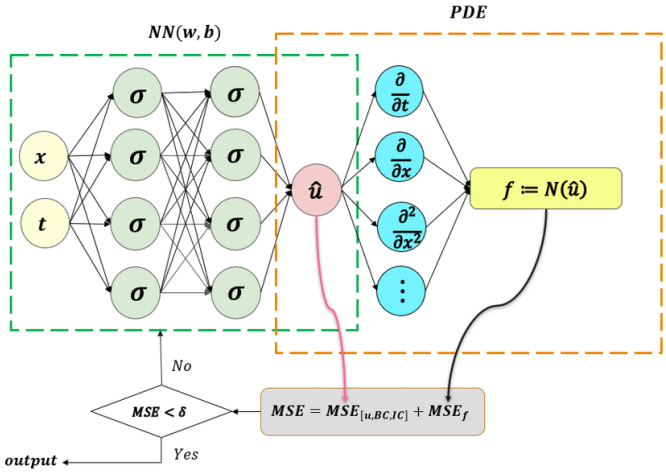


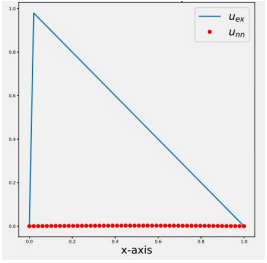
(c) Unstabilized solution



(d) Stabilized solution

PINN: Physics Informed Neural Network





Objective

- Develop a hybrid technique which can leverage the benefits of both conventional numerical schemes and neural networks
- Develop a technique capable of computing the layers sharp and accurately

Stabilization Techniques

- Orthogonal Subgrid Scale
- Continuous Interior Penalty
- Local Projection Stabilization
- Streamline Upwind Petrov Galerkin (SUPG)

Galerkin Weak Form of the SPDE

Find u such that for all $v \in H_0^1(\Omega)$

$$a(u, v) = (f, v) \tag{1}$$

where the bilinear form $a(\cdot, \cdot) : H^1(\Omega) \times H_0^1(\Omega) \to R$ is defined by

$$a(u, v) = \int_\Omega \epsilon u' v' dx + \int_\Omega b u' v dx \tag{2}$$

$$(f, v) = \int_\Omega f v dx \tag{3}$$

$(\cdot, \cdot)$ is the $L^2(\Omega)$ inner product.

Streamline Upwind Petrov Galerkin Technique(SUPG)

The residual of equation is :

$$R(u) = -\epsilon u'' + bu' - f \tag{4}$$

Modified weak form: Find $u_h \in V_h$ such that:

$$\begin{aligned} a_h(u_h, v_h) &= \epsilon(\nabla u_h, \nabla v_h) + (\mathbf{b} \cdot \nabla u_h, v_h) \\ &+ \sum_{i \in \Omega_h} \underbrace{\tau_i(-\epsilon \Delta u_h + \mathbf{b} \cdot \nabla u_h - f_h, \mathbf{b} \cdot \nabla v_h)_{\Omega_h}}_{\text{Stabilization term}} \\ &= (f, v_h) + (g, v_h)_{\Gamma^N} \quad \forall v_h \in V_h \end{aligned} \tag{5}$$

$\tau_i \in L^2(\Omega)$ is a user-chosen stabilization parameter.

Stabilization Parameter τ

Standard formula:

$$\text{For local Peclet number, } Pe = \frac{bh}{2\epsilon};$$
$$\tau = \frac{h}{2b}(\coth(Pe) - \frac{1}{Pe}); \tag{6}$$
$$\text{where } coth = \frac{\exp(x) + \exp(-x)}{\exp(x) - \exp(-x)}$$

Limitations:

- Std. τ gives the exact solution only for the 1D domain
- Std. τ technique has limited performance in complex cases

Objective: Develop a Neural Network model to identify an optimal stabilization parameter for 1D and 2D cases

SUPG Implementation in FEniCS

Add streamline stabilization term:

SUPG Implementation in FEniCS

Add streamline stabilization term:

Python

```
h = CellDiameter(mesh)
tau = h/(2*sqrt(dot(b,b)))
a += tau*dot(b, grad(u))*dot(b, grad(v))*dx
```

Proposed schemes for predicting τ for SUPG

- **SPDE-Net:**. An ANN for one-dimensional SPDEs
- **AI-stab FEM:** An optimization scheme using another Neural Network for solving two-dimensional SPDEs.
- **SPDE-ConvNet:** A convolutional neural network (CNN), for predicting the local(cell-wise) stabilization parameter.

SPDE-Net for 1D convection-diffusion equation

We use SPDE-Net an ANN to find the stabilization parameter

$$\hat{\tau}_i(\theta) = G_\theta(\epsilon_i, b_i, h_i) \tag{7}$$

$$\hat{u}_i(\theta) = H(\epsilon_i, b_i, h_i, \hat{\tau}_i) \tag{8}$$

$$\theta^*_{supervised} = argmin \sum_{i=1}^{N} loss\left(\hat{\tau}_i(\theta), \tau_i\right) \tag{9}$$

$$\theta^*_{L^2 \text{ Error Minimization}} = argmin \sum_{i=1}^{N} loss\left(\hat{u}_i(\theta), u_i\right) \tag{10}$$

where, G_θ is θ parameterized SPDE-Net, H is the FEM solution, τ_i is the stabilization parameter, u is the analytical solution and N is the number of training examples.

SPDE-Net

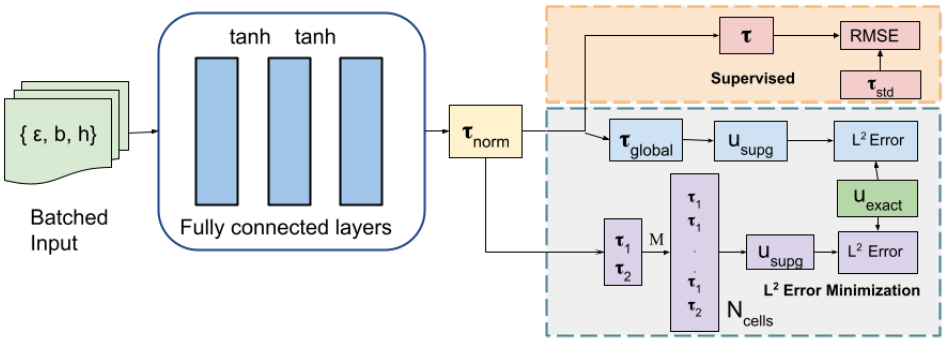


Figure: *SPDE-Net*: An end-to-end deep learning+FEM framework for solving SPDE

Algorithm SPDE-Net: L^2 error minimization

x: Input, $\hat{u}$: solution with $\hat{\tau}$, u: analytical solution,η: learning rate, bs: batchsize
Initialize network parameters with θ_0 randomly initialized parameters
Initialize the Adam optimizer and stepLR scheduler
for $k = 1$ to n_{epochs} **do**
    $loss(\theta_k) = 0$
    for $m = 1$ to n_{batches} **do**
        $\hat{\tau}(\theta_k) = \text{SPDE-Net}(x_{(m-1)\times bs:m\times bs}; \theta_k)$
        $loss(\theta_k) += (\int_\Omega (\hat{u}(\hat{\tau}(\theta_k)) - u)dx)^{\frac{1}{2}}$
    end for
    $\theta_{k+1} = \theta_k - \eta_k \frac{\nabla loss(\theta_k)}{N}$
end for

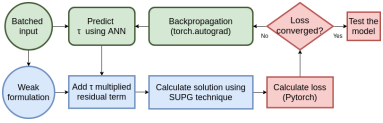


Figure: Training algorithm

L^2 Error Minimization

- Global $\hat{\tau}$: Predict single τ for whole domain.
- Local $\hat{\tau}$: Predict 2 values, $\hat{\tau}_1$ and $\hat{\tau}_2$ for non-boundary and boundary layer regions. In this particular case, the boundary region is either near $x = 0$(for $b < 0$) or $x = 1$(for $b > 0$). For $b > 0$:

$$M = \begin{bmatrix} 1 & 0 \\ 1 & 0 \\ \vdots & \vdots \\ 0 & 1 \end{bmatrix}_{N_{\text{cells}},2} \tag{11}$$

$$\tau_{pred} = [\hat{\tau}_1(\theta), \hat{\tau}_2(\theta)]^T \tag{12}$$

$$\hat{\tau}_{local} = M\tau_{pred} \tag{13}$$

Evaluation Metrics

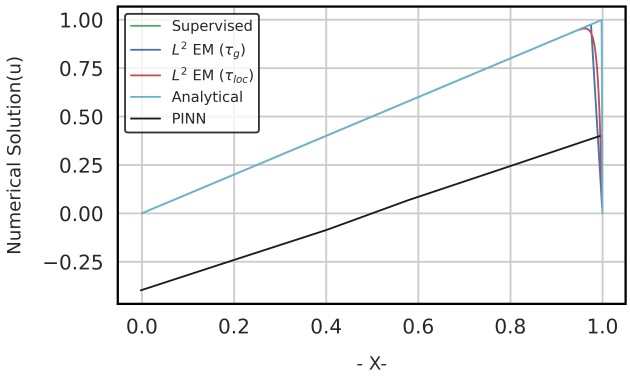
$$RMSE = \frac{\sqrt{\sum_{i=1}^{N}(\hat{\tau} - \tau)^2}}{N} \tag{14}$$

$$L^2Error = \left(\int_{\Omega} (u_{supg}(\hat{\tau}) - u_{\text{analytical}})^2 dx \right)^{\frac{1}{2}} \tag{15}$$

Qualitative Comparison

Test Case

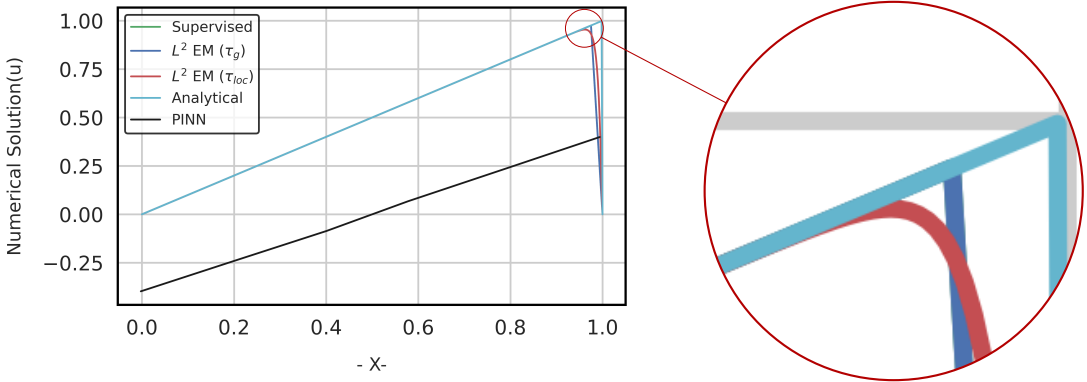
$\epsilon = 1e-11, \quad b = 1.0, \quad h = 0.0083$



Qualitative Comparison

Test Case

$\epsilon = 1e-11, \quad b = 1.0, \quad h = 0.0083$



Performance Comparison

Table: Performance comparison of different techniques for validation and test dataset

	Validation data		**Test data**	
Technique	$\|\|\hat{u}(\hat{\tau}) - u\|\|_{L^2(\Omega_h)}$	$\|\|\hat{\tau} - \tau\|\|_{L^2(\Omega_h)}$	$\|\|\hat{u}(\hat{\tau}) - u\|\|_{L^2(\Omega_h)}$	$\|\|\hat{\tau} - \tau\|\|_{L^2(\Omega_h)}$
PINN	$8.11\,\mathrm{e}{-3}$	NA	$7.82\,\mathrm{e}{-3}$	NA
Supervised	$5.13\,\mathrm{e}{-6}$	$2.79\,\mathrm{e}{-7}$	$7.88\,\mathrm{e}{-6}$	$3.72\,\mathrm{e}{-7}$
L^2 EM(τ_{loc})	$6.42\,\mathrm{e}{-5}$	NA	$1.70\,\mathrm{e}{-4}$	NA
L^2 EM(τ_g)	$\mathbf{5.00\,e{-6}}$	$3.33\,\mathrm{e}{-6}$	$\mathbf{7.76\,e{-6}}$	$4.83\,\mathrm{e}{-7}$

Perturbation analysis (error): Critical dataset

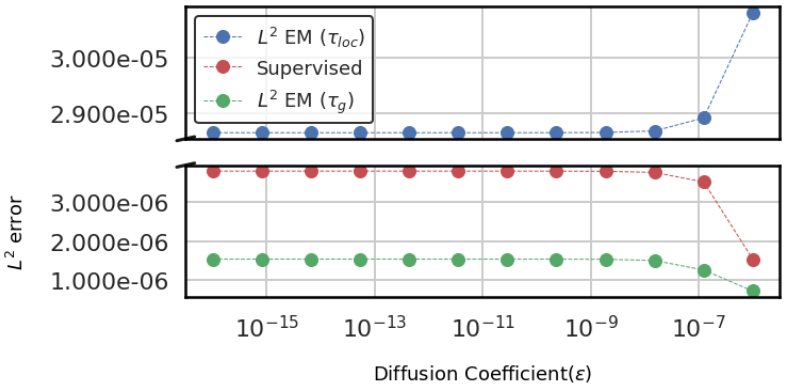


Figure: Perturbation analysis of error for critical test set

Perturbation analysis (error): Non-critical dataset

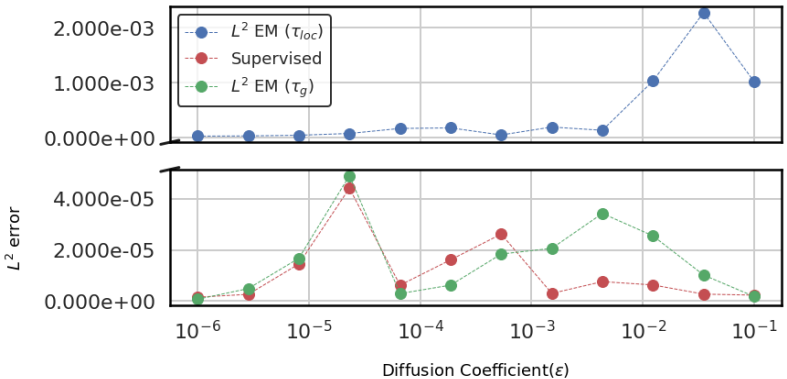


Figure: Perturbation analysis of error for non-critical test set

Mesh refinement analysis

Test cases

All the techniques are tested on a set of examples with constant ϵ, b and varying h.
$\epsilon = 10^{-11}, b = 1.6, N_{cells} = \{512, 256, 128, 64, 32, 16\}$

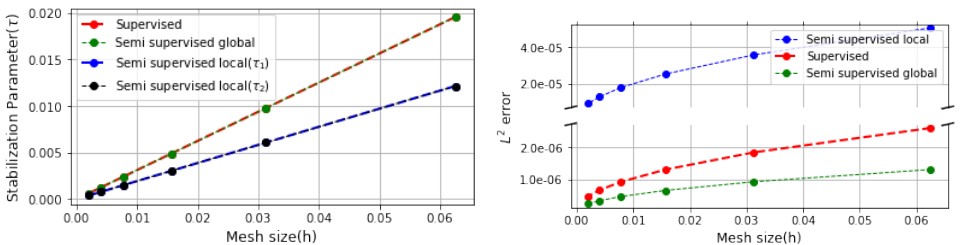


Figure: Mesh refinement analysis: (a) Stabilization parameter τ (b) L^2 Error

2D Convection Diffusion Equation

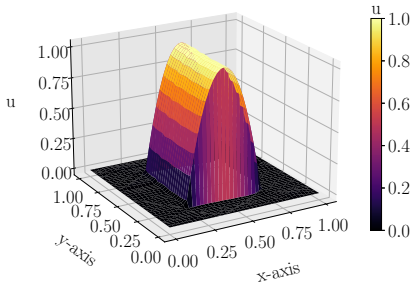
Let $\Omega \subset \mathbb{R}^2$ be a bounded domain, and $x \in \mathbb{R}^2$ be a spatial point in Ω. A 2-D convection-diffusion equation is given as follows:

$$-\epsilon \Delta u + \mathbf{b}.\nabla u = f \text{ in } \Omega, \text{ and } u = u_b \text{ on } \Gamma^D, \tag{16}$$

- $\epsilon > 0$: Diffusion coefficient
- $\mathbf{b} \in W^{1,\infty}(\Omega)^2$: The convection velocity
- $f \in L^2(\Omega)$: The external source term
- $u(x, y)$: The unknown scalar
- $u_b \in H^{1/2}(\Gamma^D)$: Dirichlet boundary condition

Challenges in 2D problem

- Convection in the cross-wind direction
- No closed-form analytical solution available for training
- Accuracy of the standard formula for stabilization parameter is low
- Presence of interior layer in addition to the boundary layer



Proposed Technique: AI-stab FEM

$$\tau(\theta) = \text{AI-stab FEM}(\epsilon, \mathbf{b}, h; \theta)$$

$u := u(x, y, \tau(\theta))$ is the SUPG solution using stabilization parameter

$$\text{Loss: } \mathcal{L}(\tau) = [-\epsilon \Delta u(\tau) + \mathbf{b} \cdot \nabla u(\tau) - f]^2 + q(\mathbf{b}^{\perp} \cdot \nabla u(\tau))^1 \tag{17}$$

$$\text{where, } q(s) = \begin{cases} \sqrt{s} & s > 1 \\ 2.5s^2 - 1.5s^3 & \text{otherwise} \end{cases}$$

$$\theta^* = argmin(\mathcal{L}(u, \tau(\theta)))$$

[1] John, Volker & Knobloch, Petr & Savescu, Simona. (2011). A posteriori optimization of parameters in stabilized methods for convection-diffusion problems-Part I. Computer Methods in Applied Mechanics and Engineering - COMPUT METHOD APPL MECH ENG. 200. 10.1016/j.cma.2011.04.016.

Algorithm AI-stab FEM

1: $\mathsf{I} = \{\varepsilon, b_1, b_2, h_T\}$

2: $Pe(I) = \dfrac{|\mathbf{b}|h_T}{2\varepsilon}$

3: **for** $k = 1$ to n_{epochs} **do**

4: $\tau(\theta_k) = \dfrac{G_{\theta_k}(Pe(I))}{||\nabla u_{Galerkin}(I)||}$

5: $q(s) = \begin{cases} \sqrt{s} & s > 1 \\ 2.5s^2 - 1.5s^3 & \text{otherwise} \end{cases}$

6: $b^{\perp} = \begin{cases} \dfrac{(b_2, -b_1)}{|b(x)|} & \text{when } b(x) \neq 0 \\ 0 & \text{when } b(x) = 0 \end{cases}$

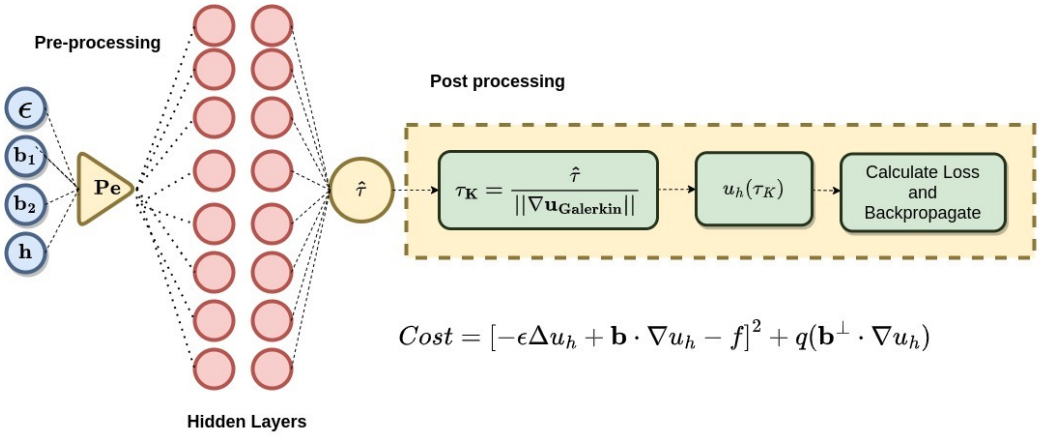
7: Solve given equation with $\tau(\theta_k)$ and get u_h

8: $loss(\theta_k) = \sum_{K_i \in \Omega} || - \varepsilon \Delta u_h + \mathbf{b} \cdot \nabla u_h - f||^2_{K_i} + \ q(\mathbf{b}^{\perp} \cdot \nabla u_h)$

9: $\theta_{k+1} = \theta_k - \eta_k \nabla loss(\theta_k)$

10: **end for**

AI-stab FEM: Proposed network architecture



$$Cost = [-\epsilon \Delta u_h + \mathbf{b} \cdot \nabla u_h - f]^2 + q(\mathbf{b}^{\perp} \cdot \nabla u_h)$$

Baselines

1. **Standard τ:** Given Pe_T is the local Peclet number, we use the following expression to compute τ:

$$\tau_{std}|_T = \frac{h_T}{2|\mathbf{b}|}\left(\coth\left(Pe_T\right) - \frac{1}{Pe_T}\right) \tag{18}$$

   where τ_{std} is used to solve SUPG weak form of the given equation.

2. **Standard τ with $||\nabla u_h||$:** We normalize the τ_{std} with the cell-wise euclidean norm of the gradient of the Galerkin solution.

$$\tau_h = \frac{\tau_{std}}{||\nabla u_h||}$$

3. **PINNs:** A neural network trained for solving PDEs by minimizing the residual of the given equation.

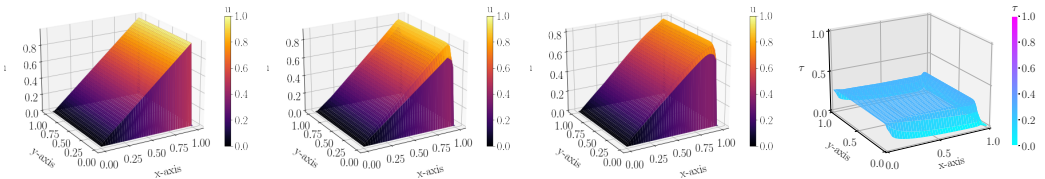
Test examples

Table: Value of τ by standard technique for different examples

Example	Internal Layer	Boundary Layer	N_{cells}	τ_{std}
1	✗	✓	40	1.77e-2
2	✗	✓	40	1.77e-2
3	✓	✓	40	1.77e-2
4	✓	✗	40	4.90e-3

Example 1

$$\epsilon = 10^{-8}, \quad \mathbf{b} = [b_1, b_2] = (1, 0), \quad f = 1, \quad \Omega = (0, 1)^2, u = 0 \text{ on } \partial\Omega_D \tag{19}$$



(a) Exact Solution (b) Solution with Std. τ (c) Solution from AI-stab FEM (d) τ predicted from AI-stab FEM

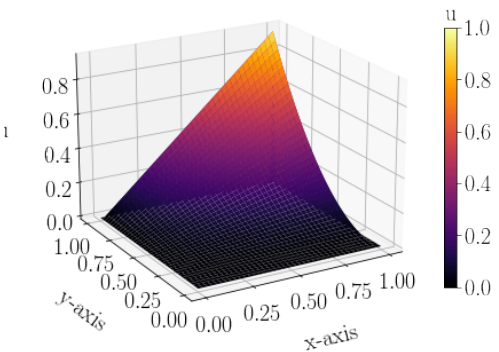
Example 2

$$\begin{aligned} &\epsilon = 10^{-8}, \mathbf{b} = (2,3), \Omega = (0,1)^2, \\ &u_b = 0 \quad \text{on} \quad \partial\Omega_D \end{aligned} \tag{20}$$

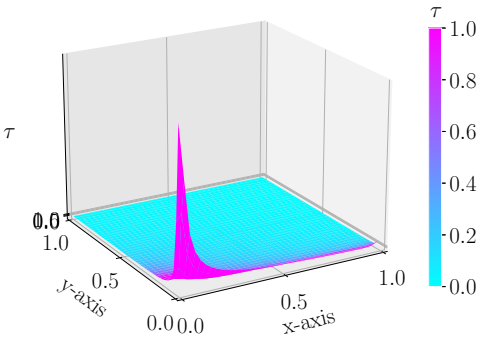
Source term f is calculated by substituting the following analytical solution(u_{exact}) in Eq. (20).

$$\begin{aligned} u_{exact}(x,y) = xy^2 - x\exp\left(\frac{3(y-1)}{\epsilon}\right) - y^2\exp\left(\frac{2(x-1)}{\epsilon}\right) \\ + \exp\left(\frac{2(x-1)+3(y-1)}{\epsilon}\right) \end{aligned} \tag{21}$$

Example 2

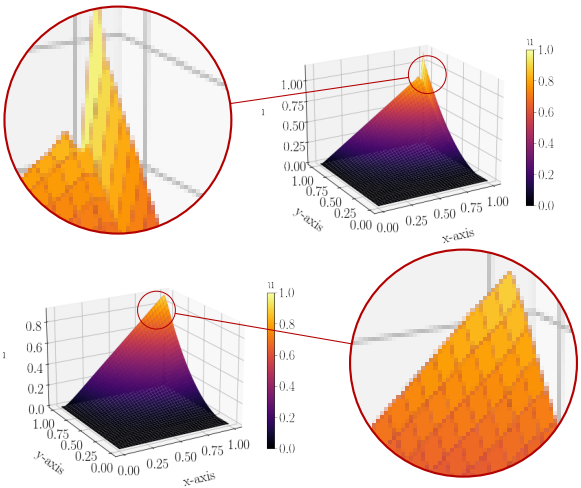


(a) Exact Solution



(b) τ predicted from SPDE-Net

Example 2: Comparison of oscillations



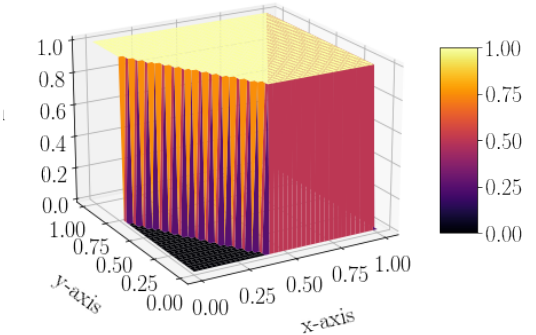
Solution with Std. τ (above) and AI-stab FEM (below)

Example 3

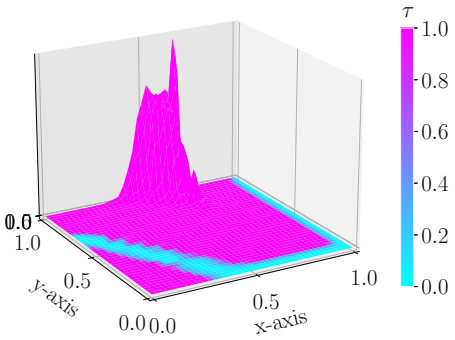
We consider the convection-diffusion equation (6) with following equation coefficients and boundary conditions:

$$
\begin{aligned}
&-\varepsilon\Delta u + \mathbf{b}\cdot\nabla u = f \text{ in } \Omega \\
&\varepsilon = 10^{-8},\ \theta = -\pi/3,\ \mathbf{b} = (\cos(\theta), \sin(\theta)),\ f = 0.0,\ \Omega = [0,1]^2, \\
&u_b = \begin{cases} 0, & \text{for } x = 1 \text{ or } y \leq 0.7 \\ 1, & \text{otherwise} \end{cases} \\
&u = u_b \quad \text{on} \quad \partial\Omega_D, \quad \partial\Omega_D = \partial\Omega
\end{aligned}
\tag{22}
$$

Example 3

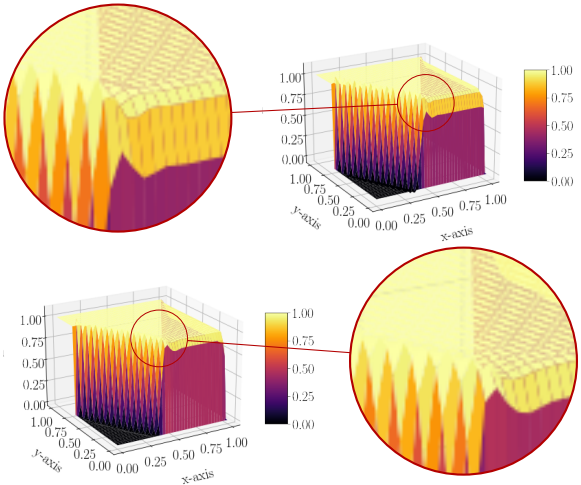


(a) Exact Solution



(b) τ predicted from AI-stab FEM

Example 3: Comparison of oscillations

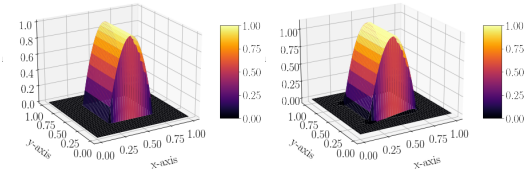


Solution with Std. τ (above) and AI-stab FEM (below)

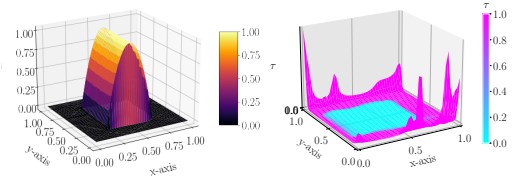
Example 4

We consider the convection-diffusion equation (6) with following equation coefficients and boundary conditions:

$$
\begin{aligned}
&-\varepsilon\Delta u + \mathbf{b}\cdot\nabla u = f \text{ in } \Omega \\
&\varepsilon = 10^{-8},\ \theta = -\pi/3,\ \mathbf{b} = (\cos(\theta), \sin(\theta)),\ f = 0.0,\ \Omega = [0,1]^2, \\
&u_b = \begin{cases} 0, & \text{for } x = 1 \text{ or } y \leq 0.7 \\ 1, & \text{otherwise} \end{cases} \\
&u = u_b \quad \text{on} \quad \partial\Omega_D, \quad \partial\Omega_D = \partial\Omega
\end{aligned}
\tag{23}
$$

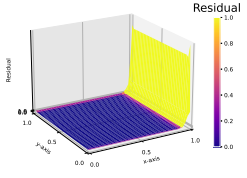


(a) Exact Solution (b) Solution with Std. τ

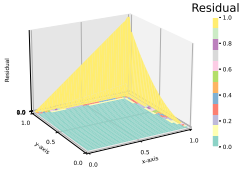


(c) Solution from AI-stab FEM (d) τ predicted from AI-stab FEM

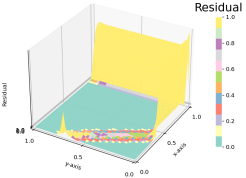
Residual and the predicted stabilization parameter



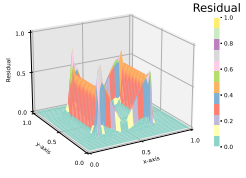
(a) Example 1: Residual



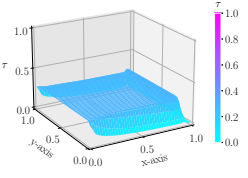
(b) Example 2: Residual



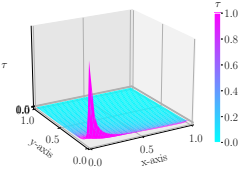
(c) Example 3: Residual



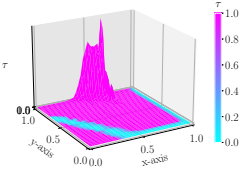
(d) Example 4: Residual



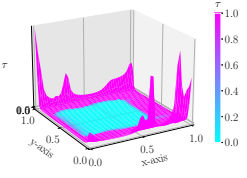
(e) Example 1: $\hat{\tau}$



(f) Example 2: $\hat{\tau}$



(g) Example 3: $\hat{\tau}$



(h) Example 4: $\hat{\tau}$

Evaluation Metrics

$$L^2 \text{ error: } ||e_h||_{L^2(\Omega)} = ||\hat{u}(\hat{\tau}) - u_{exact}||_{L^2(\Omega)} = \left(\int_\Omega (\hat{u}(\hat{\tau}) - u_{exact})^2 dx \right)^{\frac{1}{2}}$$

$$\text{Relative } l^2 \text{ error: } \sum_{i=1}^{i=N_d} \frac{||\hat{u}_{\hat{\tau}}(x_i) - u_{exact}(x_i)||_2}{||u||_2}$$

$$H^1 \text{ error: } ||e||_{H^1(\Omega)} = \sum_{|\alpha|<=1} (D^\alpha(\hat{u}(\hat{\tau}) - u_{exact}), D^\alpha(\hat{u}(\hat{\tau}) - u_{exact}))$$

$$= \sum_{\alpha<=1} \int_\Omega D^\alpha(\hat{u}(\hat{\tau}) - u_{exact}) D^\alpha(\hat{u}(\hat{\tau}) - u_{exact}) dx$$

$$L^\infty \text{ error: } ||e||_{L^\infty(\Omega)} = ess \quad sup\{|\hat{u}(\hat{\tau}) - u_{exact}| : x \in \Omega\}$$

Here N_d is the number of degrees of freedom, D^α is the weak derivative up to order α.

Results: Relative l^2-Error

Table: Relative l^2-Error

	Examples			
Techniques	1	2	3	4
PINN	4.85e-1	4.85e+1	8.69e-1	1.01e+0
Standard τ	1.17e-1	1.36e-1	8.02e-2	4.63e-2
Standard τ with $\|\|\nabla u\|\|$	3.35e-1	3.06e-1	9.68e-2	4.90e-2
AI-stab FEM	**6.17e-2**	**9.73e-2**	**6.94e-2**	**4.60e-2**

Localised stabilization parameter prediction

Limitation of AI-stab FEM:

- Same stabilization parameter is predicted for the entire domain
- Boundary/Interior layer regions are treated only in the post-processing step
- Only applicable to constant convective velocity cases

Remedies:

- Predict cell-wise stabilization parameter with a convolutional neural network
- Treat the layered region(Normalization) within the network training step

SPDE-ConvNet

$$\tau_K(\theta) = \textit{SPDE-ConvNet}(\epsilon^K, b_1^K, b_2^K, h^K, ||\nabla u_h^K||) \text{ where } K \in \Omega_h$$

$$\text{Loss } \mathcal{L}(\tau(\theta)) = \sum_{K=1}^{N_{\text{cells}}} [-\epsilon \Delta u_h + \mathbf{b} \cdot \nabla u_h - f]_K^2 + q(\mathbf{b}^\perp \cdot \nabla u_h)_K$$

$$\text{where, } q(s) = \begin{cases} \sqrt{s} & s > 1 \\ 2.5s^2 - 1.5s^3 & \text{otherwise} \end{cases} \tag{24}$$

$$\text{and, } \mathbf{b}^\perp(\mathbf{x}) = \begin{cases} \dfrac{[b_2(\mathbf{x}), -b_1(\mathbf{x})]}{||\mathbf{b}(\mathbf{x})||} & \text{if } ||\mathbf{b}(\mathbf{x})|| \neq 0 \\ 0 & \text{else } ||\mathbf{b}(\mathbf{x})|| = 0 \end{cases}$$

$$\theta^* = \arg\min_\theta \left(Loss(\tau(\theta))\right)$$

SPDE-ConvNet: Network Architecture

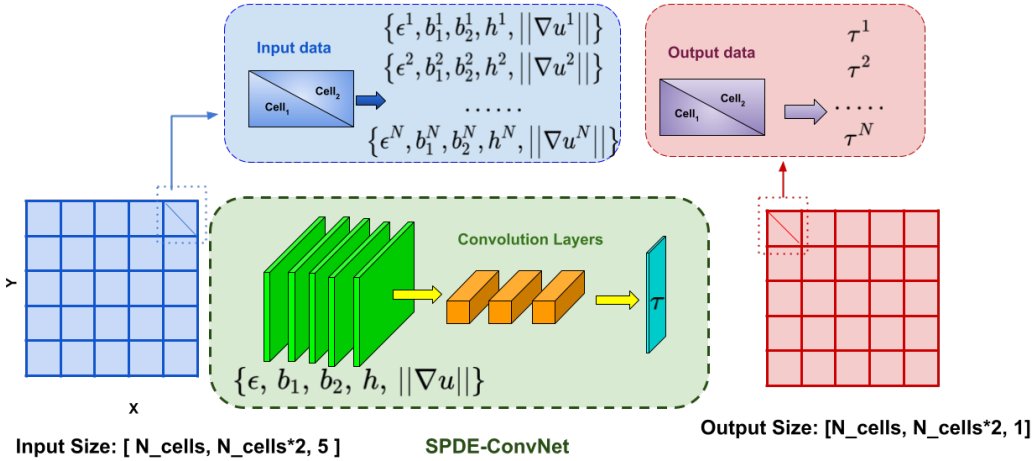


Figure: Network Architecture of *SPDE-ConvNet*

Results: Qualitative Comparison

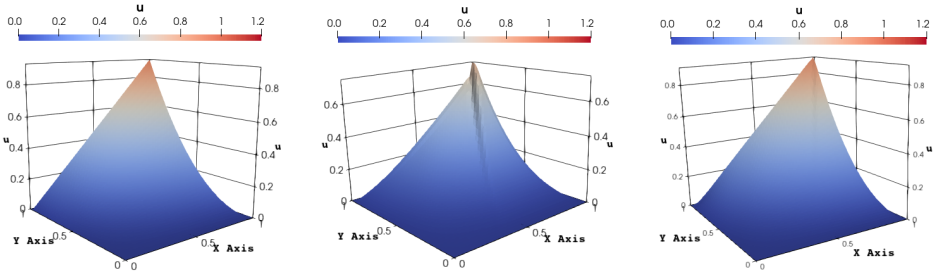
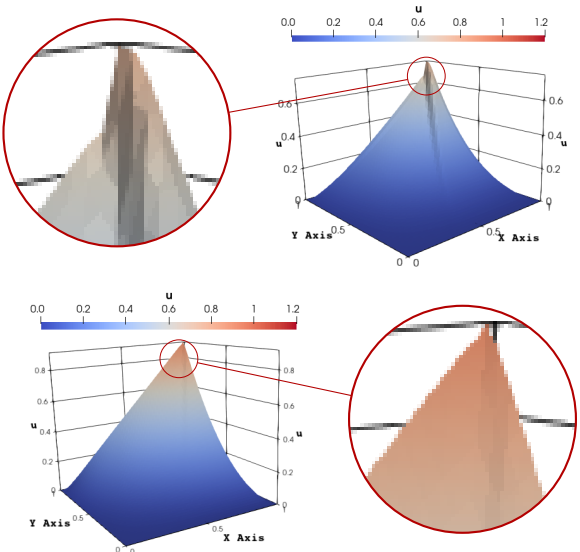
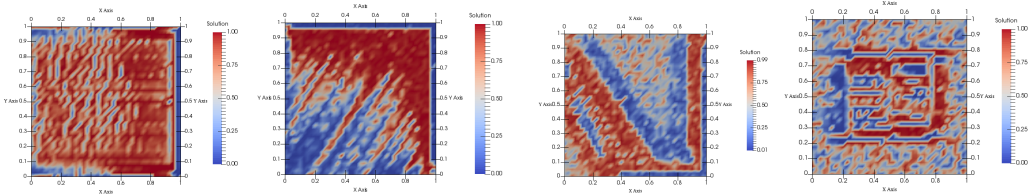


Figure: (a) Exact Solution, (b) Solution with standard τ, (c) Solution from *SPDE-ConvNet*

Std. τ vs. SPDE-ConvNet



$\hat{\tau}$ predicted from *SPDE-ConvNet*



(a) Example 1 (b) Example 2 (c) Example 3 (d) Example 4

Results

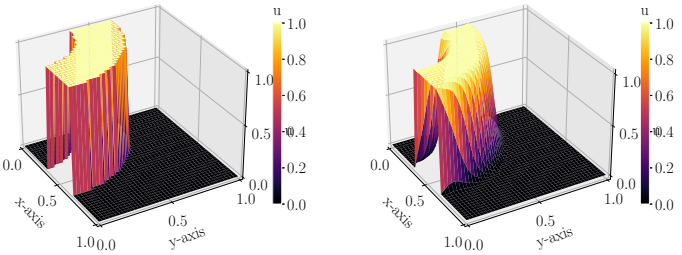
Table: Comparison of *SPDE-ConvNet* with other techniques

	L^2 Error	Relative l^2 error	H^1 error	L^∞ error
Standard τ	6.77e-6	1.36e-1	6.74e-4	7.29e-5
AI-stab FEM	5.04e-6	9.73e-2	4.80e-4	4.05e-5
SPDE-ConvNet	**3.04e-6**	**8.36e-2**	**3.20e-4**	**4.03e-5**

SPDE-ConvNet: Variable Convective velocity

$$\epsilon = 10^{-8}, \quad \mathbf{b} = (-y, x)^T, \quad \Omega = (0,1)^2, f = 0, \; u = u_b \quad \text{on} \quad \partial\Omega$$

$$u_b = \begin{cases} 1 & \text{if } \frac{1}{3} \leq x \leq \frac{2}{3} \text{ and } y = 0 \\ 0 & \text{otherwise, if } y = 0 \end{cases} \tag{25}$$



(a) Exact Solution (u) (b) u_h with $\hat{\tau}$ from SPDE-ConvNet

Conclusions

- ANN-aided solvers are more accurate than purely ANN-based PDE solvers such as PINNs and heuristic-based stabilised FEM Methods
- Of the three techniques, SPDE-ConvNet is the most promising one as it allows localised τ prediction with minimal parameters
- SPDE-ConvNet can be used for equations with varibale coefficients
- All three techniques achieved optimal order of convergence for L^2 error.

Future Work

- Extend to time-dependent convection-diffusion equations.
- Derive stability analysis of the proposed networks.
- Augment SPDE-ConvNet with adaptive mesh refinement.
- Deploy SPDE-ConvNet in practical applications, like the problems modelled by two fluids, such as viscoelastic fluids, which have multiple stabilization parameters in the SUPG formulation. One can use SPDE-ConvNet for these cases.

List of Journal publications

- Sangeeta Yadav, Sashikumaar Ganesan, "Artificial neural network-augmented stabilized finite element method, Journal of Computational Physics, 499, (15 February 2024), 112702.
- Sangeeta Yadav, Sashikumaar Ganesan, "ConvStabNet: a CNN-based approach for the prediction of local stabilization parameter for SUPG scheme", Calcolo, 61 (52)

List of publications

- Sangeeta Yadav and Sashikumaar Ganesan, "Convolutional Neural network for local stabilization parameter prediction for Singularly Perturbed PDEs", Synergy of Scientific and Machine Learning Modeling, International Conference on Machine Learning (ICML 2023)
- Sangeeta Yadav and Sashikumaar Ganesan, "Predicting the stabilization quantity with neural networks for Singularly Perturbed Partial Differential Equations", Synergy of Scientific and Machine Learning Modeling, International Conference on Machine Learning (ICML 2023).
- Sangeeta Yadav, Sashikumaar Ganesan, "SPDE-Net: Neural Network based prediction of stabilization parameter for SUPG technique", Proceedings of the 13th Asian Conference on Machine Learning, PMLR 157:268-283, 2021.
- Sangeeta Yadav, Sashikumaar Ganesan, "SPDE-ConvNet: Solve Singularly Perturbed Partial Differential Equation with deep learning", Proceedings of the European Congress on Computational Methods in Applied Sciences and Engineering-ECCOMAS CONGRESS 2022.

Conferences presentations

- Sangeeta Yadav, Sashikumaar Ganesan. "How deep learning performs with singularly perturbed problems?" IEEE Second International Conference on Artificial Intelligence and Knowledge Engineering (AIKE), pages 293–297, 2019.
- *IEEE Second International Conference on Artificial Intelligence and Knowledge Engineering (AIKE), 3-5 June 2019, held at Cagliari, Italy*
- *The 13th Asian Conference on Machine Learning(ACML), held online during November 17 - 19, 2021*
- *Reliable Methods of Mathematical Modeling(RMMM), EPFL, June 21-23, 2022*
- *8th European Congress on Computational Methods in Applied Sciences and Engineering-ECCOMAS CONGRESS 2022, 5-9 June 2022, Oslo, Norway.*
- *SIAM Conference on Mathematics of Data Science (MDS22),September 26 - 30, 2022, Town and Country Resort, San Diego, California, U.S.*
- *International Conference on Advances in Differential Equations and Numerical Analysis (ADENA2020), held online October 12-15, 2020 in Indian Institute of Technology Guwahati India.*

Thank You