Solving partial differential equations with TorchPhysics – real and artificial intelligence for science and engineering

Uwe Iben

Robert Bosch GmbH

14.11. -- 15.11.2024; Berlin

Real intelligence is needed to make artificial intelligence work

(Wil Schilders)

BOSCH

Agenda:

- Motivation and Introduction
- History of TorchPhysics
- Forward and inverse problems
- PINN and TorchPhysics
- Academic test cases and results
- Discussion on pro and limits of PINNs
- Conclusion

Solving PDEs – real and artificial intelligence for science and engineering

Motivation – our daily tasks and demands

Products

Functionalities

Electric torque and power

Mass flow

Lifetime prediction

Acoustics

Models

Algebraic equations

Ordinary differential equations

Partial differential equations

Solvers

FEM

FVM

BEM

Multi-scale problems with parameters

High computational effort

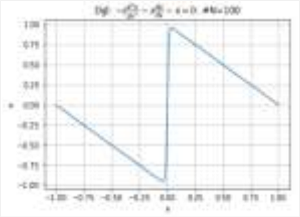
Forward problems

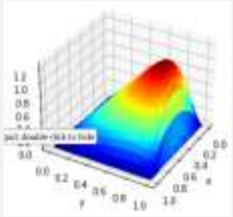
cause - effect

Multi-scale problems:

- Spatial dimensions ~ 10^{-6} … 10^{-1}
- Physical quantities ~ 10^{-6}…. 10^{8}
- Terms ~ 10^{-14} … 10^{14}

- Can lead to strong gradients
- Adapted methods are needed
- Reformulation, scaling





BOSCH

Solving PDEs – real and artificial intelligence for science and engineering

History

- **Parametric PDEs in industry**:
  - Heat transport equations
  - Flow equations
  - Electromagnetic equations
  - Mechanical equations
- **Demand**:
  - Fast and reliable solvers
  - Speed up of 10 and greater with respect to existing solvers (commercial or OpenSource)
  - Universal approach
  - Easy to use with high automatization potential
- **Possible candidates for surrogate models**:
  - Physics Informed Neural Networks
  - Kernel methods
  - Black box neural networks
- **How to do a proof of concept?**
  - Joint work with Uni Bremen
  - Student work
  - Development of an OpenSource tool TorchPhysics with training documents

Motivation and Introduction
History of TorchPhysics

- **2020:**
  - Question: Are there any AI methods available to solve inverse problems in industry?
  - Scouting and study of different approaches
- **05/2021:**
  - Student work of Tom Heilenkötter and Nick Freudenberg (Uni Bremen) – implementation of the PINN approach
  - OpenSource library TorchPhysics
- **04/2022:**
  - Running library
  - Analysis of academic test cases

Solving PDEs – real and artificial intelligence for science and engineering

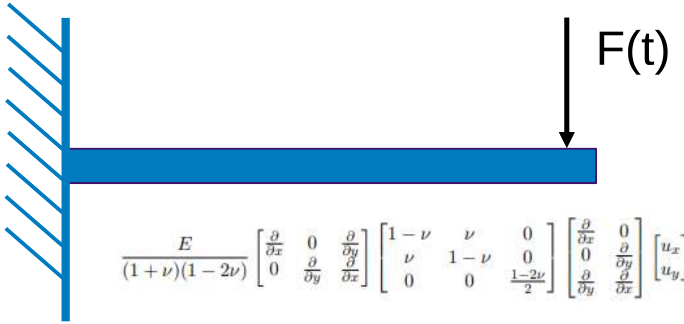
Motivation – forward and inverse problems

Forward problem:
cause -> effect
Inverse problem:
effect -> cause

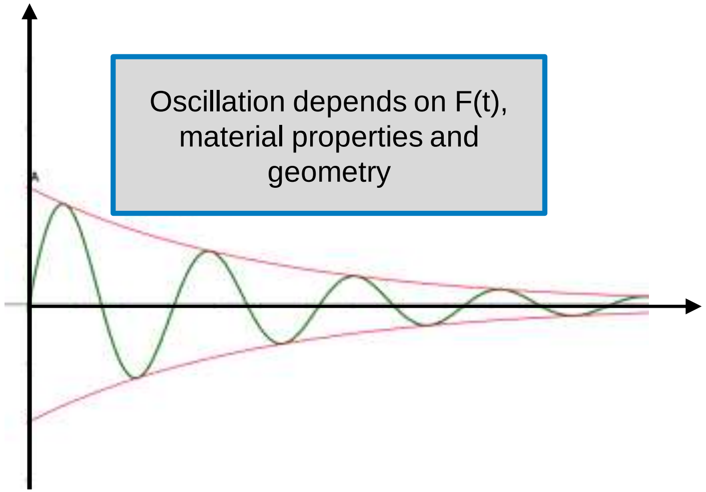
- Problem formulation: beam vibration

well-posed, unique solution, parameter with uncertainties

Forward problem



$$\frac{E}{(1+\nu)(1-2\nu)}\begin{bmatrix}\frac{\partial}{\partial x} & 0 & \frac{\partial}{\partial y}\\ 0 & \frac{\partial}{\partial y} & \frac{\partial}{\partial x}\end{bmatrix}\begin{bmatrix}1-\nu & \nu & 0\\ \nu & 1-\nu & 0\\ 0 & 0 & \frac{1-2\nu}{2}\end{bmatrix}\begin{bmatrix}\frac{\partial}{\partial x} & 0\\ 0 & \frac{\partial}{\partial y}\\ \frac{\partial}{\partial y} & \frac{\partial}{\partial x}\end{bmatrix}\begin{bmatrix}u_x\\ u_y\end{bmatrix}+\rho\begin{bmatrix}b_x\\ b_y\end{bmatrix}=\rho\begin{bmatrix}\ddot{u}_x\\ \ddot{u}_y\end{bmatrix}$$



Oscillation depends on F(t), material properties and geometry

Parameters, initial values, etc.

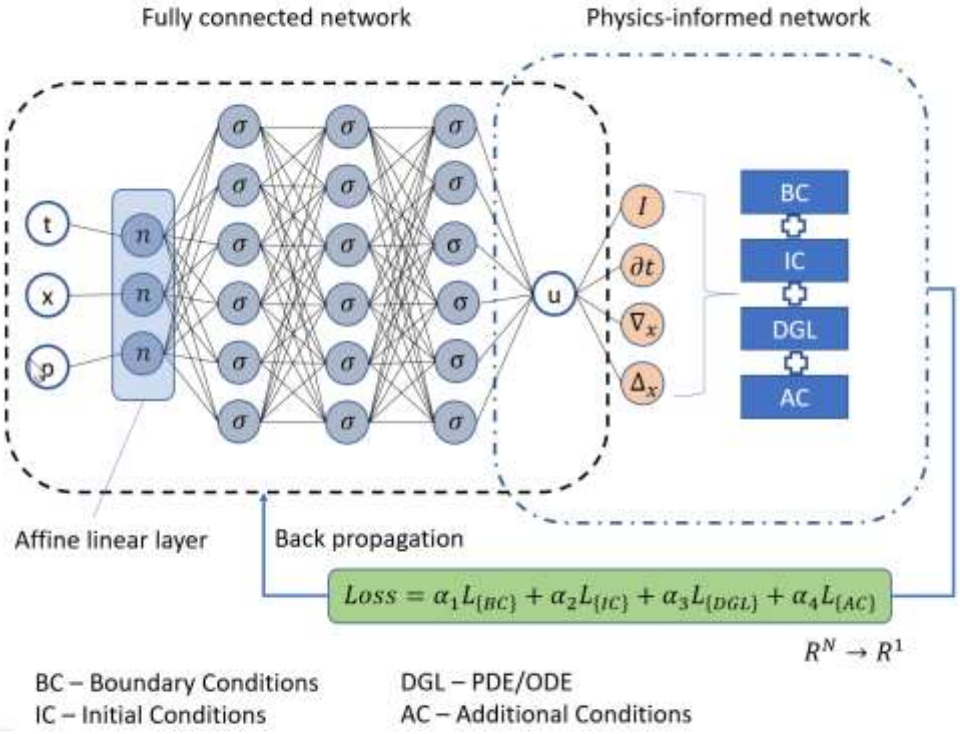
Inverse problem

Ill-posed, not unique solution, data with noise

Solving PDEs – real and artificial intelligence for science and engineering

How to speed up numerical computations for PDEs?

- Physics Informed Neural Networks
- **Idea:** Using a NN as a function approximator and including data and physics in the training procedure
- **Implementation** of this idea in an OpenSource software in a cooperation the University of Bremen (prof. Peter Maass) called [TorchPhysics]
- Framework can be used for
  - Only meshless solver for PDEs and ODEs
  - Hybrid solver (with data and physics)
  - Only with data (black box)
- Many functionalities: Deep-O-Net, Hidden Physics,….



BOSCH

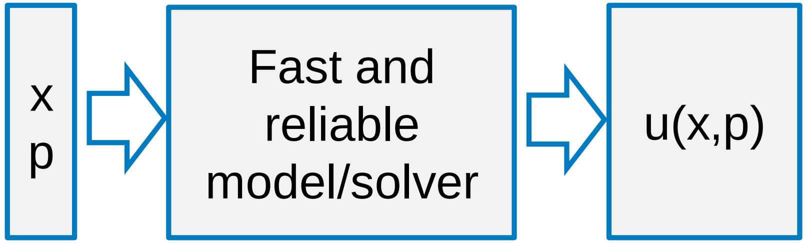
Solving PDEs – real and artificial intelligence for science and engineering

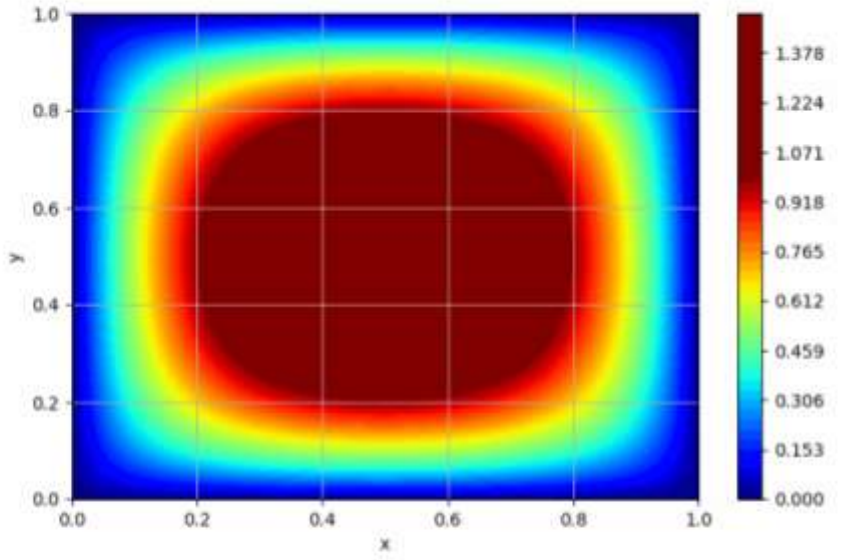
How to speed up numerical computations for PDEs?

- Parametric PDEs are often given, e.g. fluid flow problems, corrosion problems
- Example:

$$-\epsilon\Delta u - (b_1, b_2)\cdot \nabla(u) = 2 \quad x \in \Omega = [0,1]^2$$
$$u = 0 \quad \text{on} \quad \partial\Omega$$

  - transport coefficient: $\vec{b} \in [-1,1] \times [-1,1]$
  - Goal: find an approximation for $u(\vec{x}, \vec{b})$
  - Several methods are available for this task (list is not complete):
    - Kernel methods (data driven only)
    - PINNs (physics and data driven)
    - DL-MOR (as used for ODEs)





Solving PDEs – real and artificial intelligence for science and engineering

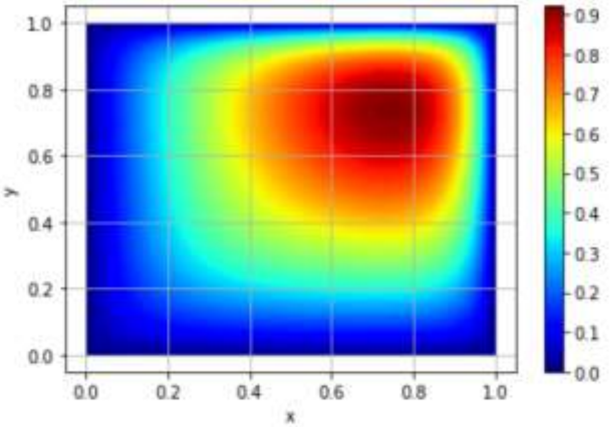
How to start?

- Some remarks to the transport-diffusion equation

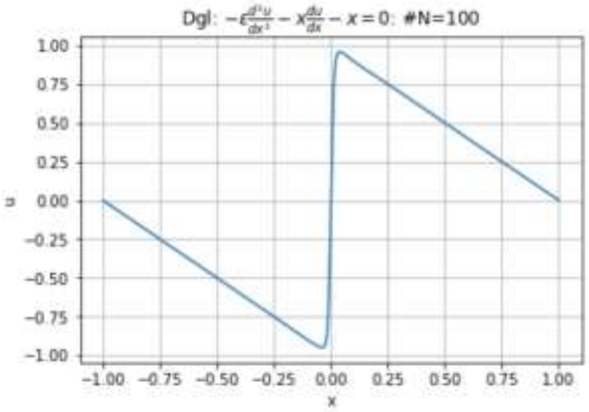
$$-\Delta u - \frac{(b_1, b_2)}{\epsilon} \cdot \nabla u = \frac{2}{\epsilon}$$

$$\lim_{\epsilon \to 0} | \nabla u | \to \infty$$

- Large gradient in one corner or at one boundary



D = [1 1]

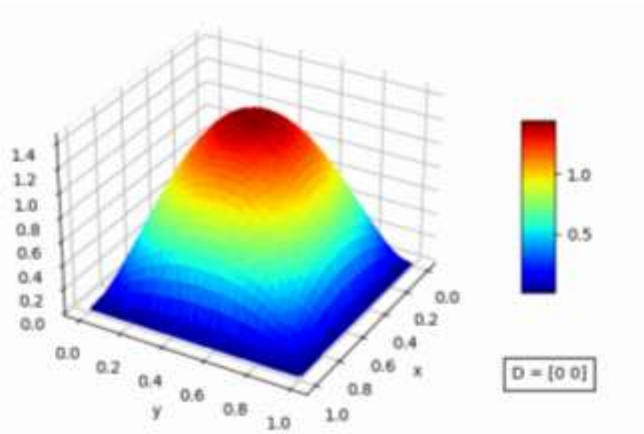


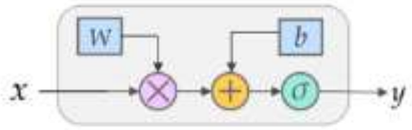
BOSCH

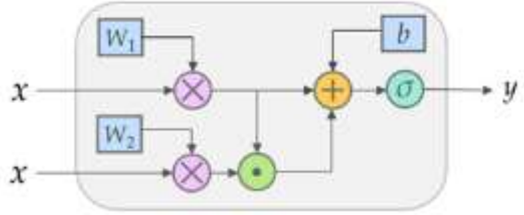
Solving PDEs – real and artificial intelligence for science and engineering

Academic test cases and results

- How to implement the PDE in TorchPhysics?
  - Definition of variables (x, b, u)
  - Definition of domain Ω
  - Definition of boundary $\partial\Omega$
  - Definition of PDE
  - Definition of NN: FCN or QRES
  - Definition of training procedure







- **Notebooks:**
  - SGR_HC_NN4
  - SGR_HC_NN6
  - SGR_HC_derivative_adapt_NN3
  - SGR_HC_derivative_adapt_NN6
  - SGR_HC_derivative_adapt_2_NN3
  - SGR_HC_derivative_adapt_2_NN6

Solving PDEs – real and artificial intelligence for science and engineering

Academic test cases and results

- Implementation: $$-\epsilon\Delta u - (b_1, b_2)\cdot\nabla(u) = 2 \quad x \in \Omega = [0,1]^2$$ $$u = 0 \quad \text{on} \quad \partial\Omega$$

$$\begin{aligned} u_t + N_x(u) &= g(x) \quad \text{on} \quad x \in \Omega,\ t \in [0,T] \\ u(t=0,x) &= IC(x) \\ u(x,t) &= BC(t,x) \quad \text{on} \quad x \in \Gamma,\ t \in [0,T] \end{aligned}$$

$$L_{pde} = \frac{1}{N_\Omega}\sum_{i=1}^{N_\Omega} |\, u_t(x_i,t_i) - N_x(u(x_i,t_i)) - g(x_i,t_i)\,|^2$$

$$L_{IC} = \frac{1}{N_{IC}}\sum_{j=1}^{N_{IC}} |\, u(0,x_j) - IC(x_j)\,|^2$$

$$L_{BC} = \frac{1}{N_{BC}}\sum_{k=1}^{N_{BC}} |\, u(t_k,x_k) - BC(x_k,t_k)\,|^2$$

```python
X = tp.spaces.R1('x')
Y = tp.spaces.R1('y')
D = tp.spaces.R2('D')
U = tp.spaces.R1('u')
```

$D = (b_1, b_2)$

```python
def pde_residual(u, x, y, D):
    u = constrain_fn(u,x,y)
    u_grad = tp.utils.grad(u, x, y)
    conv_term = torch.sum(D*tp.utils.grad(u, x, y), dim=1, keepdim=True)
    lap = tp.utils.laplacian(u, x, y, grad=u_grad)
    return -eps*lap + conv_term - 2  # dim

pde_condition = tp.conditions.PINNCondition(module=model,
                                            sampler=inner_sampler,
                                            residual_fn=pde_residual,
                                            name='pde_condition')
```

Simple notations and many predefined functions

Solving PDEs – real and artificial intelligence for science and engineering

Academic test cases and results

- Different implementations to get the best results
  - Small loss function
  - Fast training
  - Reliable results
- Options:
  - One NN for approximation of u
  - Splitting of Δu and one NN
  - Splitting of Δu and two NN

- Splitting of Δu:

$$\sigma = -\epsilon^{\frac{1}{2}} \nabla u$$

$$\sigma + \epsilon^{\frac{1}{2}} \nabla u = 0$$

$$E(u) \leq E(v) = ||\sigma + \epsilon^{\frac{1}{2}} \nabla v||^2 + ||\epsilon^{\frac{1}{2}} \nabla \cdot \sigma - D \cdot \nabla v - R||^2$$

$$\min_v E(v)$$

```python
# a entspricht sigm aus den Aufzeichnungen
# Hier wird das Resiuum der PDE gelernt
def pde_residual(u, a, x, y, D):
    u = constrain_fn(u,x,y)
    conv_term = torch.sum(D*tp.utils.grad(u, x, y), dim=1, keepdim=True)
    lap = tp.utils.div(a, x, y)
    return (eps**0.5)*lap - conv_term + R

pde_condition = tp.conditions.PINNCondition(module=model,
                                             sampler=inner_sampler,
                                             residual_fn=pde_residual,
                                             name='pde_condition')
# Hier wird der Wert von sigma gelernt, was dem gradienten von u mit
# dem Vorfaktor -\sqrt{\epsilon} entspricht
def pde2_residual(u, a, x, y, D):
    u = constrain_fn(u,x,y)
    return a + (eps**0.5)*tp.utils.grad(u,x,y)

pde2_condition = tp.conditions.PINNCondition(module=model,
                                              sampler=inner_sampler,
                                              residual_fn=pde2_residual,
                                              name='pde2_condition')
```

Solving PDEs – real and artificial intelligence for science and engineering

Academic test cases and results

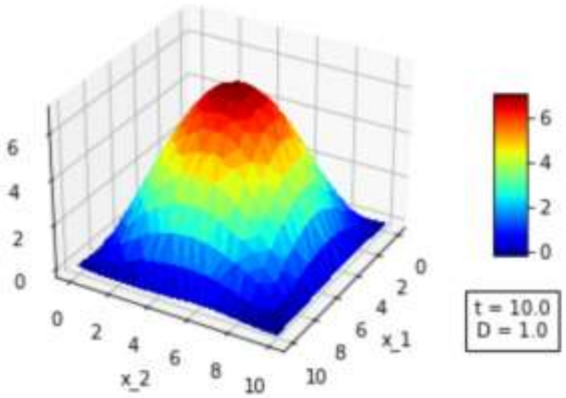
- Training with data
  - Heat diffusion equation
  - Data generation with FD solver for different parameter D
- Notebook:
  - Notebooks-Training-Heidelberg/Heat-equation/heat-equation.ipynb

```python
def heat_residual(u, x, t, D):
    return D*tp.utils.laplacian(u, x) - tp.utils.grad(u, t)

pde_condition = tp.conditions.PINNCondition(module=model,
                                            sampler=inner_sampler,
                                            residual_fn=heat_residual,
                                            name='pde_condition')
```

```python
def boundary_v_residual(u):
    return u

boundary_v_condition = tp.conditions.PINNCondition(module=model,
                                                   sampler=boundary_v_sampler,
                                                   residual_fn=boundary_v_residual,
                                                   name='boundary_condition')
```

```python
def f(x):
    return T_max*torch.sin(math.pi/w*x[:, :1])*torch.sin(math.pi/h*x[:,1:])

def initial_v_residual(u, f):
    return u-f

initial_v_condition = tp.conditions.PINNCondition(module=model,
                                                  sampler=initial_v_sampler,
                                                  residual_fn=initial_v_residual,
                                                  data_functions={'f': f},
                                                  name='initial_condition')
```

```python
optim = tp.solver.OptimizerSetting(torch.optim.Adam, lr=1e-4) #SGD, LBFGS
if data:
    solver = tp.solver.Solver([pde_condition,
                               boundary_v_condition,
                               initial_v_condition], [val_condition], optimizer_setting = optim)
else:
    solver = tp.solver.Solver([pde_condition,
                               boundary_v_condition,
                               initial_v_condition],  optimizer_setting = optim)
```



Academic test cases and results

- Deep-O-Net – learning of right-hand site of ODEs/PDEs
- Example: ODE with different activations (right-hand site)
  - $u' = f(t)\ \ with\ \ u(0) = 0\ \ and\ f \in P(t)$
  - P(t) polynomial space
- Goal:
  - Learning of solution u(t) for elements of P(t)
- Notebook: ode.ipynb

```python
# Spaces
T = tp.spaces.R1('t') # input variable
U = tp.spaces.R1('u') # output variable
K = tp.spaces.R1('k') # parameter
F = tp.spaces.R1('f') # function output space name
# Domains
T_int = tp.domains.Interval(T, 0, 1)
K_int = tp.domains.Interval(K, 0, 6) # Parameters will be scalar values
```

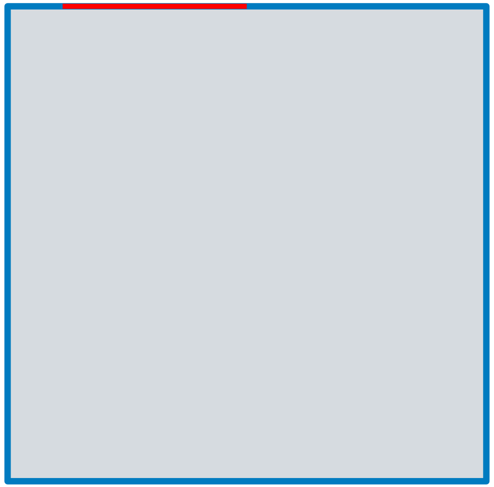
```python
# Defining function set
Fn_space = tp.spaces.FunctionSpace(T_int, F)

def f0(k, t):
    return k

def f1(k, t):
    return k*t

def f2(k, t):
    return k*t**2

def f3(k, t):
     return k*t**3

#def f4(k, t):
#    return k*torch.cos(k*t)

param_sampler = tp.samplers.RandomUniformSampler(K_int, n_points=40)
Fn_set_0 = tp.domains.CustomFunctionSet(Fn_space, param_sampler, f0)
Fn_set_1 = tp.domains.CustomFunctionSet(Fn_space, param_sampler, f1)
Fn_set_2 = tp.domains.CustomFunctionSet(Fn_space, param_sampler, f2)
Fn_set_3 = tp.domains.CustomFunctionSet(Fn_space, param_sampler, f3)
Fn_set = Fn_set_0 + Fn_set_1 + Fn_set_2 + Fn_set_3
```


BOSCH

Solving PDEs – real and artificial intelligence for science and engineering

PINN and UQ

- Models include parameters that are not known exactly – include some uncertainties
- Study of the influence of these parameters
- Model problem:
  - 2D heat transfer problem
  - Closed room with a local heater
  - Initial temperature T_0
  - Heat conduction coefficient d



$$
\begin{cases}
\frac{\partial}{\partial t} u(x,t) & = d\Delta_x u(x,t) & \text{on } \Omega \times I, \\
u(x,t) & = u^0 & \text{on } \Omega \times \{0\}, \\
u(x,t) & = h(t) & \text{at } \partial\Omega_{heater} \times I, \\
\nabla_x u(x,t) \cdot \vec{n}(x) & = 0 & \text{at } (\partial\Omega \setminus \partial\Omega_{heater}) \times I.
\end{cases}
$$

Solving PDEs – real and artificial intelligence for science and engineering

PINN and UQ

- PCE – Polynomial Chaos Expension

$$\xi = (\xi_1, \xi_2) \qquad u = u(x,t,\xi) = \sum_{i=0}^{P} u_i(x,t)\psi_i(\xi)$$

$$d = d(\xi_1) = d_0 + d_1\xi_1, \quad \xi_1 \sim \mathcal{U}(-1,1)$$

$$u^0 = u^0(\xi_2) = u_{00} + u_{01}\xi_2, \quad \xi_2 \sim \mathcal{U}(-1,1)$$

$$\begin{cases} \frac{\partial}{\partial t}u(x,t) & = d\Delta_x u(x,t) & \text{on } \Omega \times I, \\ u(x,t) & = u^0 & \text{on } \Omega \times \{0\}, \\ u(x,t) & = h(t) & \text{at } \partial\Omega_{heater} \times I, \\ \nabla_x u(x,t) \cdot \vec{n}(x) & = 0 & \text{at } (\partial\Omega \setminus \partial\Omega_{heater}) \times I. \end{cases}$$

$$\begin{cases} \sum_{i=0}^{P} \frac{\partial}{\partial t}u_i(x,t)\psi_i(\xi) & = \sum_{i=0}^{P} d(\xi_1)\Delta_x u_i(x,t)\psi_i(\xi) & \text{on } \Omega \times I, \\ \sum_{i=0}^{P} u_i(x,t)\psi_i(\xi) & = u^0(\xi_2) & \text{on } \Omega \times \{0\}, \\ \sum_{i=0}^{P} u_i(x,t)\psi_i(\xi) & = h(t) & \text{at } \partial\Omega_{heater} \times I, \\ \sum_{i=0}^{P} \nabla_x u_i(x,t)\psi_i(\xi) \cdot \vec{n}(x) & = 0 & \text{at } (\partial\Omega \setminus \partial\Omega_{heater}) \times I. \end{cases}$$

Solving PDEs – real and artificial intelligence for science and engineering

PINN and UQ

- PCE – Polynomial Chaos Expension

$$
\begin{cases}
\sum_{i=0}^{P} \frac{\partial}{\partial t} u_i(x,t)\psi_i(\xi)\psi_k(\xi) & = \sum_{i=0}^{P} d(\xi_1)\Delta_x u_i(x,t)\psi_i(\xi)\psi_k(\xi) & \text{on } \Omega \times I, \\
\sum_{i=0}^{P} u_i(x,t)\psi_i(\xi)\psi_k(\xi) & = u^0(\xi_2)\psi_k(\xi) & \text{on } \Omega \times \{0\}, \\
\sum_{i=0}^{P} u_i(x,t)\psi_i(\xi)\psi_k(\xi) & = h(t)\psi_k(\xi) & \text{at } \partial\Omega_{heater} \times I, \\
\sum_{i=0}^{P} \nabla_x u_i(x,t)\psi_i(\xi)\psi_k(\xi) \cdot \vec{n}(x) & = 0 & \text{at } (\partial\Omega \setminus \partial\Omega_{heater}) \times I.
\end{cases}
$$

$$
\begin{cases}
\langle \psi_i, \psi_j \rangle & = \int_{-1}^{1} \int_{-1}^{1} \psi_i(\xi)\psi_j(\xi)d\xi = \delta_{ij} \\
\langle \psi_i, \psi_j, \psi_k \rangle & := c_{ijk}
\end{cases}
$$

Solving PDEs – real and artificial intelligence for science and engineering

PINN and UQ

- PCE – Polynomial Chaos Expension

P=1

$$C = \begin{pmatrix} 0 & 2/3 & 0 \\ 2/3 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

$$\begin{cases} \frac{\partial}{\partial t} u_k(x,t) & = d_0 \Delta_x u_k(x,t) + \sum_{i=0}^{P} d_1 \Delta_x u_i(x,t) c_{1ik} & \text{on } \Omega \times I, \\ u_k(x,t) & = u_{00}\delta_{0k} + u_{01}\delta_{2k} & \text{on } \Omega \times \{0\}, \\ u_k(x,t) & = h(t)\delta_{0k} & \text{at } \partial\Omega_{heater} \times I, \\ \nabla_x u_k(x,t) \cdot \vec{n}(x) & = 0 & \text{at } (\partial\Omega \setminus \partial\Omega_{heater}) \times I \end{cases}$$

$$\boxed{\begin{cases} \mathbb{E}(u)(x,t) & = u_0(x,t) \\ \sigma^2(u)(x,t) & = \sum_{i=1}^{P} u_i^2(x,t) \end{cases}}$$

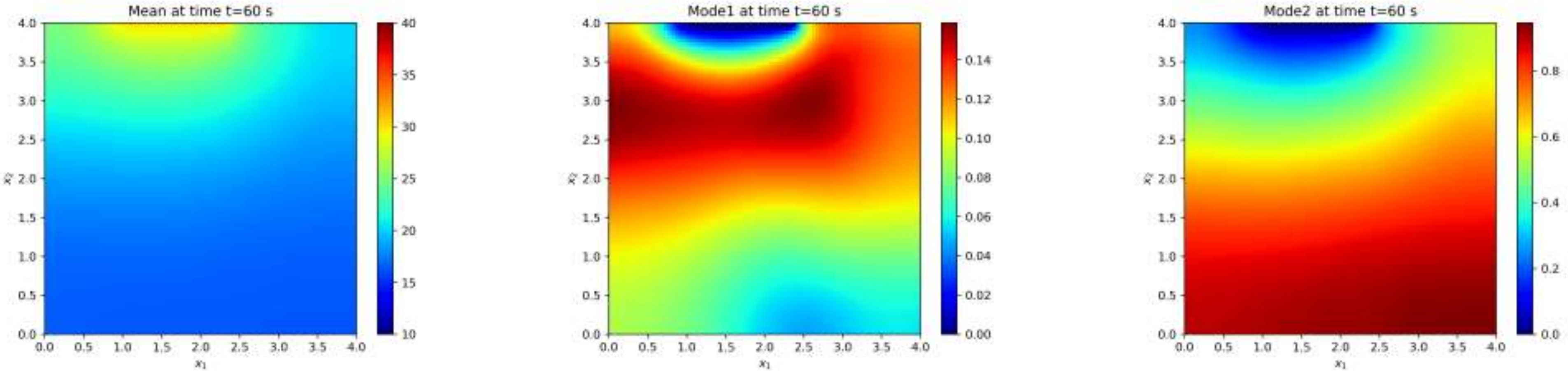
Solving PDEs – real and artificial intelligence for science and engineering

PINN and UQ

Three sub-models with
#Spatial discretication: 3600
#Time discretication: 120
= 432.000 sampling points
Training time = 1h 20 min

- Notebook:

$$d = d(\xi_1) = d_0 + d_1\xi_1, \quad \xi_1 \sim \mathcal{U}(-1,1)$$
$$u^0 = u^0(\xi_2) = u_{00} + u_{01}\xi_2, \quad \xi_2 \sim \mathcal{U}(-1,1)$$



Solving PDEs – real and artificial intelligence for science and engineering

Summary

- Pro of PINNs in TorchPhysics:
  - Very flexible meshless method for numerical solution of ODEs, PDEs
  - Surrogate solver for repeated usage
  - Powerful tool with simple natation
  - Fast implementation of additional functionalities

- Cons of PINNs:
  - Training can be time-consuming
  - Multi-scale problems can lead to a time-consuming pre-definition of the models
  - Generation of data can be a time-consuming additional procedure
  - Flexibility of PINN approach can be a challenge to find the optimal procedure

Thank you for your attention and successful solution of problems using TorchPhysics