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

Real intelligence is needed to make artificial intelligence work

(Wil Schilders)

Uwe Iben

Robert Bosch GmbH

7.11-8.11.2023; Heidelberg

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

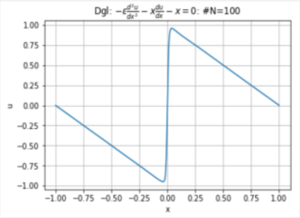
High computational effort

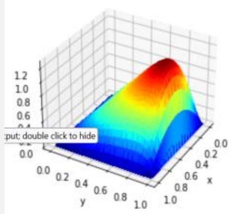
Forward problems — cause - effect

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





Intern | CR/AME CE-AM | 24.10.2023

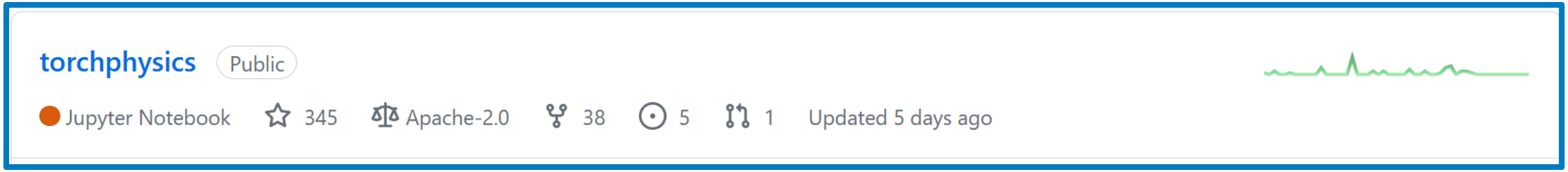
Solving PDEs – real and artificial intelligence for science and engineering

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 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



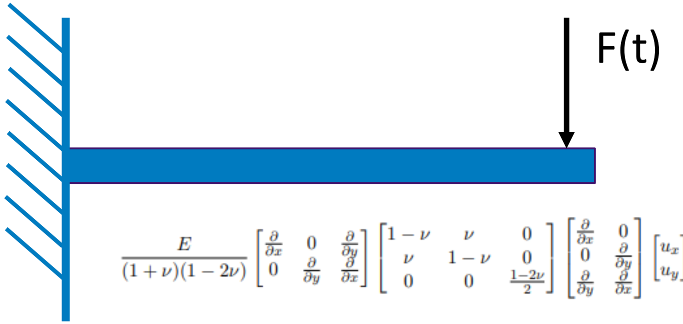
Motivation – forward and inverse problems

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

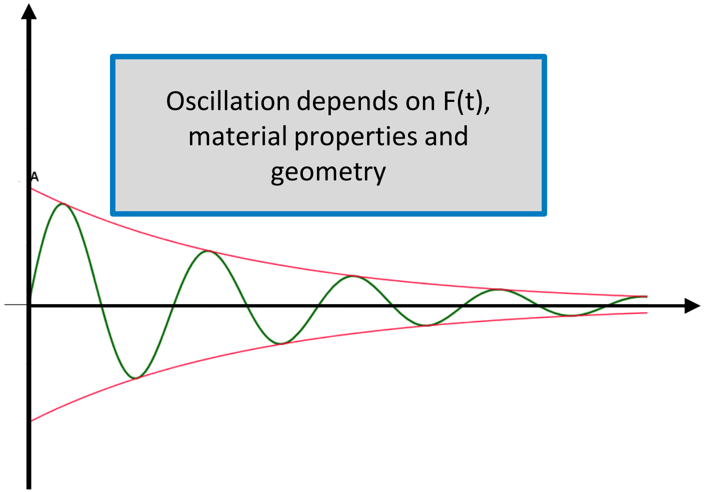
- Problem formulation: beam vibration

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.

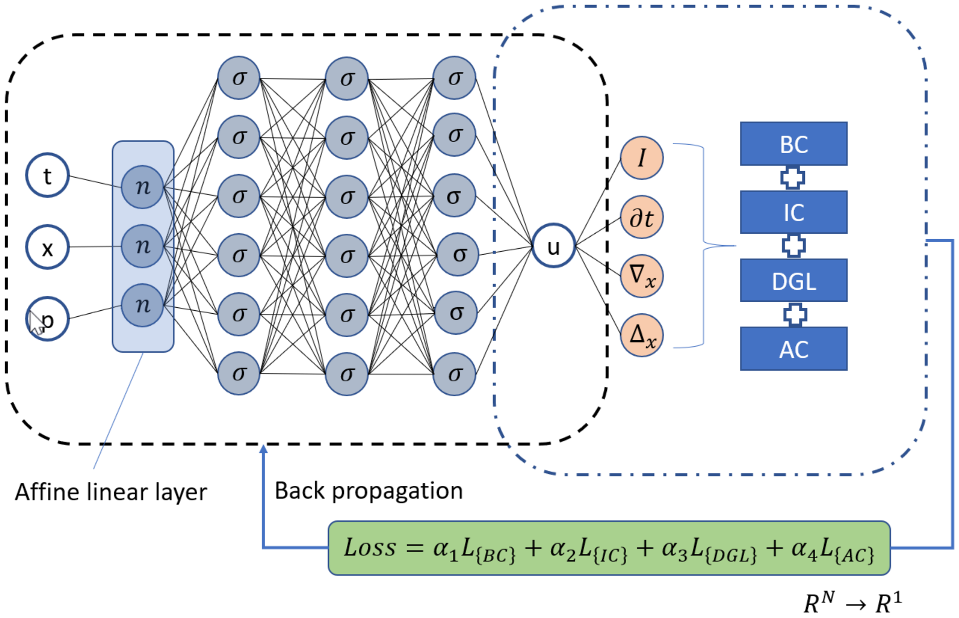
Inverse problem

Ill-posed, not unique solution, data with noise

BOSCH

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,….



BC – Boundary Conditions
IC – Initial Conditions
DGL – PDE/ODE
AC – Additional Conditions

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: $\vec{b}\in[-1,1]\times[-1,1]$

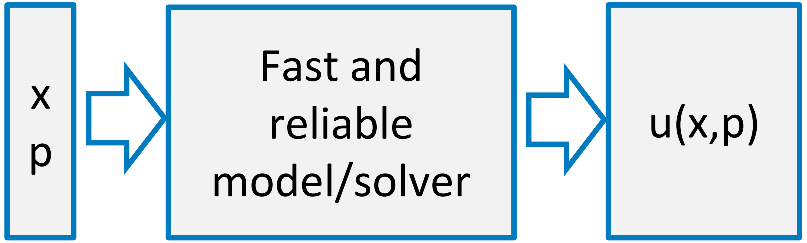
  – Goal: find an approximation for $u(\vec{x},\vec{b})$

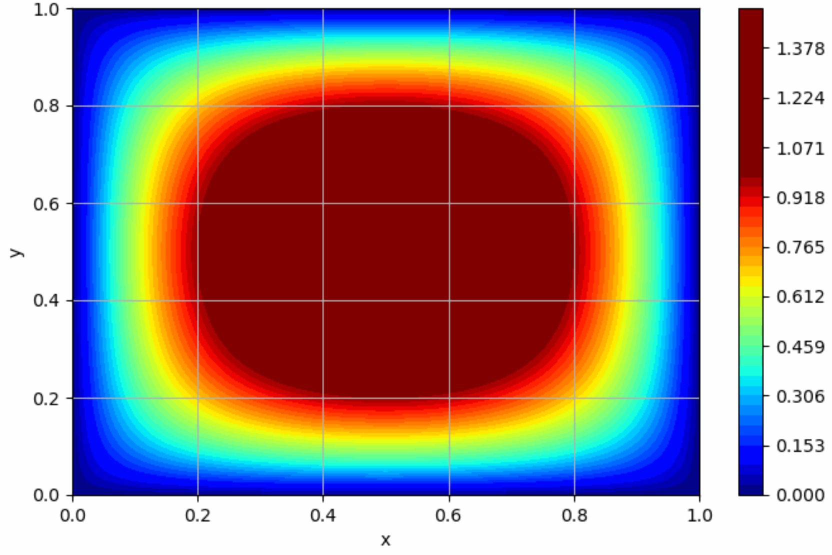
  – Several methods are available for this task (list is not complete):

    – Kernel methods (data driven only)

    – PINNs (physics and data driven)

    – DL-MOR (as used for ODEs)

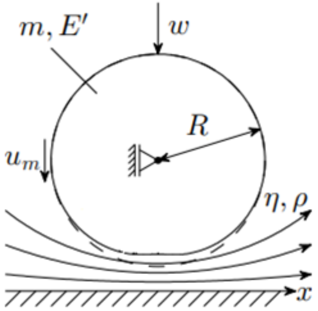




Solving PDEs – real and artificial intelligence for science and engineering

How to start?

- Given PDE problem – how to start with PINN approach?
- Nothing works automatically!!!
- First step:
  - Analysis of terms
  - Identification of scales, e.g. $p, \mu, h\ , u_m$
  - $p \in [10^{-2}, 10^{8}]$
  - $\mu \in [10^{-6}, 10^{-5}]$
  - $h \in [10^{-7}, 10^{-5}]$
  - Introduction of scaling factors, e.g. p_{skal}
  - Re-organization of terms



$$u_m \frac{\partial}{\partial x}(\rho h) - \frac{\partial}{\partial x}\left(\frac{\rho h^3}{12\eta}\frac{\partial p}{\partial x}\right) = 0 \quad \text{for } x \in (-4a, 2a)$$

$$a = \sqrt{\frac{8wR}{E'\pi}}$$

- Second step:
  - Definition of parameters and variables -> defines our complete solution space

BOSCH

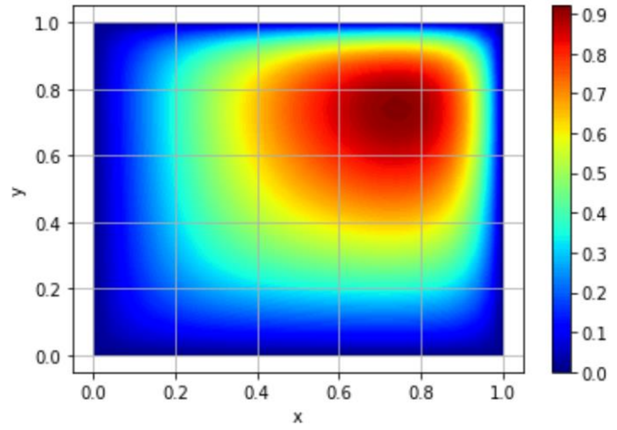
How to start?

- Some remarks to the transport-diffusion equation

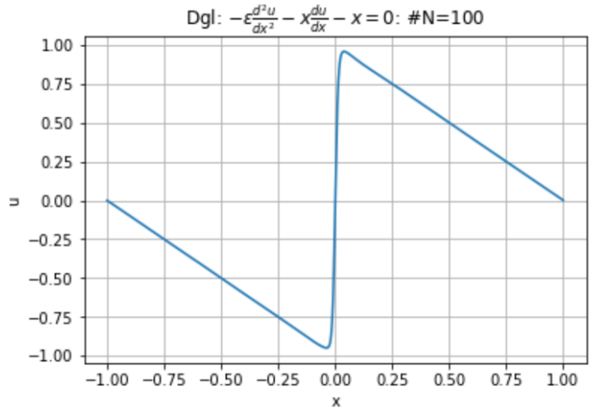
$$-\Delta u - \frac{(b_1, b_2)}{\epsilon} \cdot \nabla u = \frac{2}{\epsilon}$$

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

- Large gradient in one corner or at one boundary

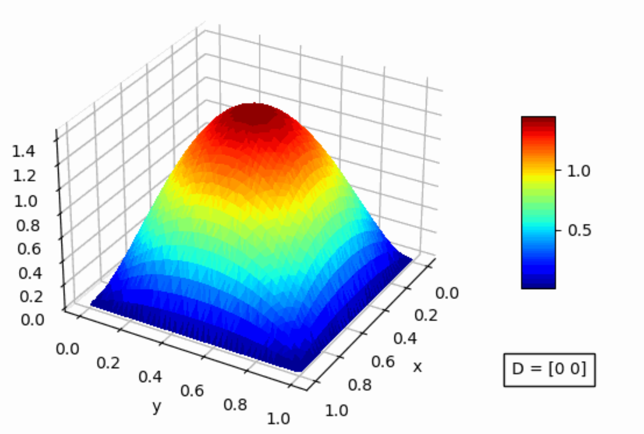


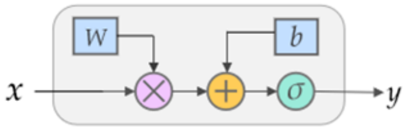
D = [1 1]

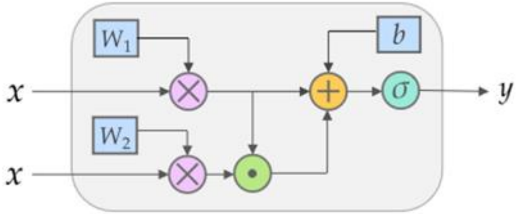


Academic test cases and results

- 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

Academic test cases and results

- Implementation:

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

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

$$L_{pde} = \frac{1}{N_\Omega}\sum_{i=1}^{N_\Omega} |\, u_t(x_i,t_i) - N_x(u(x_i,t_i)) - g(x_i,t_i)\,|^2$$
$$L_{IC} = \frac{1}{N_{IC}}\sum_{j=1}^{N_{IC}} |\, u(0,x_j) - IC(x_j)\,|^2$$
$$L_{BC} = \frac{1}{N_{BC}}\sum_{k=1}^{N_{BC}} |\, u(t_k,x_k) - BC(x_k,t_k)\,|^2$$

```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

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:

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

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

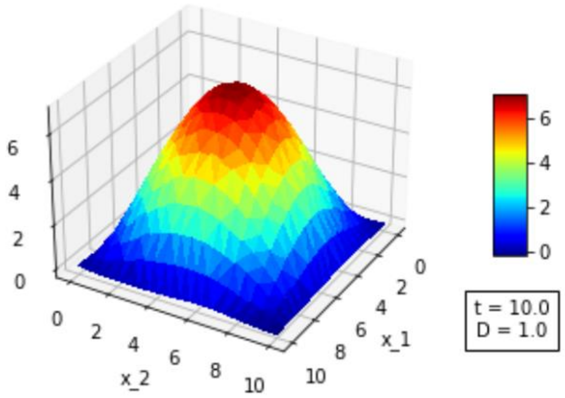
$$E(u) \leq E(v) = ||\sigma + \epsilon^{\frac{1}{2}} \nabla v||^2 + ||\epsilon^{\frac{1}{2}} \nabla \cdot \sigma - D \cdot \nabla v - R||^2$$

$$\min_v \; E(v)$$

```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')
```

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




```
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')
```

```
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')
```

```
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')
```

```
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

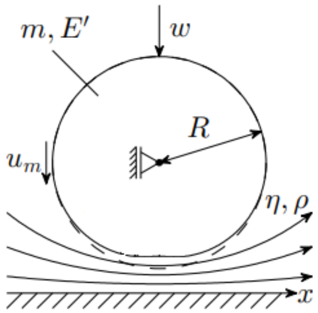
```python
# Spaces
T = tp.spaces.R1('t') # input variable
U = tp.spaces.R1('u') # output variable
K = tp.spaces.R1('k') # parameter
F = tp.spaces.R1('f') # function output space name
# Domains
T_int = tp.domains.Interval(T, 0, 1)
K_int = tp.domains.Interval(K, 0, 6) # Parameters will be scalar values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

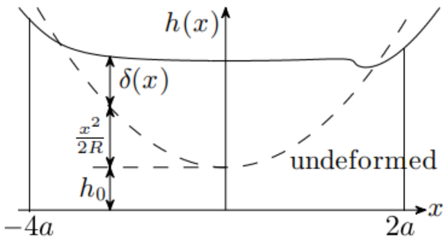
```python
# Defining function set
Fn_space = tp.spaces.FunctionSpace(T_int, F)

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
```

Industrial applications of PINNs

Tribology: **E**lasto**H**ydro**D**ynamic contact (I)

Roll-plate contact – thin gap filled with fluid - deformation due to high pressure





Gap size function with deformation

Reynolds equation

$$u_m \frac{\partial}{\partial x}(\rho h) - \frac{\partial}{\partial x}\left(\frac{\rho h^3}{12\eta}\frac{\partial p}{\partial x}\right) = 0 \quad \text{for } x \in (-4a, 2a)$$

with special challenging features:

- different orders of magnitude (x: 10^{-5} vs. p: 10^{9})
- cavitation (nonnegative pressure)
- highly non-linear (viscosity, density, deformation)
- integro-differential (gap size with deformation)

$$h(x) = h_0 + \frac{x^2}{2R} - \frac{4}{\pi E_{red}} \int_{-4a}^{2a} p(x') \ln|x - x'| dx'$$

Forward problem:

Given h_0 -> compute p

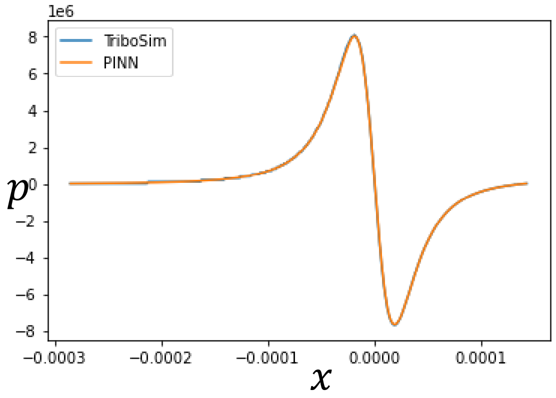
Inverse problem:

Compute h_0, p fulfilling

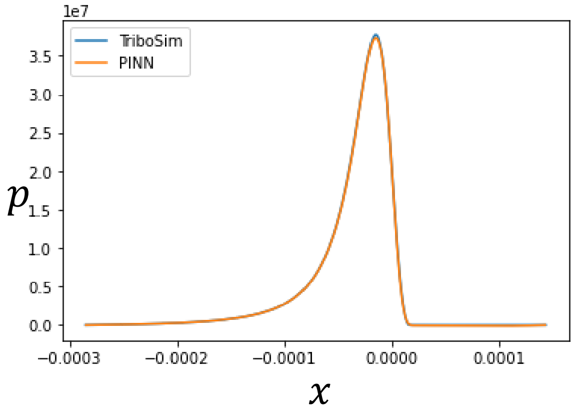
$$w = \int_{-4a}^{2a} p(x') dx'$$

Stepwise solution of forward problem

(i) Solution with constant viscosity, w/o cavitation, w/o deformation



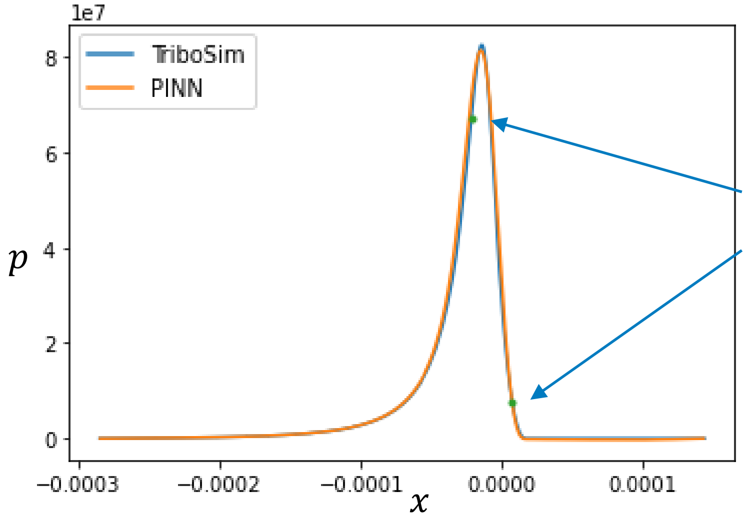
(ii) Solution with constant viscosity & cavitation, w/o deformation



Industrial applications of PINNs

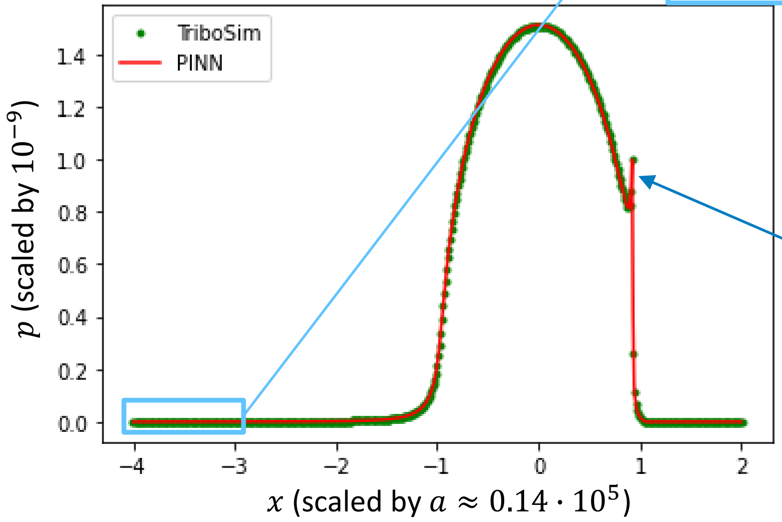
Tribology: **E**lasto**H**ydro**D**ynamic contact (II)

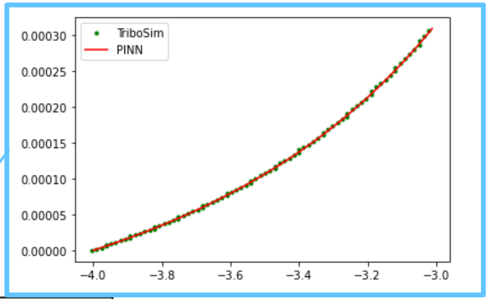
(iii) Solution with nonlinear viscosity & cavitation, w/o deformation



With additionally learned data points

(iv) Solution with nonlinear viscosity, cavitation & deformation





With many additionally learned data points (including solution peak)

PINN-method fails for forward problem due to high nonlinearities:

- PINN can only be well trained with additional data of reference solution (peak of solution has to be represented)
- Parametric PINN model for h_0 can not be learned not even with data (no convergence or bad accuracy)

BOSCH

Summary

Pick your poison

- 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 tasks using TorchPhysics