Machine Learning vs. Physics Based Modeling?

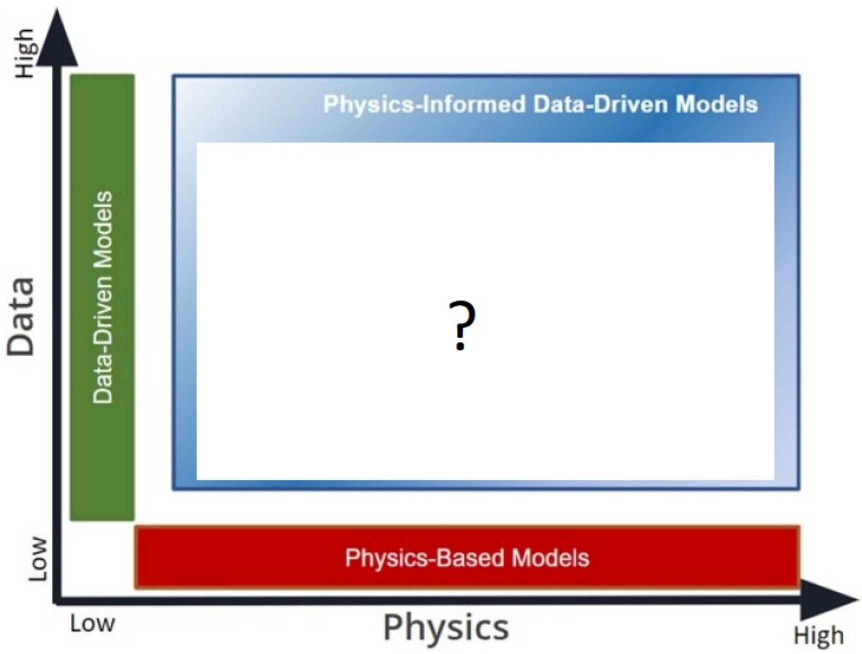
Talk Outline

- What is the problem with the ML?
- Relation between ML and PM
- Integrating ML and PM - Neural Network(NN)
- Automatic Differentiation of NN (Autograd)
- Solving Differential Equations via NN
- Physical Informed Neural Network (PINN)
- Problem with PINN
    - Curriculum Learning
    - Sequence to Sequence Learning
- Examples
    - Simple PINN Problem
    - Second Order Problem
    - Inverse Problem of PDF
    - Solving of DAE

1. Introduction

ML: Data-driven modeling- lot of data, no physics but smart algorithms

PM: Theory based modeling- solid theory, necessarily many neglections - unknown parameters



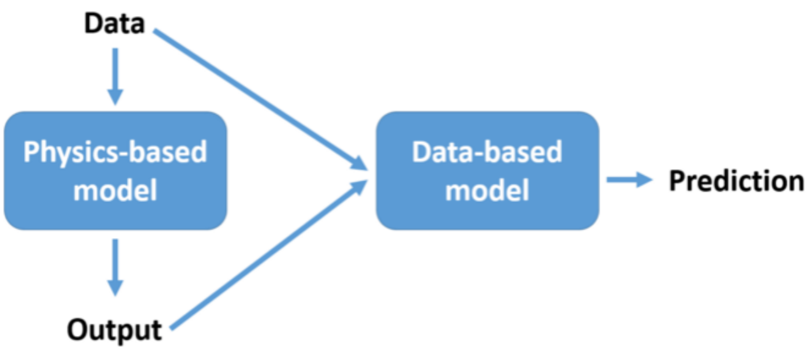
- ML can outperform PM
- PM needs parameter estimation
    - PM effectiveness can be extended

- Example: Solar Collector Modeling

---

2. ML assisted modeling

Integrating ML algorithms and physics-based simulations



It means that the best model is neither ML nor PM but the hybrid PINN model represented by a neural network including the knowledge of the measured data and the physical model simultaneously.

ML algorithms: Basically any universal approximator: Algebraic Polynomials, Radial Basis Function, Neural Networks

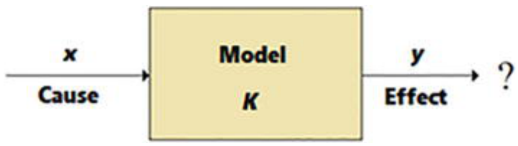
PM algorithms: Basically Differential Equations - Neural Networks provide meshless solution

Common platform: Neural Networks

Different type of Problems

Model is known inclusive its parameters: from input let us compute the output

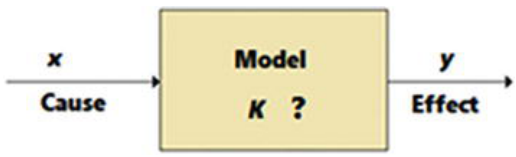
In[ ]:=



Direct problem (Physics Based Model)

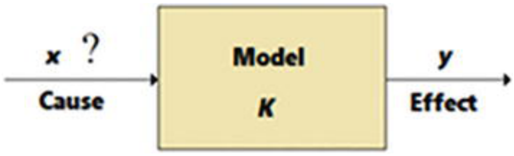
Model is known but its parameter(s) is/are unknown and in addition input and output known: find the model parameters

In[ ]:=



Identification inverse problem (parameter estimation)- Weak inverse problem(Data Based Model)

Model is known inclusive its parameters: from output let us compute the input



Causation Inverse problem (input estimation) - Strong inverse problem

3. Solving ODE via NN

In general the ODE,

$$\frac{dy(t)}{dt} = f(t, y(t)) \text{ with } y(t_0) = y_0$$

$$t \in [t_0, t_e]$$

its solution can be approximated $y(t) \approx \mathcal{N}(t)$

Example

$$\frac{d\mathcal{N}(t)}{dt} - (\exp(-t/5)\cos(t) - R\,\mathcal{N}(t)) \text{ with } y(0) = 0$$

$$t \in [0, 10]$$

It is easy to solve it even in symbolic way,

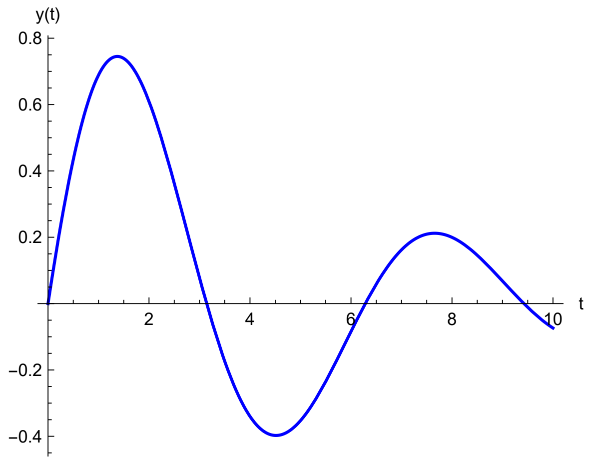
```
In[•]:= ClearAll["Global`*"]

In[•]:= soly = DSolve[{y'[t] == Exp[-t / 5] Cos[t] - R y[t], y[0] == 0}, y[t], t];

In[•]:= ysym = y[t] /. First[soly] // Simplify
```

Out[•]=

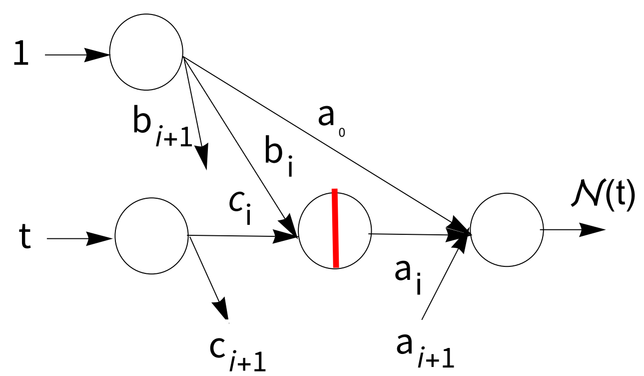
$$\frac{5\,e^{-R t}\left(1 - 5 R + e^{\left(-\frac{1}{5}+R\right) t}(-1 + 5 R)\,\text{Cos}[t] + 5\,e^{\left(-\frac{1}{5}+R\right) t}\,\text{Sin}[t]\right)}{26 - 10 R + 25 R^2}$$

In[∘]:=
```
p1 = Plot[ysym /. R → 0.2, {t, 0, 10}, AspectRatio → 0.8,
   ImageSize → 300, PlotStyle → Blue, AxesLabel → {"t", "y(t)"}]
```

Out[∘]=



Now, let us employ a simple NN,



where the activation function is sigmoid,

$$\sigma = \frac{1}{1 + e^{-x}}$$

the network using one hidden layer with *n* nodes,

$$\mathcal{N} = a_0 + \sum_{i=1}^{n} \frac{a_i}{1 + \exp(-(b_i + c_i\, t))}$$

In order to find the parameters a_i, b_i, c_i the function should be minimized the objective consisting of the model errors and the initial value error in the collocation points t_j

$$G(p) = \sum_j \left(\frac{\mathcal{N}(p, t_j)}{dt} - \exp(-t_j/5)\cos(t_j) + 0.2\,\mathcal{N}(p, t_j) \right)^2 + (\mathcal{N}(p, 0) - 0)^2$$

$$t_j \in [0, 10]$$

Let us consider four nodes

```
In[•]:= n = 4;

In[•]:= p = {a0, Table[{ai, bi, ci}, {i, 1, n}]} // Flatten
Out[•]=
```

$\{a_0, a_1, b_1, c_1, a_2, b_2, c_2, a_3, b_3, c_3, a_4, b_4, c_4\}$

In[•]:= $\mathcal{N} = a_0 + \sum_{i=1}^{n} \frac{a_i}{1 + \text{Exp}[-(b_i + c_i\, t)]}$

Out[•]=

$$a_0 + \frac{a_1}{1 + e^{-b_1 - t\, c_1}} + \frac{a_2}{1 + e^{-b_2 - t\, c_2}} + \frac{a_3}{1 + e^{-b_3 - t\, c_3}} + \frac{a_4}{1 + e^{-b_4 - t\, c_4}}$$

and

```
In[•]:= dN = D[N, t] // Simplify
Out[•]=
```

$$\frac{e^{b_1 + t\, c_1} a_1 c_1}{\left(1 + e^{b_1 + t\, c_1}\right)^2} + \frac{e^{b_2 + t\, c_2} a_2 c_2}{\left(1 + e^{b_2 + t\, c_2}\right)^2} + \frac{e^{b_3 + t\, c_3} a_3 c_3}{\left(1 + e^{b_3 + t\, c_3}\right)^2} + \frac{e^{b_4 + t\, c_4} a_4 c_4}{\left(1 + e^{b_4 + t\, c_4}\right)^2}$$

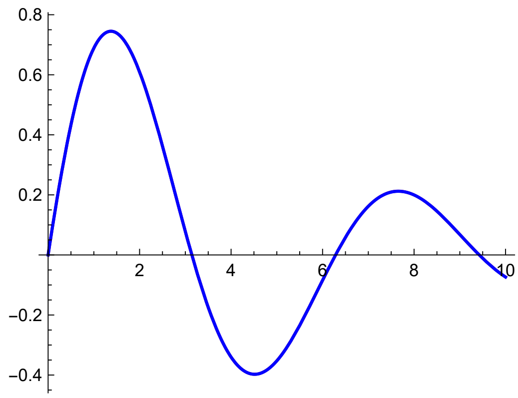
```
In[•]:= tcolloc = Table[i 0.5, {i, 0, 20}];

In[•]:= modelerr = Total[Map[(dN - (Exp[-t / 5] Cos[t] - 0.2 N) /. t → #1)^2 &, tcolloc]];

In[•]:= iniconderr = ((N /. t → 0) - 0)^2;

In[•]:= G = modelerr + iniconderr;

In[•]:= sol = NMinimize[G, p]
Out[•]=
```

$\{5.14591 \times 10^{-6}, \{a_0 \to -8.47006, a_1 \to 9.84616, b_1 \to 5.01709, c_1 \to -0.613147, a_2 \to -11.4885, b_2 \to -1.90641, c_2 \to 0.713029, a_3 \to 20.8162, b_3 \to -2.10444, c_3 \to 0.372928, a_4 \to -4.59165, b_4 \to -0.182518, c_4 \to -1.0631\}\}$

In[•]:=
```
myNet = 𝒩 /. sol⟦2⟧
```
Out[•]=

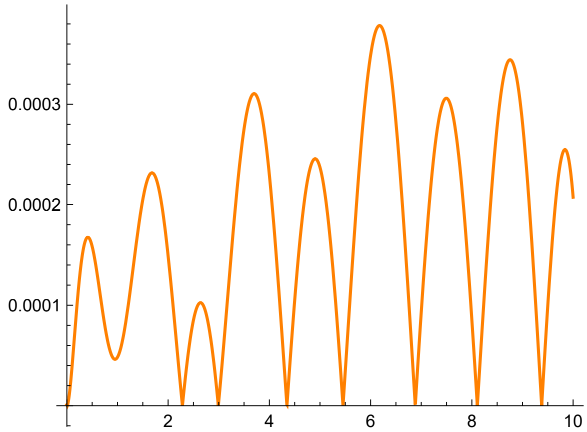
$$-8.47006-\frac{11.4885}{1+e^{1.90641-0.713029\,t}}+\frac{20.8162}{1+e^{2.10444-0.372928\,t}}+\frac{9.84616}{1+e^{-5.01709+0.613147\,t}}-\frac{4.59165}{1+e^{0.182518+1.0631\,t}}$$

In[•]:=
```
Show[Plot[myNet, {t, 0, 10}, PlotStyle → Purple, AspectRatio → 0.8], p1]
```
Out[•]=



Error of the approximation

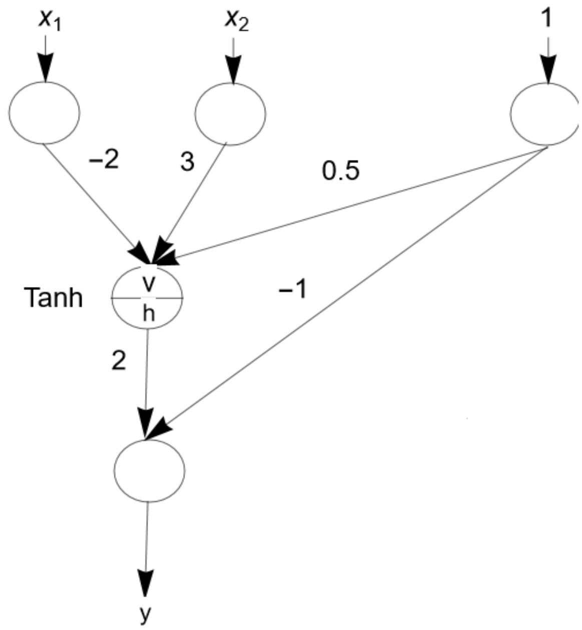
In[•]:=
```
perrM = Plot[Abs[(ysym /. R → 0.2) - myNet], {t, 0, 10.},
  AspectRatio → 0.8, ImageSize → 300, PlotRange → All, PlotStyle → Orange]
```
Out[•]=



4. Automatic Differentiation of Neural Net

In case of real network a new technique should be employed: Autograd -it is neither finite elements nor symbolic differentiation can be employed. So it has not truncation error and does not need cumbersome symbolic compotation.

Example



v = -2 x_1 + 3 x_2 + 0.5 1.
h = tanh(v)
y = 2 h - 1

Let us compute the $\frac{\partial y}{\partial x_1}$ and the $\frac{\partial y}{\partial x_2}$ at x_1=2 and x_2=1

Table I *Example of AD to compute the partial derivatives*

Forward pass	Backward pass
$x_1 = 2$ $x_2 = 1$	$\frac{\partial y}{\partial y} = 1$
$v = -2x_1 + 3x_2 + 0.5 = -0.5$ $h = \tanh v \approx -0.462$	$\frac{\partial y}{\partial h} = \frac{\partial(2h-1)}{\partial h} =$ $\frac{\partial y}{\partial v} = \frac{\partial y}{\partial h}\frac{\partial h}{\partial v} =$
$y = 2h - 1 = -1.924$	$\frac{\partial y}{\partial x_1} = \frac{\partial y}{\partial v}\frac{\partial v}{\partial x_1} =$ $\frac{\partial y}{\partial x_2} = \frac{\partial y}{\partial v}\frac{\partial v}{\partial x_2} =$

(https : // epubs . siam . org/doi/pdf/10.1137/19 M1274067)

In[•]:= D[Tanh[x], x]

Out[•]=

Sech[x]^2

Special differentiation technique Autograd is built in Python .

In[•]:= session = StartExternalSession["Python"]

Out[•]=

ExternalSessionObject[System: Python Version: 3.10.4 UUID: ae728a9b-e500-4f2b-bbfc-5334debb042f]

In[•]:=
```
import torch
import torch.nn as nn
import matplotlib.pyplot as plt
import numpy as np

## check if GPU is available and use it; otherwise use CPU
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
import matplotlib.pyplot as plt
import numpy as np
```

In[•]:=
```
N = nn.Sequential(nn.Linear(1, 100),nn.Tanh(), nn.Tanh(),nn.Tanh(), nn.Linear(100,1,
bias=False))
A = 0.
y_t = lambda t: A + t * N(t)
f = lambda t, y: torch.exp(-t / 5.0) * torch.cos(t) - y / 5.0
```

In[•]:=
```
def loss(t):

    t.requires_grad = True
    outputs = y_t(t)
    y_t_t = torch.autograd.grad(outputs, t,grad_outputs=torch.ones_like(outputs),
                        create_graph=True)[0]

    return  torch.mean( ( y_t_t - f(t, outputs) )** 2)
```

Out[•]=

ExternalFunction[System: Python Arguments: {t} Command: loss]

In[•]:=
```
optimizer = torch.optim.LBFGS(N.parameters())
```

In[•]:=
```
t = torch.Tensor(np.linspace(0, 10, 1000)[:, None])
def closure():

    optimizer.zero_grad()
    l = loss(t)
    l.backward()

    return l

for i in range(500):
    optimizer.step(closure)
```

In[•]:=
```
xx = np.linspace(0, 10, 1000)[:, None]
```

```
with torch.no_grad():
    yy = y_t(torch.Tensor(xx)).numpy()
yt = np.exp(-xx / 5.0) * np.sin(xx)

fig, ax = plt.subplots(dpi=100)
ax.plot(xx, yt, label='True')
ax.plot(xx, yy, '--', label='Neural network approximation')
ax.set_xlabel('$t$')
ax.set_ylabel('$y(t)$')
plt.legend(loc='best');
plt.show()
```

In[∘]:=
```
quit()
```

5. ML Solution via NN

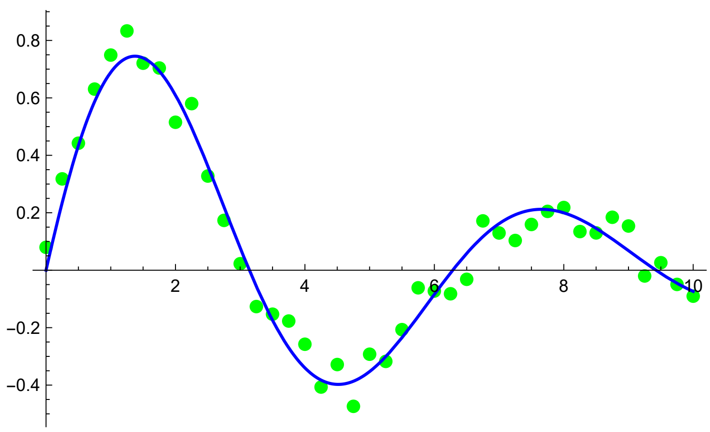
Now let us suppose that we have noisy measurements (t_i, y_i).

In[∘]:=
```
noisydata =
  Table[{i 0.25, ((ysym /. {R → 0.2, t → i 0.25}) + RandomReal[{-0.1, 0.1}])}, {i, 0, 40}];
```

In[∘]:=
```
p2 = ListPlot[noisydata, PlotStyle → {PointSize [0.020], Green}];
```

In[∘]:=
```
Show[p2, p1]
```

Out[∘]=



In[∘]:=
```
net = NetInitialize@NetChain[{LinearLayer[10, "Input" → 1], ElementwiseLayer[Tanh],
    LinearLayer[5], ElementwiseLayer[Tanh], LinearLayer[1]}, "Input" → {1}]
```

Out[∘]=

NetChain[Input port: vector (size: 1) Output port: vector (size: 1)]

In[∘]:=
```
NetGraph[net]
```

Out[∘]=

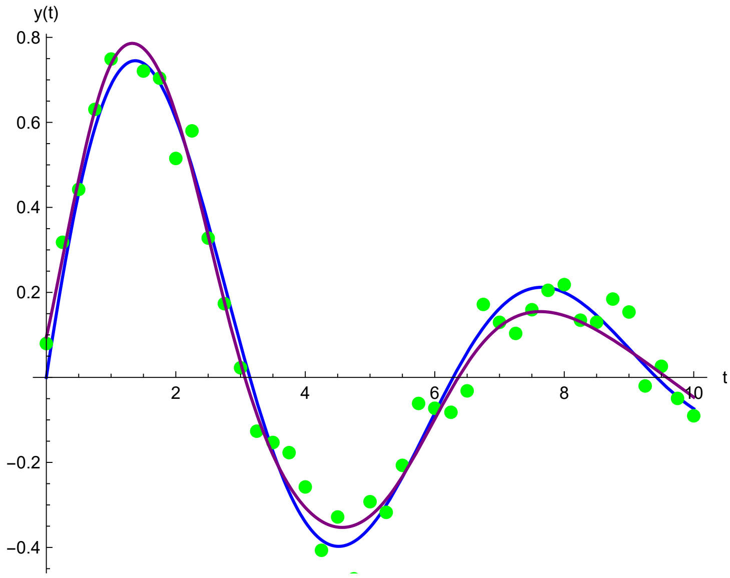
NetGraph[Input port: vector (size: 1) Output port: vector (size: 1)]

In[∘]:=
```
trainingdata = Map[{#[[1]]} → {#[[2]]} &, noisydata];
```

```
In[∘]:= 𝒩𝒩 = NetTrain[net, trainingdata, ValidationSet → Scaled[0.2]]
Out[∘]=
```

NetChain[Input port: vector (size: 1) Output port: vector (size: 1)]

```
In[∘]:= p3 = Plot[𝒩𝒩[t], {t, 0, 10}, PlotStyle → Purple];

In[∘]:= Show[p1, p2, p3, ImageSize → 300]
Out[∘]=
```



The PM and ML can be solve via NN!

Physics Informed Neural Network (PINN)

Simple PINN problem

$$G(p) = \sum_i ((\mathcal{N}(p, t_i) - y_i)^2 + \rho \sum_i \left(\frac{d\mathcal{N}(p, t_i)}{dt} - \mathcal{L}(t_i, y_i) \right)^2 + (\mathcal{N}(p, 0) - 0)^2$$

$$\mathcal{L}(t_i, y_i) = \exp(-t_i / 5) \cos(t_i) - R\, \mathcal{N}(p, t_i)$$

```
In[∘]:= datat = Transpose[noisydata]〚1〛;

In[∘]:= datay = Transpose[noisydata]〚2〛;
```

In[*]:=
```
n = 5;
```

In[*]:=
$$\mathcal{N} = a_0 + \sum_{i=1}^{n} \frac{a_i}{1 + \text{Exp}[-(b_i + c_i\, t)]}$$

Out[*]=
$$a_0 + \frac{a_1}{1 + e^{-b_1 - t\, c_1}} + \frac{a_2}{1 + e^{-b_2 - t\, c_2}} + \frac{a_3}{1 + e^{-b_3 - t\, c_3}} + \frac{a_4}{1 + e^{-b_4 - t\, c_4}} + \frac{a_5}{1 + e^{-b_5 - t\, c_5}}$$

and

In[*]:=
```
dN = D[N, t] // Simplify
```

Out[*]=
$$\frac{e^{b_1 + t\, c_1} a_1 c_1}{\left(1 + e^{b_1 + t\, c_1}\right)^2} + \frac{e^{b_2 + t\, c_2} a_2 c_2}{\left(1 + e^{b_2 + t\, c_2}\right)^2} + \frac{e^{b_3 + t\, c_3} a_3 c_3}{\left(1 + e^{b_3 + t\, c_3}\right)^2} + \frac{e^{b_4 + t\, c_4} a_4 c_4}{\left(1 + e^{b_4 + t\, c_4}\right)^2} + \frac{e^{b_5 + t\, c_5} a_5 c_5}{\left(1 + e^{b_5 + t\, c_5}\right)^2}$$

Composition of the residual

First term, the data error: $\sum_i \left(\mathcal{N}(p, t_i) - y_i \right)^2$

In[*]:=
```
dataerr = Total[MapThread[((N /. t → #1) - #2)^2 &, {datat, datay}]];
```

Second term, the model error: $\sum_i ((d\mathcal{N}(p, t_i) - (\exp(-t_i / 5) \cos(t_i) - R\, \mathcal{N}(p, t_i)))^2$

In[*]:=
```
modelerr = Total[MapThread[(dN - (Exp[-t / 5] Cos[t] - R N) /. t → #1)^2 &, {datat, datay}]];
```

Third term, initial condition $(\mathcal{N}(p, 0) - 0)^2$

In[*]:=
```
iniconderr = ((N /. t → 0) - 0)^2;
R = 0.2;
```

Fitting network and model parameters

In[*]:=
```
vars = {a0, Table[{ai, bi, ci}, {i, 1, n}], R} // Flatten
```

Out[*]=
$\{a_0, a_1, b_1, c_1, a_2, b_2, c_2, a_3, b_3, c_3, a_4, b_4, c_4, a_5, b_5, c_5, R\}$

```
ρ = 0.17;(*Hyperparameter*)
```

In[*]:=
```
G = dataerr + ρ modelerr + iniconderr;
```

PINN can be considered a modeling technique where the machine learning model is regularized by the physical model!

```
sol = NMinimize[G, vars, MaxIterations → 200] // Quiet
```

Out[*]=
$\{0.113142, \{a_0 \to 6.67798, a_1 \to -6.11769, b_1 \to -4.57232, c_1 \to 0.694326, a_2 \to -2.42818, b_2 \to -0.154402, c_2 \to -1.73998, a_3 \to 0.297474, b_3 \to -16.9887, c_3 \to 3.25016, a_4 \to -5.43924, b_4 \to 6.0283, c_4 \to -0.927631, a_5 \to -1.25222, b_5 \to -4.10501, c_5 \to 1.69228, R \to 0.232054\}\}$

In[•]:=
```
myNet = 𝒩 /. sol[[2]]
```

Out[•]=

$$6.67798 + \frac{0.297474}{1 + e^{16.9887 - 3.25016\,t}} - \frac{1.25222}{1 + e^{4.10501 - 1.69228\,t}} - \frac{6.11769}{1 + e^{4.57232 - 0.694326\,t}} - \frac{5.43924}{1 + e^{-6.0283 + 0.927631\,t}} - \frac{2.42818}{1 + e^{0.154402 + 1.73998\,t}}$$

In[•]:=
```
Show[p1, Plot[myNet, {t, 0, 10}, PlotStyle → Purple, AspectRatio → 0.8], p2]
```

Out[•]=

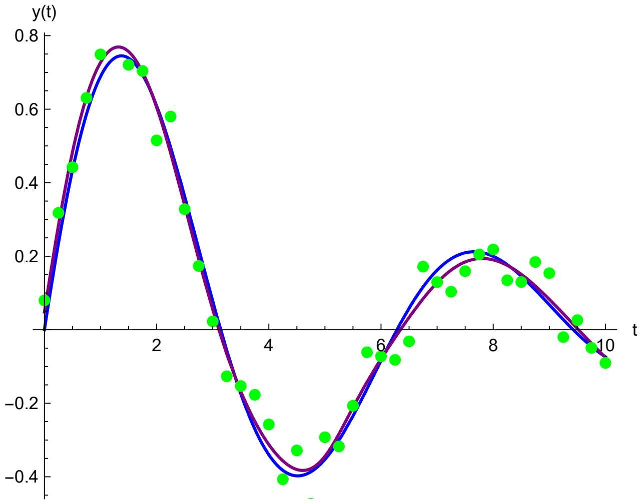


Fig.9 PINN approximation - purple

Identification Inverse Problems

Nonlinear parameter Identification

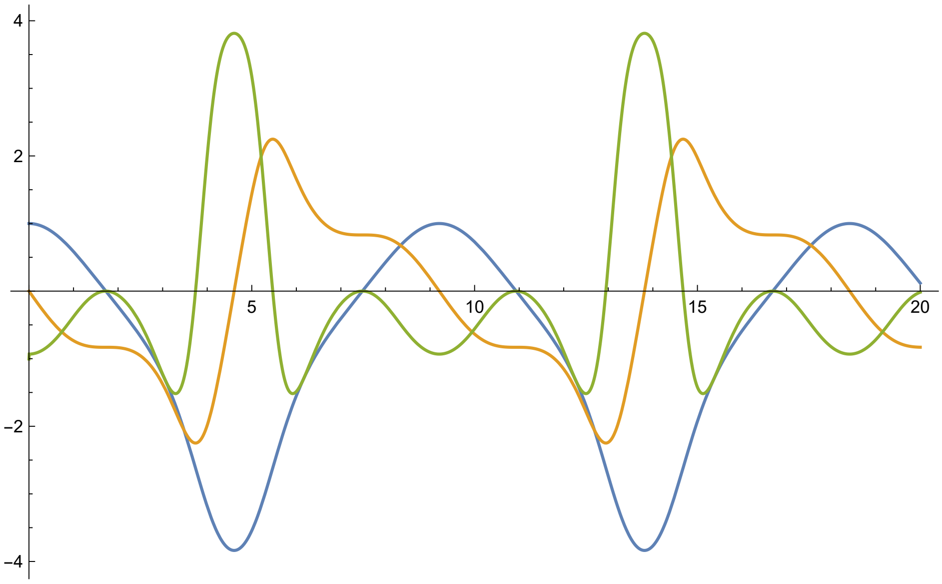
In[•]:=
```
solN =
  NDSolve[{u''[t] + (Sin[λ u[t]] u[t] /. λ → 1.2) == 0, u[0] == 1, u'[0] == 0}, u, {t, 0, 20}]
```

Out[•]=

```
{{u → InterpolatingFunction[ Domain: {{0., 20.}} Output: scalar ]}}
```

In[∘]:= `pinv1 = Plot[Evaluate[{u[t], u'[t], u''[t]} /. solN], {t, 0, 20}, PlotStyle → Automatic]`

Out[∘]=



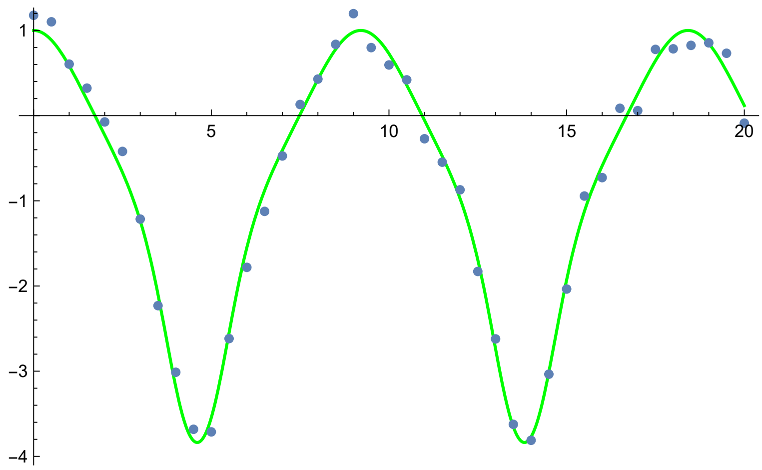
Blue - u(t), Yellow - u’(t), Green -u’’(t)

In[∘]:= `tm = Range[0, 20, 0.5];`

In[∘]:= `um = Map[(u[#] /. solN) + RandomReal[{-0.25, 0.25}] &, tm] // Flatten;`

In[∘]:=
```
Show[Plot[u[t] /. solN, {t, 0, 20}, PlotStyle → Green],
 ListPlot[Transpose[{tm, um}], AspectRatio → 0.5]]
```

Out[∘]=



In[∘]:= `n = 8;`

In[∘]:= $\mathcal{N} = a_0 + \sum_{i=1}^{n} a_i \, \mathrm{Exp}\left[b_i \, (t - c_i)^2\right]$

Out[∘]=

$$a_0 + e^{b_1 (t-c_1)^2} a_1 + e^{b_2 (t-c_2)^2} a_2 + e^{b_3 (t-c_3)^2} a_3 + e^{b_4 (t-c_4)^2} a_4 + e^{b_5 (t-c_5)^2} a_5 + e^{b_6 (t-c_6)^2} a_6 + e^{b_7 (t-c_7)^2} a_7 + e^{b_8 (t-c_8)^2} a_8$$

In[∘]:=
```
d𝒩 = D[𝒩, t] // Simplify
```
Out[∘]=

$$2\left(e^{b_1(t-c_1)^2}a_1 b_1 (t-c_1)+e^{b_2(t-c_2)^2}a_2 b_2 (t-c_2)+e^{b_3(t-c_3)^2}a_3 b_3 (t-c_3)+e^{b_4(t-c_4)^2}a_4 b_4 (t-c_4)+e^{b_5(t-c_5)^2}a_5 b_5 (t-c_5)+e^{b_6(t-c_6)^2}a_6 b_6 (t-c_6)+e^{b_7(t-c_7)^2}a_7 b_7 (t-c_7)+e^{b_8(t-c_8)^2}a_8 b_8 (t-c_8)\right)$$

In[∘]:=
```
d2𝒩 = D[𝒩, {t, 2}] // Simplify
```
Out[∘]=

$$2\left(e^{b_1(t-c_1)^2}a_1 b_1\left(1+2b_1(t-c_1)^2\right)+e^{b_2(t-c_2)^2}a_2 b_2\left(1+2b_2(t-c_2)^2\right)+e^{b_3(t-c_3)^2}a_3 b_3\left(1+2b_3(t-c_3)^2\right)+e^{b_4(t-c_4)^2}a_4 b_4\left(1+2b_4(t-c_4)^2\right)+e^{b_5(t-c_5)^2}a_5 b_5\left(1+2b_5(t-c_5)^2\right)+e^{b_6(t-c_6)^2}a_6 b_6\left(1+2b_6(t-c_6)^2\right)+e^{b_7(t-c_7)^2}a_7 b_7\left(1+2b_7(t-c_7)^2\right)+e^{b_8(t-c_8)^2}a_8 b_8\left(1+2b_8(t-c_8)^2\right)\right)$$

In[∘]:=
```
dataerr = Sqrt[Total[MapThread[((𝒩 /. t → #1) - #2)^2 &, {tm, um}]]];
```

In[∘]:=
```
modelerr = Sqrt[Total[MapThread[((d2𝒩 + Sin[𝒩] 𝒩) /. t → #1)^2 &, {tm, um}]]];
```

In[∘]:=
```
inierr = Sqrt[(1 - 𝒩 /. t → 0)^2] + Sqrt[(d𝒩 /. t → 0)^2];
```

In[∘]:=
```
ρ = 0.1;
```

In[∘]:=
```
G = dataerr + ρ (modelerr + inierr);
```

In[∘]:=
```
vars = {a0, Table[{ai, bi, ci}, {i, 1, n}], λ} // Flatten
```
Out[∘]=

$\{a_0, a_1, b_1, c_1, a_2, b_2, c_2, a_3, b_3, c_3, a_4, b_4, c_4, a_5, b_5, c_5, a_6, b_6, c_6, a_7, b_7, c_7, a_8, b_8, c_8, \lambda\}$

In[∘]:=
```
sol = NMinimize[{G, 2. > λ > 1.}, vars,
   Method → {"DifferentialEvolution", "ScalingFactor" → 0.9, "CrossProbability" → 0.1,
     "PostProcess" → {FindMinimum, Method → "QuasiNewton"}}] // Quiet
```
Out[∘]=

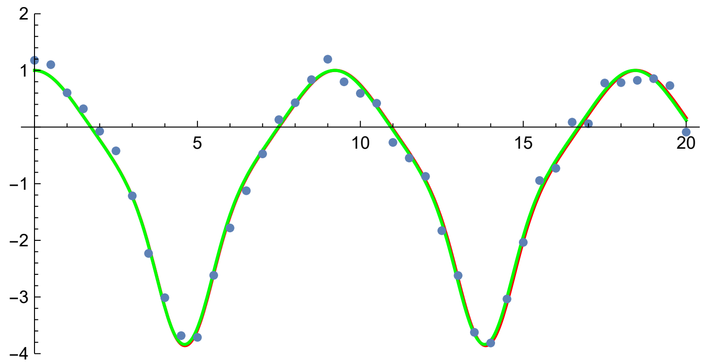
$\{10.1483, \{a_0 \to -1.14459, a_1 \to 0.322759, b_1 \to -1.25787, c_1 \to -1.06809, a_2 \to 0.763326, b_2 \to -3.62707, c_2 \to 1.35905, a_3 \to -0.521679, b_3 \to -0.98995, c_3 \to 4.51511, a_4 \to 3.22351, b_4 \to -0.149608, c_4 \to -1.88423, a_5 \to -1.29337, b_5 \to -2.65401, c_5 \to -1.25614, a_6 \to -0.030139, b_6 \to -2.6551, c_6 \to -0.77204, a_7 \to 4.63802, b_7 \to -2.03177, c_7 \to -2.1545, a_8 \to -0.455765, b_8 \to -3.00959, c_8 \to -2.51537, \lambda \to 1.19275\}\}$

In[∘]:=
```
solE =
 NDSolve[{u''[t] + (Sin[λ u[t]] u[t] /. sol[[2]]) == 0, u[0] == 1, u'[0] == 0}, u, {t, 0, 20}]
```
Out[∘]=

{{u → InterpolatingFunction[Domain: {{0., 20.}} Output: scalar]}}

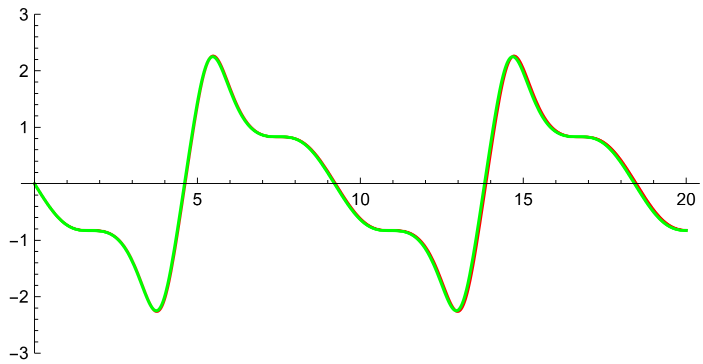
In[•]:=
```
Show[
 {Plot[u[t] /. solE, {t, 0, 20}, PlotStyle → Red, AspectRatio → 0.5, PlotRange → {2, -4}],
  Plot[u[t] /. solN, {t, 0, 20}, PlotStyle → Green, PlotRange → All],
  ListPlot[Transpose[{tm, um}]]}]
```

Out[•]=



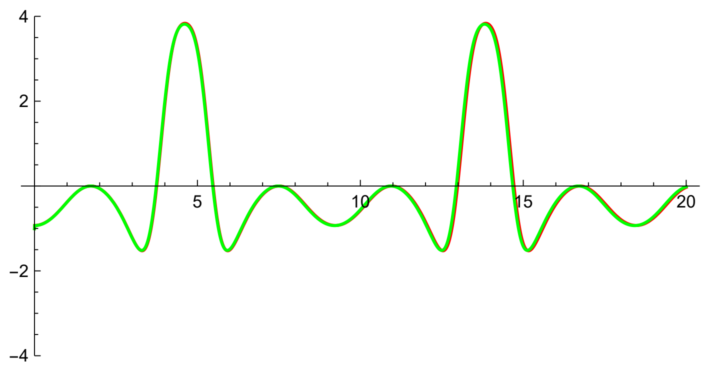
In[•]:=
```
Show[{Plot[u'[t] /. solE, {t, 0, 20},
   PlotStyle → Red, AspectRatio → 0.5, PlotRange → {-3, 3}],
  Plot[u'[t] /. solN, {t, 0, 20}, PlotStyle → Green, PlotRange → All]}]
```

Out[•]=

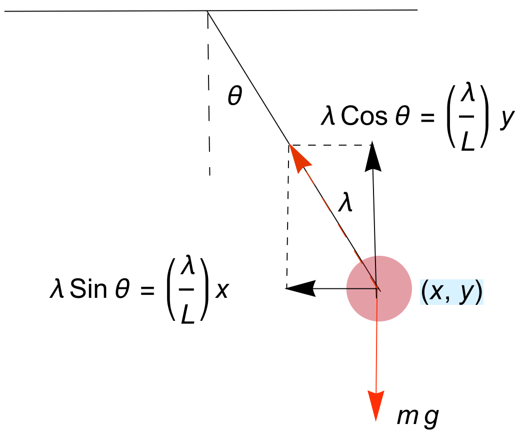


In[•]:=
```
Show[{Plot[u''[t] /. solE, {t, 0, 20},
   PlotStyle → Red, AspectRatio → 0.5, PlotRange → {-4, 4}],
  Plot[u''[t] /. solN, {t, 0, 20}, PlotStyle → Green, PlotRange → All]}]
```

Out[•]=



Inverse Problem of a DAE -Differential-Algebraic Equation System



Model the motion of a pendulum in Cartesian coordinates of 2 dimensions. The governing equations using Newton's second law of motion, $m\,x''(t) = \sum F_x$ and $m\,y''(t) = \sum F_y$,

$$m \frac{d^2 x(t)}{dt^2} = \lambda(t) \frac{x(t)}{L} - \mathcal{D} \frac{dx(t)}{dt} m$$

$$m \frac{d^2 y(t)}{dt^2} = \lambda(t) \frac{y(t)}{L} - m\,g - \mathcal{D} \frac{dy(t)}{dt} m$$

$$x^2(t) + y^2(t) = L$$

where there is a mass m at the point $\{x(t),\ y(t)\}$ constrained by a string of length L and λ is the tension in the string. For simplicity in the description of index reduction, take $m = L = 1$. The figure above shows the schematic of the pendulum system

The equations of motion with damping $\mathcal{D}$,

$$\frac{d^2 x(t)}{dt^2} = \lambda(t)\,x(t) - \mathcal{D} \frac{dx(t)}{dt}$$

$$\frac{d^2 y(t)}{dt^2} = \lambda(t)\,y(t) - g - \mathcal{D} \frac{dy(t)}{dt}$$

The geometric constraint,

$$x^2(t) + y^2(t) = 1$$

The initial conditions,

$$x(0) = 1$$

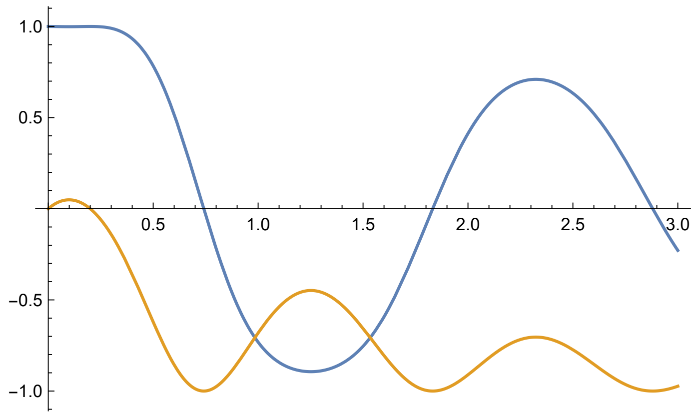
$$\left.\frac{dy(t)}{dt}\right|_{t=0} = 1$$

This is a system of ODE with AE = DAE (Differential-Algebraic Equation)

Numerical Solution

First let us solve the problem with numerical method employing *Index Reduction Method* in case $\mathcal{D} = 0.6$

```
In[]:= deqns = {x''[t] == λ[t] × x[t] - 0.6 x'[t], y''[t] == λ[t] × y[t] - 9.81 - 0.6 y'[t]};
       (*Differential Equations*)

In[]:= aeqns = {x[t]^2 + y[t]^2 == 1^2}; (*Algebraic Equation*)

In[]:= ics = {x[0] == 1, y'[0] == 1}; (*Initial Conditions*)

In[]:= sol1 = NDSolve[{deqns, aeqns, ics},
          {x, y, λ}, {t, 0, 5}, Method → {"IndexReduction" → True}];

In[]:= Plot[Evaluate[{x[t], y[t]} /. sol1], {t, 0, 3}]
Out[]=
```



x(t) - blue, y(t) - yellow

PINN Solution

Networks

$$x(t) \approx \mathcal{N}_x(t)$$
$$\lambda(t) \approx \mathcal{N}_\lambda(t)$$
$$y(t) \approx \mathcal{N}_y(t)$$

Activation Function

$$\text{Tanh}(\eta) = \frac{e^{\eta} - e^{-\eta}}{e^{\eta} + e^{-\eta}}$$

Diff.Eqs. errors

$$\mathcal{L}_x = \sqrt{\sum_i \left(\frac{d^2 \mathcal{N}_x(t_i)}{dt^2} - \mathcal{N}_\lambda(t_i)\,\mathcal{N}_x(t_i) + \mathcal{D}\,\frac{d\mathcal{N}_x(t_i)}{dt} \right)^2}$$

$$\mathcal{L}_y = \sqrt{\sum_i \left(\frac{d^2 \mathcal{N}_y(t_i)}{dt^2} - \mathcal{N}_\lambda(t_i)\,\mathcal{N}_y(t_i) + g + \mathcal{D}\,\frac{d\mathcal{N}_y(t_i)}{dt} \right)^2}$$

Alg.Eq.errors

$$\mathcal{L}_{xy} = \sqrt{\sum_i \left(\mathcal{N}_x^{\,2}(t_i) + \mathcal{N}_y^{\,2}(t_i) - 1 \right)^2}$$

Init.Cond.errors

$$\alpha_{xy} = \sqrt{(\mathcal{N}_x(0) - 1)^2 + \left(\frac{d\mathcal{N}_y(0)}{dt} - 1 \right)^2}$$

Data errors

$$\delta_x = \sqrt{\sum_i (x_m(t_i) - \mathcal{N}_x(t_i))^2}$$

$$\delta_y = \sqrt{\sum_i (y_m(t_i) + \mathcal{N}_y(t_i))^2}$$

Objective Function - Inverse Problem

$$\mathcal{G}(px, py, p\lambda, \mathcal{D}) = \mathcal{L}_x + \mathcal{L}_y + \mathcal{L}_{xy} + \alpha_{xy} + \delta_x + \delta_y$$

In[∘]:=
```
n = 8;
```

In[∘]:=
```
px = {ax0, Table[{axi, bxi, cxi}, {i, 1, n}]} // Flatten
```

Out[∘]=
```
{ax0, ax1, bx1, cx1, ax2, bx2, cx2, ax3, bx3, cx3, ax4,
 bx4, cx4, ax5, bx5, cx5, ax6, bx6, cx6, ax7, bx7, cx7, ax8, bx8, cx8}
```

In[ ]:= $\mathcal{N}x = ax_0 + \sum_{i=1}^{n} ax_i \,\text{Tanh}[bx_i + cx_i\, t]$

Out[ ]=

$ax_0 + ax_1 \text{Tanh}[bx_1 + t\, cx_1] + ax_2 \text{Tanh}[bx_2 + t\, cx_2] + ax_3 \text{Tanh}[bx_3 + t\, cx_3] + ax_4 \text{Tanh}[bx_4 + t\, cx_4] + ax_5 \text{Tanh}[bx_5 + t\, cx_5] + ax_6 \text{Tanh}[bx_6 + t\, cx_6] + ax_7 \text{Tanh}[bx_7 + t\, cx_7] + ax_8 \text{Tanh}[bx_8 + t\, cx_8]$

In[ ]:= **d𝒩x = D[𝒩x, t]**

Out[ ]=

$\text{Sech}[bx_1 + t\, cx_1]^2 ax_1\, cx_1 + \text{Sech}[bx_2 + t\, cx_2]^2 ax_2\, cx_2 + \text{Sech}[bx_3 + t\, cx_3]^2 ax_3\, cx_3 + \text{Sech}[bx_4 + t\, cx_4]^2 ax_4\, cx_4 + \text{Sech}[bx_5 + t\, cx_5]^2 ax_5\, cx_5 + \text{Sech}[bx_6 + t\, cx_6]^2 ax_6\, cx_6 + \text{Sech}[bx_7 + t\, cx_7]^2 ax_7\, cx_7 + \text{Sech}[bx_8 + t\, cx_8]^2 ax_8\, cx_8$

In[ ]:= **d2𝒩x = D[𝒩x, {t, 2}]**

Out[ ]=

$-2\, \text{Sech}[bx_1 + t\, cx_1]^2 ax_1\, cx_1^2 \text{Tanh}[bx_1 + t\, cx_1] - 2\, \text{Sech}[bx_2 + t\, cx_2]^2 ax_2\, cx_2^2 \text{Tanh}[bx_2 + t\, cx_2] - 2\, \text{Sech}[bx_3 + t\, cx_3]^2 ax_3\, cx_3^2 \text{Tanh}[bx_3 + t\, cx_3] - 2\, \text{Sech}[bx_4 + t\, cx_4]^2 ax_4\, cx_4^2 \text{Tanh}[bx_4 + t\, cx_4] - 2\, \text{Sech}[bx_5 + t\, cx_5]^2 ax_5\, cx_5^2 \text{Tanh}[bx_5 + t\, cx_5] - 2\, \text{Sech}[bx_6 + t\, cx_6]^2 ax_6\, cx_6^2 \text{Tanh}[bx_6 + t\, cx_6] - 2\, \text{Sech}[bx_7 + t\, cx_7]^2 ax_7\, cx_7^2 \text{Tanh}[bx_7 + t\, cx_7] - 2\, \text{Sech}[bx_8 + t\, cx_8]^2 ax_8\, cx_8^2 \text{Tanh}[bx_8 + t\, cx_8]$

In[ ]:= **pλ = {aλ$_0$, Table[{aλ$_i$, bλ$_i$, cλ$_i$}, {i, 1, n}]} // Flatten**

Out[ ]=

$\{a\lambda_0, a\lambda_1, b\lambda_1, c\lambda_1, a\lambda_2, b\lambda_2, c\lambda_2, a\lambda_3, b\lambda_3, c\lambda_3, a\lambda_4, b\lambda_4, c\lambda_4, a\lambda_5, b\lambda_5, c\lambda_5, a\lambda_6, b\lambda_6, c\lambda_6, a\lambda_7, b\lambda_7, c\lambda_7, a\lambda_8, b\lambda_8, c\lambda_8\}$

In[ ]:= $\mathcal{N}\lambda = a\lambda_0 + \sum_{i=1}^{n} a\lambda_i \,\text{Tanh}[b\lambda_i + c\lambda_i\, t]$

Out[ ]=

$a\lambda_0 + a\lambda_1 \text{Tanh}[b\lambda_1 + t\, c\lambda_1] + a\lambda_2 \text{Tanh}[b\lambda_2 + t\, c\lambda_2] + a\lambda_3 \text{Tanh}[b\lambda_3 + t\, c\lambda_3] + a\lambda_4 \text{Tanh}[b\lambda_4 + t\, c\lambda_4] + a\lambda_5 \text{Tanh}[b\lambda_5 + t\, c\lambda_5] + a\lambda_6 \text{Tanh}[b\lambda_6 + t\, c\lambda_6] + a\lambda_7 \text{Tanh}[b\lambda_7 + t\, c\lambda_7] + a\lambda_8 \text{Tanh}[b\lambda_8 + t\, c\lambda_8]$

In[ ]:= **py = {ay$_0$, Table[{ay$_i$, by$_i$, cy$_i$}, {i, 1, n}]} // Flatten**

Out[ ]=

$\{ay_0, ay_1, by_1, cy_1, ay_2, by_2, cy_2, ay_3, by_3, cy_3, ay_4, by_4, cy_4, ay_5, by_5, cy_5, ay_6, by_6, cy_6, ay_7, by_7, cy_7, ay_8, by_8, cy_8\}$

In[ ]:= $\mathcal{N}y = ay_0 + \sum_{i=1}^{n} ay_i \,\text{Tanh}[by_i + cy_i\, t]$

Out[ ]=

$ay_0 + ay_1 \text{Tanh}[by_1 + t\, cy_1] + ay_2 \text{Tanh}[by_2 + t\, cy_2] + ay_3 \text{Tanh}[by_3 + t\, cy_3] + ay_4 \text{Tanh}[by_4 + t\, cy_4] + ay_5 \text{Tanh}[by_5 + t\, cy_5] + ay_6 \text{Tanh}[by_6 + t\, cy_6] + ay_7 \text{Tanh}[by_7 + t\, cy_7] + ay_8 \text{Tanh}[by_8 + t\, cy_8]$

In[ ]:= **d𝒩y = D[𝒩y, t]**

Out[ ]=

$\text{Sech}[by_1 + t\, cy_1]^2 ay_1\, cy_1 + \text{Sech}[by_2 + t\, cy_2]^2 ay_2\, cy_2 + \text{Sech}[by_3 + t\, cy_3]^2 ay_3\, cy_3 + \text{Sech}[by_4 + t\, cy_4]^2 ay_4\, cy_4 + \text{Sech}[by_5 + t\, cy_5]^2 ay_5\, cy_5 + \text{Sech}[by_6 + t\, cy_6]^2 ay_6\, cy_6 + \text{Sech}[by_7 + t\, cy_7]^2 ay_7\, cy_7 + \text{Sech}[by_8 + t\, cy_8]^2 ay_8\, cy_8$

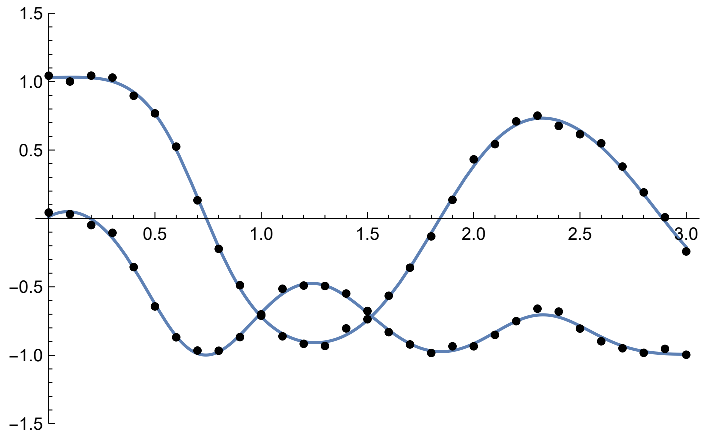
```
In[•]:= d2𝒩y = D[𝒩y, {t, 2}]
Out[•]=
-2 Sech[by1 + t cy1]^2 ay1 cy1^2 Tanh[by1 + t cy1] - 2 Sech[by2 + t cy2]^2 ay2 cy2^2 Tanh[by2 + t cy2] -
 2 Sech[by3 + t cy3]^2 ay3 cy3^2 Tanh[by3 + t cy3] - 2 Sech[by4 + t cy4]^2 ay4 cy4^2 Tanh[by4 + t cy4] -
 2 Sech[by5 + t cy5]^2 ay5 cy5^2 Tanh[by5 + t cy5] - 2 Sech[by6 + t cy6]^2 ay6 cy6^2 Tanh[by6 + t cy6] -
 2 Sech[by7 + t cy7]^2 ay7 cy7^2 Tanh[by7 + t cy7] - 2 Sech[by8 + t cy8]^2 ay8 cy8^2 Tanh[by8 + t cy8]

In[•]:= Clear[𝒟]

In[•]:= vars = Join[{px, pλ, py}, {𝒟}] // Flatten
Out[•]=
{ax0, ax1, bx1, cx1, ax2, bx2, cx2, ax3, bx3, cx3, ax4, bx4, cx4, ax5, bx5, cx5, ax6, bx6, cx6,
 ax7, bx7, cx7, ax8, bx8, cx8, aλ0, aλ1, bλ1, cλ1, aλ2, bλ2, cλ2, aλ3, bλ3, cλ3, aλ4, bλ4, cλ4,
 aλ5, bλ5, cλ5, aλ6, bλ6, cλ6, aλ7, bλ7, cλ7, aλ8, bλ8, cλ8, ay0, ay1, by1, cy1, ay2, by2, cy2,
 ay3, by3, cy3, ay4, by4, cy4, ay5, by5, cy5, ay6, by6, cy6, ay7, by7, cy7, ay8, by8, cy8, 𝒟}

In[•]:= tm = Range[0, 3, 0.1] // Flatten;

In[•]:= datamx = Map[(x[#] /. sol1) + RandomReal[{-0.05, 0.05}] &, tm] // Flatten;

In[•]:= datamy = Map[(y[#] /. sol1) + RandomReal[{-0.05, 0.05}] &, tm] // Flatten;

In[•]:= pp = Show[ListPlot[Transpose[{tm, datamx}], PlotStyle → Black],
    ListPlot[Transpose[{tm, datamy}], PlotStyle → Black]];

In[•]:= nm = Length[tm]
Out[•]=
31

In[•]:= g = 9.81;

In[•]:= dataerrx = Sqrt[Total[MapThread[(#2 - (𝒩x /. t → #1))^2 &, {tm, datamx}]]];

In[•]:= dataerry = Sqrt[Total[MapThread[(#2 - (𝒩y /. t → #1))^2 &, {tm, datamy}]]];

In[•]:= dataerr = dataerrx + dataerry;

In[•]:= diffeqx = Sqrt[Total[Map[((d2𝒩x - 𝒩λ 𝒩x + 𝒟 d𝒩x) /. t → #)^2 &, tm]]];

In[•]:= diffeqy = Sqrt[Total[Map[((d2𝒩y - 𝒩λ 𝒩y + g + 𝒟 d𝒩y) /. t → #)^2 &, tm]]];

In[•]:= algeqxy = Sqrt[Total[Map[((𝒩x^2 + 𝒩y^2 - 1) /. t → #)^2 &, tm]]];

In[•]:= modelerr = diffeqx + diffeqy + algeqxy;

In[•]:= initconderr = Sqrt[((𝒩x /. t → 0) - 1)^2 + ((d𝒩y /. t → 0) - 1)^2];

In[•]:= ρ = 0.013;

In[•]:= G = dataerr + ρ (modelerr + initconderr);
```

In[•]:=
```
solDAE = NMinimize[{G, 1 > 𝒟 > 0}, vars,
  Method → {"DifferentialEvolution", "ScalingFactor" → 0.9,
    "CrossProbability" → 0.1, "PostProcess" → {FindMinimum, Method → "QuasiNewton"}}]
```

Out[•]=
```
{0.462489, {ax0 → -0.423507, ax1 → 1.00389, bx1 → 2.51915,
  cx1 → -3.55853, ax2 → 0.357212, bx2 → 8.41567, cx2 → -2.93006, ax3 → -0.368722,
  bx3 → 1.12565, cx3 → 1.09006, ax4 → -0.883042, bx4 → 1.26948, cx4 → 2.02464,
  ax5 → -3.1319, bx5 → 3.36877, cx5 → -1.82236, ax6 → -1.43643, bx6 → -2.17909,
  cx6 → -1.06897, ax7 → 0.528212, bx7 → -0.128072, cx7 → 1.51288,
  ax8 → 3.05324, bx8 → 2.04039, cx8 → -1.0357, aλ0 → -1.96087, aλ1 → -0.907585,
  bλ1 → 1.25911, cλ1 → 1.48929, aλ2 → -1.64409, bλ2 → 1.95824, cλ2 → -1.08018,
  aλ3 → -1.3049, bλ3 → 0.529591, cλ3 → 1.94706, aλ4 → 3.93317, bλ4 → 9.41813,
  cλ4 → -6.6065, aλ5 → -5.23629, bλ5 → -1.56493, cλ5 → 5.20473, aλ6 → -11.5701,
  bλ6 → 5.2041, cλ6 → -5.77432, aλ7 → -0.926242, bλ7 → 0.33118, cλ7 → 1.98292,
  aλ8 → 9.02398, bλ8 → 3.95694, cλ8 → -7.49236, ay0 → -1.64378, ay1 → 2.01753,
  by1 → 1.58504, cy1 → -2.91446, ay2 → -1.49766, by2 → -6.84212, cy2 → 2.80872,
  ay3 → 1.62558, by3 → 1.42016, cy3 → 3.85221, ay4 → -0.85095, by4 → 1.02637,
  cy4 → 0.740683, ay5 → 1.11088, by5 → 3.2272, cy5 → -2.23208, ay6 → 1.78644,
  by6 → -5.17205, cy6 → 2.23648, ay7 → -2.34473, by7 → 1.65892, cy7 → -2.0721,
  ay8 → -0.404431, by8 → -1.61306, cy8 → -1.87289, 𝒟 → 0.583894}}
```

In[•]:=
```
Show[{Plot[{𝒩x, 𝒩y} /. solDAE⟦2⟧, {t, 0, 3}, PlotRange → {-1.5, 1.5}], pp}]
```

Out[•]=

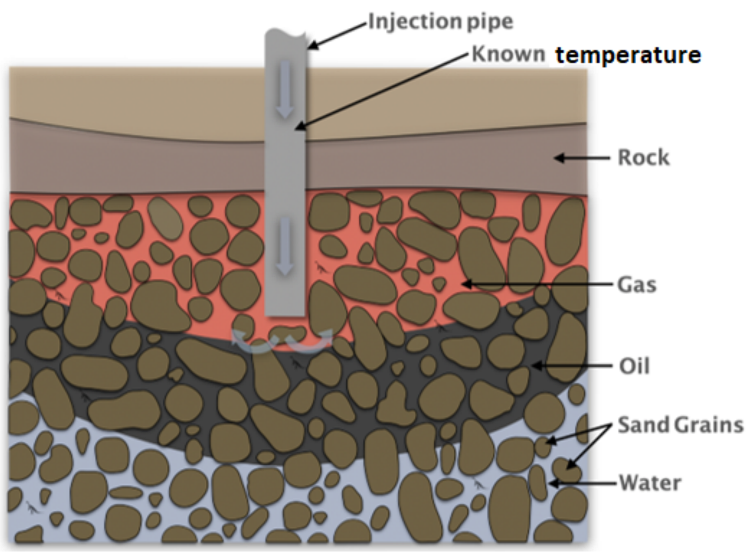


Identification Inverse Problem(parameter estimation)

In inverse problems, there are some unknown parameters, but we have some extra information on some points besides the differential equation and boundary conditions:

https://www.intechopen.com/online-first/83203

Consider the problem of modelling the dynamics of oil & gas fields in porous - media of earth subsurface . To successfully determine the fluid flow one has to deal with a system of PDEs with unknown distributions of media properties (porosity for instance) and unknown state of the syste . Yet, to make the problem tractable, there' s data present about the temperature of one of the fluids (water component) in an injection well :



PDF problem

Let us consider the following the PDE representing a physical model,

$$\frac{\partial T(t, x)}{\partial t} = \kappa \frac{\partial T^2(t, x)}{\partial x^2}$$

where $T(0, x) = 1, \; T(t, 0) = 1 + \sin(t), \; T(t, 5) = 1$

Numerical solution in case of known κ

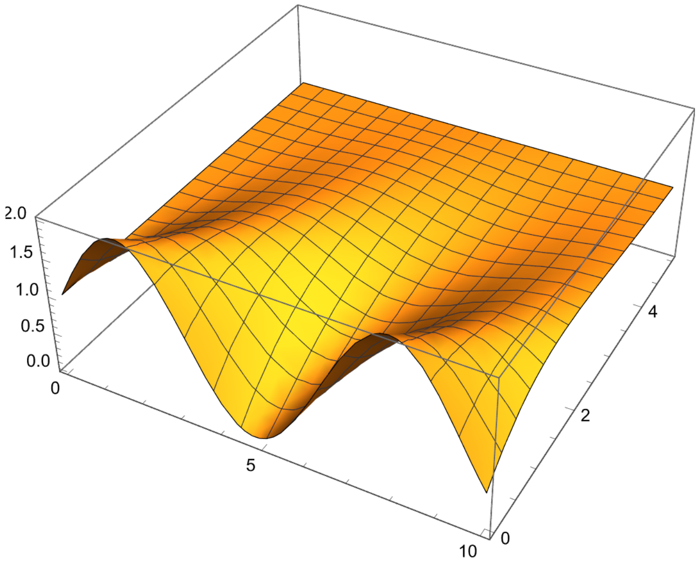
In[•]:=
```
ClearAll["Global`*"]
```

In[•]:=
```
κ = 1;
```

In[•]:=
```
sol = NDSolve[{D[T[t, x], t] == κ D[T[t, x], x, x],
   T[0, x] == 1, T[t, 0] == 1 + Sin[t], T[t, 5] == 1}, T, {t, 0, 10}, {x, 0, 5}]
```

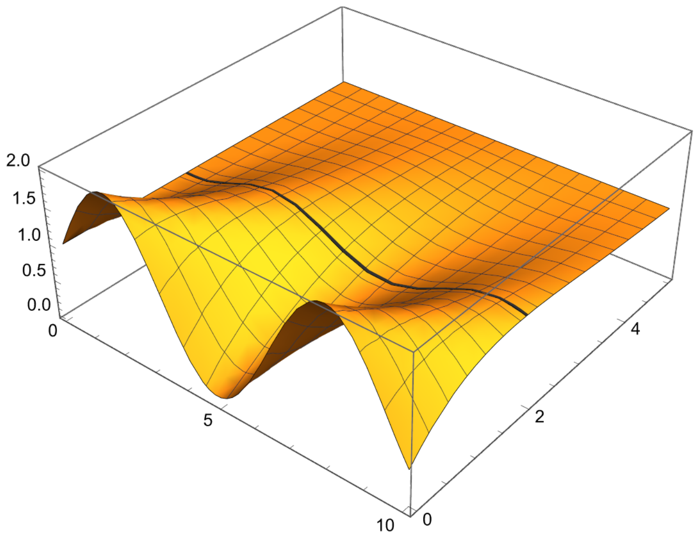
Out[•]=
```
{{T → InterpolatingFunction[ Domain: {{0., 10.}, {0., 5.}} Output: scalar ]}}
```

In[•]:=
```
pinv1 = Plot3D[Evaluate[T[t, x] /. sol], {t, 0, 10}, {x, 0, 5}, PlotRange → All]
```
Out[•]=



Measured value of T(t, 2) - we use 1 sensor at x = 2

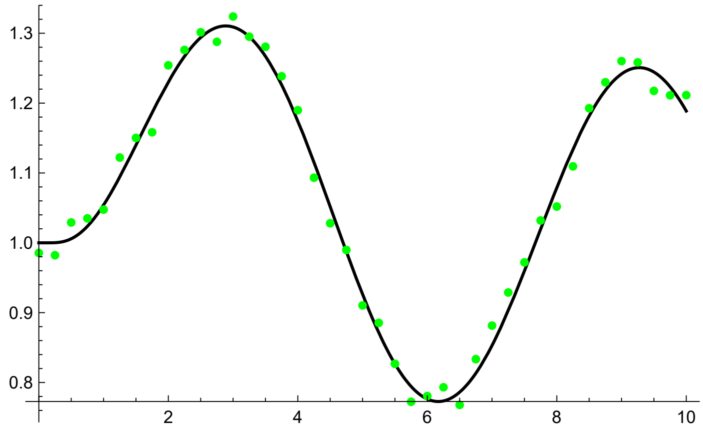
In[•]:=
```
pinv2 = Plot3D[Evaluate[T[t, x] /. sol], {t, 0, 10}, {x, 1.975, 2.0125}, PlotRange → All];
```
In[•]:=
```
Show[pinv2, pinv1]
```
Out[•]=



In[•]:=
```
pinv3 = Plot[Evaluate[T[t, 2] /. sol], {t, 0, 10}, PlotStyle → Black];
```
In[•]:=
```
noisydata = Map[Flatten[#] &,
    Table[{i 0.25, ((Evaluate[T[t, 2] /. sol] /. t → i 0.25)) + RandomReal[{-0.03, 0.03}]},
     {i, 0, 40}]];
```
In[•]:=
```
pinv4 = ListPlot[noisydata, PlotStyle → Green];
```

In[∘]:=
```
Show[pinv3, pinv4]
```

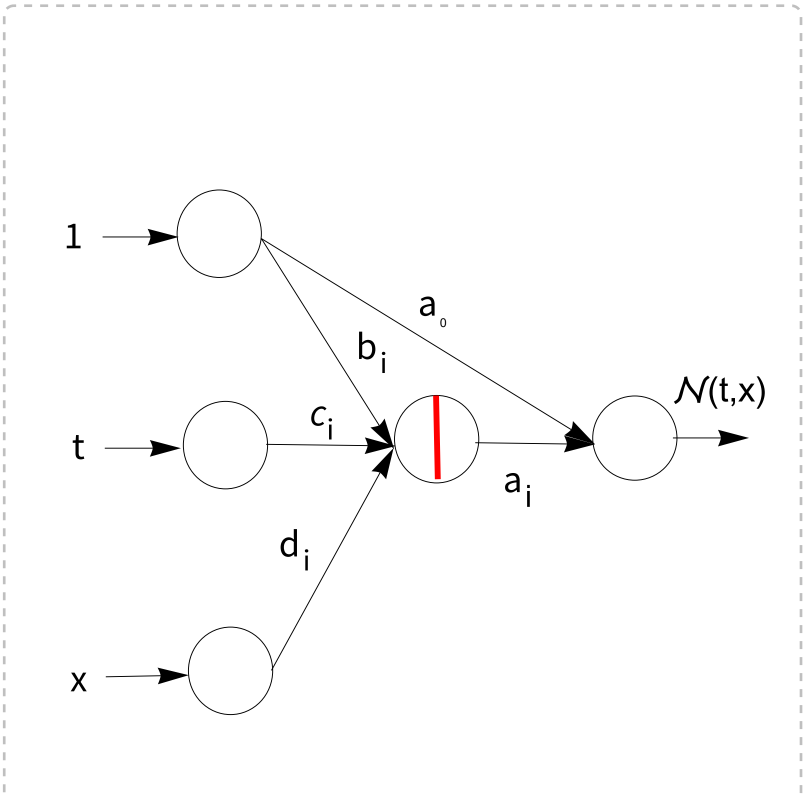
Out[∘]=



Solution of Inverse Problem via PINN

Now we have two active input nodes,

In[∘]:=



Let us select again one hidden layer with six nodes,

In[∘]:=
```
n = 6;
```

In[•]:=
$$\mathcal{N} = a_0 + \sum_{i=1}^{n} \frac{a_i}{1 + \text{Exp}[-(b_i + c_i\, t + d_i\, x)]}$$

Out[•]=
$$a_0 + \frac{a_1}{1+e^{-b_1 - t c_1 - x d_1}} + \frac{a_2}{1+e^{-b_2 - t c_2 - x d_2}} + \frac{a_3}{1+e^{-b_3 - t c_3 - x d_3}} + \frac{a_4}{1+e^{-b_4 - t c_4 - x d_4}} + \frac{a_5}{1+e^{-b_5 - t c_5 - x d_5}} + \frac{a_6}{1+e^{-b_6 - t c_6 - x d_6}}$$

The derivations of the network can be carried out

In[•]:=
```
dtN = D[N, t] // Simplify
```

Out[•]=
$$\frac{e^{b_1+t c_1+x d_1} a_1 c_1}{\left(1+e^{b_1+t c_1+x d_1}\right)^2} + \frac{e^{b_2+t c_2+x d_2} a_2 c_2}{\left(1+e^{b_2+t c_2+x d_2}\right)^2} + \frac{e^{b_3+t c_3+x d_3} a_3 c_3}{\left(1+e^{b_3+t c_3+x d_3}\right)^2} + \frac{e^{b_4+t c_4+x d_4} a_4 c_4}{\left(1+e^{b_4+t c_4+x d_4}\right)^2} + \frac{e^{b_5+t c_5+x d_5} a_5 c_5}{\left(1+e^{b_5+t c_5+x d_5}\right)^2} + \frac{e^{b_6+t c_6+x d_6} a_6 c_6}{\left(1+e^{b_6+t c_6+x d_6}\right)^2}$$

In[•]:=
```
dx2N = D[N, {x, 2}] // Simplify
```

Out[•]=
$$-\frac{e^{b_1+t c_1+x d_1}\left(-1+e^{b_1+t c_1+x d_1}\right) a_1 d_1^2}{\left(1+e^{b_1+t c_1+x d_1}\right)^3} - \frac{e^{b_2+t c_2+x d_2}\left(-1+e^{b_2+t c_2+x d_2}\right) a_2 d_2^2}{\left(1+e^{b_2+t c_2+x d_2}\right)^3} - \frac{e^{b_3+t c_3+x d_3}\left(-1+e^{b_3+t c_3+x d_3}\right) a_3 d_3^2}{\left(1+e^{b_3+t c_3+x d_3}\right)^3} - \frac{e^{b_4+t c_4+x d_4}\left(-1+e^{b_4+t c_4+x d_4}\right) a_4 d_4^2}{\left(1+e^{b_4+t c_4+x d_4}\right)^3} - \frac{e^{b_5+t c_5+x d_5}\left(-1+e^{b_5+t c_5+x d_5}\right) a_5 d_5^2}{\left(1+e^{b_5+t c_5+x d_5}\right)^3} - \frac{e^{b_6+t c_6+x d_6}\left(-1+e^{b_6+t c_6+x d_6}\right) a_6 d_6^2}{\left(1+e^{b_6+t c_6+x d_6}\right)^3}$$

The variables are the weights and the model parameter κ,

In[•]:=
```
vars = {a0, Table[{ai, bi, ci, di}, {i, 1, n}], κ} // Flatten
```

Out[•]=
$$\{a_0, a_1, b_1, c_1, d_1, a_2, b_2, c_2, d_2, a_3, b_3, c_3, d_3, a_4, b_4, c_4, d_4, a_5, b_5, c_5, d_5, a_6, b_6, c_6, d_6, \kappa\}$$

The model and data will be fitted to the measured sensor data measured at x = 2 at t ∈ [0,10], so the collocation points for the time t_i∈[0,10] and the corresponding measured pressure values p_i, i= 0,40.

In[•]:=
```
datat = Transpose[noisydata][[1]];
```

In[•]:=
```
datap = Transpose[noisydata][[2]];
```

The data error,

In[•]:=
```
dataerr = Mean[MapThread[Sqrt[((N /. {x → 2, t → #1}) - #2)^2] &, {datat, datap}]];
```

The model error at x =2 for any time

In[∘]:=
```
modelerr = Mean[Map[√((dt𝒩 - ϰ dx2𝒩)^2) /. {t → #1, x → 2} &, datat]];
```

The initial condition at x=2

In[∘]:=
```
iniconderr = √(((𝒩 /. {x → 2, t → 0}) - 1)^2);
```

In[∘]:=
```
Clear[ϰ]
ρ = 0.4;
```

In[∘]:=
```
G = dataerr + modelerr + iniconderr;
```

In[∘]:=
```
sol = NMinimize[{G, ϰ > 0}, vars]
```

Out[∘]=

$\{0.152062, \{a_0 \to 0.0937335, a_1 \to 0.1678, b_1 \to 0.0924803, c_1 \to 0.265322, d_1 \to -0.0921352, a_2 \to 0.444355, b_2 \to 0.130647, c_2 \to 0.202259, d_2 \to 0.13308, a_3 \to 0.461101, b_3 \to 0.0139889, c_3 \to -0.101652, d_3 \to -0.324678, a_4 \to 0.368121, b_4 \to -0.12316, c_4 \to 0.0594066, d_4 \to -0.237981, a_5 \to 0.20769, b_5 \to 0.114372, c_5 \to 0.687957, d_5 \to -0.0170683, a_6 \to 0.473863, b_6 \to 0.228777, c_6 \to -0.340583, d_6 \to -0.440351, \varkappa \to 1.05534\}\}$

In[∘]:=
```
myNet = 𝒩 /. sol[[2]]
```

Out[∘]=

$$0.0937335 + \frac{0.444355}{1 + e^{-0.130647 - 0.202259\,t - 0.13308\,x}} + \frac{0.20769}{1 + e^{-0.114372 - 0.687957\,t + 0.0170683\,x}} + \frac{0.1678}{1 + e^{-0.0924803 - 0.265322\,t + 0.0921352\,x}} + \frac{0.368121}{1 + e^{0.12316 - 0.0594066\,t + 0.237981\,x}} + \frac{0.461101}{1 + e^{-0.0139889 + 0.101652\,t + 0.324678\,x}} + \frac{0.473863}{1 + e^{-0.228777 + 0.340583\,t + 0.440351\,x}}$$

In[∘]:=
```
myϰ = ϰ /. sol[[2]]
```

Out[∘]=

1.05534

Software

Python - DeepXDE

https : // archive . softwareheritage . org/browse/origin/directory/?origin_url=https://github.com/lululxvi/deepxde