物理深層学習PINNの基本プログラミング講習

Physics **I**nformed **N**eural **N**etwork

岡崎智久
理化学研究所革新知能統合研究センター

内容

1. チュートリアル（約30分）
2. 実習（約15分）

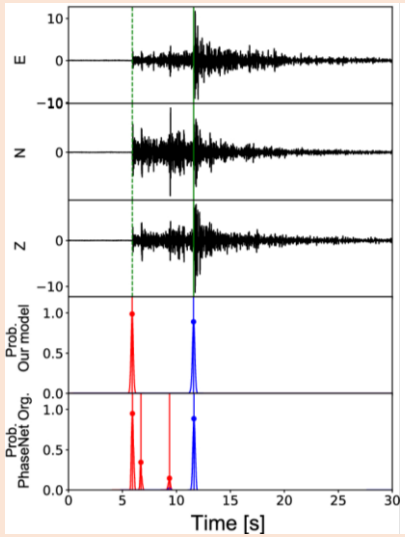
GitHub: https://github.com/okazakitomo/pinn_damped-oscillation

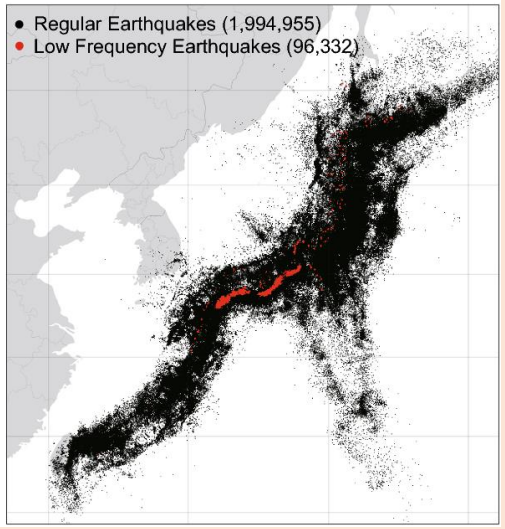
日本地震学会2025年度秋季大会ランチョンセミナー
2025年10月20日
主催：東京大学地震研究所共同利用特定共同研究B「科学的機械学習(SciML)による固体地球科学の加速」有志，大会･企画委員会

自然科学（地震学）におけるデータと理論

データ ← → 理論

深層学習



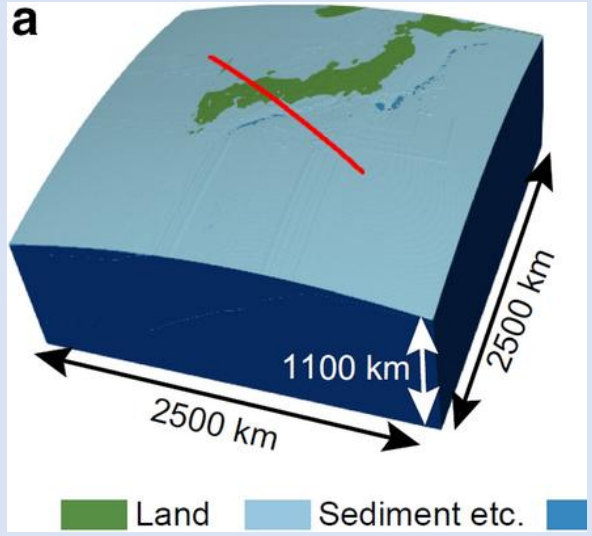


Naoi et al. (2024)

- データが豊富ならば圧倒的な性能
- 理論を活用しづらいのが惜しい。

物理深層学習
PINN

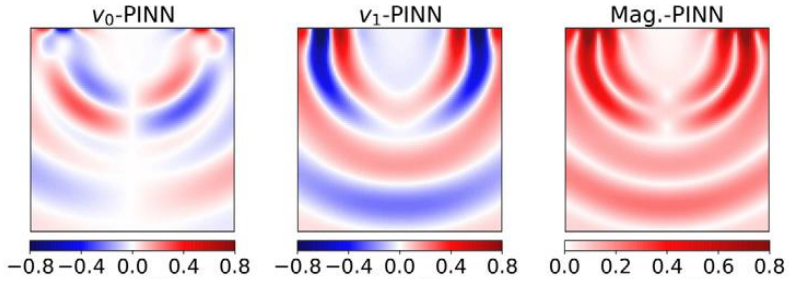
数値シミュレーション



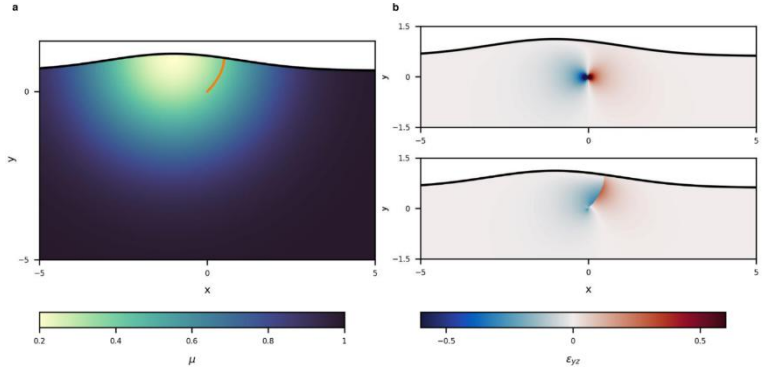
Hori et al. (2021)

- 物理法則に基づく高精細計算
- 観測を取込む柔軟性・計算量に課題

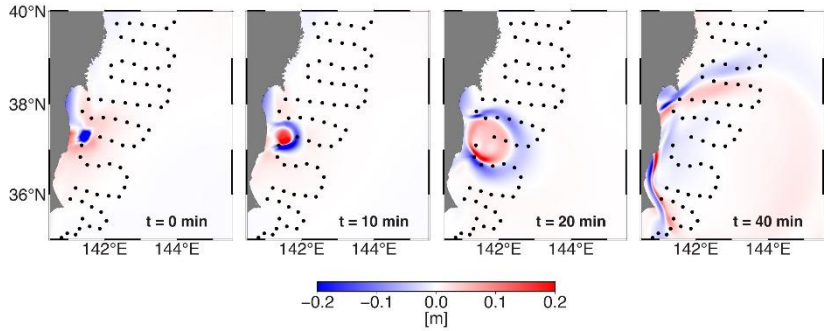
PINNによる地震モデリング研究



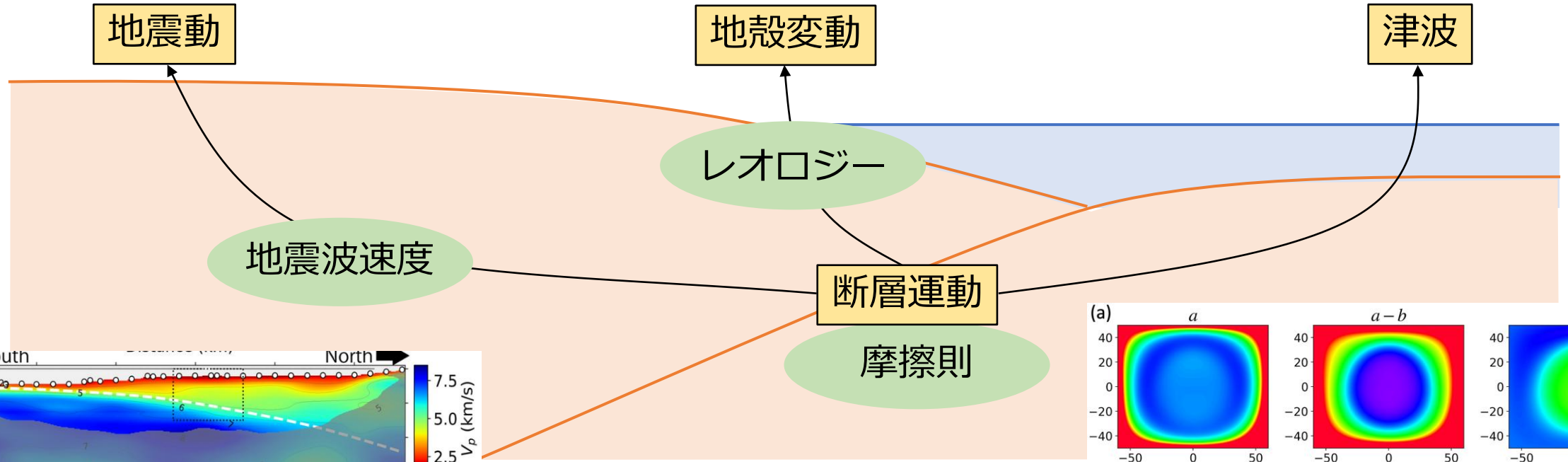
Ren et al. (2024)



Okazaki et al. (2022)



Someya & Furumura (2025)



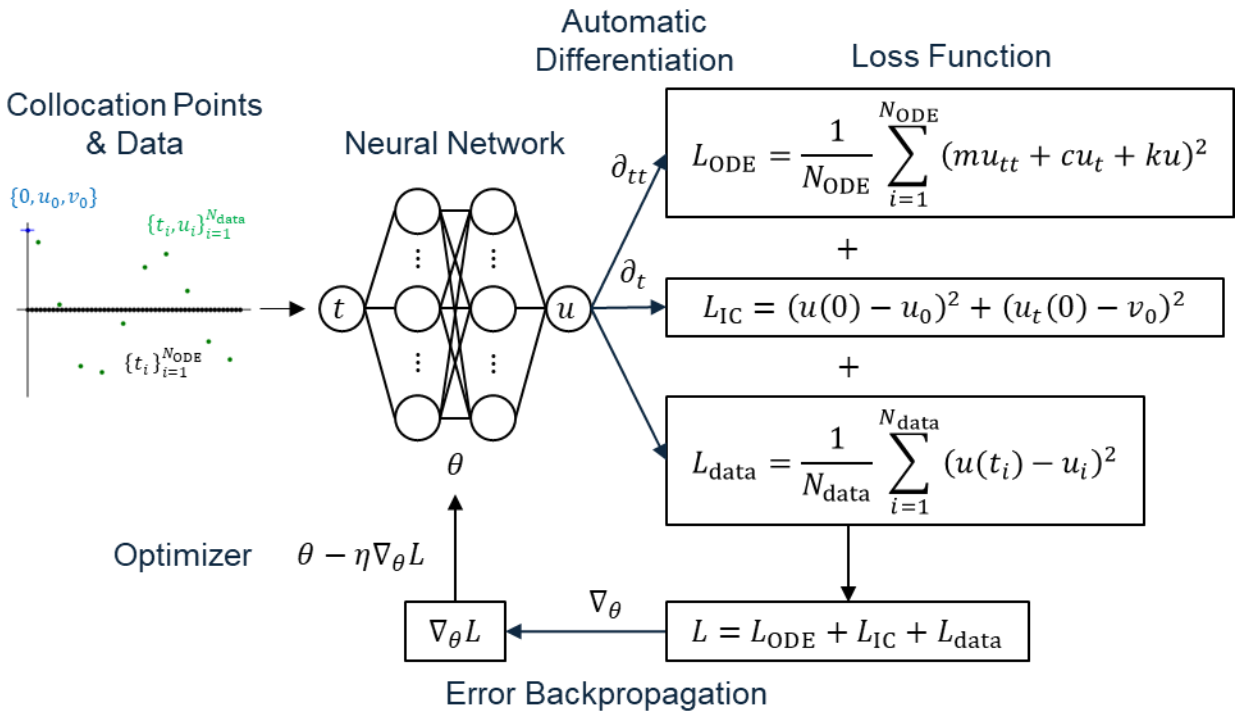
Agata et al. (2025)

Fukushima et al. (2025)

本チュートリアルの目的・内容

- PINNの「実装のための」基礎知識
- 順・逆解析を「できるだけ平易に」実装

GitHub: https://github.com/okazakitomo/pinn_damped-oscillation



理論編

解析対象：減衰振動

データ ⟵ ⟶ 理論

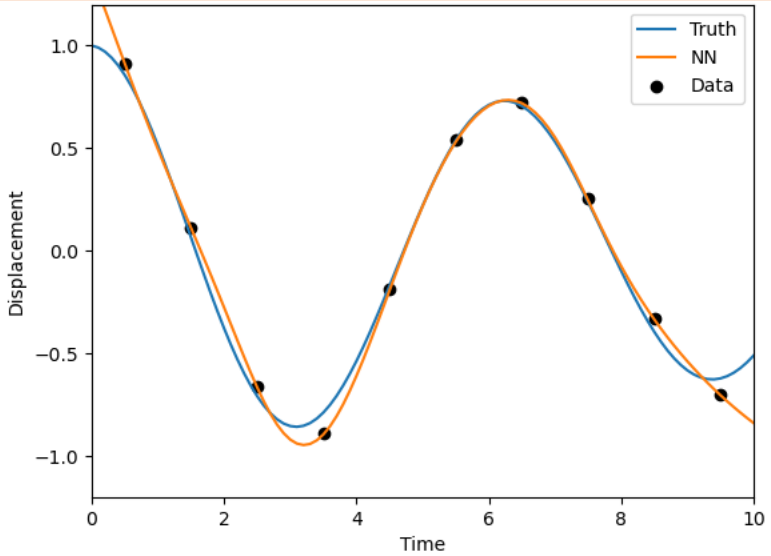
深層学習

1. 教師あり学習

データ $\{t_i, u_i\}_{i=1}^{N_{\text{data}}}$

⇓

解 $u(t)$



物理深層学習

3. 逆解析

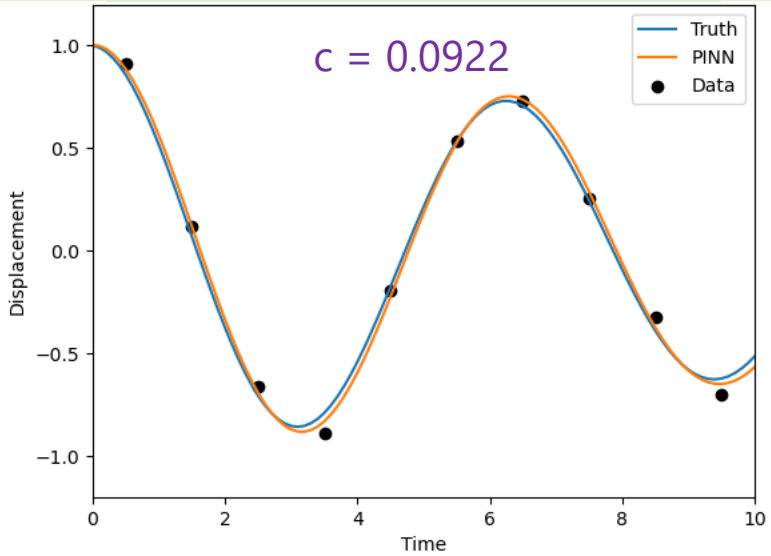
方程式 $\frac{d^2u}{dt^t} + c\frac{du}{dt} + u = 0$

初期値 $u(0) = 1, \frac{du}{dt}(0) = 0$

データ $\{t_i, u_i\}_{i=1}^{N_{\text{data}}}$

⇓

未知パラメタ c, 解 $u(t)$



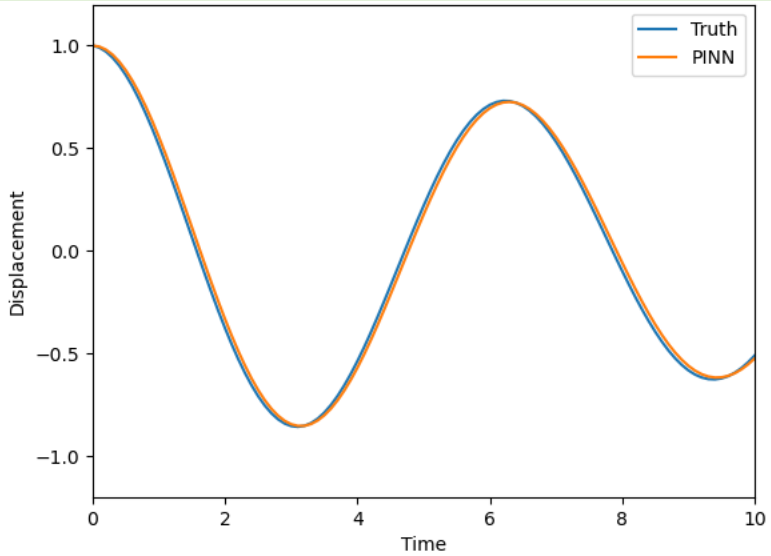
2. 順解析

方程式 $\frac{d^2u}{dt^t} + 0.1\frac{du}{dt} + u = 0$

初期値 $u(0) = 1, \frac{du}{dt}(0) = 0$

⇓

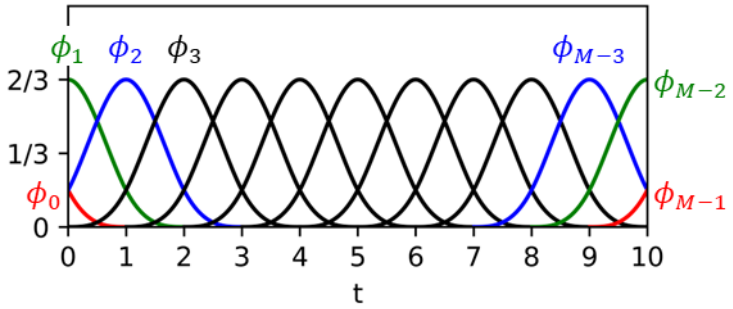
解 $u(t)$



線形回帰モデル

回帰問題（教師あり学習）

入出力データ$\{t_i, u_i\}_{i=1}^{N_{\text{data}}}$から未知関数$u = u(t)$を推定



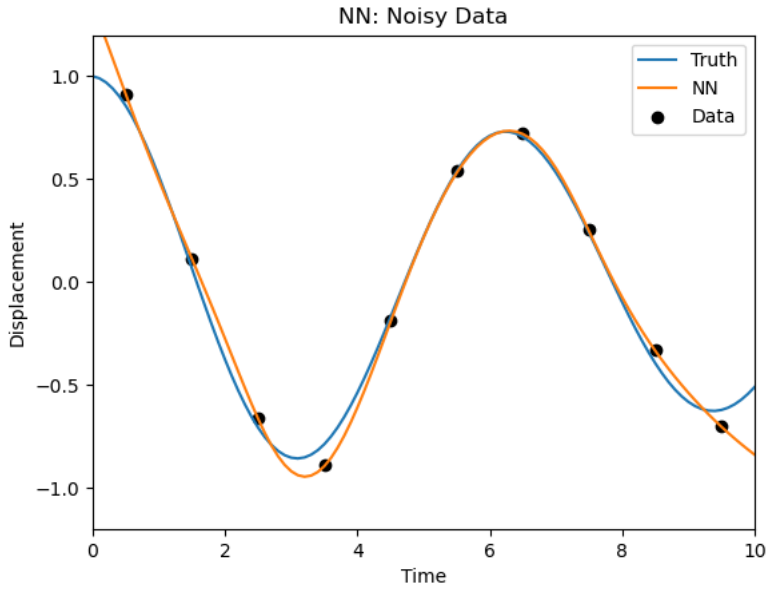
- **データ**：$\{t_i, u_i\}_{i=1}^{N_{\text{data}}}$
- **モデル**：$u_\theta(t) = \sum_{j=1}^{M} \theta_j \phi_j(t)$　パラメタ $\theta = (\theta_1, \ldots, \theta_M)$
- **損失関数**：$L_{\text{data}}(\theta) = \frac{1}{N_{\text{data}}} \sum_{i=1}^{N_{\text{data}}} (u_\theta(t_i) - u_i)^2$　データ残差

⇒**学習**：$\theta^* = \underset{\theta}{\operatorname{argmin}}\, L_{\text{data}}(\theta)$　（最小二乗解）

Neural Network (NN)

回帰問題（教師あり学習）

入出力データ$\{t_i, u_i\}_{i=1}^{N_{\text{data}}}$から未知関数$u = u(t)$を推定



- **データ**：$\{t_i, u_i\}_{i=1}^{N_{\text{data}}}$
- **モデル**：$u_\theta(t) = W_3\big(\sigma(W_2\sigma(W_1 t + b_1) + b_2)\big) + b_3$

  パラメタ $\theta = (W_1, b_1, W_2, b_2, W_3, b_3)$

  σ：非線形関数（固定）
- **損失関数**：$L_{\text{data}}(\theta) = \frac{1}{N_{\text{data}}}\sum_{i=1}^{N_{\text{data}}}(u_\theta(t_i) - u_i)^2$　データ残差

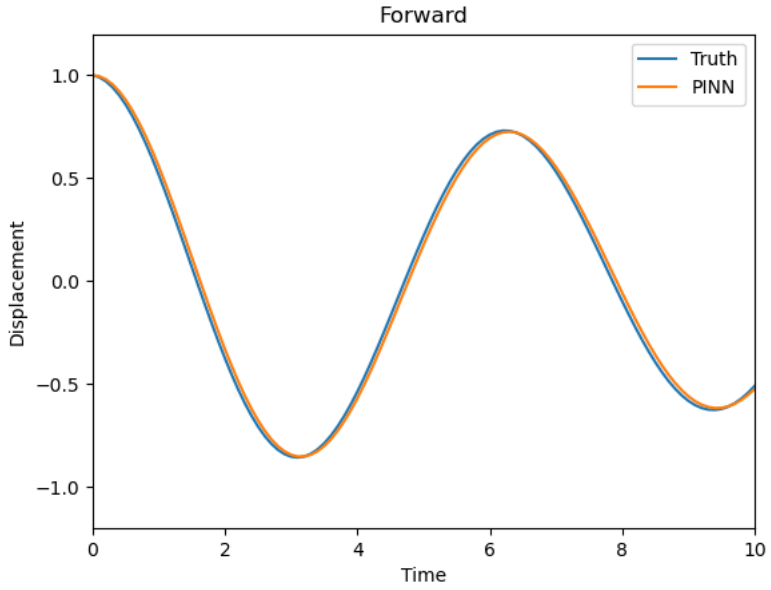
⇒**学習**：$\theta^* = \underset{\theta}{\operatorname{argmin}}\, L_{\text{data}}(\theta)$　（勾配降下法）

Physics Informed Neural Network (PINN)

微分方程式の解法

以下の初期値問題の解$u = u(t)$を求める：

$$m\frac{d^2u}{dt^2} + c\frac{du}{dt} + ku = 0 \text{ subject to } u(0) = u_0, \frac{du}{dt}(0) = v_0$$



（データ）

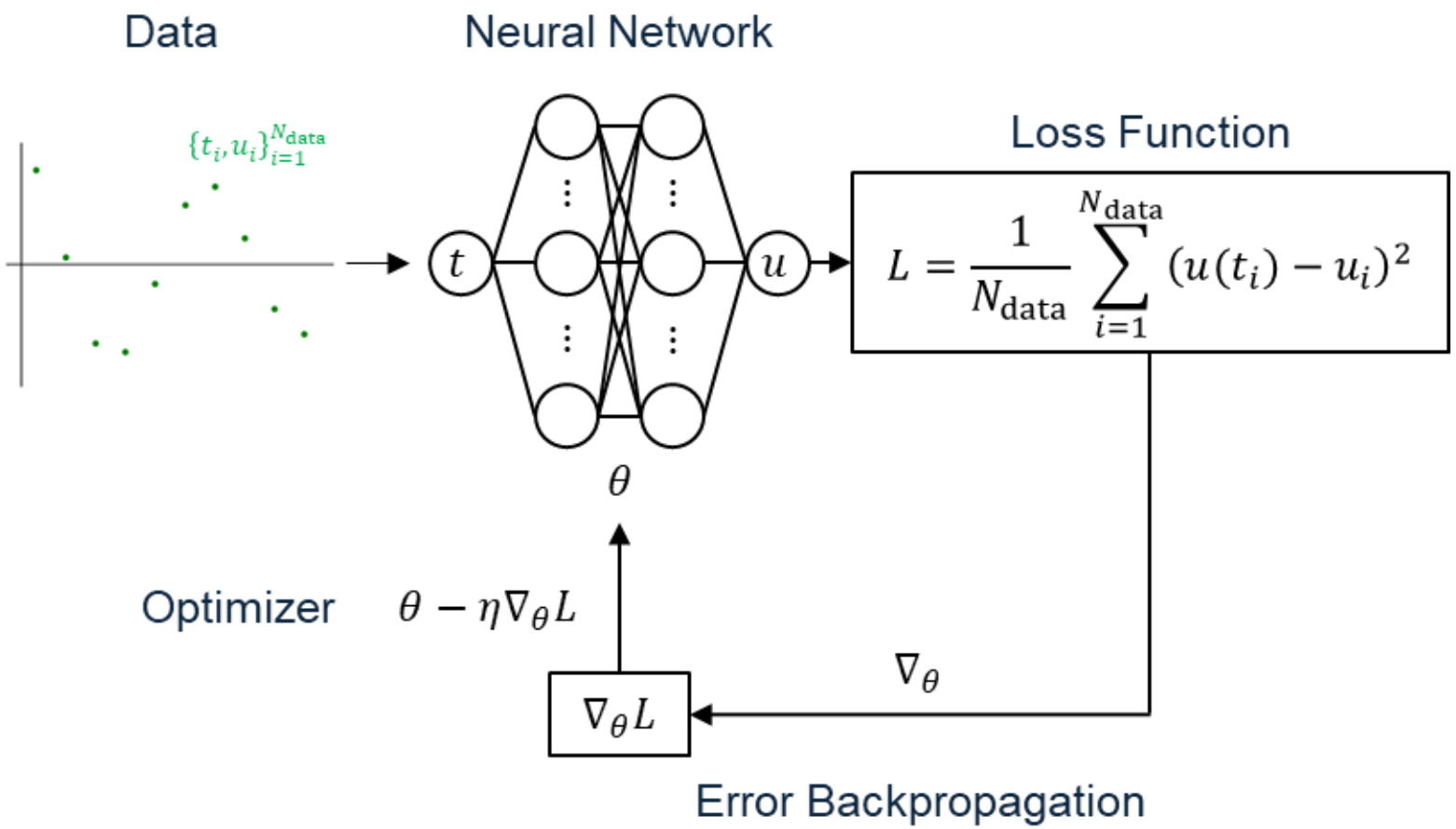
- **モデル**：$u_\theta(t) = W_3\big(a(W_2a(W_1t + b_1) + b_2)\big) + b_3$

  パラメタ $\theta = (W_1, b_1, W_2, b_2, W_3, b_3)$

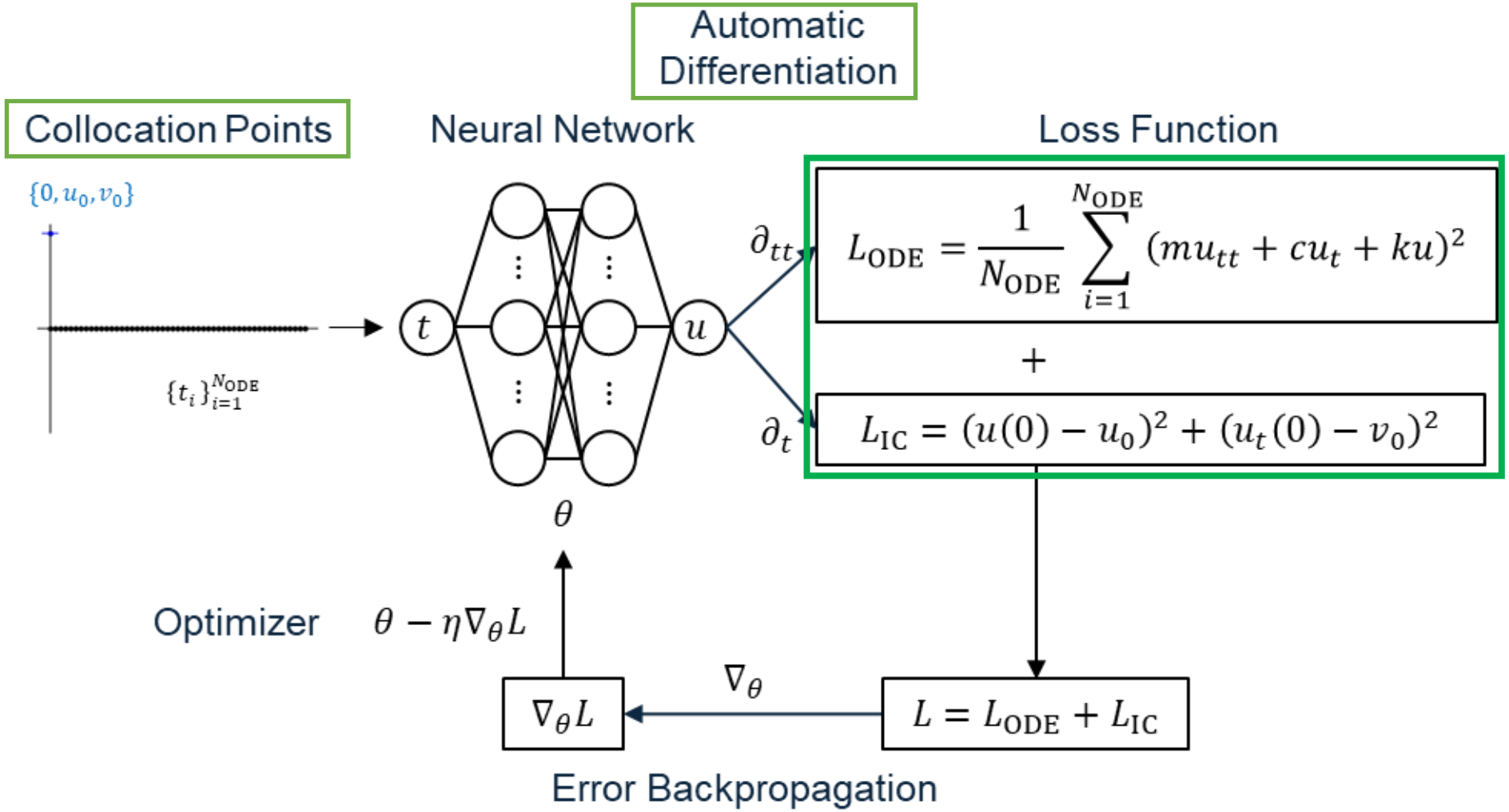
- **損失関数**： $L_{\text{phys}}(\theta) = \frac{1}{N_{\text{ODE}}}\sum_{i=1}^{N_{\text{ODE}}}(mu_{tt} + cu_t + ku)^2 + (u(0) - u_0)^2 + (u_t(0) - v_0)^2$

⇒**学習**：$\theta^* = \underset{\theta}{\text{argmin}}\, L_{\text{phys}}(\theta)$　　（勾配降下法）

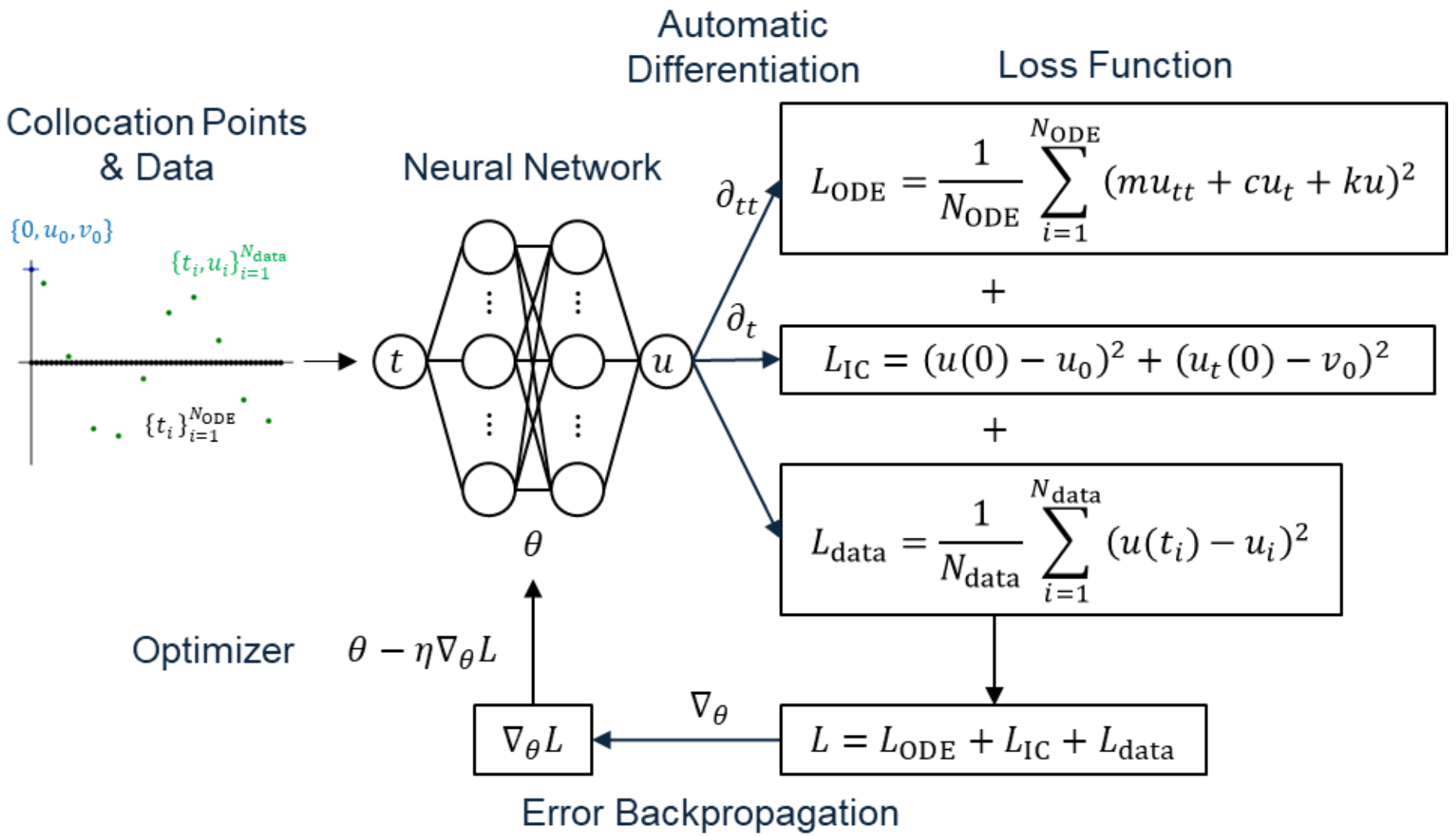
NNの基本構成（１．教師あり学習）



PINNの基本構成（２．順解析）



PINNの基本構成（３．逆解析）



実装編

GitHub: https://github.com/okazakitomo/pinn_damped-oscillation

解析対象：減衰振動

プログラミング環境：Python（3系）, PyTorch, Numpy, Matplotlib

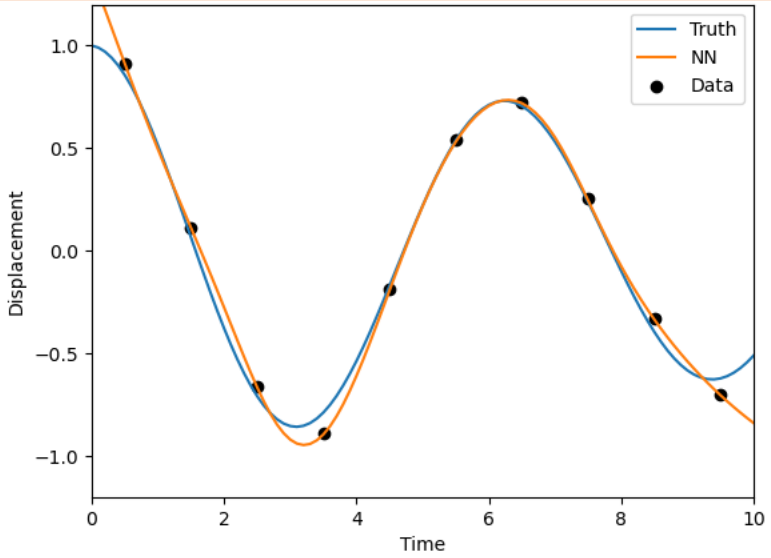
深層学習

1. 教師あり学習

データ $\{t_i, u_i\}_{i=1}^{N_{\text{data}}}$

⇓

解 $u(t)$



物理深層学習

3. 逆解析

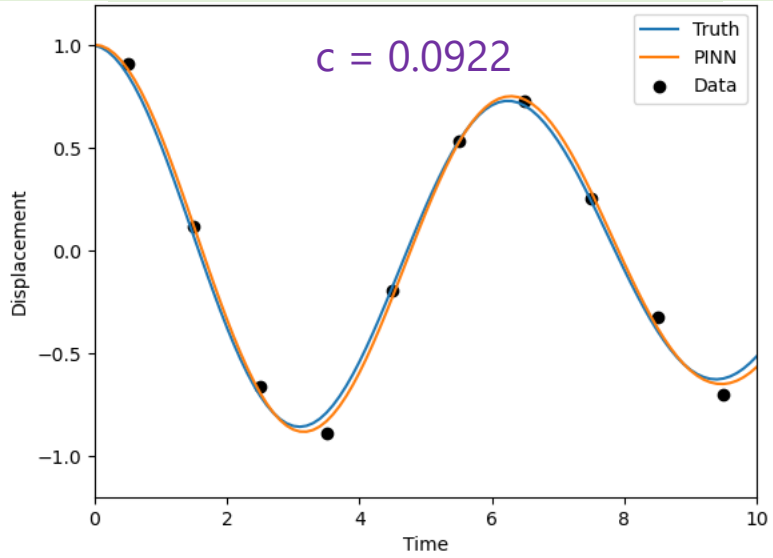
方程式 $\frac{d^2u}{dt^t} + c\frac{du}{dt} + u = 0$

初期値 $u(0) = 1, \frac{du}{dt}(0) = 0$

データ $\{t_i, u_i\}_{i=1}^{N_{\text{data}}}$

⇓

未知パラメタ c, 解 $u(t)$



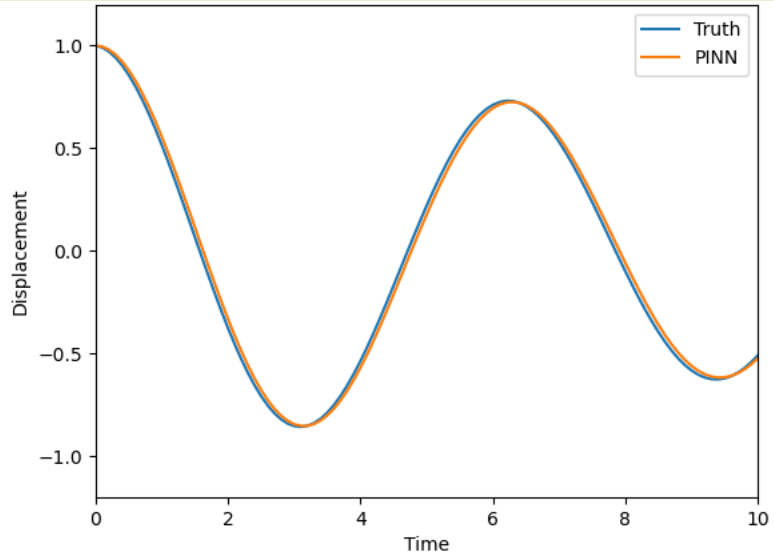
2. 順解析

方程式 $\frac{d^2u}{dt^t} + 0.1\frac{du}{dt} + u = 0$

初期値 $u(0) = 1, \frac{du}{dt}(0) = 0$

⇓

解 $u(t)$



0．データセットの準備

入出力の組 $\{(t_{\mathrm{data}}, u_{\mathrm{data}})\}$を作成しテキストファイルに保存

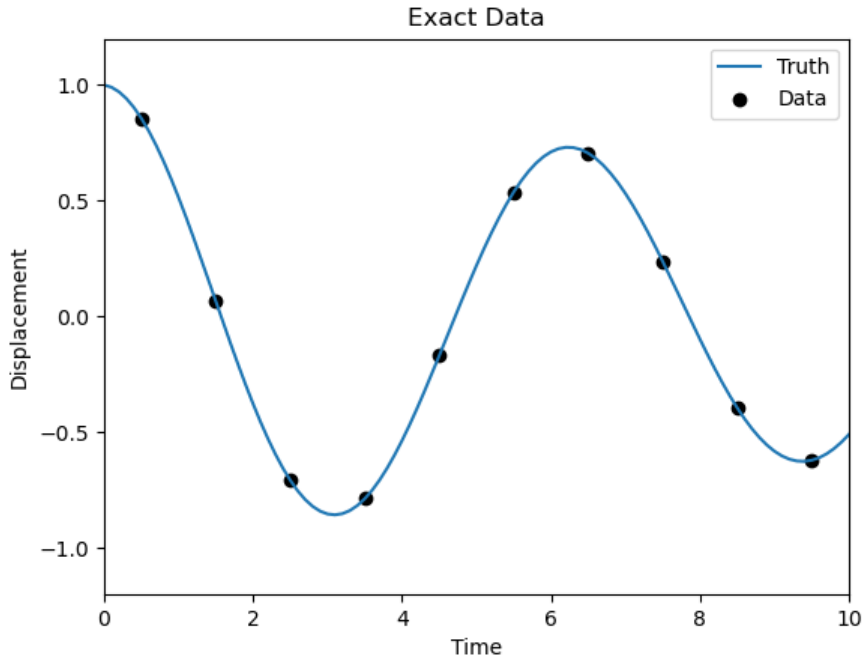
● 真値

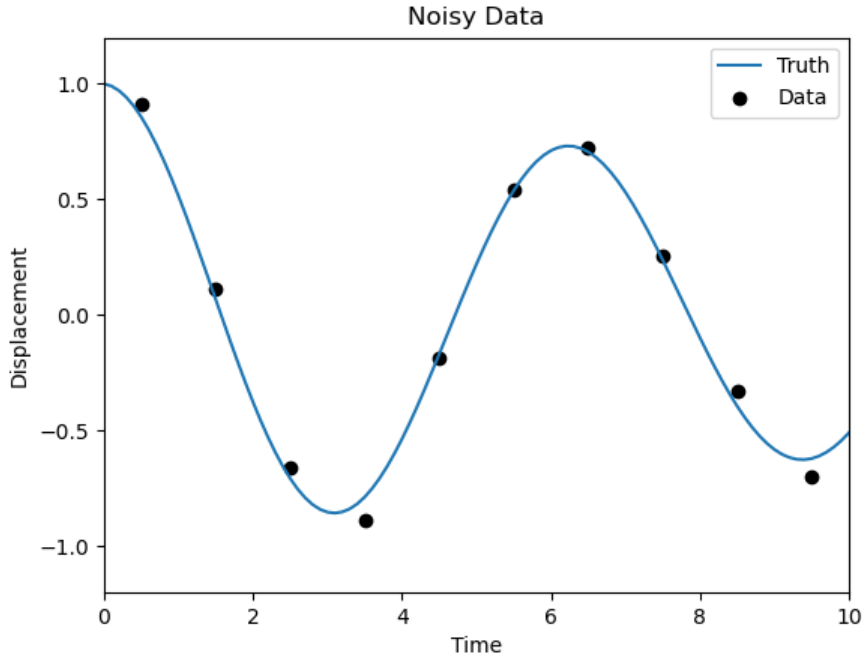
101点 $t = \{0.00, 0.01, 0.02, \ldots, 0.99, 1.00\}$ における理論解

● 教師データ

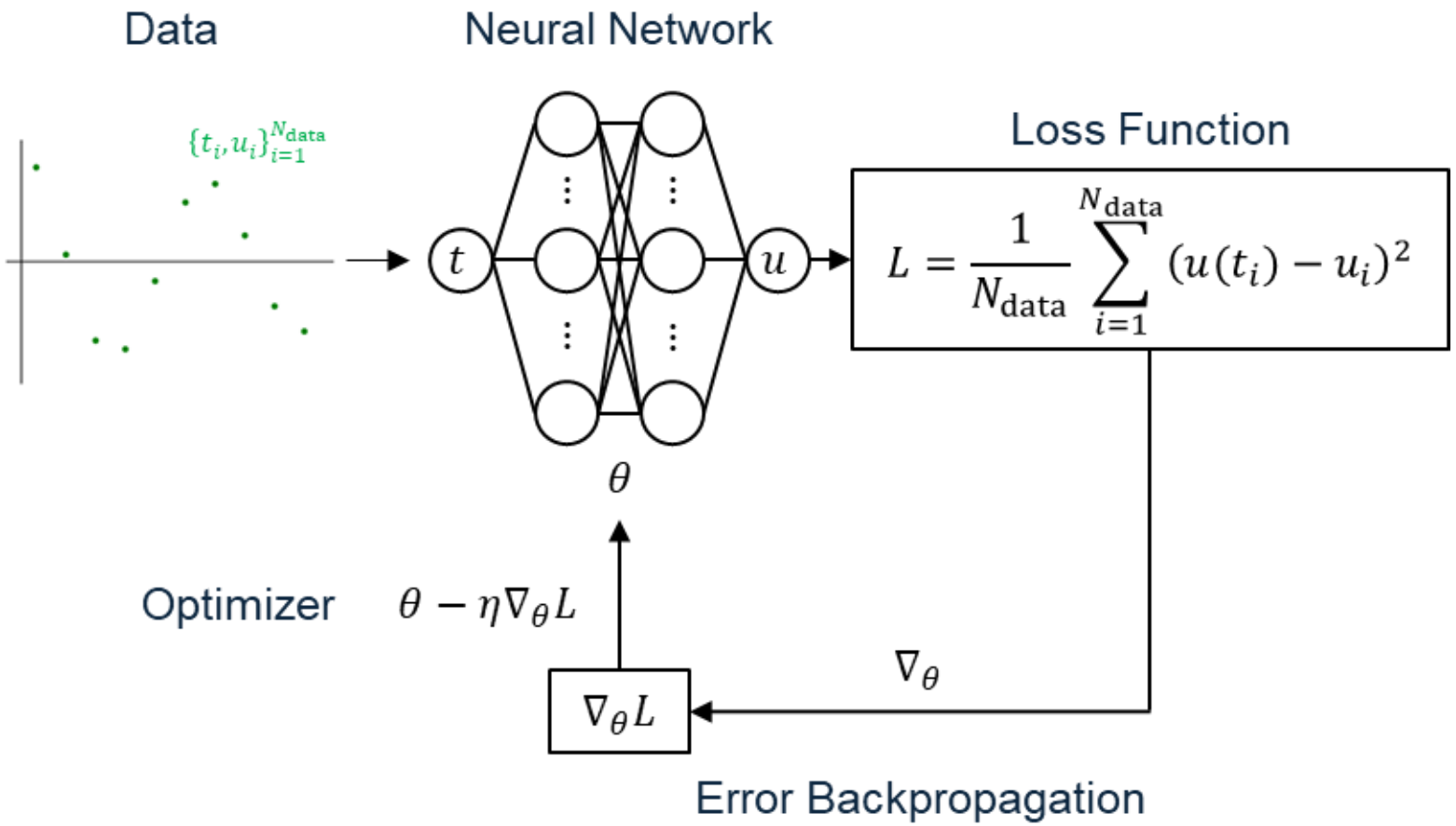
10点 $t = \{0.5, 1.5, 2.5, \ldots, 8.5, 9.5\}$ において

1. Exact data: 理論解
2. Noisy data: 理論解＋独立正規分布（標準偏差0.05）





1．教師あり学習


Data
Neural Network
$\{t_i, u_i\}_{i=1}^{N_{\text{data}}}$
t
u
Loss Function
$L = \frac{1}{N_{\text{data}}} \sum_{i=1}^{N_{\text{data}}} (u(t_i) - u_i)^2$
θ
Optimizer
$\theta - \eta \nabla_\theta L$
∇_θ
$\nabla_\theta L$
Error Backpropagation

1．教師あり学習（実装1/2）

```
class NN(nn.Module):
    def __init__(self):
        super(NN, self).__init__()
        self.fc1 = nn.Linear(1, 20)
        self.fc2 = nn.Linear(20, 20)
        self.fc3 = nn.Linear(20, 1)

    def forward(self, x):
        x = torch.tanh(self.fc1(x))
        x = torch.tanh(self.fc2(x))
        x = self.fc3(x)
        return x
```

- **ニューラル・ネットワーク**
- 全結合層：入力1→20→20→出力1
- 活性化関数：tanh
- 順伝播： $u = f^{\mathrm{NN}}(t)$

```
def loss_function(net, t_data, u_data):

    # Data
    u_pred = net(t_data)
    loss_data = torch.mean((u_pred - u_data) ** 2)

    return loss_data
```

- **損失関数**

$$u_{\mathrm{pred}} = f^{\mathrm{NN}}(t_{\mathrm{data}})$$

$$L_{\mathrm{data}}(t_{\mathrm{data}}, u_{\mathrm{data}}) = \frac{1}{N_{\mathrm{data}}} \sum (u_{\mathrm{pred}} - u_{\mathrm{data}})^2$$

1．教師あり学習（実装2/2）

```
tu_data = torch.from_numpy(np.loadtxt('data/data_exact.txt', dtype='float32'))
t_data = tu_data[:,[0]]
u_data = tu_data[:,[1]]
```

- **データ読込み**

```
net = NN()
optimizer = torch.optim.Adam(net.parameters(), lr=0.001)
```

- **モデル定義**

NN構造・最適化関数(Adam)

```
for epoch in range(num_epoch):
    optimizer.zero_grad()
    loss = loss_function(net, t_data, u_data)
    loss.backward()
    optimizer.step()
```

- **訓練**

勾配の初期化
損失関数
誤差逆伝播
パラメタ更新

```
net.eval()
with torch.no_grad():
    u_pred = net(t_true)
```

- **推論**

1．教師あり学習（実行）

```
programs> python 1a_nn_exact.py
```

Pythonプログラム1a_nn_exact.pyの実行

- 画面出力

```
#epoch= 10000
Epoch 0 Loss: 6.7383e-01
Epoch 1000 Loss: 4.7363e-03
Epoch 2000 Loss: 1.0838e-03
Epoch 3000 Loss: 1.0647e-04
Epoch 4000 Loss: 1.6798e-06
Epoch 5000 Loss: 2.5085e-09
Epoch 6000 Loss: 1.9962e-09
Epoch 7000 Loss: 8.1482e-08
Epoch 8000 Loss: 4.7654e-06
Epoch 9000 Loss: 1.4591e-06
Epoch 10000 Loss: 1.7120e-07
Ablolute error: 0.0329
Relative error: 0.0574
```

解析条件

学習過程

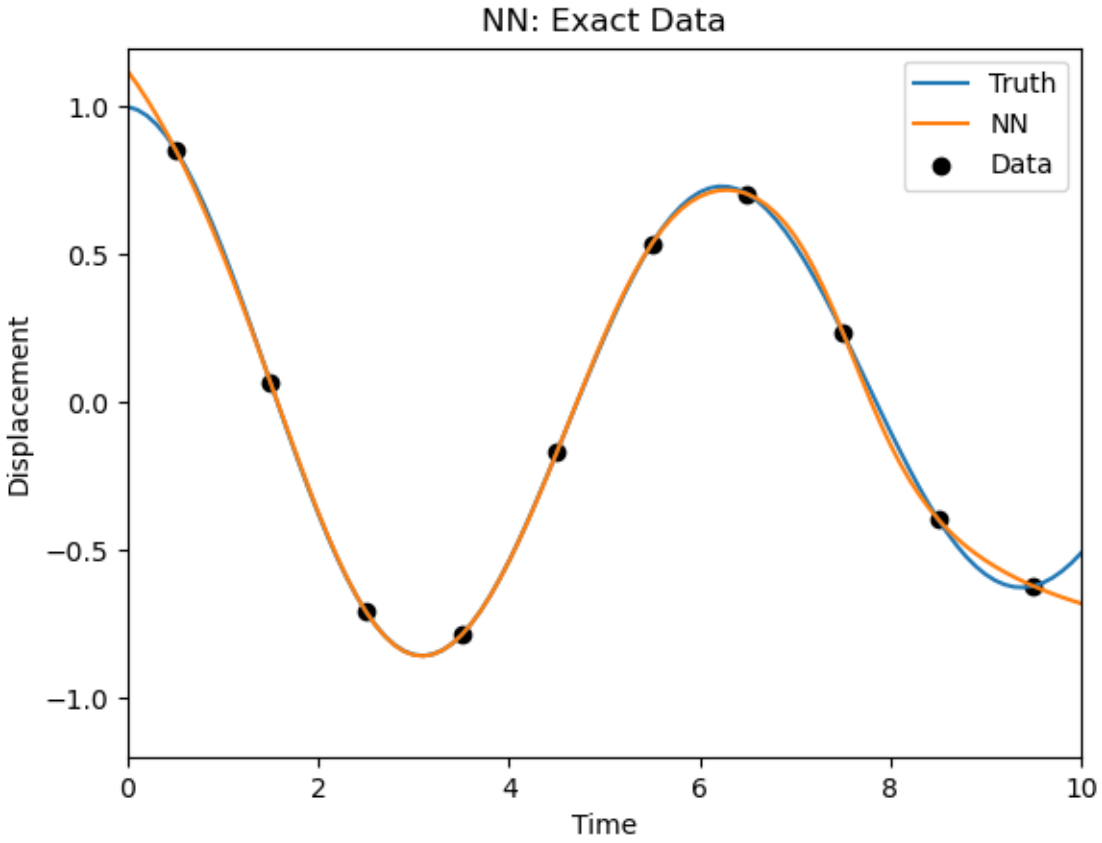
誤差評価

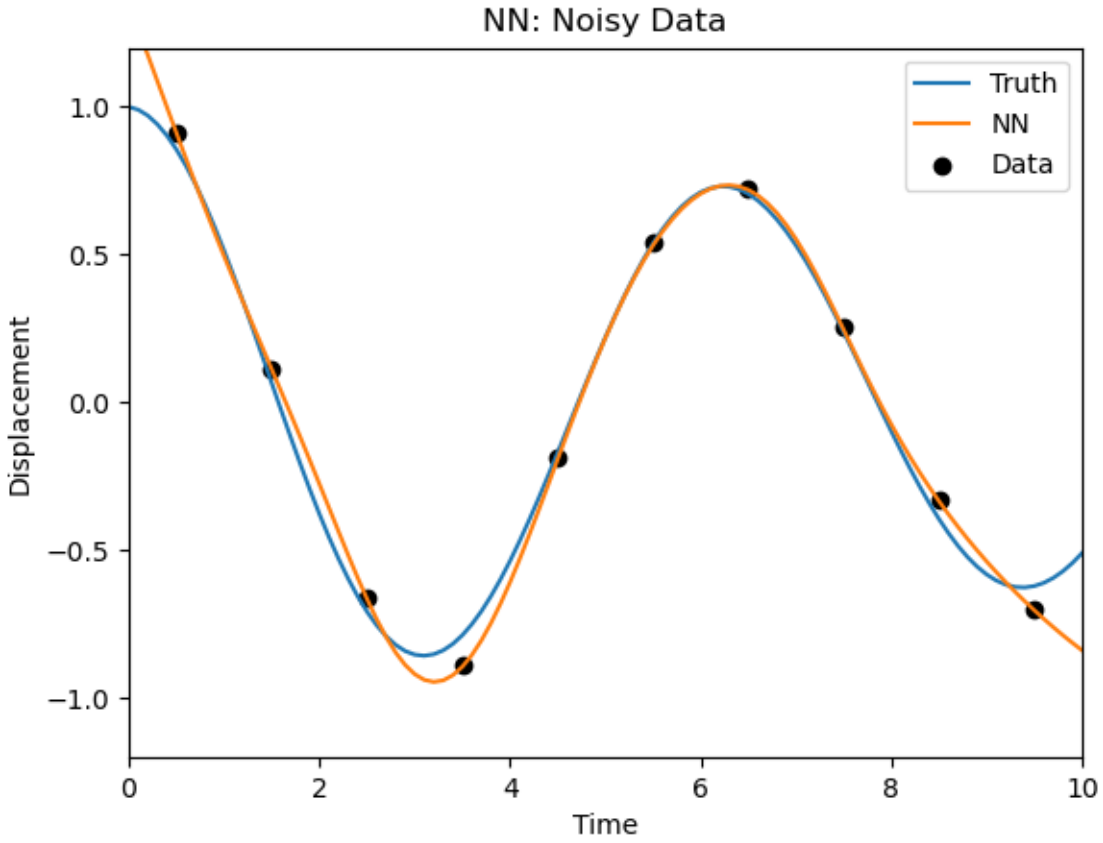
- ファイル出力

```
result:損失関数 (loss_1a_nn_exact.txt)
       訓練済みNN (net_1a_nn_exact.ckpt)
figure:推定結果 (1a_nn_exact.png)
       学習曲線 (loss_1a_nn_exact.png)
```

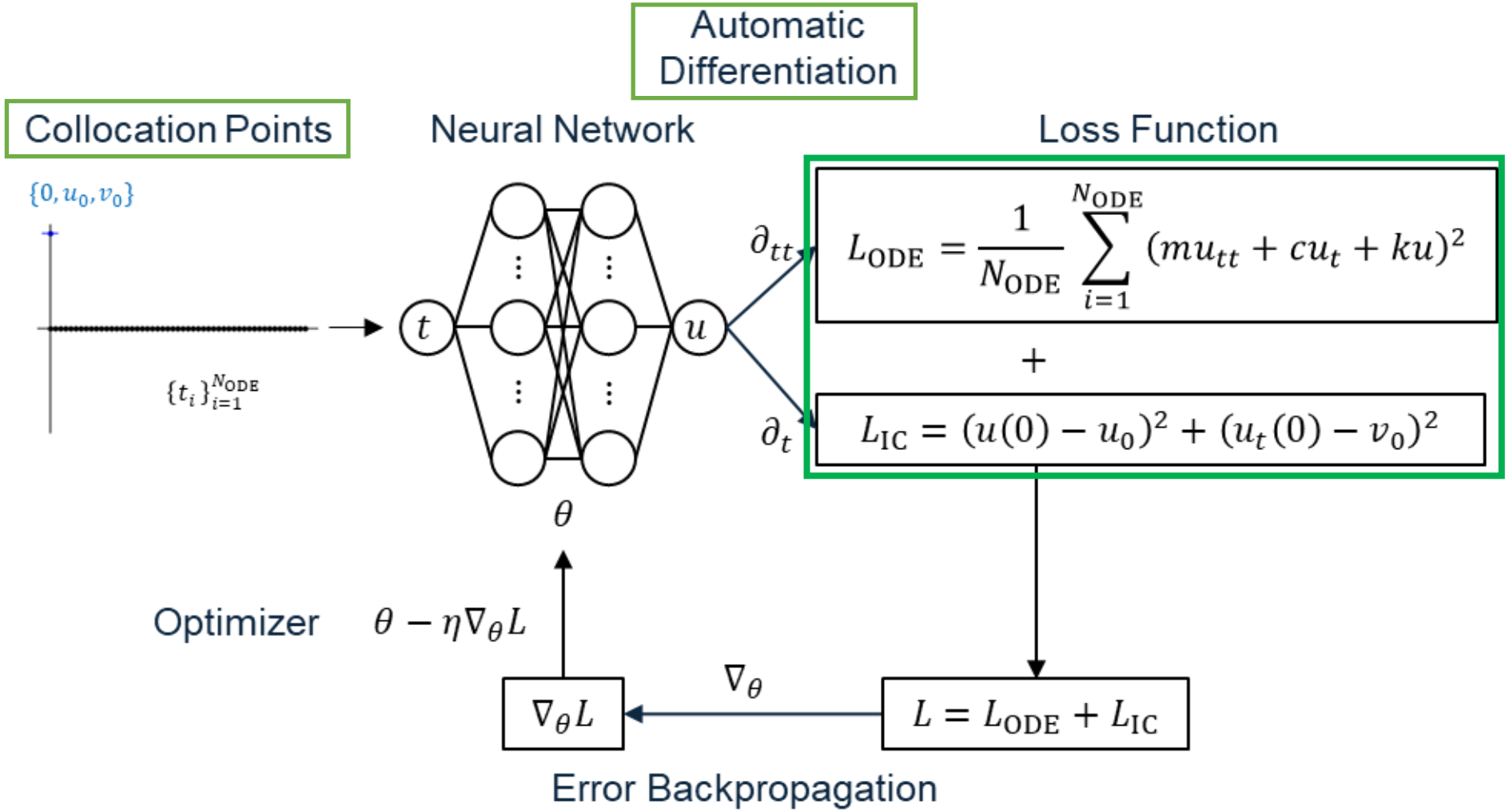
（参考）
手元のノートPCで
実行時間：約30秒

１．教師あり学習（解析結果）





2．順解析


Automatic Differentiation
Collocation Points
Neural Network
Loss Function
$\{0, u_0, v_0\}$
$\{t_i\}_{i=1}^{N_{ODE}}$
t
u
∂_{tt}
∂_t
$L_{ODE} = \frac{1}{N_{ODE}} \sum_{i=1}^{N_{ODE}} (mu_{tt} + cu_t + ku)^2$
+
$L_{IC} = (u(0) - u_0)^2 + (u_t(0) - v_0)^2$
θ
Optimizer
$\theta - \eta \nabla_\theta L$
$\nabla_\theta L$
∇_θ
$L = L_{ODE} + L_{IC}$
Error Backpropagation

B1. 自動微分

```
def y(x):
    return x ** 2 + torch.sin(x)

# Input values
x = torch.tensor([-1., 0., 1.], requires_grad=True)

# Output values
y = y(x)
dy_auto = torch.autograd.grad(y, x, grad_outputs=torch.ones_like(y), create_graph=True)[0]
ddy_auto = torch.autograd.grad(dy_auto, x, grad_outputs=torch.ones_like(y))[0]
```

NNを含む様々な関数の指定された点における導関数を計算機精度で効率的に計算できる。

- 実行画面

```
>>> x
tensor([-1.,  0.,  1.], requires_grad=True)
>>> y
tensor([0.1585, 0.0000, 1.8415], grad_fn=<AddBackward0>)
>>> dy_auto
tensor([-1.4597,  1.0000,  2.5403], grad_fn=<AddBackward0>)
>>> ddy_auto
tensor([2.8415, 2.0000, 1.1585])
```

$$y = x^2 + \sin(x)$$
$$\frac{dy}{dx} = 2x + \cos(x)$$
$$\frac{d^2y}{dx^2} = 2 - \sin(x)$$

$$\sin(1) \approx 0.8415$$
$$\cos(1) \approx 0.5403$$

2．順解析（実装1/2）

```
# Physical constants
m = 1.0  # Mass
c = 0.1  # Damping coefficient
k = 1.0  # Spring constant

# Initial conditions
u0 = 1.0  # Initial displacement
v0 = 0.0  # Initial velocity
```

- **物理条件**

物理定数 m, c, k
初期条件 u_0, v_0

※教師あり学習では不要だった。
　PINN順解析では物理条件を完全に指定する必要がある。

```
tmin = 0
tmax = 10
num_coll = 101  # Number of collocation points

def uniform_grid(tmin, tmax, num_coll):
    return torch.linspace(tmin, tmax, num_coll,
        requires_grad=True).view(-1, 1)

t_coll = uniform_grid(tmin, tmax, num_coll)
```

- **選点**

選点の範囲 $[t_{\min}, t_{\max}]$と個数N_{coll}を指定

一様グリッドを生成
`requires_grad=True`により自動微分を可能に。

2．順解析（実装2/2）

本チュートリアルのハイライト

```
def loss_function(net, t_coll, m, c, k, u0, v0):
    # ODE
    u = net(t_coll)  # u(t)
    u_t = torch.autograd.grad(u, t_coll,
        grad_outputs=torch.ones_like(u),
        create_graph=True)[0]  # du/dt
    u_tt = torch.autograd.grad(u_t, t_coll,
        grad_outputs=torch.ones_like(u_t),
        create_graph=True)[0]  # d2u/dt2
    r = m * u_tt + c * u_t + k * u  # ODE residual
    loss_ode = torch.mean(r ** 2)
    # IC
    t0 = torch.zeros((1, 1), requires_grad=True)
    u0_pred = net(t0) # u(0)
    v0_pred = torch.autograd.grad(u0_pred, t0,
        grad_outputs=torch.ones_like(u0_pred),
        create_graph=True)[0]  # du/dt(0)
    loss_ic = torch.mean((u0_pred - u0) ** 2
        + (v0_pred - v0) ** 2)
    # Total
    loss = loss_ode + loss_ic
    return loss, loss_ode, loss_ic
```

- **損失関数**

物理定数 m, c, k
初期条件 u_0, v_0

自動微分

ODE残差 $r = mu_{tt} + cu_t + ku$
ODE損失 $L_{\mathrm{ODE}} = \frac{1}{N_{\mathrm{ODE}}} \sum_{i=1}^{N_{\mathrm{ODE}}} r^2$

IC残差 $\mathbf{r} = (u - u_0, u_t - v_0)$
IC損失 $L_{\mathrm{IC}} = \frac{1}{N_{\mathrm{IC}}} \sum_{i=1}^{N_{\mathrm{IC}}} \|\mathbf{r}\|^2$

損失関数 $L = L_{\mathrm{ODE}} + L_{\mathrm{IC}}$

2．順解析（実行）

```
programs> python 2_forward.py
```

Pythonプログラム2_forward.pyの実行

```
m = 1.0 , c = 0.1 , k = 1.0
u0 = 1.0 , v0 = 0.0
tmin = 0 , tmax = 10
#coll = 101 , #epoch= 10000
Epoch 0 Loss: 9.1625e-01 ODE: 2.2456e-03 IC: 9.1400e-01
Epoch 1000 Loss: 2.6881e-02 ODE: 2.5966e-02 IC: 9.1530e-04
Epoch 2000 Loss: 1.2663e-02 ODE: 1.2441e-02 IC: 2.2209e-04
Epoch 3000 Loss: 9.5411e-03 ODE: 9.4369e-03 IC: 1.0426e-04
Epoch 4000 Loss: 4.2631e-03 ODE: 4.2242e-03 IC: 3.8916e-05
Epoch 5000 Loss: 1.8358e-03 ODE: 1.8209e-03 IC: 1.4851e-05
Epoch 6000 Loss: 5.5981e-04 ODE: 5.5596e-04 IC: 3.8480e-06
Epoch 7000 Loss: 1.5442e-04 ODE: 1.5358e-04 IC: 8.4452e-07
Epoch 8000 Loss: 5.1221e-05 ODE: 5.1033e-05 IC: 1.8792e-07
Epoch 9000 Loss: 2.4638e-05 ODE: 2.4584e-05 IC: 5.3964e-08
Epoch 10000 Loss: 1.6149e-05 ODE: 1.6129e-05 IC: 1.9765e-08
Ablolute error: 0.0281
Relative error: 0.0490
```

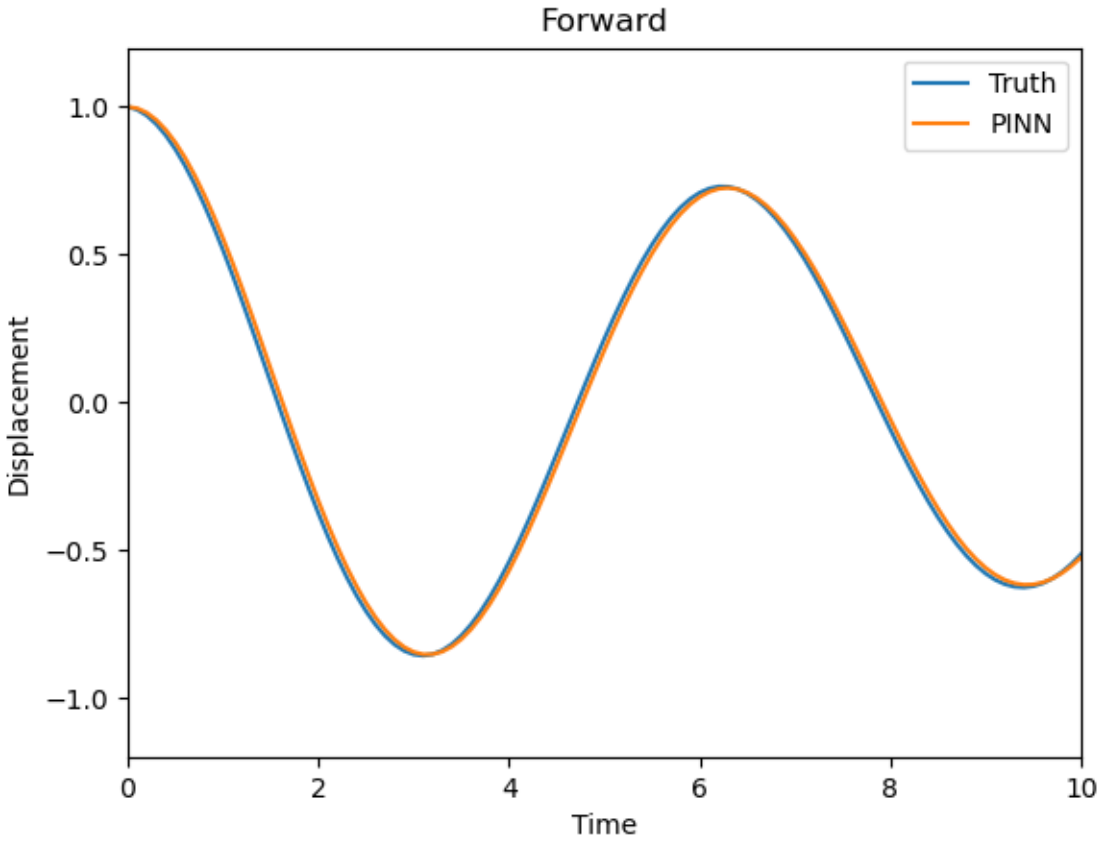
解析条件

学習過程

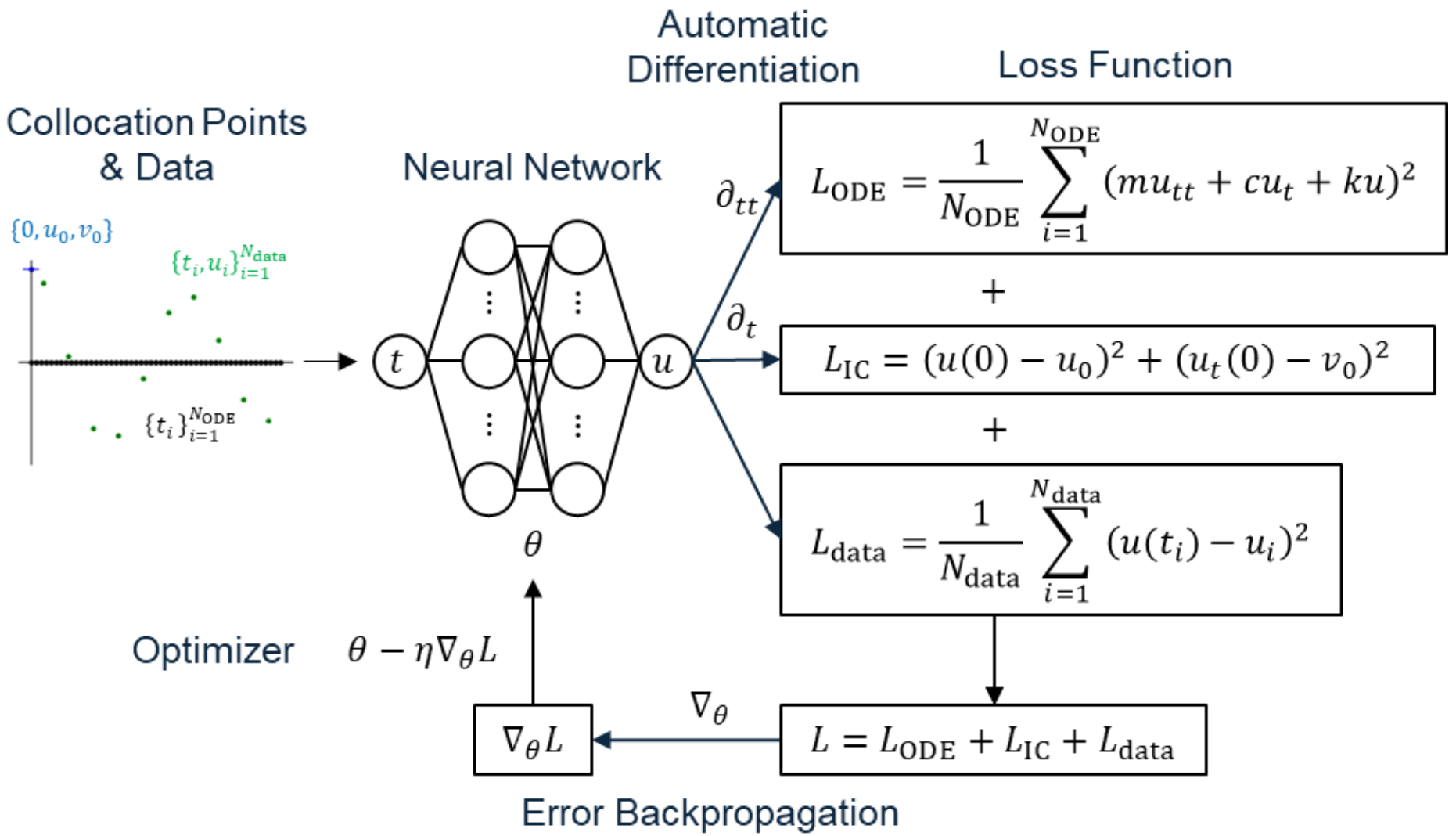
誤差評価

（参考）
手元のノートPCで
実行時間：約70秒

2．順解析（解析結果）



3．逆解析


Automatic
Differentiation
Loss Function
Collocation Points
& Data
Neural Network
$\{0, u_0, v_0\}$
$\{t_i, u_i\}_{i=1}^{N_{\text{data}}}$
$\{t_i\}_{i=1}^{N_{\text{ODE}}}$
t
u
∂_{tt}
∂_t
$L_{\text{ODE}} = \frac{1}{N_{\text{ODE}}} \sum_{i=1}^{N_{\text{ODE}}} (mu_{tt} + cu_t + ku)^2$
+
$L_{\text{IC}} = (u(0) - u_0)^2 + (u_t(0) - v_0)^2$
+
$L_{\text{data}} = \frac{1}{N_{\text{data}}} \sum_{i=1}^{N_{\text{data}}} (u(t_i) - u_i)^2$
θ
Optimizer
$\theta - \eta \nabla_\theta L$
∇_θ
$\nabla_\theta L$
$L = L_{\text{ODE}} + L_{\text{IC}} + L_{\text{data}}$
Error Backpropagation

3．逆解析（実装1/1）

```
c0 = 1.0  # Initial guess for c

c_est = torch.tensor(c0, dtype=torch.float32, requires_grad=True)
params = [c_est]

optimizer = optim.Adam(list(net.parameters()) + params, lr=0.001)
```

訓練可能パラメタ

- NNパラメタ：net.parameters()
- 推定パラメタ：params = [c_est]（初期化 c0 = 1.0）

まとめて最適化関数の引数とする。

損失関数：$L = L_{\mathrm{ODE}} + L_{\mathrm{IC}} + L_{\mathrm{data}}$

```
# Loss function
def loss_function(net, t_coll, t_data, u_data, m, c, k, u0, v0):
     (中略)
    loss = loss_ode + loss_ic + loss_data
    return loss, loss_ode, loss_ic, loss_data

loss = loss_function(net, t_coll, t_data, u_data, m, c_est, k, u0, v0)
```

3．逆解析（実行）

```
programs> python 3a_inverse_exact.py
```

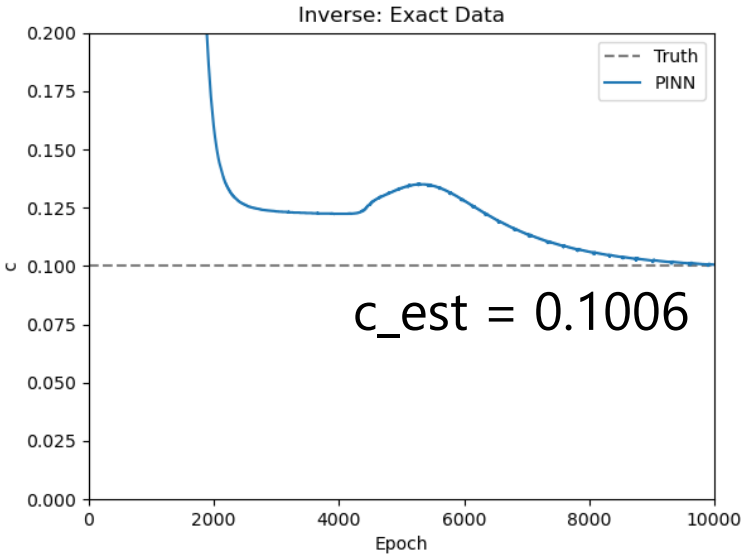
Pythonプログラム3a_inverse_exact.pyの実行

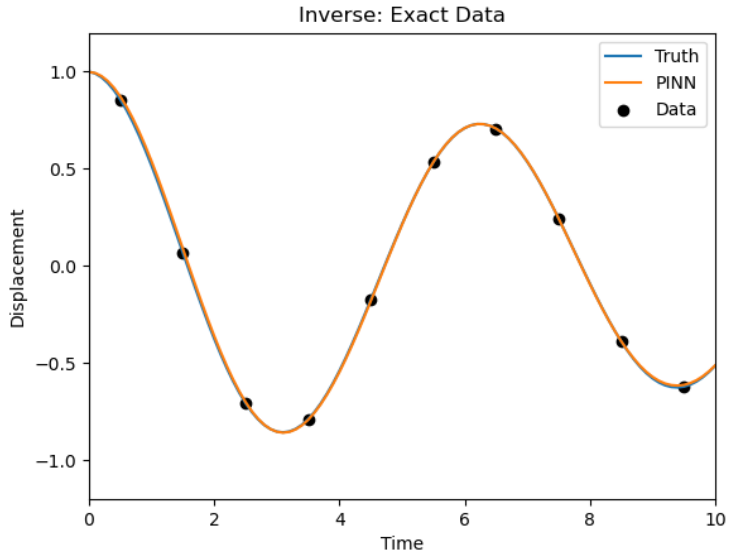
（参考）
手元のノートPCで
実行時間：約80秒

```
m = 1.0 , c = 0.1 , k = 1.0
u0 = 1.0 , v0 = 0.0
c0 = 1.0
tmin = 0 , tmax = 10
#coll = 101 , #epoch= 10000
Epoch 0 Loss: 2.1064e+00 ODE: 1.4204e-01 IC: 1.5356e+00 Data: 4.2870e-01 c: 0.9990
Epoch 1000 Loss: 2.0303e-01 ODE: 1.2657e-02 IC: 5.0307e-04 Data: 1.8987e-01 c: 0.5606
Epoch 2000 Loss: 4.8740e-02 ODE: 2.1929e-02 IC: 5.1751e-05 Data: 2.6759e-02 c: 0.1590
Epoch 3000 Loss: 3.2205e-02 ODE: 2.1482e-02 IC: 2.8750e-05 Data: 1.0695e-02 c: 0.1234
Epoch 4000 Loss: 2.9165e-02 ODE: 2.0284e-02 IC: 2.2791e-05 Data: 8.8579e-03 c: 0.1224
Epoch 5000 Loss: 1.8986e-02 ODE: 1.3368e-02 IC: 2.5341e-05 Data: 5.5920e-03 c: 0.1334
Epoch 6000 Loss: 5.8666e-03 ODE: 4.3074e-03 IC: 2.8742e-05 Data: 1.5305e-03 c: 0.1280
Epoch 7000 Loss: 1.6570e-03 ODE: 1.2466e-03 IC: 1.6210e-05 Data: 3.9424e-04 c: 0.1137
Epoch 8000 Loss: 7.2629e-04 ODE: 5.5249e-04 IC: 1.6759e-05 Data: 1.5704e-04 c: 0.1061
Epoch 9000 Loss: 5.4932e-04 ODE: 4.0388e-04 IC: 3.7561e-05 Data: 1.0788e-04 c: 0.1023
Epoch 10000 Loss: 3.7826e-04 ODE: 2.7151e-04 IC: 1.7332e-05 Data: 8.9419e-05 c: 0.1006
Ablolute error: 0.0094
Relative error: 0.0164
```

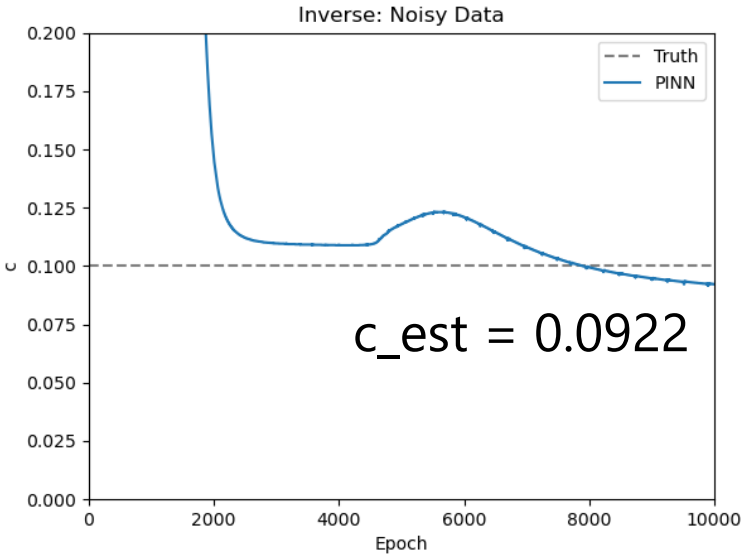
3．逆解析（解析結果）

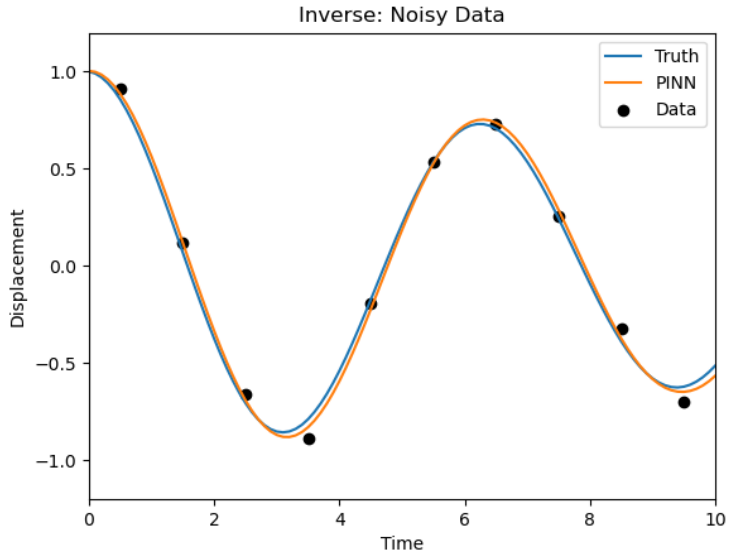
誤差なしデータ





誤差ありデータ





解析結果まとめ

データ ⟵ ⟶ 理論

深層学習

1. 教師あり学習

データ $\{t_i, u_i\}_{i=1}^{N_{\text{data}}}$

⇓

解 $u(t)$

物理深層学習

3. 逆解析

方程式 $\frac{d^2u}{dt^t} + c\frac{du}{dt} + u = 0$

初期値 $u(0) = 1, \frac{du}{dt}(0) = 0$

データ $\{t_i, u_i\}_{i=1}^{N_{\text{data}}}$

⇓

未知パラメタ c, 解 $u(t)$

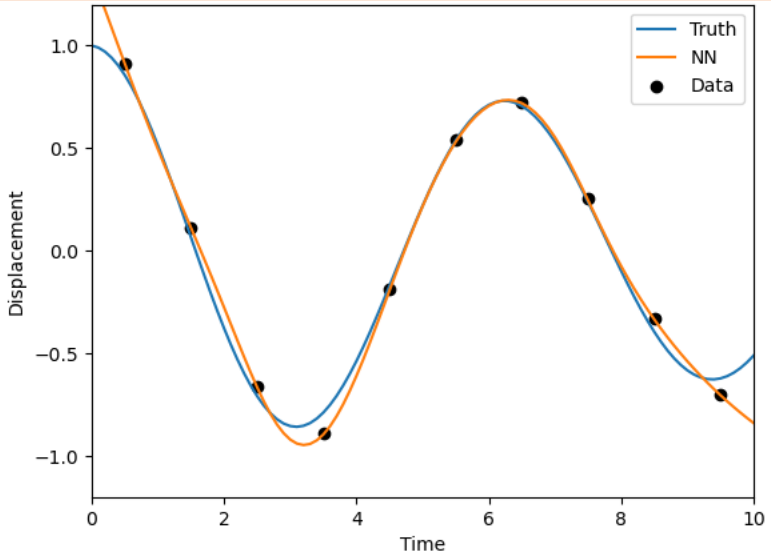
2. 順解析

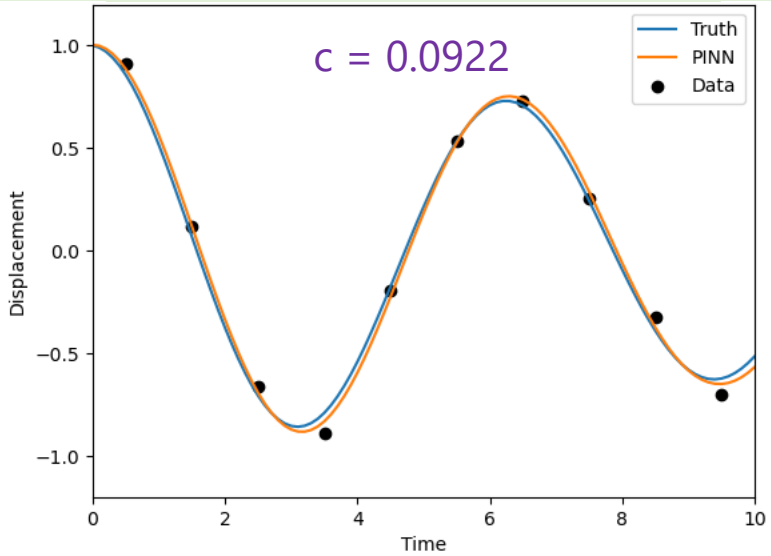
方程式 $\frac{d^2u}{dt^t} + 0.1\frac{du}{dt} + u = 0$

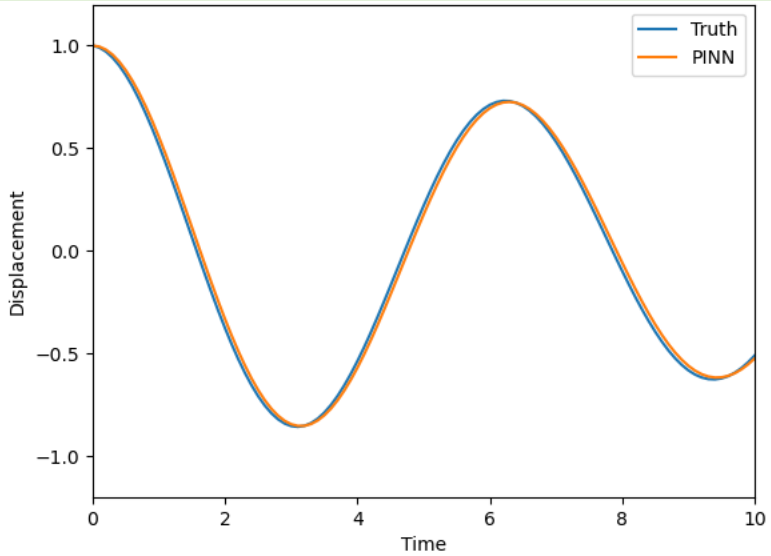
初期値 $u(0) = 1, \frac{du}{dt}(0) = 0$

⇓

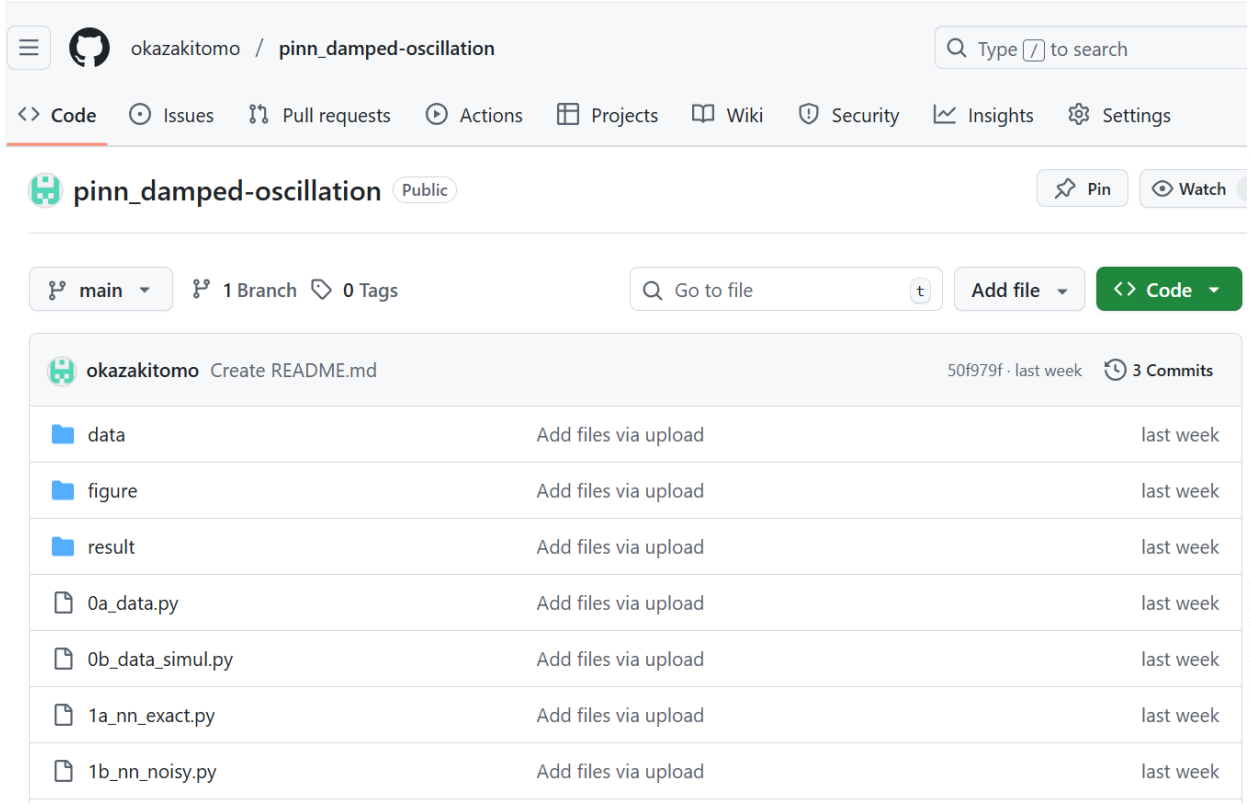
解 $u(t)$

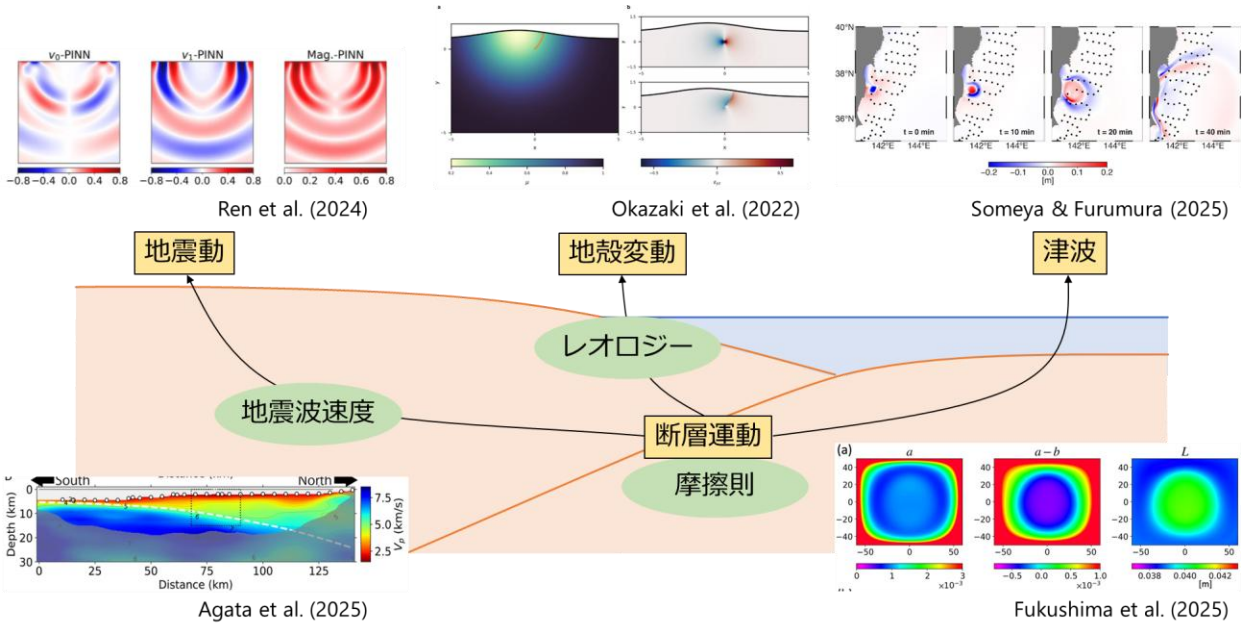






おわりに





本チュートリアルのGitHub
https://github.com/okazakitomo/pinn_damped-oscillation

岡﨑, Scientific Machine Learning 地震学
地震, 77, 101–120 (2025).
https://doi.org/10.4294/zisin.2024-9
(arXiv:2409.18397)