DHOscillator_FNN_PINN_example

March 10, 2024

[1]:
```
# import numpy, scipy, and matplotlib
import numpy as np
import scipy as sp
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split

import torch
%matplotlib widget

import os
import tempfile
```

1 Compare FNN and PINN for the damped harmonic oscillator ODE

In this notebook we train two model with the same architecture and for the same number of epochs to solve the damped harmonic oscillator problem ODE.

The first model is a simple feedforward neural network (FNN) and the second model is a physics-informed neural network (PINN).

We compare the performance of the two models as extrapolator, so predicting the solution of the ODE outside the training domain for the FFNN. Instead, since no solution is needed in the PINN training, the extrapolation range will be included in the time span where the ODE is enforced on the net.

[2]:
```
# Number of epochs
n_epochs = 50000

# Batch size
batch_size = 595

# Learning rate and scheduler
lr = 0.01
factor = 0.9
patience = 200
```

```
# Model architecture
n_layers = 3
n_neurons = 20
```

[3]:
```
# Model class
class FFNN(torch.nn.Module):
    def __init__(self, n_layers, n_neurons):
        super(FFNN, self).__init__()
        layers = []
        for i in range(n_layers):
            if i == 0:
                layers.append(torch.nn.Linear(1, n_neurons))
            else:
                layers.append(torch.nn.Linear(n_neurons, n_neurons))
            layers.append(torch.nn.Tanh())
        layers.append(torch.nn.Linear(n_neurons, 2))
        self.model = torch.nn.Sequential(*layers)
    def forward(self, x):
        return self.model(x)
```

1.1 Load data

[4]:
```
# import data
# data are generated by "src/DHOscillator_data_gen.py"
data = np.load('../data/DHOscillator_data.npy')
X = data[:,0]
Y = data[:,1:]
```

[5]:
```
def data_loader(X, Y, batch_size):
    """
    Function to load data and divide it in batches
    input: X, Y, batch_size
    output: train_X_batches, train_Y_batches, val_X, val_Y, test_X, test_Y
    """

    # divide in train, validation and test
    train_frac = 0.7
    val_frac = 0.15
    test_frac = 0.15

    train_val_X = X[:int((train_frac+val_frac)*len(X))]
    train_val_Y = Y[:int((train_frac+val_frac)*len(X)), :]
    train_X, val_X, train_Y, val_Y = train_test_split(
        train_val_X,
        train_val_Y,
        test_size=val_frac/(train_frac+val_frac),
        random_state=42
```

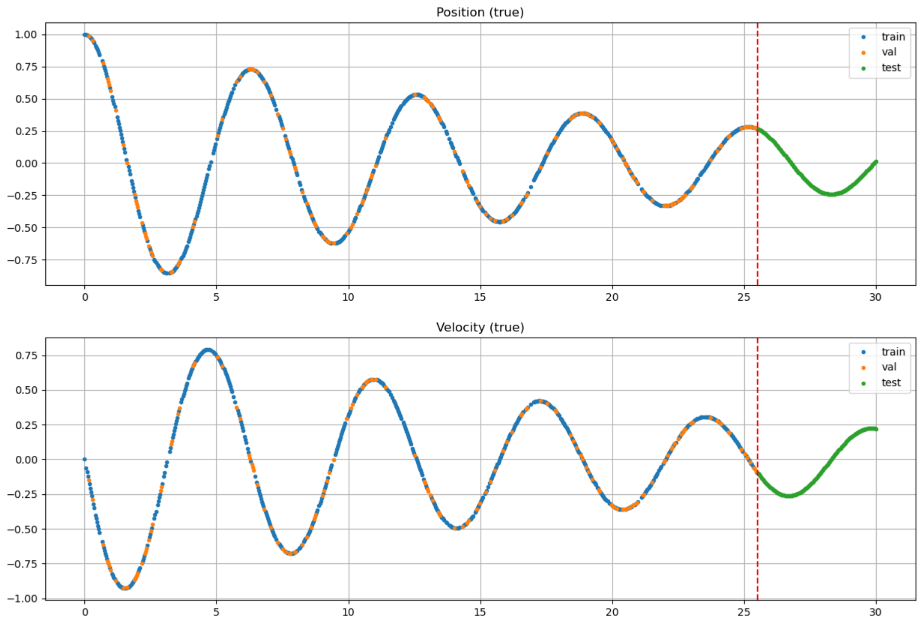
```
        )

    test_X = X[int(0.85*len(X)):]
    test_Y = Y[int(0.85*len(X)):, :]

    # convert to torch tensor
    train_X = torch.tensor(train_X, dtype=torch.float32).view(-1, 1)
    train_Y = torch.tensor(train_Y, dtype=torch.float32)
    val_X = torch.tensor(val_X, dtype=torch.float32).view(-1, 1)
    val_Y = torch.tensor(val_Y, dtype=torch.float32)
    test_X = torch.tensor(test_X, dtype=torch.float32).view(-1, 1)
    test_Y = torch.tensor(test_Y, dtype=torch.float32)

    # divide in batches train
    train_X_batches = torch.split(train_X, batch_size)
    train_Y_batches = torch.split(train_Y, batch_size)


    return train_X_batches, train_Y_batches, val_X, val_Y, test_X, test_Y
```

[6]:
```
# use the data loader to get the data, in this example we use only one batch
train_X_batches, train_Y_batches, val_X, val_Y, test_X, test_Y = data_loader(X,␣
↪Y, batch_size)
```

[7]:
```
# plot the position
plt.figure(figsize=(15, 10))
plt.subplot(2, 1, 1)
plt.plot(train_X_batches[0].detach().numpy(), train_Y_batches[0][:, 0].detach().
↪numpy(), '.', label='train')
plt.plot(val_X.detach().numpy(), val_Y[:, 0].detach().numpy(), '.', label='val')
plt.plot(test_X.detach().numpy(), test_Y[:, 0].detach().numpy(), '.',␣
↪label='test')
plt.grid()
plt.title('Position (true)')
plt.axvline(x=30*0.85, color='r', linestyle='--')
plt.legend()

# plot the velocity
plt.subplot(2, 1, 2)
plt.plot(train_X_batches[0].detach().numpy(), train_Y_batches[0][:, 1].detach().
↪numpy(), '.', label='train')
plt.plot(val_X.detach().numpy(), val_Y[:, 1].detach().numpy(), '.', label='val')
plt.plot(test_X.detach().numpy(), test_Y[:, 1].detach().numpy(), '.',␣
↪label='test')
plt.grid()
plt.title('Velocity (true)')
plt.axvline(x=30*0.85, color='r', linestyle='--')
```

```
plt.legend()
```

[7]: <matplotlib.legend.Legend at 0x7ff8e31cae50>



1.2 FFNN

The FFNN will be trained on data point with time < 25, discarding validation data this means 595 point of the ODE.

[8]:
```
# define the model
model_FFNN = FFNN(n_layers, n_neurons)

# define the loss function, mean squared error
loss_fn = torch.nn.MSELoss()

# define the optimizer and lr scheduler
optimizer = torch.optim.Adam(model_FFNN.parameters(), lr=lr)
scheduler = torch.optim.lr_scheduler.ReduceLROnPlateau(optimizer, 'min',␣
 ↪factor=factor, patience=patience)
```

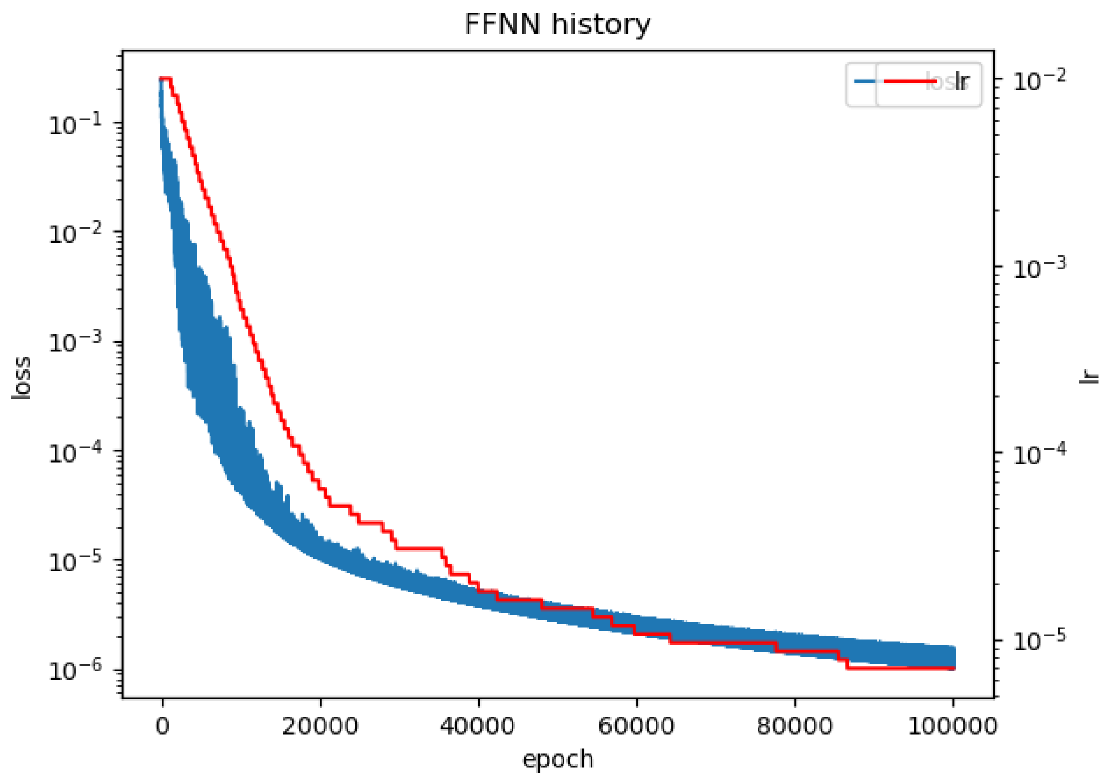
[9]:
```
%%time
history_FFNN = []
```

```
# train the model
for epoch in range(n_epochs):
    for i, (X, Y) in enumerate(zip(train_X_batches, train_Y_batches)):
        optimizer.zero_grad()
        Y_pred = model_FFNN(X)
        loss = loss_fn(Y_pred, Y)
        loss.backward()
        optimizer.step()
        optimizer.step()
        scheduler.step(loss)
        history_FFNN.append([loss.item(), optimizer.param_groups[0]['lr']])
    if epoch % 10000 == 0:
        print(epoch, loss.item())
```

```
0 0.24801737070083618
10000 1.0190393368247896e-05
20000 3.811971282630111e-06
30000 2.1196483430685475e-06
40000 1.414661369381065e-06
CPU times: user 12min 53s, sys: 7.47 s, total: 13min
Wall time: 3min 15s
```

[10]:
```
# plot history_FFNN loss and lr in two subplots
history_FFNN = np.array(history_FFNN)
fig, ax = plt.subplots()
ax.plot(history_FFNN[:, 0], label='loss')
ax.legend()
ax.set_yscale('log')
ax.set_xlabel('epoch')
ax.set_ylabel('loss')

ax2 = ax.twinx()
ax2.plot(history_FFNN[:, 1], label='lr', color='r')
ax2.set_yscale('log')
ax2.set_ylabel('lr')
ax2.legend()

plt.title('FFNN history')
```

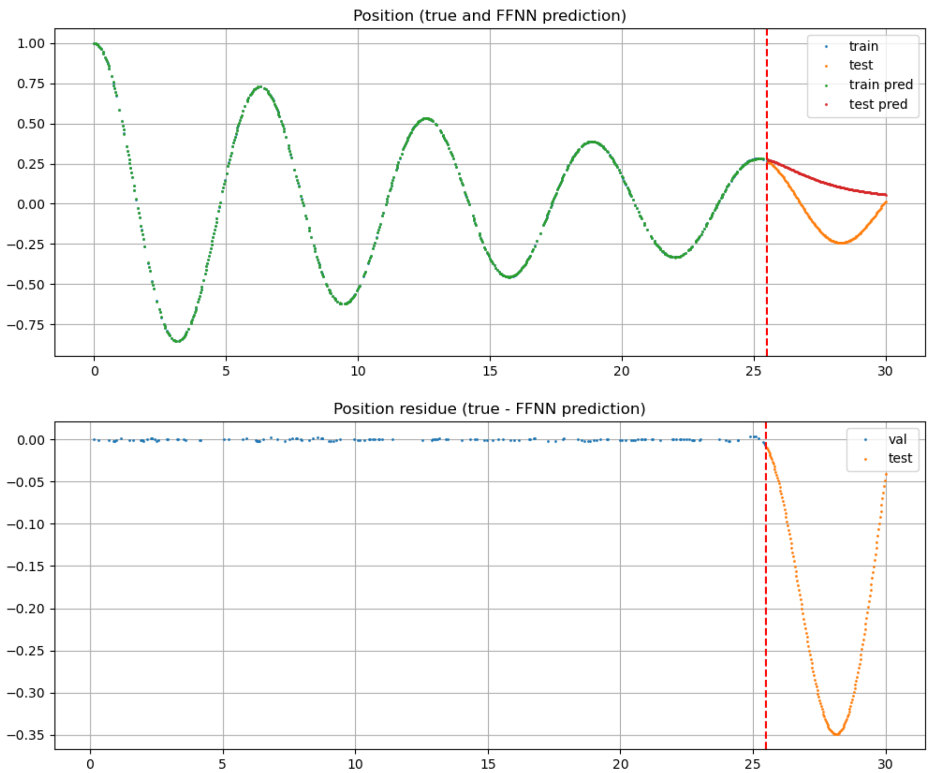
[10]:
```
Text(0.5, 1.0, 'FFNN history')
```

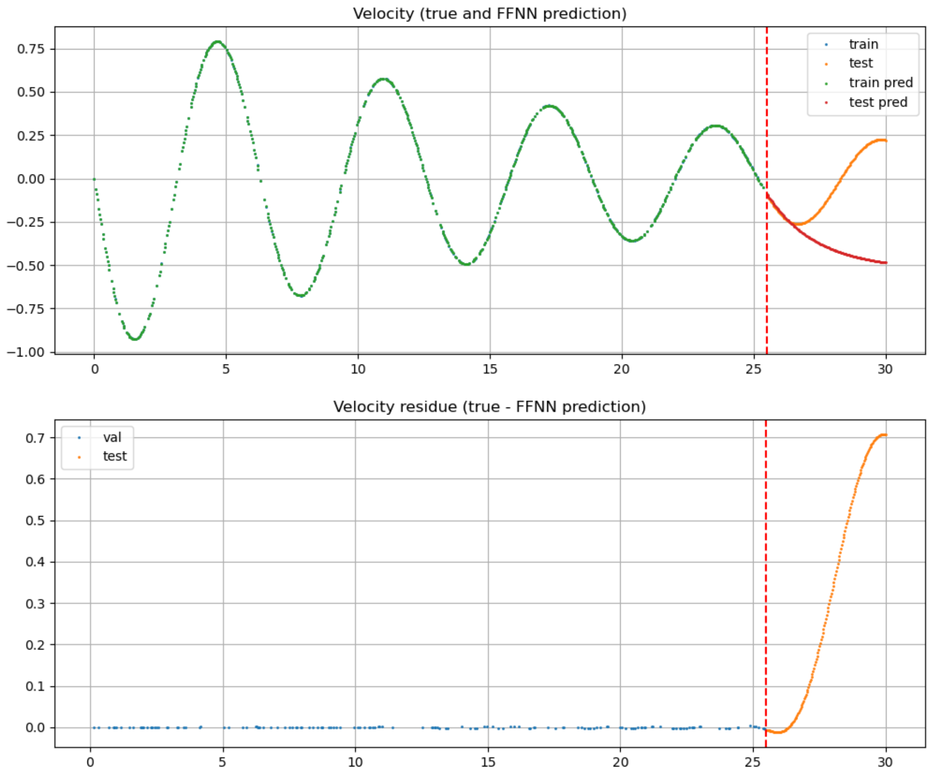


[11]:
```
# get predictions
Y_pred_train = model_FFNN(train_X_batches[0])
Y_pred_val = model_FFNN(val_X)
Y_pred_test = model_FFNN(test_X)

# plot the position, and subplot the residue
plt.figure(figsize=(12, 10))
plt.subplot(2, 1, 1)

marker='.'
markersize=2

plt.plot(train_X_batches[0].detach().numpy(), train_Y_batches[0][:, 0].detach().
 ↪numpy(), marker, label='train', markersize=markersize)
plt.plot(test_X.detach().numpy(), test_Y[:, 0].detach().numpy(), marker,␣
 ↪label='test', markersize=markersize)
plt.plot(train_X_batches[0].detach().numpy(), Y_pred_train[:, 0].detach().
 ↪numpy(), marker, label='train pred', markersize=markersize)
plt.plot(test_X.detach().numpy(), Y_pred_test[:, 0].detach().numpy(), marker,␣
 ↪label='test pred', markersize=markersize)
```

```
plt.grid()

plt.title('Position (true and FFNN prediction)')
plt.axvline(x=30*0.85, color='r', linestyle='--')
plt.legend()

plt.subplot(2, 1, 2)
plt.plot(val_X.detach().numpy(), val_Y[:, 0].detach().numpy()-Y_pred_val[:, 0].detach().numpy(), marker, label='val', markersize=markersize)
plt.plot(test_X.detach().numpy(), test_Y[:, 0].detach().numpy()-Y_pred_test[:, 0].detach().numpy(), marker, label='test', markersize=markersize)
plt.grid()
plt.title('Position residue (true - FFNN prediction)')
plt.legend()
plt.axvline(x=30*0.85, color='r', linestyle='--')


# new figure for the velocity
plt.figure(figsize=(12, 10))
plt.subplot(2, 1, 1)

plt.plot(train_X_batches[0].detach().numpy(), train_Y_batches[0][:, 1].detach().numpy(), marker, label='train', markersize=markersize)
plt.plot(test_X.detach().numpy(), test_Y[:, 1].detach().numpy(), marker, label='test', markersize=markersize)
plt.plot(train_X_batches[0].detach().numpy(), Y_pred_train[:, 1].detach().numpy(), marker, label='train pred', markersize=markersize)
plt.plot(test_X.detach().numpy(), Y_pred_test[:, 1].detach().numpy(), marker, label='test pred', markersize=markersize)
plt.grid()

plt.title('Velocity (true and FFNN prediction)')
plt.axvline(x=30*0.85, color='r', linestyle='--')
plt.legend()

plt.subplot(2, 1, 2)
plt.plot(val_X.detach().numpy(), val_Y[:, 1].detach().numpy()-Y_pred_val[:, 1].detach().numpy(), marker, label='val', markersize=markersize)
plt.plot(test_X.detach().numpy(), test_Y[:, 1].detach().numpy()-Y_pred_test[:, 1].detach().numpy(), marker, label='test', markersize=markersize)
plt.grid()
plt.title('Velocity residue (true - FFNN prediction)')
plt.legend()
plt.axvline(x=30*0.85, color='r', linestyle='--')
```

[11]:
```
<matplotlib.lines.Line2D at 0x7ff8da2b8df0>
```


Position (true and FFNN prediction)
train
test
train pred
test pred
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
0
5
10
15
20
25
30
Position residue (true - FFNN prediction)
val
test
0.00
−0.05
−0.10
−0.15
−0.20
−0.25
−0.30
−0.35
0
5
10
15
20
25
30



[12]:
```
# test loss
loss_test_FFNN = loss_fn(Y_pred_test, test_Y)
print('test loss:', loss_test_FFNN.item())
```

```
test loss: 0.11087661981582642
```

Comment: The model do not reproduce qualitativly the beheviuor of the ODE outside the train time span.

1.3 PINN

The PINN will be trained again with 595 point, but since this time we do not need the solution of the ODE to train the model, we can use point also with t>25 until t = 30. We chose a uniform distribution of the points in the time span (0,30). To enlight the fact that this train method do not need the solution of the ODE, we will use dummy_train_Y = zeros as the target of the training (it is not used anyway).

[13]:
```
dummy_train_X = np.linspace(0, 30, 595)
dummy_train_Y = np.zeros((595, 2))
```

```
dummy_train_X_batches = torch.split(torch.tensor(dummy_train_X, dtype=torch.
↪float32).view(-1, 1), 595)
dummy_train_Y_batches = torch.split(torch.tensor(dummy_train_Y, dtype=torch.
↪float32), 595)
```

[14]:
```
# define the model
model_PINN = FFNN(n_layers, n_neurons)

# define the loss function, mean squared error
loss_fn = torch.nn.MSELoss()

# define the optimizer
optimizer = torch.optim.Adam(model_PINN.parameters(), lr=lr)
scheduler = torch.optim.lr_scheduler.ReduceLROnPlateau(optimizer, 'min',␣
↪factor=factor, patience=patience)
```

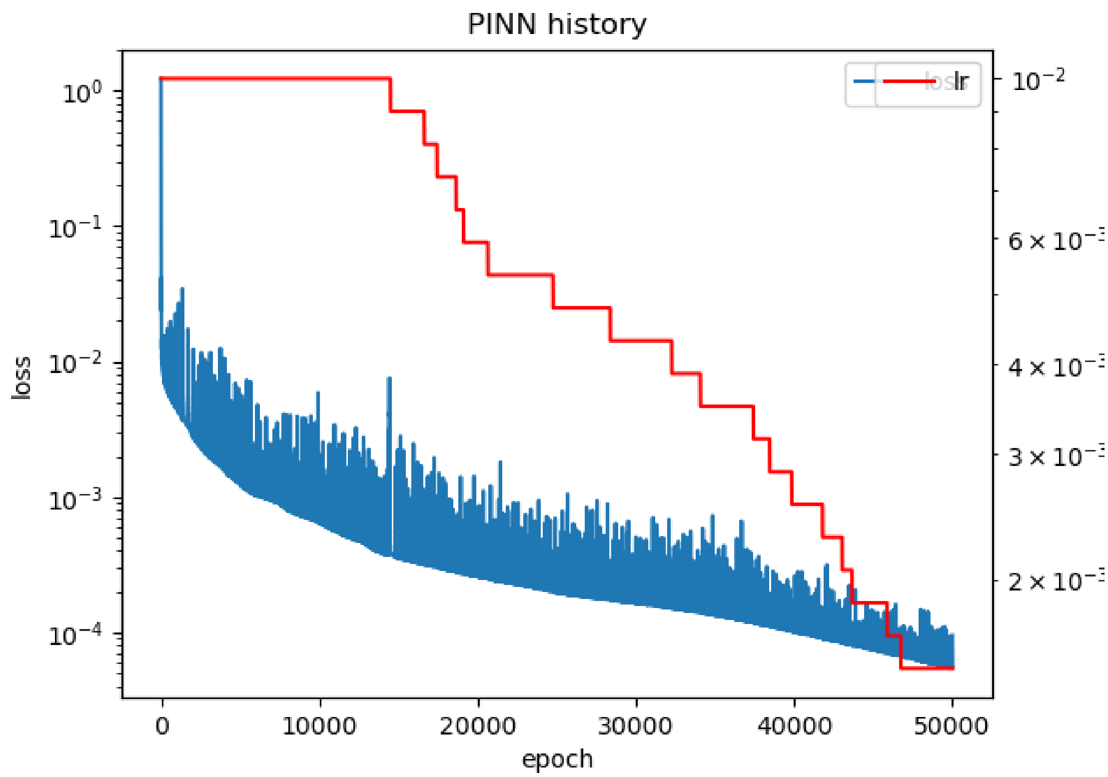
[15]:
```
%%time
history_PINN = []

# train the model
for epoch in range(n_epochs):
    for i, (X, Y) in enumerate(zip(dummy_train_X_batches,␣
↪dummy_train_Y_batches)):
        optimizer.zero_grad()
        X.requires_grad = True
        Y_pred = model_PINN(X)

        # get the derivatives
        dx_dt = torch.autograd.grad(Y_pred[:,0], X, grad_outputs=torch.
↪ones_like(Y_pred[:,0]), create_graph=True)[0]
        dv_dt = torch.autograd.grad(Y_pred[:,1], X, grad_outputs=torch.
↪ones_like(Y_pred[:,1]), create_graph=True)[0]

        # loss function is the error in the ODE and the initial condition
        loss_ode = torch.mean((dx_dt[:,0] - Y_pred[:,1])**2 + (dv_dt[:,0] + 0.
↪1*Y_pred[:,1] + Y_pred[:,0])**2)
        loss_ic = ((Y_pred[0,0] - 1)**2 + (Y_pred[0,1] - 0)**2)

        loss = loss_ode + loss_ic

        loss.backward()
        optimizer.step()
        scheduler.step(loss)
        history_PINN.append([loss.item(), optimizer.param_groups[0]['lr']])
    if epoch % 10000 == 0:
```

```
        print(epoch, loss.item())
```

```
0 1.2286367416381836
10000 0.0009441034053452313
20000 0.0005221162573434412
30000 0.00022039702162146568
40000 0.00010028920223703608
CPU times: user 12min 22s, sys: 6.32 s, total: 12min 29s
Wall time: 3min 7s
```

[16]:
```
# plot history_PINN loss and lr in two subplots
history_PINN = np.array(history_PINN)
fig, ax = plt.subplots()
ax.plot(history_PINN[:, 0], label='loss')
ax.legend()
ax.set_yscale('log')
ax.set_xlabel('epoch')
ax.set_ylabel('loss')

ax2 = ax.twinx()
ax2.plot(history_PINN[:, 1], label='lr', color='r')
ax2.set_yscale('log')
ax2.set_ylabel('lr')
ax2.legend()

plt.title('PINN history')
```

[16]:
```
Text(0.5, 1.0, 'PINN history')
```

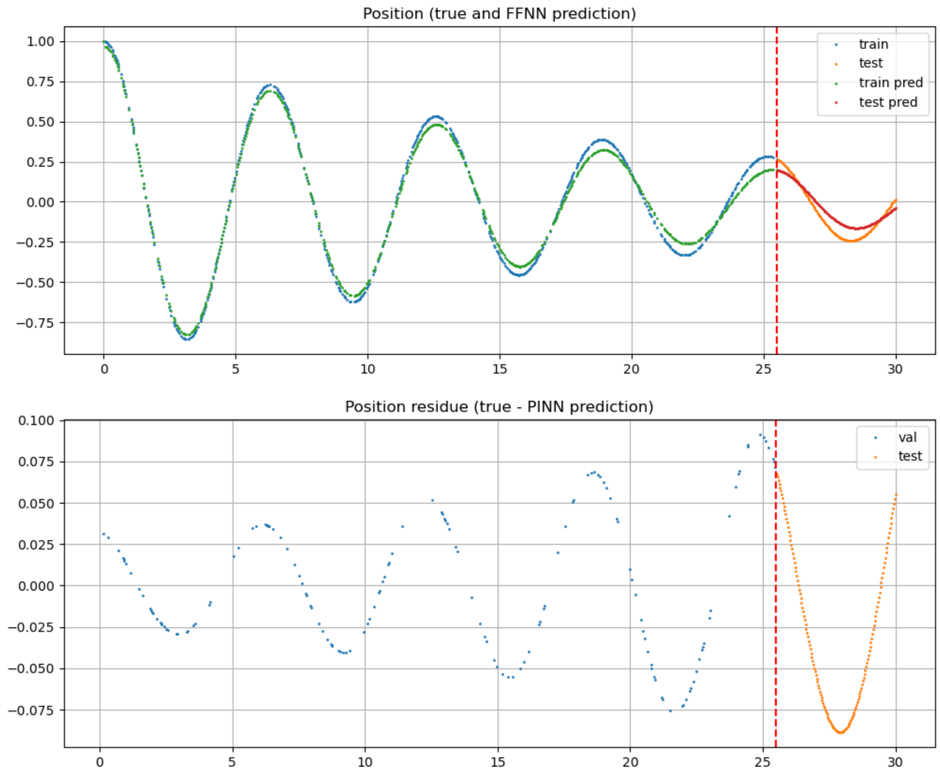


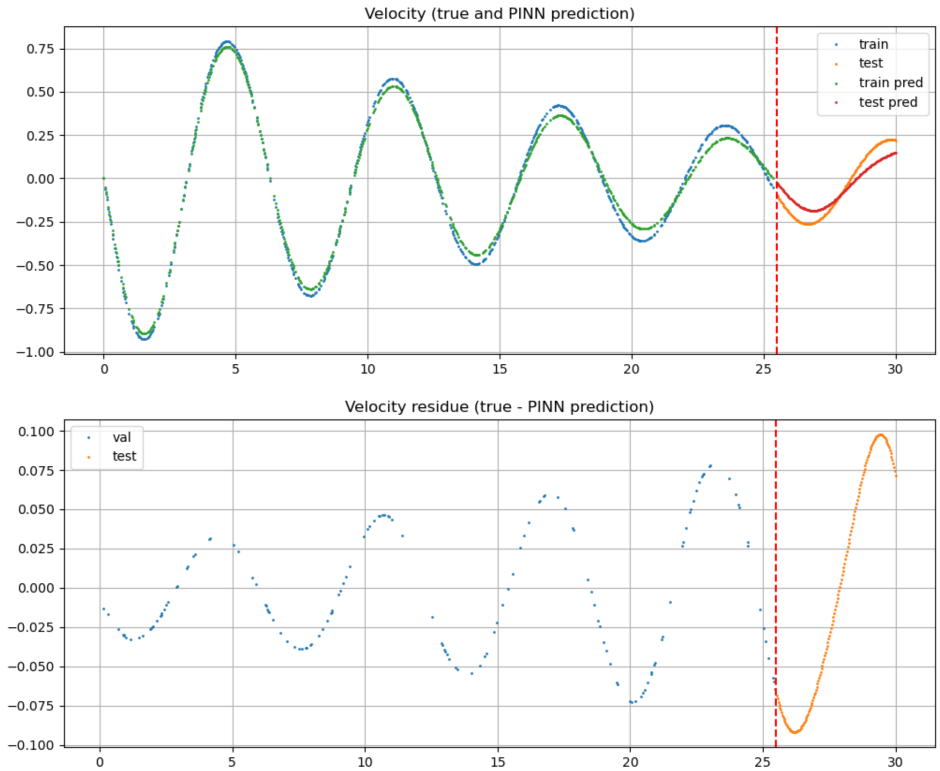
[17]:

```
# get predictions
Y_pred_train = model_PINN(train_X_batches[0])
Y_pred_val = model_PINN(val_X)
Y_pred_test = model_PINN(test_X)

# plot the position, and subplot the residue
plt.figure(figsize=(12, 10))
plt.subplot(2, 1, 1)

marker='.'
markersize=2

plt.plot(train_X_batches[0].detach().numpy(), train_Y_batches[0][:, 0].detach().
  ↪numpy(), marker, label='train', markersize=markersize)
plt.plot(test_X.detach().numpy(), test_Y[:, 0].detach().numpy(), marker,␣
  ↪label='test', markersize=markersize)
plt.plot(train_X_batches[0].detach().numpy(), Y_pred_train[:, 0].detach().
  ↪numpy(), marker, label='train pred', markersize=markersize)
plt.plot(test_X.detach().numpy(), Y_pred_test[:, 0].detach().numpy(), marker,␣
  ↪label='test pred', markersize=markersize)
```

```
plt.grid()

plt.title('Position (true and FFNN prediction)')
plt.axvline(x=30*0.85, color='r', linestyle='--')
plt.legend()

plt.subplot(2, 1, 2)
plt.plot(val_X.detach().numpy(), val_Y[:, 0].detach().numpy()-Y_pred_val[:, 0].
 ↪detach().numpy(), marker, label='val', markersize=markersize)
plt.plot(test_X.detach().numpy(), test_Y[:, 0].detach().numpy()-Y_pred_test[:,␣
 ↪0].detach().numpy(), marker, label='test', markersize=markersize)
plt.grid()
plt.title('Position residue (true - PINN prediction)')
plt.legend()
plt.axvline(x=30*0.85, color='r', linestyle='--')


# new figure for the velocity
plt.figure(figsize=(12, 10))
plt.subplot(2, 1, 1)

plt.plot(train_X_batches[0].detach().numpy(), train_Y_batches[0][:, 1].detach().
 ↪numpy(), marker, label='train', markersize=markersize)
plt.plot(test_X.detach().numpy(), test_Y[:, 1].detach().numpy(), marker,␣
 ↪label='test', markersize=markersize)
plt.plot(train_X_batches[0].detach().numpy(), Y_pred_train[:, 1].detach().
 ↪numpy(), marker, label='train pred', markersize=markersize)
plt.plot(test_X.detach().numpy(), Y_pred_test[:, 1].detach().numpy(), marker,␣
 ↪label='test pred', markersize=markersize)
plt.grid()

plt.title('Velocity (true and PINN prediction)')
plt.axvline(x=30*0.85, color='r', linestyle='--')
plt.legend()

plt.subplot(2, 1, 2)
plt.plot(val_X.detach().numpy(), val_Y[:, 1].detach().numpy()-Y_pred_val[:, 1].
 ↪detach().numpy(), marker, label='val', markersize=markersize)
plt.plot(test_X.detach().numpy(), test_Y[:, 1].detach().numpy()-Y_pred_test[:,␣
 ↪1].detach().numpy(), marker, label='test', markersize=markersize)
plt.grid()
plt.title('Velocity residue (true - PINN prediction)')
plt.legend()
plt.axvline(x=30*0.85, color='r', linestyle='--')
```

```
[17]: <matplotlib.lines.Line2D at 0x7ff8d9e69250>
```


Position (true and FFNN prediction)
train
test
train pred
test pred
Position residue (true - PINN prediction)
val
test




```
[18]: # test loss
loss_test_PINN = loss_fn(Y_pred_test, test_Y)
print('test loss:', loss_test_PINN.item())
```

```
test loss: 0.004159581381827593
```

Comment: the model reproduce qualitative the behaviuor of the solution for all the time span.

1.4 Conclusion

The architecture of the two models is the same, the clock time (3 min on this machine) of training and the number of data point is the same. If we compare the two loss error in the test time span we obtain:

```
[19]: # print the two test losses
print('FFNN test loss:', loss_test_FFNN.item())
print('PINN test loss:', loss_test_PINN.item())
# print the ratio
print('ratio:', loss_test_PINN.item()/loss_test_FFNN.item())
```

```
FFNN test loss: 0.11087661981582642
PINN test loss: 0.004159581381827593
ratio: 0.037515405761259134
```