In [ ]:
```
import DW_oscillator as DW
import numpy as np
from IPython.display import clear_output
from torchdiffeq import odeint
import torch
import torch.nn as nn
import torch.optim as optim
from torch.nn import functional as F
import matplotlib.pyplot as plt
```

In [ ]:
```
def app (indices):
    omega = torch.tensor(2 * torch.pi * 0.5)  # f = 0.5
    if flag == True:
        h_accumulator = torch.zeros_like(torch.tensor([0.]), dtype=torch.float64)
        omega = omega * time_train[0]
        h = h_t_[0] * torch.sin(omega)
        h_accumulator = h
        return h_accumulator.unsqueeze(0)
    h_accumulator = torch.zeros_like(indices, dtype=torch.float64)  # Initialize an

    for i in range(len(indices)):
        omega = omega * time_train[indices[i]]  # Assuming t is defined elsewhere
        h = h_t_[indices[i]] * torch.sin(omega)  # Assuming h_t is defined elsewher
        h_accumulator[i] = h  # Store the calculated h in the accumulator tensor

    return h_accumulator.unsqueeze(0)  # Return the accumulator tensor
```

In [ ]:
```
def get_batch(true_y,time, batch_size):
    num_samples = len(true_y)
    indices = np.random.choice(np.arange(num_samples - batch_size, dtype=np.int64),
    indices.sort()
    #print(indices)
    batch_y0 = true_y[indices]  # (batch_size, D)
    batch_t = time[:batch_size]  # (batch_size)
    batch_y = torch.stack([true_y[indices + i] for i in range(batch_size)], dim=0)
    indices = torch.tensor(indices)
    return batch_y0,batch_t,batch_y,indices
```

In [ ]:
```
t, y, h_t, fields, periods = DW.run_field_sequence(field_low = 100, field_high = 10
```

```
[100.]
[60.]
```

In [ ]:
```
h_t_ = torch.tensor(torch.div(h_t_, 1000.), dtype=torch.float64)  # Converting to c
y_0_ = torch.tensor(y[0], dtype=torch.float64)  # Converting to column vector
y_1_ = torch.tensor(y[1], dtype=torch.float64)  # Converting to column vector

# Stack the tensors horizontally
data = torch.stack((torch.div(y_0_, 1000.),y_1_))
```

```
C:\Users\jagpr\AppData\Local\Temp\ipykernel_38068\4139661924.py:1: UserWarning: To copy construct from a tensor, it is recommended to use sourceTensor.clone().detach() or sourceTensor.clone().detach().requires_grad_(True), rather than torch.tensor(sourceTensor).
  h_t_ = torch.tensor(torch.div(h_t_, 1000.), dtype=torch.float64)  # Converting to column vector
```

In [ ]:
```
data.shape
```

Out[ ]:
```
torch.Size([2, 600])
```

In [ ]:
```
train = data[:,:].transpose(0,1)
```

In [ ]:
```
train.shape
```

Out[ ]:
```
torch.Size([600, 2])
```

In [ ]:
```
class DWODE(nn.Module):
    """
    neural network for learning the chaotic lorenz system
    """
    def __init__(self):
        super(DWODE, self).__init__()
        self.lin = nn.Linear(2, 128)
        self.lin2 = nn.Linear(128, 256)
        self.lin3 = nn.Linear(256,512)
        self.lin4 = nn.Linear(512,2)
        self.tanh = nn.Tanh()
        self.lrelu = nn.LeakyReLU()


    def forward(self,t,x):
        h = app(indices).view(-1,1)
        x_aug = torch.cat([x, h], 1)
        x = self.lrelu(self.lin(x))
        x = self.lrelu(self.lin2(x))
        x = self.tanh(self.lin3(x))
        x = self.lin4(x)
        return x
```

In [ ]:
```
model = DWODE().double()
```

In [ ]:
```
optimizer = optim.Adam(model.parameters(), lr=1e-3)
```

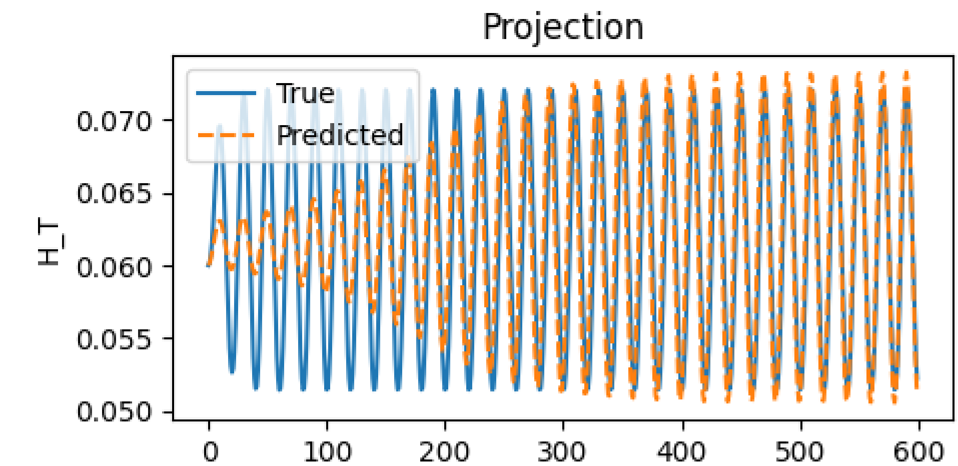
In [ ]:
```
from torchdiffeq import odeint_adjoint as adjoint
```

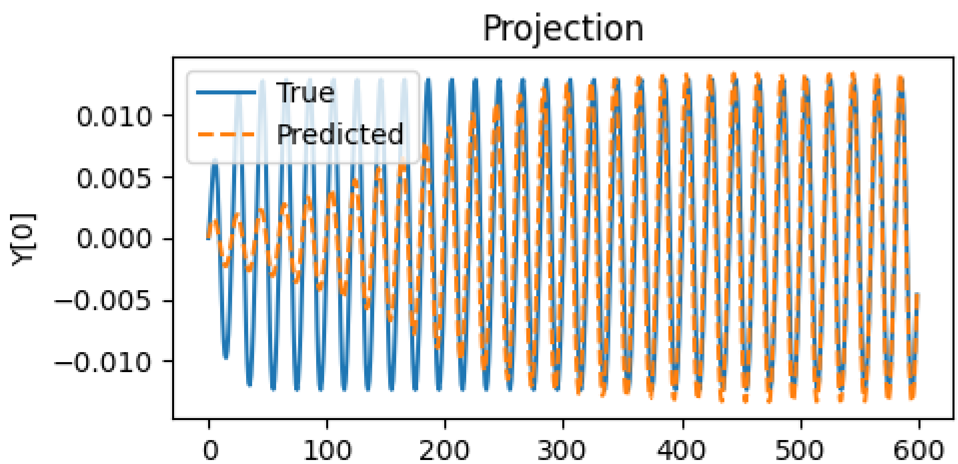
In [ ]:
```
time_train = torch.tensor(t)
```

In [ ]:
```
losses = []
whole_losses = []
best_loss = 100.0
for i in range(1000):
```

```
    optimizer.zero_grad()

    init,batch_t,truth,indices = get_batch(train,time_train,8)
    #print(init,batch_t,truth)
    pred_y = adjoint(model,init,batch_t,method='dopri5')
    loss = F.mse_loss(pred_y, truth)
    loss.backward()
    losses.append(loss.item())
    optimizer.step()
    if loss.item() < best_loss:
                best_loss = loss.item()
                torch.save(model.state_dict(), 'saved_models/forward_field.pth')
    if i % 100 == 0:

        with torch.no_grad():
            flag = True
            pred_y = adjoint(model, train[0].view(1,-1), time_train,method='dopri5'
            pred_y = pred_y.squeeze(1)
            loss = F.mse_loss(pred_y, train)
            whole_losses.append(loss.item())
            flag = False
            print('Iter {:04d} | Total Loss {:.6f}'.format(i, loss.item()))
            x_pred = pred_y[:,0].cpu()
            y_pred = pred_y[:,1].cpu()


            # Extract the x, y, z coordinates from X_train_plt
            x_train = train[:,0].cpu()
            y_train = train[:,1].cpu()


            fig, ax = plt.subplots(2, 1, figsize=(5, 5))
            ax[0].plot(x_train, label='True')
            ax[0].plot(x_pred, label='Predicted', linestyle='--')
            ax[0].set_ylabel('H_T')
            ax[0].set_title('Projection')
            ax[0].legend()

            ax[1].plot(y_train, label='True')
            ax[1].plot(y_pred, label='Predicted',linestyle='--')
            ax[1].set_ylabel('Y[0]')
            ax[1].set_title('Projection')
            ax[1].legend()


            plt.tight_layout()
            plt.show()
            clear_output(wait=True)
```

```
Iter 0900 | Total Loss 0.000020
```



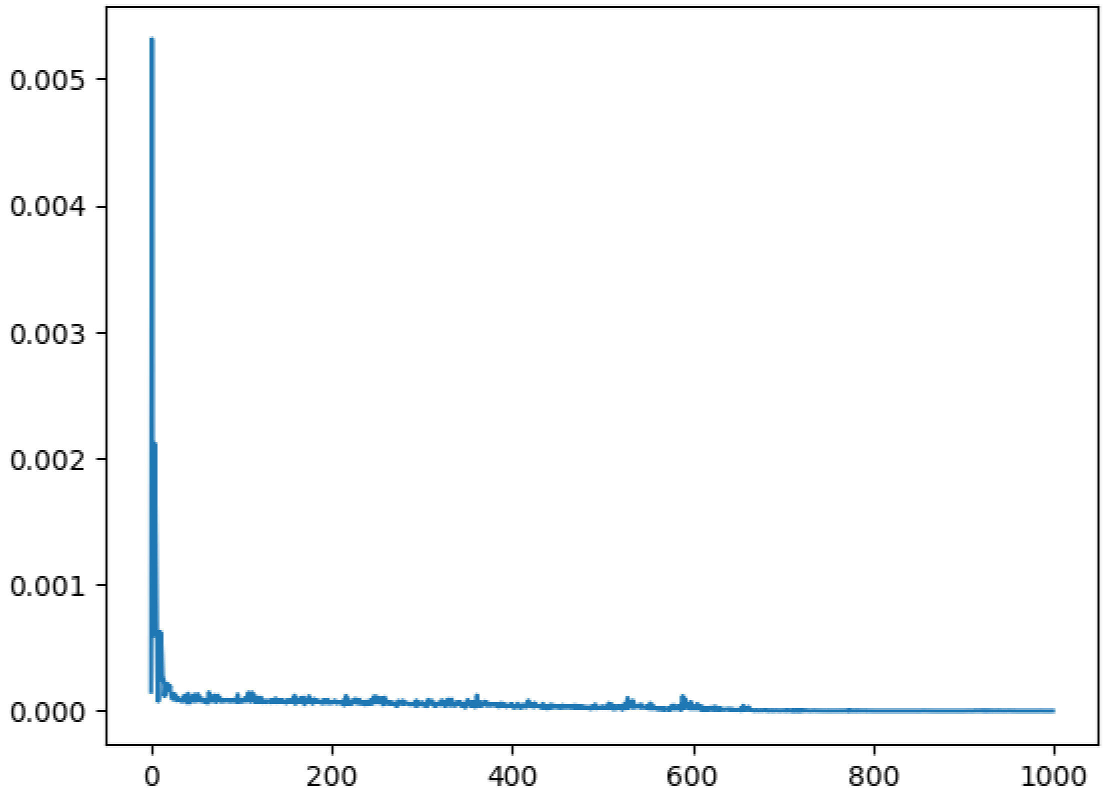


In [ ]:
```
test_model = DWODE().double()
```

In [ ]:
```
test_model.load_state_dict(torch.load('saved_models/forward_field.pth'))
```

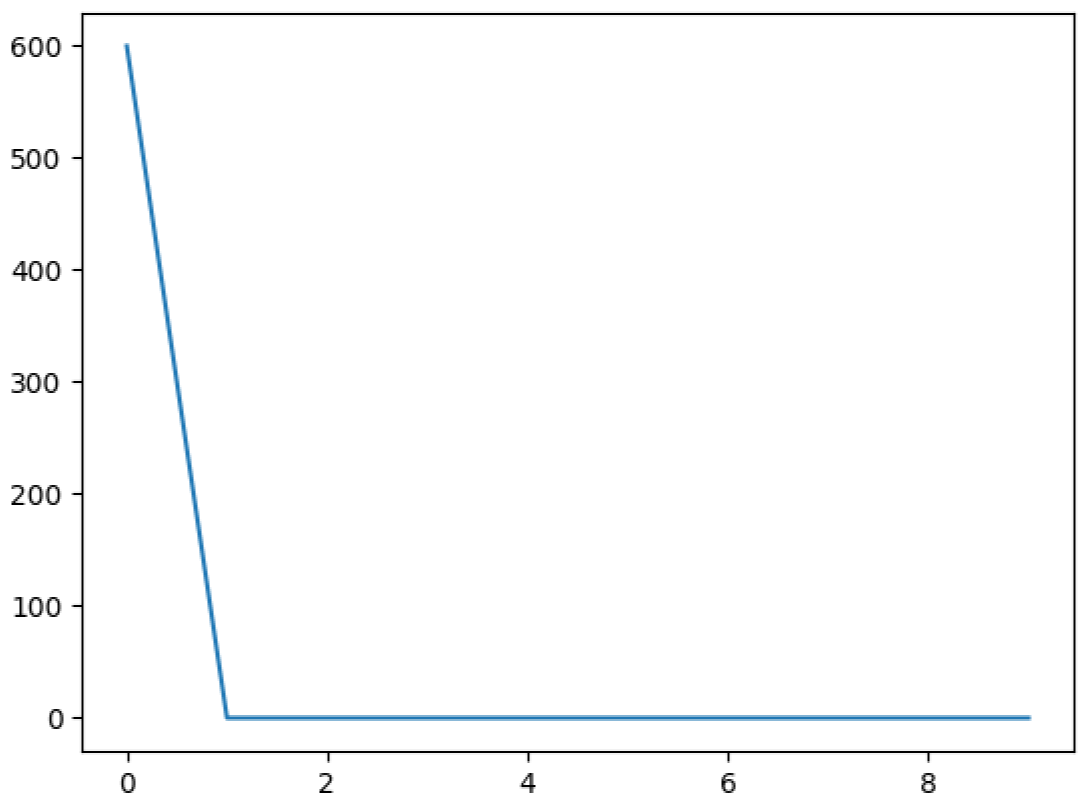
Out[ ]:
```
<All keys matched successfully>
```

In [ ]:
```
plt.plot(losses)
plt.show()
```



In [ ]:

```
plt.plot(whole_losses)
plt.show()
```



In [ ]:

```
with torch.no_grad():
    flag = True
```

```
    pred = adjoint(test_model, train[0].view(1,-1), time_train,method='dopri5')
    flag = False
```

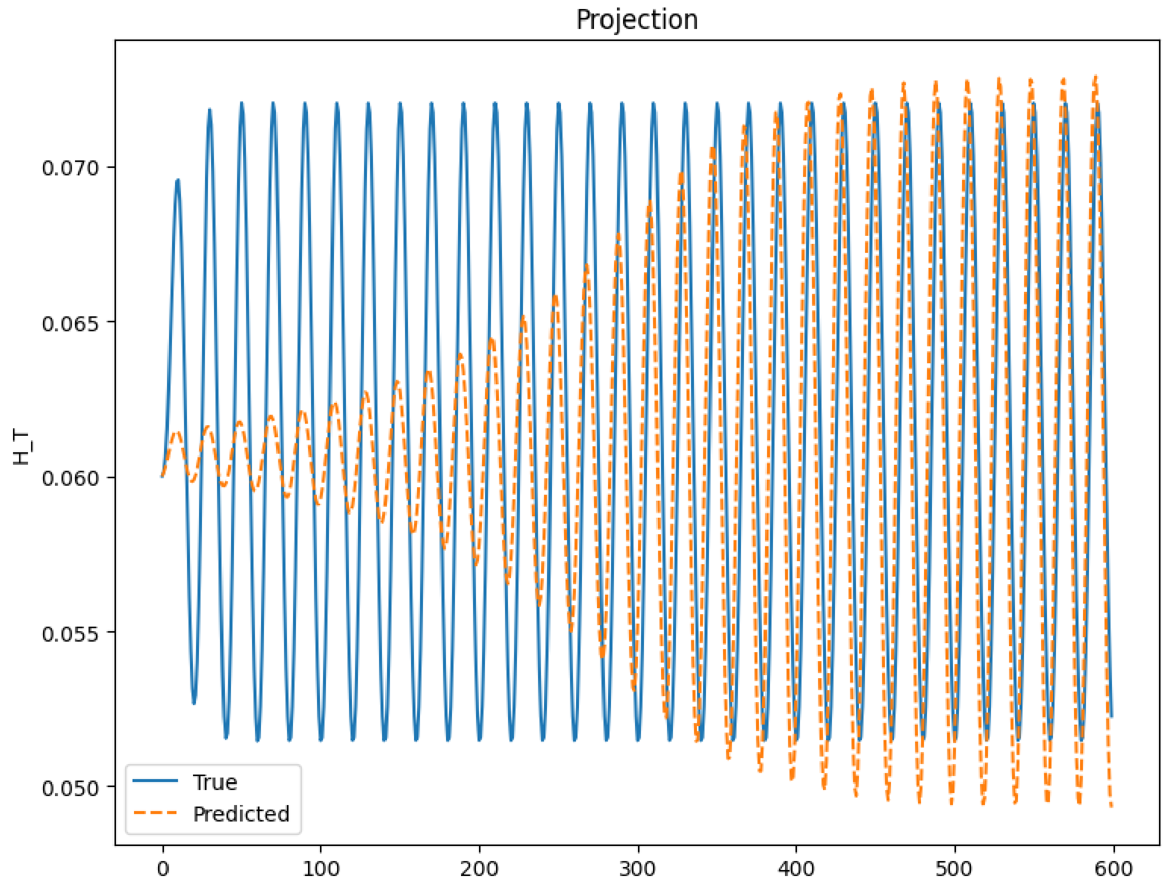
In [ ]:
```
pred = pred.cpu().detach().numpy()
```

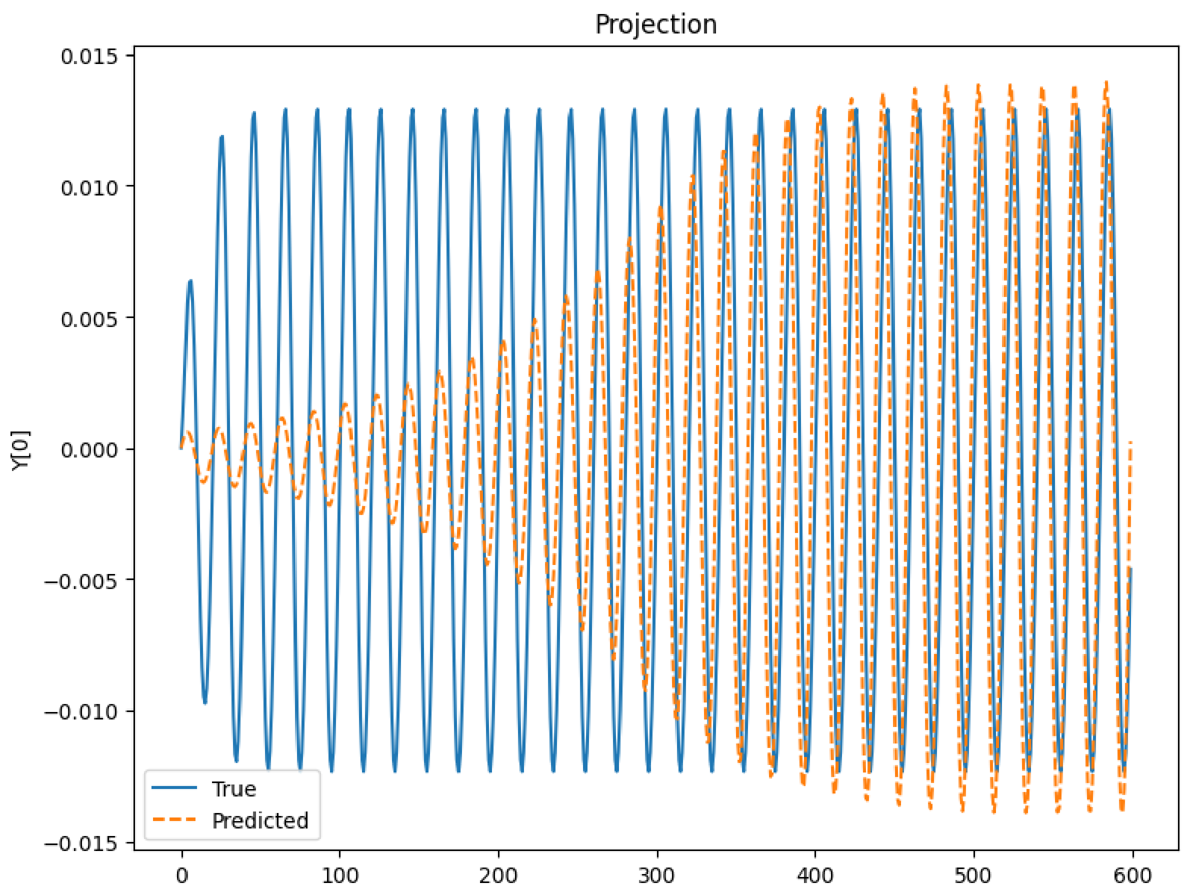
In [ ]:
```
pred = pred.squeeze(1)
```

In [ ]:
```
pred.shape,train.shape
```

Out[ ]:
```
((600, 2), torch.Size([600, 2]))
```

In [ ]:
```
# Extract the x, y, z coordinates from predictions_plt
x_pred = pred[:,0]
y_pred = pred[:,1]


# Extract the x, y, z coordinates from X_train_plt
x_train = train[:,0].cpu()
y_train = train[:,1].cpu()

fig, ax = plt.subplots(2, 1, figsize=(8, 12))
ax[0].plot(x_train, label='True')
ax[0].plot(x_pred, label='Predicted',linestyle='--')
ax[0].set_ylabel('H_T')
ax[0].set_title('Projection')
ax[0].legend()

ax[1].plot(y_train, label='True')
ax[1].plot(y_pred, label='Predicted',linestyle='--')
ax[1].set_ylabel('Y[0]')
ax[1].set_title('Projection')

ax[1].legend()
plt.savefig('forward_field.png')
plt.tight_layout()
plt.show()
```


Projection
H_T
0.070
0.065
0.060
0.055
0.050
0 100 200 300 400 500 600
True
Predicted


Projection
Y[0]
0.015
0.010
0.005
0.000
−0.005
−0.010
−0.015
0 100 200 300 400 500 600
True
Predicted