```
import numpy as np
from scipy.optimize import minimize

N = 50
dt = 0.1
a = 0.1
b = 1.0
```

```
def velocity_constraint(x):
    q1 = x[:N]
    q2 = x[N:]


    v1_start = (q1[1] - q1[0]) / dt
    v2_start = (q2[1] - q2[0]) / dt
    v1_end = (q1[-1] - q1[-2]) / dt
    v2_end = (q2[-1] - q2[-2]) / dt

    return np.array([
        v1_start,
        v2_start,
        v1_end,
        v2_end
    ])
```

```
def max_velocity_constraint(x):
    q1 = x[:N]
    q2 = x[N:]

    dq1 = (q1[1:] - q1[:-1]) / dt
    dq2 = (q2[1:] - q2[:-1]) / dt

    constraints = []
    dq_max_1=2
    dq_max_2=2
    # Joint 1
    constraints.extend(dq_max_1 - dq1)
    constraints.extend(dq_max_1 + dq1)

    # Joint 2
    constraints.extend(dq_max_2 - dq2)
    constraints.extend(dq_max_2 + dq2)

    return np.array(constraints)
```

```
def start_constraint(x, q_start):
    return np.array([
        x[0]      - q_start[0],  # q1(0)
        x[N]      - q_start[1]   # q2(0)
    ])

def end_constraint(x, q_end):
    return np.array([
        x[N-1]    - q_end[0],    # q1(T)
        x[2*N-1]  - q_end[1]     # q2(T)
    ])
```

```
def cost(x):
    q1 = x[:N]
    q2 = x[N:]

    dq1 = (q1[1:] - q1[:-1]) / dt
    dq2 = (q2[1:] - q2[:-1]) / dt

    ddq1 = (q1[2:] - 2*q1[1:-1] + q1[:-2]) / dt**2
    ddq2 = (q2[2:] - 2*q2[1:-1] + q2[:-2]) / dt**2

    Jv = np.sum(dq1**2) + np.sum(dq2**2)
    Ja = np.sum(ddq1**2) + np.sum(ddq2**2)

    return a*Jv + b*Ja
```

```
def optimized_trajectory(q_start, q_end):

    q1_init = np.linspace(q_start[0], q_end[0], N)
    q2_init = np.linspace(q_start[1], q_end[1], N)
    x0 = np.hstack([q1_init, q2_init])

    constraints = [
    {'type': 'eq',   'fun': start_constraint, 'args': (q_start,)},
    {'type': 'eq',   'fun': end_constraint,   'args': (q_end,)},
    {'type': 'eq',   'fun': velocity_constraint},
    #{'type': 'ineq', 'fun': max_velocity_constraint}
]


    result = minimize(
        cost,
        x0,
        method='SLSQP'
```

```
        method='SLSQP',
        constraints=constraints,
        options={'maxiter': 500, 'ftol': 1e-6}
    )

    return result.x
```

Same constraints as assignment 3 but i also experimented with max velocity constraint found it to be to time taking for generating datasets, hence i have commented it out

```
def datasetgen (num_samples):
    X = []
    Y = []

    for i in range(num_samples):

        q_start = np.random.uniform(-np.pi, np.pi, size=2)
        q_end   = np.random.uniform(-np.pi, np.pi, size=2)
        traj = optimized_trajectory(q_start, q_end)


        X.append(np.hstack([q_start, q_end]))

        Y.append(traj)

        if i % 10 == 0:
            print(f"Generated {i}/{num_samples} trajectories")

    return np.array(X), np.array(Y)
```

```
X, Y = datasetgen (500)

print("X shape:", X.shape)
print("Y shape:", Y.shape)
```

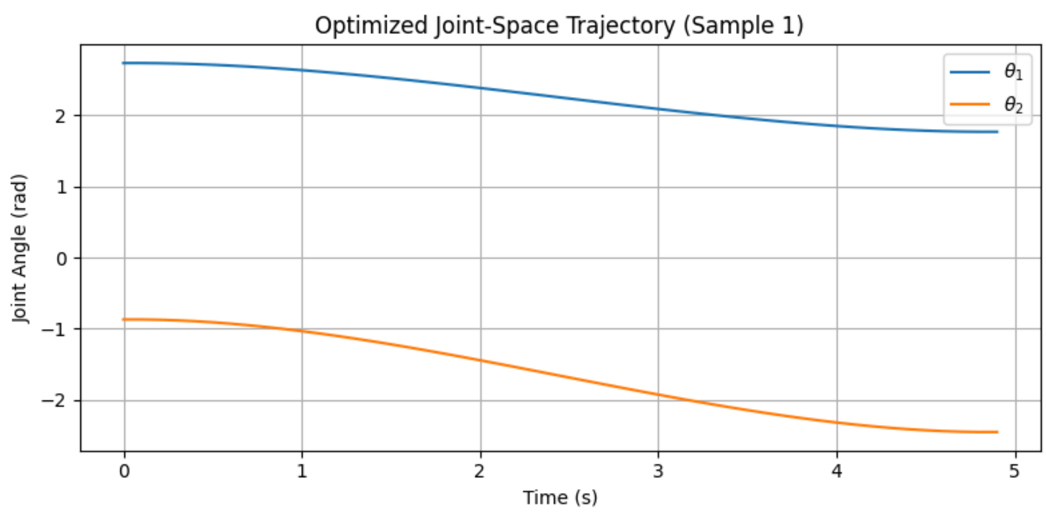
```
Generated 0/500 trajectories
Generated 10/500 trajectories
Generated 20/500 trajectories
Generated 30/500 trajectories
Generated 40/500 trajectories
Generated 50/500 trajectories
Generated 60/500 trajectories
Generated 70/500 trajectories
Generated 80/500 trajectories
Generated 90/500 trajectories
Generated 100/500 trajectories
Generated 110/500 trajectories
Generated 120/500 trajectories
Generated 130/500 trajectories
Generated 140/500 trajectories
Generated 150/500 trajectories
Generated 160/500 trajectories
Generated 170/500 trajectories
Generated 180/500 trajectories
Generated 190/500 trajectories
Generated 200/500 trajectories
Generated 210/500 trajectories
Generated 220/500 trajectories
Generated 230/500 trajectories
Generated 240/500 trajectories
Generated 250/500 trajectories
Generated 260/500 trajectories
Generated 270/500 trajectories
Generated 280/500 trajectories
Generated 290/500 trajectories
Generated 300/500 trajectories
Generated 310/500 trajectories
Generated 320/500 trajectories
Generated 330/500 trajectories
Generated 340/500 trajectories
Generated 350/500 trajectories
Generated 360/500 trajectories
Generated 370/500 trajectories
Generated 380/500 trajectories
Generated 390/500 trajectories
Generated 400/500 trajectories
Generated 410/500 trajectories
Generated 420/500 trajectories
Generated 430/500 trajectories
Generated 440/500 trajectories
Generated 450/500 trajectories
Generated 460/500 trajectories
Generated 470/500 trajectories
Generated 480/500 trajectories
Generated 490/500 trajectories
X shape: (500, 4)
Y shape: (500, 100)
```

```
import numpy as np

np.save("X_inputs.npy", X)
np.save("Y_trajectories.npy", Y)
```

```
X = np.load("X_inputs.npy")
Y = np.load("Y_trajectories.npy")
```

```
q_start = X[0, :2]
q_end   = X[0, 2:]

traj = Y[0]
q1 = traj[:N]
q2 = traj[N:]
```

```
import matplotlib.pyplot as plt
import numpy as np

t = np.arange(N) * dt

plt.figure(figsize=(8, 4))
plt.plot(t, q1, label=r'$\theta_1$')
plt.plot(t, q2, label=r'$\theta_2$')
plt.xlabel("Time (s)")
plt.ylabel("Joint Angle (rad)")
plt.title("Optimized Joint-Space Trajectory (Sample 1)")
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```




```
from sklearn.model_selection import train_test_split

X_train, X_test, Y_train, Y_test = train_test_split(
    X, Y, test_size=0.2, random_state=42
)
```

```
import numpy as np
import torch
from torch.utils.data import TensorDataset, DataLoader
```

```
X_train = torch.tensor(X_train, dtype=torch.float32)
Y_train = torch.tensor(Y_train, dtype=torch.float32)

X_test = torch.tensor(X_test, dtype=torch.float32)
Y_test = torch.tensor(Y_test, dtype=torch.float32)
```

```
train_ds = TensorDataset(X_train, Y_train)
test_ds  = TensorDataset(X_test, Y_test)

train_loader = DataLoader(train_ds, batch_size=64, shuffle=True)
test_loader  = DataLoader(test_ds, batch_size=64, shuffle=False)
```

```
import torch.nn as nn

class TrajectoryMLP(nn.Module):
    def __init__(self, N):
        super().__init__()
        self.net = nn.Sequential(
            nn.Linear(4, 128),
            nn.ReLU(),
            nn.Linear(128, 256),
            nn.ReLU(),
            nn.Linear(256, 2*N)
        )

    def forward(self, x):
        return self.net(x)
```

```
N = Y.shape[1] // 2

model = TrajectoryMLP(N)
```

```
optimizer = torch.optim.Adam(model.parameters(), lr=1e-3)
criterion = nn.MSELoss()
```

```
num_epochs = 1000

for epoch in range(num_epochs):
    model.train()
    train_loss = 0.0

    for xb, yb in train_loader:
        optimizer.zero_grad()
        pred = model(xb)
        loss = criterion(pred, yb)
        loss.backward()
        optimizer.step()

        train_loss += loss.item()

    train_loss /= len(train_loader)

    # Validation
    model.eval()
    test_loss = 0.0
    with torch.no_grad():
        for xb, yb in test_loader:
            pred = model(xb)
            loss = criterion(pred, yb)
            test_loss += loss.item()

    test_loss /= len(test_loader)

    if epoch % 20 == 0:
        print(f"Epoch {epoch:03d} | Train MSE: {train_loss:.6f} | Test MSE: {test_loss:.6f}")
```

```
Epoch 000 | Train MSE: 0.000076 | Test MSE: 0.000253
Epoch 020 | Train MSE: 0.000093 | Test MSE: 0.000372
Epoch 040 | Train MSE: 0.000177 | Test MSE: 0.000403
Epoch 060 | Train MSE: 0.000107 | Test MSE: 0.000345
Epoch 080 | Train MSE: 0.000113 | Test MSE: 0.000295
Epoch 100 | Train MSE: 0.000291 | Test MSE: 0.000454
Epoch 120 | Train MSE: 0.000090 | Test MSE: 0.000537
Epoch 140 | Train MSE: 0.000104 | Test MSE: 0.000264
Epoch 160 | Train MSE: 0.000168 | Test MSE: 0.000343
Epoch 180 | Train MSE: 0.000225 | Test MSE: 0.000392
Epoch 200 | Train MSE: 0.000147 | Test MSE: 0.000323
Epoch 220 | Train MSE: 0.000108 | Test MSE: 0.000284
Epoch 240 | Train MSE: 0.000169 | Test MSE: 0.000248
Epoch 260 | Train MSE: 0.000230 | Test MSE: 0.000402
Epoch 280 | Train MSE: 0.000099 | Test MSE: 0.000259
Epoch 300 | Train MSE: 0.000090 | Test MSE: 0.000240
Epoch 320 | Train MSE: 0.000142 | Test MSE: 0.000321
Epoch 340 | Train MSE: 0.000062 | Test MSE: 0.000280
Epoch 360 | Train MSE: 0.000289 | Test MSE: 0.000496
Epoch 380 | Train MSE: 0.000081 | Test MSE: 0.000303
Epoch 400 | Train MSE: 0.000037 | Test MSE: 0.000211
Epoch 420 | Train MSE: 0.000063 | Test MSE: 0.000334
Epoch 440 | Train MSE: 0.000786 | Test MSE: 0.001016
Epoch 460 | Train MSE: 0.000063 | Test MSE: 0.000266
Epoch 480 | Train MSE: 0.000042 | Test MSE: 0.000197
Epoch 500 | Train MSE: 0.000038 | Test MSE: 0.000232
Epoch 520 | Train MSE: 0.000053 | Test MSE: 0.000222
Epoch 540 | Train MSE: 0.000223 | Test MSE: 0.000494
Epoch 560 | Train MSE: 0.000295 | Test MSE: 0.000498
Epoch 580 | Train MSE: 0.000042 | Test MSE: 0.000225
Epoch 600 | Train MSE: 0.000073 | Test MSE: 0.000236
Epoch 620 | Train MSE: 0.000289 | Test MSE: 0.001140
Epoch 640 | Train MSE: 0.000196 | Test MSE: 0.000630
Epoch 660 | Train MSE: 0.000142 | Test MSE: 0.000231
Epoch 680 | Train MSE: 0.000094 | Test MSE: 0.000232
Epoch 700 | Train MSE: 0.000090 | Test MSE: 0.000209
Epoch 720 | Train MSE: 0.000478 | Test MSE: 0.000612
Epoch 740 | Train MSE: 0.000244 | Test MSE: 0.000423
Epoch 760 | Train MSE: 0.000205 | Test MSE: 0.000436
Epoch 780 | Train MSE: 0.000050 | Test MSE: 0.000236
Epoch 800 | Train MSE: 0.000057 | Test MSE: 0.000201
Epoch 820 | Train MSE: 0.000069 | Test MSE: 0.000224
Epoch 840 | Train MSE: 0.000068 | Test MSE: 0.000230
Epoch 860 | Train MSE: 0.000206 | Test MSE: 0.000394
Epoch 880 | Train MSE: 0.000116 | Test MSE: 0.000241
Epoch 900 | Train MSE: 0.000582 | Test MSE: 0.000612
Epoch 920 | Train MSE: 0.000116 | Test MSE: 0.000221
Epoch 940 | Train MSE: 0.000057 | Test MSE: 0.000203
Epoch 960 | Train MSE: 0.000033 | Test MSE: 0.000208
Epoch 980 | Train MSE: 0.000187 | Test MSE: 0.000443
```

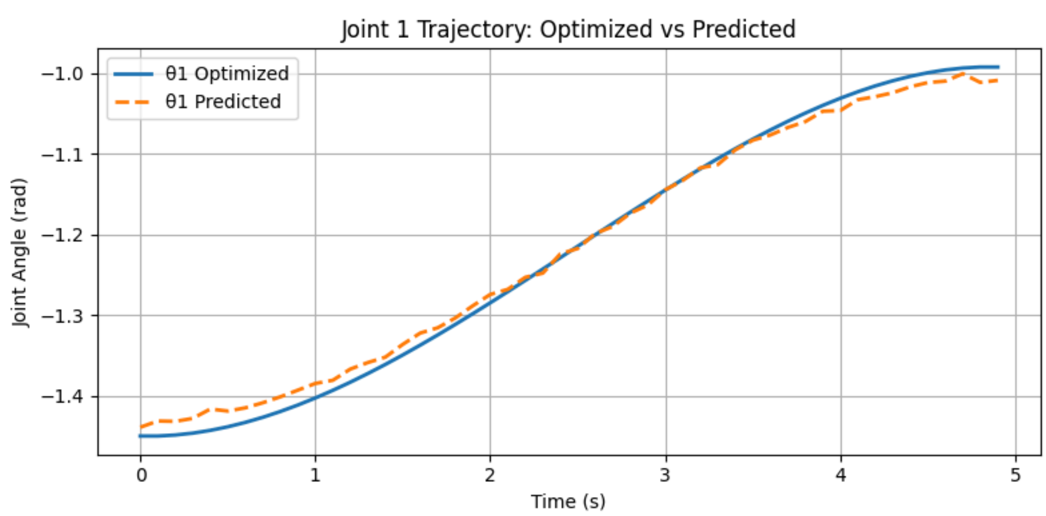
```
def predict_trajectory(model, q_start, q_end):
    model.eval()
    with torch.no_grad():
        inp = torch.tensor(
            [[q_start[0], q_start[1], q_end[0], q_end[1]]],
            dtype=torch.float32
        )
        traj = model(inp).numpy().flatten()
    return traj
```

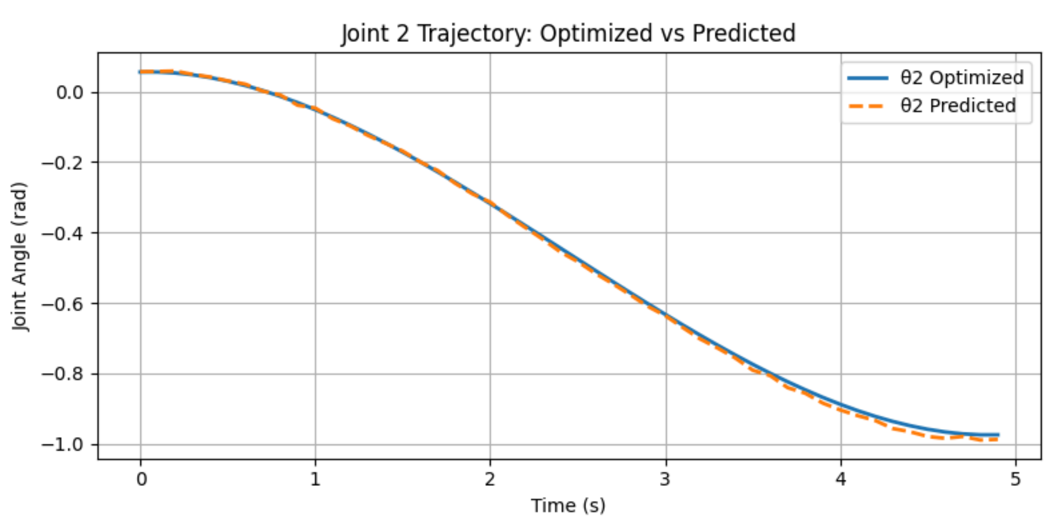
```
idx=1
q_start = X_test[idx, :2].numpy()
q_end   = X_test[idx, 2:].numpy()
```

```
pred_traj = predict_trajectory(model, q_start, q_end)
print("cost of learned trajectory ",cost(pred_traj))
q1_pred = pred_traj[:N]
q2_pred = pred_traj[N:]

_opti = Y_test[idx].detach().cpu().numpy()
print("cost of optimal trajectory ",cost(_opti))    # shape (2N,)
q1_opti = _opti[:N]
q2_opti = _opti[N:]
```

```
cost of learned trajectory  84.03821
cost of optimal trajectory  1.6972684
```

```
plt.figure(figsize=(8, 4))
plt.plot(t, q1_opti, label="θ1 Optimized", linewidth=2)
plt.plot(t, q1_pred, '--', label="θ1 Predicted", linewidth=2)
plt.xlabel("Time (s)")
plt.ylabel("Joint Angle (rad)")
plt.title("Joint 1 Trajectory: Optimized vs Predicted")
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()

plt.figure(figsize=(8, 4))
plt.plot(t, q2_opti, label="θ2 Optimized", linewidth=2)
plt.plot(t, q2_pred, '--', label="θ2 Predicted", linewidth=2)
plt.xlabel("Time (s)")
plt.ylabel("Joint Angle (rad)")
plt.title("Joint 2 Trajectory: Optimized vs Predicted")
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```






```
from google.colab import output
output.enable_custom_widget_manager()
```

```
import numpy as np
import matplotlib.pyplot as plt
import ipywidgets as widgets
from IPython.display import display, clear_output
```

```
q1s = widgets.FloatSlider(min=-np.pi, max=np.pi, step=0.1, value=0.0, description='θ1 start')
q2s = widgets.FloatSlider(min=-np.pi, max=np.pi, step=0.1, value=0.0, description='θ2 start')
q1e = widgets.FloatSlider(min=-np.pi, max=np.pi, step=0.1, value=1.0, description='θ1 end')
q2e = widgets.FloatSlider(min=-np.pi, max=np.pi, step=0.1, value=1.0, description='θ2 end')
```

```
run_button = widgets.Button(
    description="Run Optimization + NN",
    button_style='success'
)
```

```
def run_simulation(button):
    clear_output(wait=True)

    display(q1s, q2s, q1e, q2e, run_button)

    q_start = np.array([q1s.value, q2s.value])
    q_end   = np.array([q1e.value, q2e.value])

    # Compute trajectories
    q_opti = optimized_trajectory(q_start, q_end)
    q_pred = predict_trajectory(model,q_start, q_end)

    print( "optimal",cost(q_opti)," predicted ",cost(q_pred))
    q1_opti, q2_opti = q_opti[:N], q_opti[N:]
    q1_pred, q2_pred = q_pred[:N], q_pred[N:]

    t = np.arange(N) * dt

    plt.figure(figsize=(10,4))

    plt.subplot(1,2,1)
    plt.plot(t, q1_opti, label='θ1 Optimized', linewidth=2)
    plt.plot(t, q1_pred, '--', label='θ1 Predicted', linewidth=2)
    plt.xlabel("Time (s)")
    plt.ylabel("Angle (rad)")
    plt.legend(); plt.grid()

    plt.subplot(1,2,2)
    plt.plot(t, q2_opti, label='θ2 Optimized', linewidth=2)
    plt.plot(t, q2_pred, '.', label='θ2 Predicted', linewidth=2)
    plt.xlabel("Time (s)")
    plt.ylabel("Angle (rad)")
    plt.legend(); plt.grid()

    plt.tight_layout()
    plt.show()
```

```
run_button.on_click(run_simulation)
display(run_button)
```

θ1 start 3.14

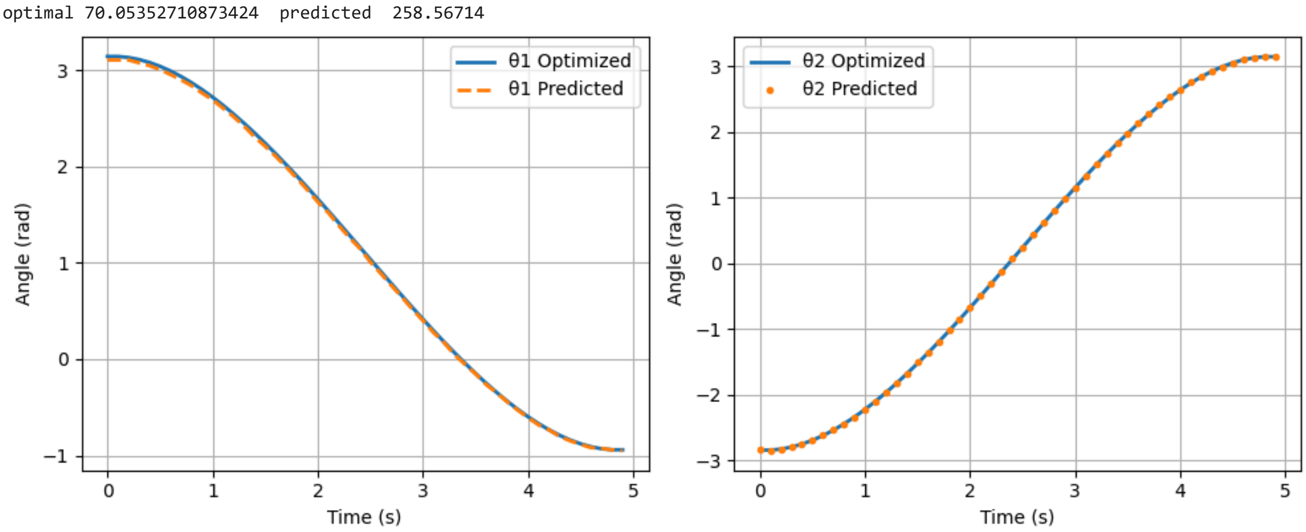
θ2 start −2.84

θ1 end −0.94

θ2 end 3.14

Run Optimization + ...

```
optimal 70.05352710873424  predicted  258.56714
```




```
import time
import numpy as np
```

```
n_runs = 10
times = []
q_start = np.asarray(q_start, dtype=np.float64)
q_end   = np.asarray(q_end, dtype=np.float64)

for _ in range(n_runs):
    t0 = time.perf_counter()
    optimized_trajectory(q_start,q_end)
    t1 = time.perf_counter()
    times.append(t1 - t0)

print(f"Optimization time (avg over {n_runs} runs): {np.mean(times):.6f} seconds")
```

```
print(f"Std dev: {np.std(times):.6f} seconds")
```

```
Optimization time (avg over 10 runs): 1.199814 seconds
Std dev: 0.221076 seconds
```

```
times = []

for _ in range(10):
    t0 = time.perf_counter()
    predict_trajectory(model,q_start, q_end)
    t1 = time.perf_counter()
    times.append(t1 - t0)

print(f"NN prediction time (avg): {np.mean(times):.8f} seconds")
print(f"Std dev: {np.std(times):.8f} seconds")
```

```
NN prediction time (avg): 0.00043583 seconds
Std dev: 0.00066673 seconds
```

we hae 500 solved trajectories as my dataset and a neural network that is learning the trajectory pattern over 1000 epochs, Relu activation was chosen to induce non linearity in the logic. The comparison of predicted and optimized trajectory is also shown, where we can see the network actually is able to come really close to the optimal trajectory. In the last cell we see the whole point of the using ML, for given intial conditions Ml solves the trajectory 10^4 times quicker than optimization algorithm, which is what makes it the best option for industrial use particularly for machine that undergo a lot of cycles.

```
Start coding or generate with AI.
```