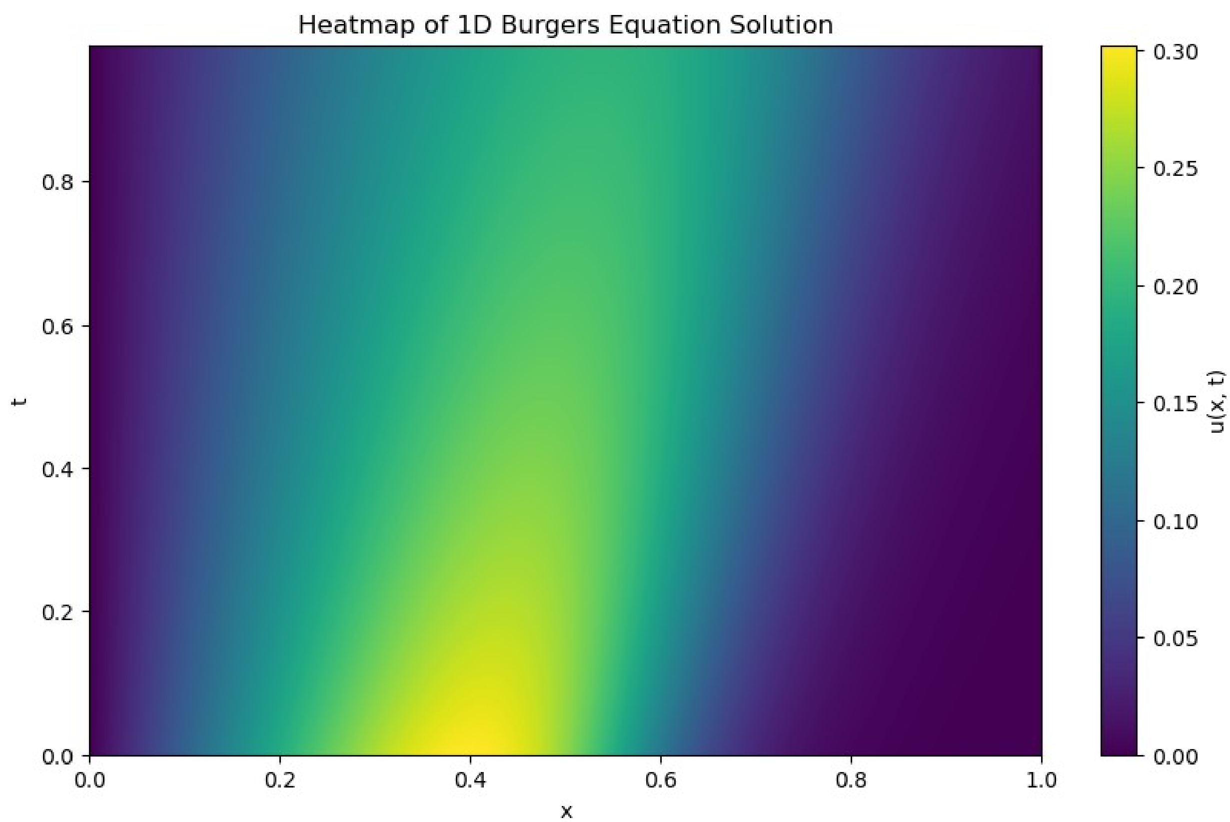
```python
import deepxde as dde
import numpy as np
from deepxde.backend import tf
import matplotlib.pyplot as plt
```

Analytical Solution

```python
# Define the Reynolds number for the analytical solution
# Re = 1,50,100,300
Re_fixed = 50

# Define the spatial and temporal grid
N_t = 100  # Number of time points
N_x = 256  # Number of spatial points
t = np.linspace(0, 0.99, N_t)  # Time grid
x = np.linspace(0, 1, N_x)    # Spatial grid

# Define the analytical solution for the 1D Burgers equation
def analytical_solution(x, t, Re_fixed):
    to = np.exp(Re_fixed / 8)  # Parameter in the equation
    u = x / (t + 1) / (1 + np.sqrt((t + 1) / to) * np.exp(Re_fixed * 
x**2 / (4 * (t + 1))))
    return u

# Compute the solution
usol = np.array([[analytical_solution(xi, ti, Re_fixed) for xi in x] 
for ti in t])

# Explicitly enforce boundary conditions
usol[:, 0] = 0  # u(0, t) = 0
usol[:, -1] = 0  # u(1, t) = 0

# Verify the boundary conditions
assert np.allclose(usol[:, 0], 0), "Boundary condition u(0, t) = 0 not 
satisfied"
assert np.allclose(usol[:, -1], 0), "Boundary condition u(1, t) = 0 
not satisfied"

# Save the data to a .npz file
np.savez("dataset/Burgers.npz", t=t, x=x, usol=usol)

# Create a new figure window with a size of 10x6 inches
plt.figure(figsize=(10, 6))

# Plot the heatmap
# `usol` is the 2D array containing the solution, `extent` specifies 
the range for x and t axes,
# `origin='lower'` ensures the heatmap starts from the bottom-left 
corner,
# `aspect='auto'` adjusts the aspect ratio automatically, and
```

```
`cmap='viridis'` sets the colormap
plt.imshow(usol, extent=[x.min(), x.max(), t.min(), t.max()],
           origin='lower', aspect='auto', cmap='viridis')

# Add a colorbar to indicate the value of u(x, t) for each color
plt.colorbar(label="u(x, t)")

plt.title("Heatmap of 1D Burgers Equation Solution")
plt.xlabel("x")
plt.ylabel("t")

plt.show()
```



Part (a): Forward Problem

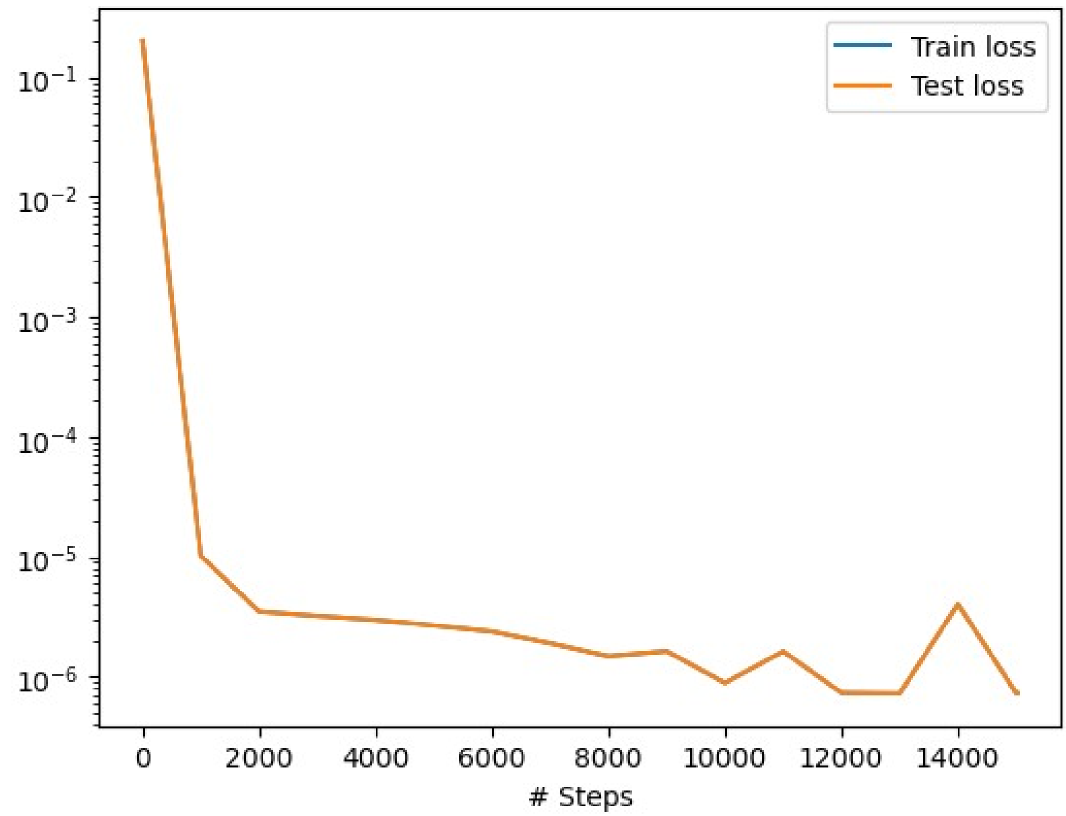
```
# Define fixed Reynolds number
# Re = 1,50,100,300
Re_fixed = 50

# Generate analytical test data
def gen_testdata():
    data = np.load("dataset/Burgers.npz")  # Ensure the file is
present in the correct location
    t, x, exact = data["t"], data["x"], data["usol"].T
```

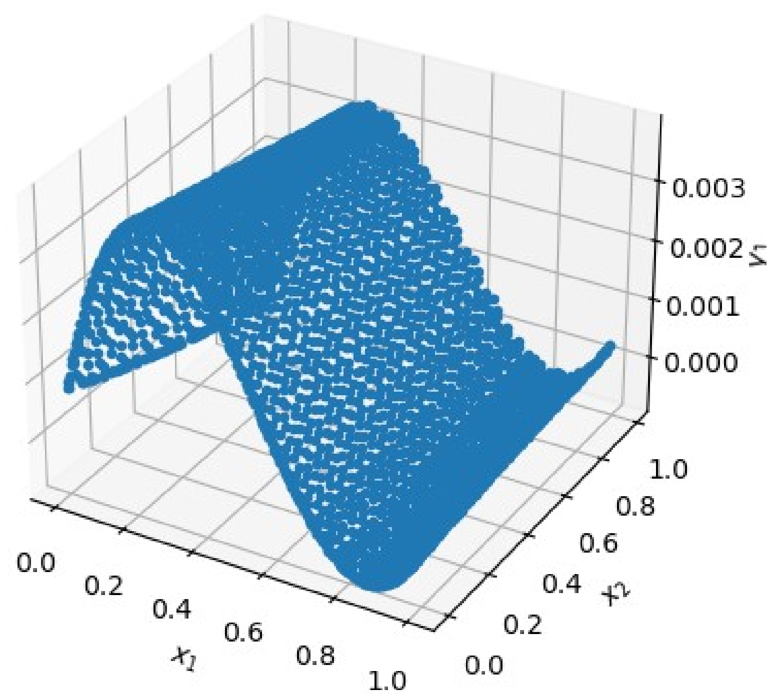
```
    xx, tt = np.meshgrid(x, t)
    X = np.vstack((np.ravel(xx), np.ravel(tt))).T
    y = exact.flatten()[:, None]
    return X, y

# Define the PDE
def pde(x, y, Re):
    dy_x = dde.grad.jacobian(y, x, i=0, j=0)
    dy_t = dde.grad.jacobian(y, x, i=0, j=1)
    dy_xx = dde.grad.hessian(y, x, i=0, j=0)
    return dy_t + y * dy_x - 0.5 / Re * dy_xx # Re=Re_fixed/100

# Define the domain and conditions
geom = dde.geometry.Interval(0, 1)
timedomain = dde.geometry.TimeDomain(0, 0.99)
geomtime = dde.geometry.GeometryXTime(geom, timedomain)
bc = dde.DirichletBC(geomtime, lambda x: 0, lambda _, on_boundary:
on_boundary)
ic = dde.IC(
    geomtime, lambda x: x[:, 0:1] / (1 + np.sqrt(np.exp(Re_fixed / 8))
* np.exp(Re_fixed * x[:, 0:1]**2 / 4)),
    lambda _, on_initial: on_initial
)

# Solve the forward problem for fixed Re
print(f"Training for fixed Re = {Re_fixed}")

# Define dataset for fixed Re
data_fixed = dde.data.TimePDE(
    geomtime, lambda x, y: pde(x, y, Re_fixed), [bc, ic],
num_domain=2540, num_boundary=80, num_initial=160
)

# Define the neural network
net_fixed = dde.maps.FNN([2] + [20] * 3 + [1], "tanh", "Glorot
normal")
model_fixed = dde.Model(data_fixed, net_fixed)
model_fixed.compile("adam", lr=1e-3)

# Train the model
model_fixed.train(epochs=15000)
model_fixed.compile("L-BFGS")
losshistory_fixed, train_state_fixed = model_fixed.train()

# Save results
dde.saveplot(losshistory_fixed, train_state_fixed, issave=True,
isplot=True)

# Test the model
X_fixed, y_true_fixed = gen_testdata()
```

```
y_pred_fixed = model_fixed.predict(X_fixed)
f_fixed = model_fixed.predict(X_fixed, operator=lambda x, y: pde(x, y,
Re_fixed))

# Print errors for fixed Re
print(f"Fixed Re = {Re_fixed}, Mean residual:
{np.mean(np.absolute(f_fixed))}")
print(f"Fixed Re = {Re_fixed}, L2 relative error:
{dde.metrics.l2_relative_error(y_true_fixed, y_pred_fixed)}")
np.savetxt(f"test_fixed_Re_{Re_fixed}.dat", np.hstack((X_fixed,
y_true_fixed, y_pred_fixed)))
```

```
Training for fixed Re = 50
Compiling model...
Building feed-forward neural network...
'build' took 0.060913 s

'compile' took 0.462253 s

Warning: epochs is deprecated and will be removed in a future version.
Use iterations instead.
Training model...

Step      Train loss                        Test loss
Test metric
0         [2.99e-02, 8.98e-02, 8.23e-02]    [2.99e-02, 8.98e-02,
8.23e-02]    []
1000      [4.80e-06, 1.77e-06, 3.61e-06]    [4.80e-06, 1.77e-06,
3.61e-06]    []
2000      [4.91e-07, 4.45e-07, 2.55e-06]    [4.91e-07, 4.45e-07,
2.55e-06]    []
3000      [2.96e-07, 4.06e-07, 2.50e-06]    [2.96e-07, 4.06e-07,
2.50e-06]    []
4000      [2.49e-07, 3.81e-07, 2.33e-06]    [2.49e-07, 3.81e-07,
2.33e-06]    []
5000      [2.21e-07, 3.49e-07, 2.10e-06]    [2.21e-07, 3.49e-07,
2.10e-06]    []
6000      [2.03e-07, 3.61e-07, 1.81e-06]    [2.03e-07, 3.61e-07,
1.81e-06]    []
7000      [1.69e-07, 2.51e-07, 1.48e-06]    [1.69e-07, 2.51e-07,
1.48e-06]    []
8000      [1.25e-07, 1.94e-07, 1.16e-06]    [1.25e-07, 1.94e-07,
1.16e-06]    []
9000      [4.66e-07, 1.97e-07, 9.64e-07]    [4.66e-07, 1.97e-07,
9.64e-07]    []
10000     [5.46e-08, 1.05e-07, 7.26e-07]    [5.46e-08, 1.05e-07,
7.26e-07]    []
11000     [1.21e-07, 3.92e-07, 1.11e-06]    [1.21e-07, 3.92e-07,
1.11e-06]    []
12000     [3.55e-08, 1.08e-07, 5.87e-07]    [3.55e-08, 1.08e-07,
```

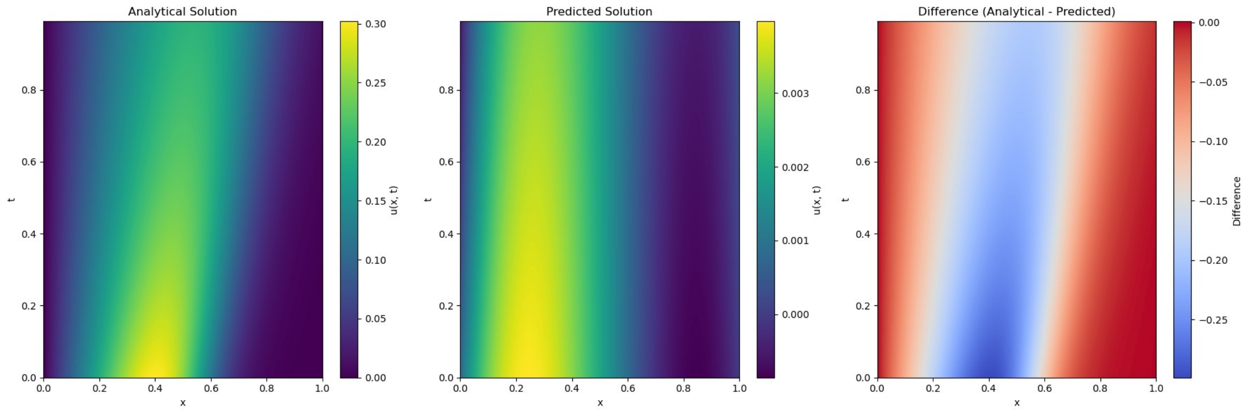
```
5.87e-07]    []
13000     [3.52e-08, 1.03e-07, 5.84e-07]    [3.52e-08, 1.03e-07,
5.84e-07]    []
14000     [4.00e-07, 2.23e-06, 1.40e-06]    [4.00e-07, 2.23e-06,
1.40e-06]    []
15000     [4.37e-08, 9.57e-08, 5.87e-07]    [4.37e-08, 9.57e-08,
5.87e-07]    []

Best model at step 13000:
  train loss: 7.23e-07
  test loss: 7.23e-07
  test metric: []

'train' took 41.743165 s

Compiling model...
'compile' took 0.283328 s

Training model...

Step      Train loss                        Test loss
Test metric
15000     [4.37e-08, 9.57e-08, 5.87e-07]    [4.37e-08, 9.57e-08,
5.87e-07]    []
INFO:tensorflow:Optimization terminated with:
  Message: CONVERGENCE: REL_REDUCTION_OF_F_<=_FACTR*EPSMCH
  Objective function value: 0.000001
  Number of iterations: 1
  Number of functions evaluations: 32
15019     [4.37e-08, 9.57e-08, 5.87e-07]    [4.37e-08, 9.57e-08,
5.87e-07]    []

Best model at step 13000:
  train loss: 7.23e-07
  test loss: 7.23e-07
  test metric: []

'train' took 0.705540 s

Saving loss history to c:\Users\jhyang\OneDrive\文档\GitHub_Projects\
ME_964\Final_Project\loss.dat ...
Saving training data to c:\Users\jhyang\OneDrive\文档\GitHub_Projects\
ME_964\Final_Project\train.dat ...
Saving test data to c:\Users\jhyang\OneDrive\文档\GitHub_Projects\
ME_964\Final_Project\test.dat ...
```


Train loss
Test loss
10^{-1}
10^{-2}
10^{-3}
10^{-4}
10^{-5}
10^{-6}
0
2000
4000
6000
8000
10000
12000
14000
Steps


0.003
0.002
0.001
0.000
v_1
0.0
0.2
0.4
0.6
0.8
1.0
x_1
0.0
0.2
0.4
0.6
0.8
1.0
x_2

```
Fixed Re = 50, Mean residual: 0.00013992008462082595
Fixed Re = 50, L2 relative error: 0.9921797726363656
```

```
# Reshape y_true and y_pred back into the shape of the grid for
plotting
y_true_reshaped = y_true_fixed.reshape(len(t), len(x))
y_pred_reshaped = y_pred_fixed.reshape(len(t), len(x))

# Create a figure with subplots for comparison
fig, axs = plt.subplots(1, 3, figsize=(18, 6))

# Plot analytical solution heatmap
# im1 = axs[0].imshow(y_true_reshaped, extent=[x.min(), x.max(),
t.min(), t.max()],
#                      origin='lower', aspect='auto', cmap='viridis')
im1 = axs[0].imshow(usol, extent=[x.min(), x.max(), t.min(), t.max()],

           origin='lower', aspect='auto', cmap='viridis')
axs[0].set_title("Analytical Solution")
axs[0].set_xlabel("x")
axs[0].set_ylabel("t")
fig.colorbar(im1, ax=axs[0], label="u(x, t)")

# Plot predicted solution heatmap
im2 = axs[1].imshow(y_pred_reshaped, extent=[x.min(), x.max(),
t.min(), t.max()],
                    origin='lower', aspect='auto', cmap='viridis')
axs[1].set_title("Predicted Solution")
axs[1].set_xlabel("x")
axs[1].set_ylabel("t")
fig.colorbar(im2, ax=axs[1], label="u(x, t)")

# Plot difference heatmap
diff = y_pred_reshaped - usol
im3 = axs[2].imshow(diff, extent=[x.min(), x.max(), t.min(), t.max()],
                    origin='lower', aspect='auto', cmap='coolwarm')
axs[2].set_title("Difference (Analytical - Predicted)")
axs[2].set_xlabel("x")
axs[2].set_ylabel("t")
fig.colorbar(im3, ax=axs[2], label="Difference")

# Adjust layout and show the plot
plt.tight_layout()
plt.show()
```



Part (b): Combined Inverse-Forward Problem

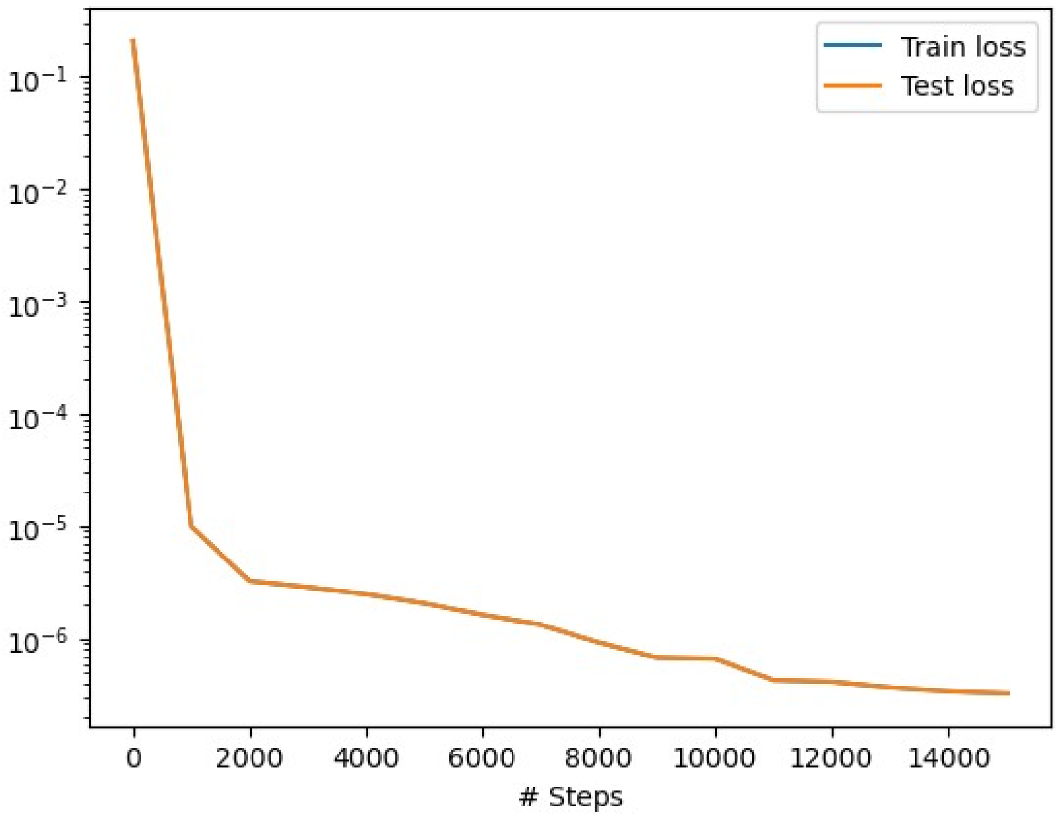
```
# Define Reynolds number as a trainable variable
Re_trainable = tf.Variable(Re_fixed, trainable=True, dtype=tf.float32)

# Generate analytical test data
def gen_testdata():
    data = np.load("dataset/Burgers.npz")  # Ensure the file is
present in the correct location
    t, x, exact = data["t"], data["x"], data["usol"].T
    xx, tt = np.meshgrid(x, t)
    X = np.vstack((np.ravel(xx), np.ravel(tt))).T
    y = exact.flatten()[:, None]
    return X, y

# Define the PDE
def pde_trainable(x, y):
    dy_x = dde.grad.jacobian(y, x, i=0, j=0)
    dy_t = dde.grad.jacobian(y, x, i=0, j=1)
    dy_xx = dde.grad.hessian(y, x, i=0, j=0)
    return dy_t + y * dy_x - 0.5 / Re_trainable * dy_xx #
Re=Re_fixed/100

# Define the domain and conditions
geom = dde.geometry.Interval(0, 1)
timedomain = dde.geometry.TimeDomain(0, 0.99)
geomtime = dde.geometry.GeometryXTime(geom, timedomain)

# Initial condition matches the analytical solution
to = tf.exp(Re_trainable / 8)
ic = dde.IC(
    geomtime,
    lambda x: x[:, 0:1] / (1 + tf.sqrt(to) * tf.exp(Re_trainable *
x[:, 0:1]**2 / 4)),
    lambda _, on_initial: on_initial,
)
```

```python
# Dirichlet boundary conditions
bc = dde.DirichletBC(geomtime, lambda x: 0, lambda _, on_boundary:
on_boundary)

# Solve the combined inverse-forward problem
print("Training for combined inverse-forward problem")

# Define dataset for trainable Re
data_trainable = dde.data.TimePDE(
    geomtime, pde_trainable, [bc, ic], num_domain=2540,
num_boundary=80, num_initial=160
)

# Define the neural network
net_trainable = dde.maps.FNN([2] + [20] * 3 + [1], "tanh", "Glorot
normal")

# Compile the model
model_trainable = dde.Model(data_trainable, net_trainable)
model_trainable.compile("adam", lr=1e-3)

# Train the model
model_trainable.train(epochs=15000)
model_trainable.compile("L-BFGS")
losshistory_trainable, train_state_trainable = model_trainable.train()

# Save results
dde.saveplot(losshistory_trainable, train_state_trainable,
issave=True, isplot=True)

# Test the model
X_trainable, y_true_trainable = gen_testdata()
y_pred_trainable = model_trainable.predict(X_trainable)
f_trainable = model_trainable.predict(X_trainable,
operator=pde_trainable)

# Print errors
print("Mean residual for trainable Re:",
np.mean(np.absolute(f_trainable)))
print("L2 relative error for trainable Re:",
dde.metrics.l2_relative_error(y_true_trainable, y_pred_trainable))

# Use a TensorFlow session to evaluate Re_trainable
with tf.compat.v1.Session() as sess:
    sess.run(tf.compat.v1.global_variables_initializer())
    learned_Re = sess.run(Re_trainable)

print("Learned Reynolds number:", learned_Re)

# Save test results
```

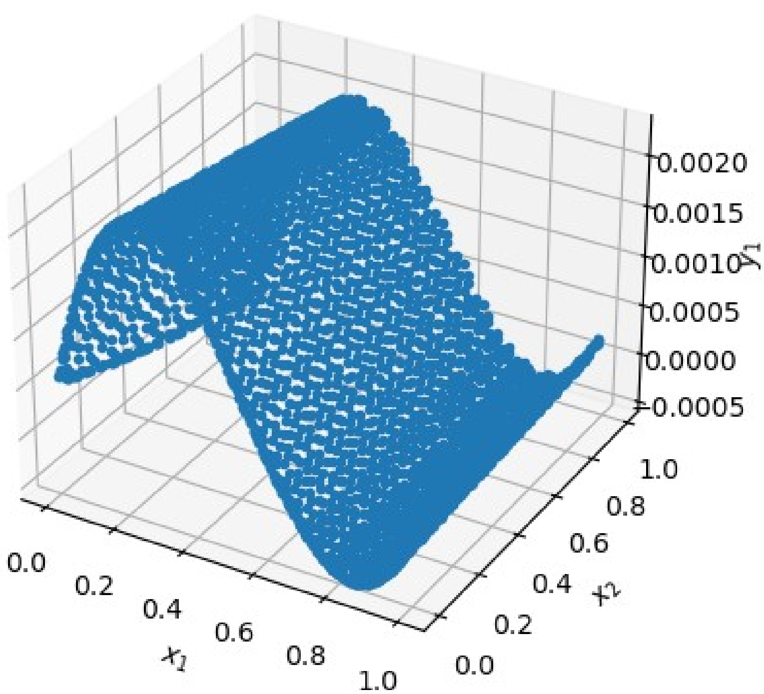
```
np.savetxt(f"test_trainable_Re_{learned_Re}.dat",
np.hstack((X_trainable, y_true_trainable, y_pred_trainable)))
```

```
Training for combined inverse-forward problem
Compiling model...
Building feed-forward neural network...
'build' took 0.059386 s

'compile' took 0.518670 s

Warning: epochs is deprecated and will be removed in a future version.
Use iterations instead.
Training model...

Step      Train loss                        Test loss
Test metric
0         [2.99e-02, 9.48e-02, 8.23e-02]    [2.99e-02, 9.48e-02,
8.23e-02]    []
1000      [4.57e-06, 1.81e-06, 3.55e-06]    [4.57e-06, 1.81e-06,
3.55e-06]    []
2000      [4.40e-07, 4.34e-07, 2.38e-06]    [4.40e-07, 4.34e-07,
2.38e-06]    []
3000      [2.69e-07, 3.74e-07, 2.22e-06]    [2.69e-07, 3.74e-07,
2.22e-06]    []
4000      [2.17e-07, 3.29e-07, 1.94e-06]    [2.17e-07, 3.29e-07,
1.94e-06]    []
5000      [1.79e-07, 2.79e-07, 1.61e-06]    [1.79e-07, 2.79e-07,
1.61e-06]    []
6000      [1.51e-07, 2.19e-07, 1.26e-06]    [1.51e-07, 2.19e-07,
1.26e-06]    []
7000      [1.27e-07, 2.46e-07, 9.58e-07]    [1.27e-07, 2.46e-07,
9.58e-07]    []
8000      [8.93e-08, 1.27e-07, 7.05e-07]    [8.93e-08, 1.27e-07,
7.05e-07]    []
9000      [5.56e-08, 9.21e-08, 5.30e-07]    [5.56e-08, 9.21e-08,
5.30e-07]    []
10000     [1.57e-07, 7.80e-08, 4.28e-07]    [1.57e-07, 7.80e-08,
4.28e-07]    []
11000     [2.28e-08, 5.46e-08, 3.49e-07]    [2.28e-08, 5.46e-08,
3.49e-07]    []
12000     [1.99e-08, 4.13e-08, 3.51e-07]    [1.99e-08, 4.13e-08,
3.51e-07]    []
13000     [1.52e-08, 3.27e-08, 3.19e-07]    [1.52e-08, 3.27e-08,
3.19e-07]    []
14000     [1.65e-08, 4.14e-08, 2.82e-07]    [1.65e-08, 4.14e-08,
2.82e-07]    []
15000     [1.50e-08, 4.64e-08, 2.64e-07]    [1.50e-08, 4.64e-08,
2.64e-07]    []

Best model at step 15000:
```

```
  train loss: 3.25e-07
  test loss: 3.25e-07
  test metric: []

'train' took 42.691809 s

Compiling model...
'compile' took 0.347429 s

Training model...

Step      Train loss                        Test loss
Test metric
15000     [1.50e-08, 4.64e-08, 2.64e-07]    [1.50e-08, 4.64e-08,
2.64e-07]    []
INFO:tensorflow:Optimization terminated with:
  Message: CONVERGENCE: REL_REDUCTION_OF_F_<=_FACTR*EPSMCH
  Objective function value: 0.000000
  Number of iterations: 1
  Number of functions evaluations: 35
15018     [1.50e-08, 4.64e-08, 2.64e-07]    [1.50e-08, 4.64e-08,
2.64e-07]    []

Best model at step 15000:
  train loss: 3.25e-07
  test loss: 3.25e-07
  test metric: []

'train' took 0.913704 s

Saving loss history to c:\Users\jhyang\OneDrive\文档\GitHub_Projects\
ME_964\Final_Project\loss.dat ...
Saving training data to c:\Users\jhyang\OneDrive\文档\GitHub_Projects\
ME_964\Final_Project\train.dat ...
Saving test data to c:\Users\jhyang\OneDrive\文档\GitHub_Projects\
ME_964\Final_Project\test.dat ...
```



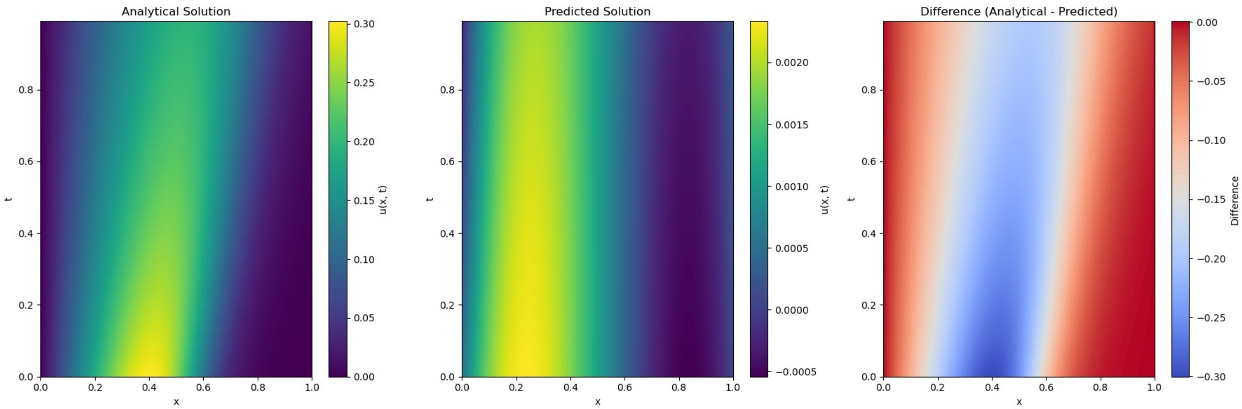
Train loss
Test loss
10^{-1}
10^{-2}
10^{-3}
10^{-4}
10^{-5}
10^{-6}
0
2000
4000
6000
8000
10000
12000
14000
Steps



0.0020
0.0015
0.0010
0.0005
0.0000
−0.0005
y_1
0.0
0.2
0.4
0.6
0.8
1.0
x_1
0.0
0.2
0.4
0.6
0.8
1.0
x_2

```
Mean residual for trainable Re: 9.480372e-05
L2 relative error for trainable Re: 0.9953817047510793
Learned Reynolds number: 50.0
```

```
# Reshape y_true and y_pred back into the shape of the grid for
plotting
y_true_reshaped = y_true_trainable.reshape(len(t), len(x))
y_pred_reshaped = y_pred_trainable.reshape(len(t), len(x))

# Create a figure with subplots for comparison
fig, axs = plt.subplots(1, 3, figsize=(18, 6))

# Plot analytical solution heatmap
# im1 = axs[0].imshow(y_true_reshaped, extent=[x.min(), x.max(),
t.min(), t.max()],
#                     origin='lower', aspect='auto', cmap='viridis')
im1 = axs[0].imshow(usol, extent=[x.min(), x.max(), t.min(), t.max()],

           origin='lower', aspect='auto', cmap='viridis')
axs[0].set_title("Analytical Solution")
axs[0].set_xlabel("x")
axs[0].set_ylabel("t")
fig.colorbar(im1, ax=axs[0], label="u(x, t)")

# Plot predicted solution heatmap
im2 = axs[1].imshow(y_pred_reshaped, extent=[x.min(), x.max(),
t.min(), t.max()],
                    origin='lower', aspect='auto', cmap='viridis')
axs[1].set_title("Predicted Solution")
axs[1].set_xlabel("x")
axs[1].set_ylabel("t")
fig.colorbar(im2, ax=axs[1], label="u(x, t)")

# Plot difference heatmap
diff = y_pred_reshaped - usol
im3 = axs[2].imshow(diff, extent=[x.min(), x.max(), t.min(), t.max()],
                    origin='lower', aspect='auto', cmap='coolwarm')
axs[2].set_title("Difference (Analytical - Predicted)")
axs[2].set_xlabel("x")
axs[2].set_ylabel("t")
fig.colorbar(im3, ax=axs[2], label="Difference")

# Adjust layout and show the plot
plt.tight_layout()
plt.show()
```



Analytical Solution
t
x
u(x, t)
Predicted Solution
t
x
u(x, t)
Difference (Analytical - Predicted)
t
x
Difference