Pinn

April 12, 2025

[1]:
```
import numpy as np
import os
import sys
import gym
import zipfile
import autograd
import matplotlib.gridspec as gridspec
# Use tf.random.set_seed for TensorFlow 2.0 and above
#from scipy.signal.waveforms import square
import matplotlib.pyplot as plt
from scipy.integrate import solve_ivp
from sklearn.model_selection import train_test_split
import random
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.models import Sequential, model_from_json
from keras.layers import Dense
from keras.layers import Input
from tensorflow.keras import layers
```

```
2025-04-12 22:13:58.973915: I external/local_xla/xla/tsl/cuda/cudart_stub.cc:32]
Could not find cuda drivers on your machine, GPU will not be used.
2025-04-12 22:13:58.985565: I external/local_xla/xla/tsl/cuda/cudart_stub.cc:32]
Could not find cuda drivers on your machine, GPU will not be used.
2025-04-12 22:13:59.019039: E
external/local_xla/xla/stream_executor/cuda/cuda_fft.cc:485] Unable to register
cuFFT factory: Attempting to register factory for plugin cuFFT when one has
already been registered
2025-04-12 22:13:59.066104: E
external/local_xla/xla/stream_executor/cuda/cuda_dnn.cc:8454] Unable to register
cuDNN factory: Attempting to register factory for plugin cuDNN when one has
already been registered
2025-04-12 22:13:59.081321: E
external/local_xla/xla/stream_executor/cuda/cuda_blas.cc:1452] Unable to
register cuBLAS factory: Attempting to register factory for plugin cuBLAS when
one has already been registered
2025-04-12 22:13:59.117395: I tensorflow/core/platform/cpu_feature_guard.cc:210]
This TensorFlow binary is optimized to use available CPU instructions in
```

```
performance-critical operations.
To enable the following instructions: AVX2 FMA, in other operations, rebuild
TensorFlow with the appropriate compiler flags.
2025-04-12 22:14:03.309084: W
tensorflow/compiler/tf2tensorrt/utils/py_utils.cc:38] TF-TRT Warning: Could not
find TensorRT
```

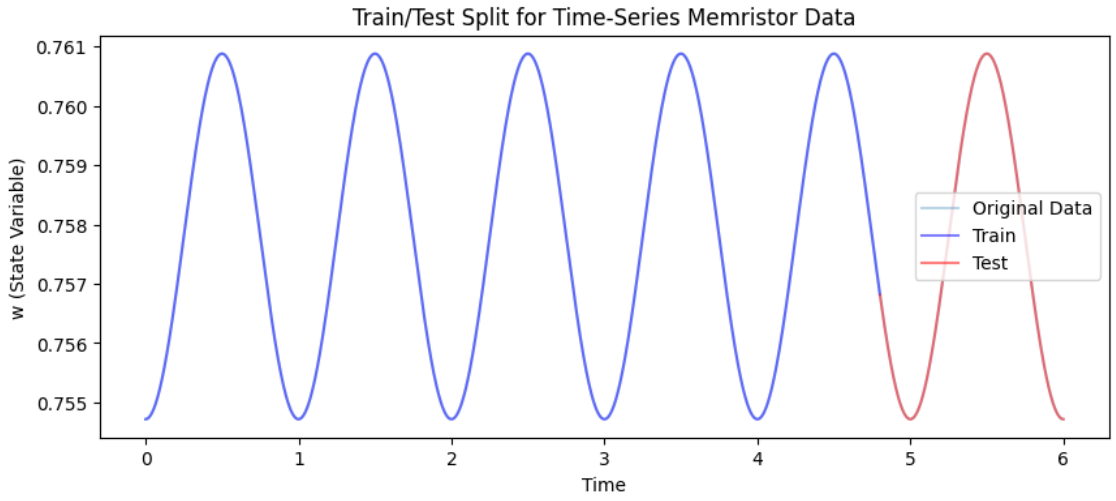
[1]:
```latex
# @title  Hp meristor's state variable:
from IPython.display import display, Math

latex_equation = r"""
\text{State variable:}\quad \frac{dw}{dt} = \mu_\text{v} \cdot \left(␣
 ↪\frac{R_{\text{on}}}{D^2} \right) \cdot i(t) \cdot f(w) \\
\text{Window function:}\quad f(w) = {w(1 - w)}\\
\text{state variable in this code is w:}\quad w =  \frac{X}{D}
"""
display(Math(latex_equation))
```

State variable: $\frac{dw}{dt} = \mu_{\text{v}} \cdot \left(\frac{R_{\text{on}}}{D^2}\right) \cdot i(t) \cdot f(w)$
Window function: $f(w) = w(1 - w)$
state variable in this code is w: $w = \frac{X}{D}$

[9]:
```python
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import solve_ivp
from sklearn.preprocessing import StandardScaler, MinMaxScaler

# -------------- Physical Parameters ----------------
frequency = 1
A_train = 1.5
W_train = 2 * np.pi * frequency

mu_v = 10**4
D = 60
r_on = 0.1
r_off = 16
r0 = 4
w0 = (r0 - r_off) / (r_on - r_off)

points_per_period = 600

total_points = 10 * points_per_period

# -------------- Solving ODE ----------------
def f(t, w, A, W, mu_v, D, r_on, r_off):
    k = mu_v * (r_on / D**2)
```

```
    f_w = w * (1 - w)
    r = r_on * w + r_off * (1 - w)
    I = A * np.sin(W * t) / r
    return I * f_w * k

t_all = np.linspace(0, 6, total_points)
sol_all=solve_ivp(f, (0, 6), [w0], t_eval=t_all, args=(A_train, W_train, mu_v,␣
 ↪D, r_on, r_off),
          method='RK45', max_step=0.001)


w_all = sol_all.y[0]
v_all = A_train * np.sin(W_train * t_all)
r_all = r_on * w_all + r_off * (1 - w_all)
I_all = v_all / r_all

X_all = np.column_stack([t_all[:-1], w_all[:-1], I_all[:-1]])
y_all = w_all[1:]

# -------------- Split Data -----------------
test_ratio = 0.2
test_size = int(test_ratio * len(X_all))


test_index = np.arange(len(X_all) - test_size, len(X_all))
train_index = np.arange(0, len(X_all) - test_size)


X_train, X_test = X_all[train_index], X_all[test_index]
y_train, y_test = y_all[train_index], y_all[test_index]

plt.figure(figsize=(10, 4))
plt.plot(t_all[:-1], w_all[:-1], label="Original Data", alpha=0.3)
plt.plot(X_train[:, 0], y_train, color='blue', alpha=0.5, label='Train')
plt.plot(X_test[:, 0], y_test, color='red', alpha=0.5, label='Test')
plt.xlabel("Time")
plt.ylabel("w (State Variable)")
plt.title("Train/Test Split for Time-Series Memristor Data")

plt.legend()
plt.savefig("rungkutta_train_test.pdf")
plt.show()

from sklearn.preprocessing import StandardScaler, MinMaxScaler
import numpy as np

# ------------------       ------------------
```

```
scaler_time = StandardScaler()
X_train[:, 0] = scaler_time.fit_transform(X_train[:, 0].reshape(-1, 1)).
 ↪flatten()
X_test[:, 0] = scaler_time.transform(X_test[:, 0].reshape(-1, 1)).flatten()

scaler_features = MinMaxScaler(feature_range=(-1, 1))
X_train[:, 1:] = scaler_features.fit_transform(X_train[:, 1:])
X_test[:, 1:] = scaler_features.transform(X_test[:, 1:])

scaler_y = MinMaxScaler(feature_range=(-1, 1))
y_train_scaled = scaler_y.fit_transform(y_train.reshape(-1, 1))
y_test_scaled = scaler_y.transform(y_test.reshape(-1, 1))

X_train_scaled = X_train
X_test_scaled = X_test


print("Mean of X_train_scaled:", np.mean(X_train, axis=0))
print("Std of X_train_scaled:", np.std(X_train, axis=0))
print("Mean of X_test_scaled:", np.mean(X_test, axis=0))
print("Std of X_test_scaled:", np.std(X_test, axis=0))
print("Mean of y_train_scaled:", np.mean(y_train_scaled))
print("Std of y_train_scaled:", np.std(y_train_scaled))
print("Mean of y_test_scaled:", np.mean(y_test_scaled))
print("Std of y_test_scaled:", np.std(y_test_scaled))
```




```
Mean of X_train_scaled: [-6.65671222e-17  3.04609012e-02  2.26622042e-02]
Std of X_train_scaled: [1.         0.70298873 0.71031816]
Mean of X_test_scaled: [ 2.16470271 -0.12662702 -0.09072466]
Std of X_test_scaled: [0.24979159 0.70958015 0.68660316]
```

```
Mean of y_train_scaled: 0.03060324437440435
Std of y_train_scaled: 0.7028492870931501
Mean of y_test_scaled: -0.12719823824974452
Std of y_test_scaled: 0.7100073018388313
```

[3]:
```python
import random
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.layers import Dense, Input, Dropout
from tensorflow.keras import Sequential, regularizers

# ---------------- SEED ----------------
SEED = 42
tf.random.set_seed(SEED)
np.random.seed(SEED)
random.seed(SEED)

# ---------------- PINN Model ----------------
class PhysicsInformedNN(tf.keras.Model):
    def __init__(self):
        super().__init__()
        self.NN = tf.keras.Sequential([
            tf.keras.layers.Input((3,)),
            tf.keras.layers.Dense(128, activation='tanh'),
            tf.keras.layers.Dense(128, activation='tanh'),
            tf.keras.layers.Dense(128, activation='tanh'),
            tf.keras.layers.Dense(128, activation='tanh'),
            tf.keras.layers.Dense(1)
        ])

    def call(self, inputs):
        return self.NN(inputs)


    def get_config(self):
        config = super().get_config()
        return config


    @classmethod
    def from_config(cls, config):
        return cls()

# ---------------- Initialize Model ----------------
pinn_model = PhysicsInformedNN()
pinn_model.build((None, 3))
```

```
pinn_model.summary()
```

```
WARNING: All log messages before absl::InitializeLog() is called are written to
STDERR
I0000 00:00:1744483456.437439   14174 cuda_executor.cc:1015] successful NUMA
node read from SysFS had negative value (-1), but there must be at least one
NUMA node, so returning NUMA node zero. See more at
https://github.com/torvalds/linux/blob/v6.0/Documentation/ABI/testing/sysfs-bus-
pci#L344-L355
2025-04-12 22:14:16.438724: W
tensorflow/core/common_runtime/gpu/gpu_device.cc:2343] Cannot dlopen some GPU
libraries. Please make sure the missing libraries mentioned above are installed
properly if you would like to use GPU. Follow the guide at
https://www.tensorflow.org/install/gpu for how to download and setup the
required libraries for your platform.
Skipping registering GPU devices…
```

Model: "physics_informed_nn"

Layer (type)	Output Shape	Param #
sequential (Sequential)	?	50,177

Total params: 50,177 (196.00 KB)

Trainable params: 50,177 (196.00 KB)

Non-trainable params: 0 (0.00 B)

[4]:
```
N0 = 1
Nf = X_train.shape[0]
Nd = y_train.shape[0]

#col_weights = tf.Variable(1.0)   # for ode loss
#u_weights = tf.Variable(1.0)     # for ic loss
#data_weights = tf.Variable(1.5)  # for data loss
#----------------------------------------
col_weights = tf.Variable(tf.ones(Nf), dtype=tf.float32)   #  weight of ODE
data_weights = tf.Variable(tf.ones(Nd), dtype=tf.float32)  # weight of data
u_weights = tf.Variable(tf.ones(N0), dtype=tf.float32)

optimizer_col_weights = tf.keras.optimizers.Adam(learning_rate=1e-2)
optimizer_data_weights = tf.keras.optimizers.Adam(learning_rate=1e-4)
```

```
print("done")
```

```
done
```

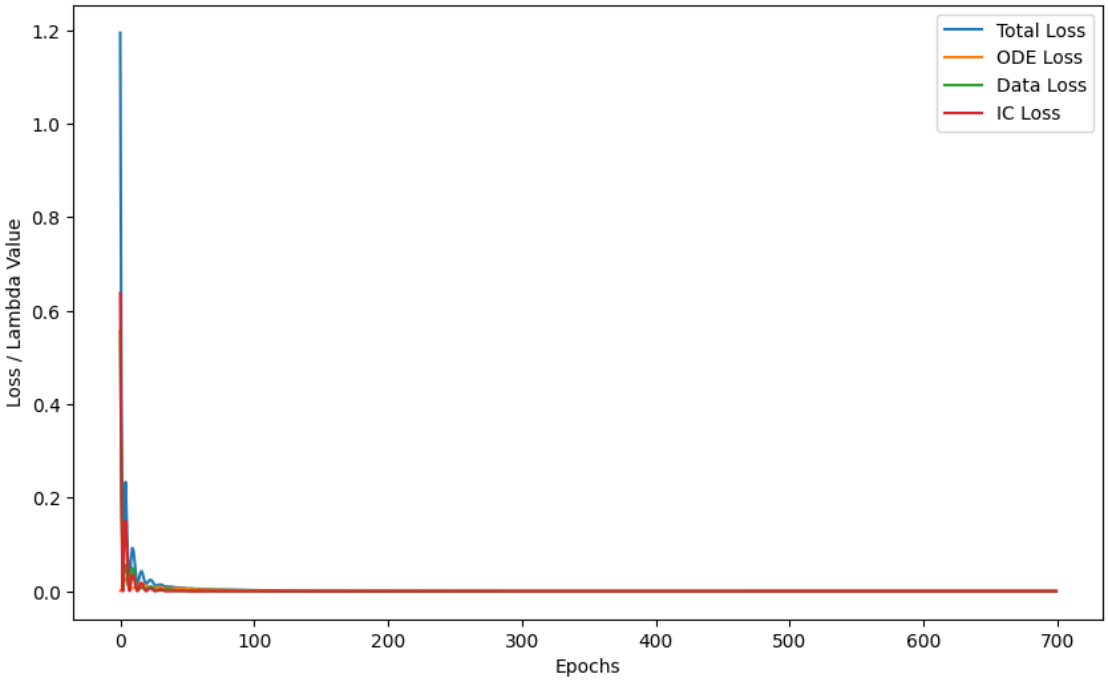
[5]:
```
def compute_loss(X, y_true, model, col_weights, u_weights, data_weights):
    X = tf.convert_to_tensor(X, dtype=tf.float32)
    y_true = tf.convert_to_tensor(y_true, dtype=tf.float32)

    with tf.GradientTape() as tape:
        w_pred = model(X)

        T = X[:, 0:1]
        w_prev = X[:, 1:2]
        I_t = X[:, 2:3]
        f_w = w_pred * (1 - w_pred)

        with tf.GradientTape() as g:
            g.watch(T)
            w_pred_g = model(tf.stack([T[:, 0], w_prev[:, 0], I_t[:, 0]],␣
 ↪axis=1))
        dw_dt = g.gradient(w_pred_g, T)

        ode_res = dw_dt - mu_v * (r_on / D**2) * I_t * f_w

        ode_loss = tf.reduce_mean(tf.square(col_weights * ode_res))
        data_loss = tf.reduce_mean(tf.square(data_weights * (w_pred - y_true)))


        ic_input = tf.convert_to_tensor([[0.0, y_train[0], X_train[0, 2]]],␣
 ↪dtype=tf.float32)
        ic_pred = model(ic_input)
        ic_true = tf.convert_to_tensor(y_train[0], dtype=tf.float32)
        ic_loss = tf.reduce_mean(tf.square(u_weights * (ic_pred - ic_true)))

        total_loss = data_loss + ode_loss + ic_loss

        return total_loss, ode_loss, data_loss, ic_loss


########################################################## trainig loop

from tensorflow.keras.optimizers.schedules import ExponentialDecay
#earning_rate = tf.keras.optimizers.schedules.ExponentialDecay(
   #initial_learning_rate=1e-3
    #ecay_steps=10000 ,
    #ecay_rate=0.75
```

```
#
learning_rate=1e-3



optimizer=tf.keras.optimizers.Adam(learning_rate, clipnorm=1.0)

epochs=700

train_loss_record,ode_loss_record,data_loss_record,ic_loss_record=[],[],[],[]
for epoch in range(epochs):
    with tf.GradientTape(persistent=True) as tape:
        total_loss, ode_loss, data_loss, ic_loss = compute_loss(X_train_scaled,␣
 ↪y_train_scaled
                                                , pinn_model, col_weights,␣
 ↪u_weights, data_weights)

    grads = tape.gradient(total_loss, pinn_model.trainable_variables)

    grads_data = tape.gradient(data_loss, pinn_model.trainable_variables)
    grads_ode = tape.gradient(ode_loss, pinn_model.trainable_variables)

    grads_col = tape.gradient(ode_loss, col_weights)
    grads_data_weights = tape.gradient(data_loss, data_weights)


    optimizer.apply_gradients(zip(grads, pinn_model.trainable_variables))

    if grads_col is not None:
        optimizer_col_weights.apply_gradients(zip([grads_col], [col_weights]))
    if grads_data_weights is not None:
        optimizer_data_weights.apply_gradients(zip([grads_data_weights],␣
 ↪[data_weights]))




    train_loss_record.append(total_loss.numpy())
    ode_loss_record.append(ode_loss.numpy())
    data_loss_record.append(data_loss.numpy())
    ic_loss_record.append(ic_loss.numpy())


    if (epoch + 1) % 100 == 0:
        print(f"Epoch {epoch+1}/{epochs} | "
              f"Total Loss: {train_loss_record[-1]:.6f} | "
              f"ODE Loss: {ode_loss_record[-1]:.6f} | "
```

```
              f"Data Loss: {data_loss_record[-1]:.6f} | "
              f"IC Loss: {ic_loss_record[-1]:.6f} | ")


# ---------------- Plot Adaptive Lambda Values ----------------
plt.figure(figsize=(10, 6))
plt.plot(train_loss_record, label='Total Loss')
plt.plot(ode_loss_record, label='ODE Loss')
plt.plot(data_loss_record, label='Data Loss')
plt.plot(ic_loss_record, label='IC Loss')
plt.xlabel("Epochs")
plt.ylabel("Loss / Lambda Value")
plt.legend()

plt.show()
```

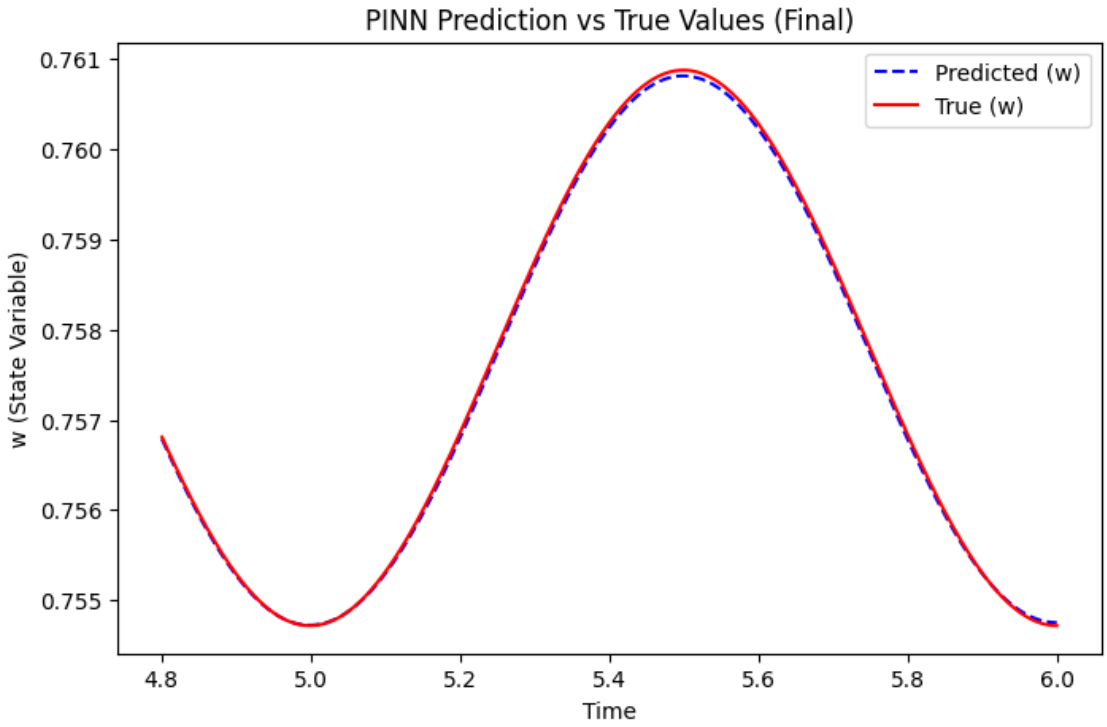
```
2025-04-12 22:14:24.242539: W
external/local_tsl/tsl/framework/cpu_allocator_impl.cc:83] Allocation of
92160000 exceeds 10% of free system memory.
2025-04-12 22:14:24.264016: W
external/local_tsl/tsl/framework/cpu_allocator_impl.cc:83] Allocation of
92160000 exceeds 10% of free system memory.
2025-04-12 22:14:24.303932: W
external/local_tsl/tsl/framework/cpu_allocator_impl.cc:83] Allocation of
92160000 exceeds 10% of free system memory.
2025-04-12 22:14:24.325020: W
external/local_tsl/tsl/framework/cpu_allocator_impl.cc:83] Allocation of
92160000 exceeds 10% of free system memory.
2025-04-12 22:14:24.385884: W
external/local_tsl/tsl/framework/cpu_allocator_impl.cc:83] Allocation of
92160000 exceeds 10% of free system memory.
```

```
Epoch 100/700 | Total Loss: 0.001871 | ODE Loss: 0.001518 | Data Loss: 0.000271
| IC Loss: 0.000082 |
Epoch 200/700 | Total Loss: 0.000147 | ODE Loss: 0.000043 | Data Loss: 0.000088
| IC Loss: 0.000015 |
Epoch 300/700 | Total Loss: 0.000062 | ODE Loss: 0.000000 | Data Loss: 0.000055
| IC Loss: 0.000007 |
Epoch 400/700 | Total Loss: 0.000047 | ODE Loss: 0.000000 | Data Loss: 0.000041
| IC Loss: 0.000006 |
Epoch 500/700 | Total Loss: 0.000037 | ODE Loss: 0.000000 | Data Loss: 0.000032
| IC Loss: 0.000005 |
Epoch 600/700 | Total Loss: 0.000030 | ODE Loss: 0.000000 | Data Loss: 0.000026
| IC Loss: 0.000004 |
Epoch 700/700 | Total Loss: 0.000025 | ODE Loss: 0.000000 | Data Loss: 0.000021
| IC Loss: 0.000004 |
```



[6]:
```
w_pred = pinn_model.predict(X_test_scaled)
w_pred_rescaled = scaler_y.inverse_transform(w_pred.reshape(-1, 1))

y_test_rescaled = scaler_y.inverse_transform(y_test_scaled.reshape(-1, 1))

plt.figure(figsize=(8, 5))
plt.plot(t_all[test_index], w_pred_rescaled, label="Predicted (w)",␣
 ↪linestyle="dashed", color="blue")
plt.plot(t_all[test_index], y_test_rescaled, label="True (w)",␣
 ↪linestyle="solid", color="red")

plt.xlabel("Time")
plt.ylabel("w (State Variable)")
plt.legend()

plt.title("PINN Prediction vs True Values (Final)")
plt.savefig("test_data2.pdf")
plt.show()
```

```
38/38             0s 5ms/step
```



[11]:
```
import os

save_dir = "./models"
if not os.path.exists(save_dir):
    os.makedirs(save_dir)

save_path = os.path.join(save_dir, "my_pinn_model.keras")

pinn_model.save(save_path)
print(f"Model saved at: {save_path}")
```

```
Model saved at: ./models/my_pinn_model.keras
```

[12]:
```
from tensorflow.keras.models import load_model

save_path = "./models/my_pinn_model.keras"

if os.path.exists(save_path):
    loaded_pinn_model = load_model(save_path,␣
 ↪custom_objects={'PhysicsInformedNN': PhysicsInformedNN})
    print("  Model loaded successfully!")
else:
    print("  Error: Model file not found!")
```

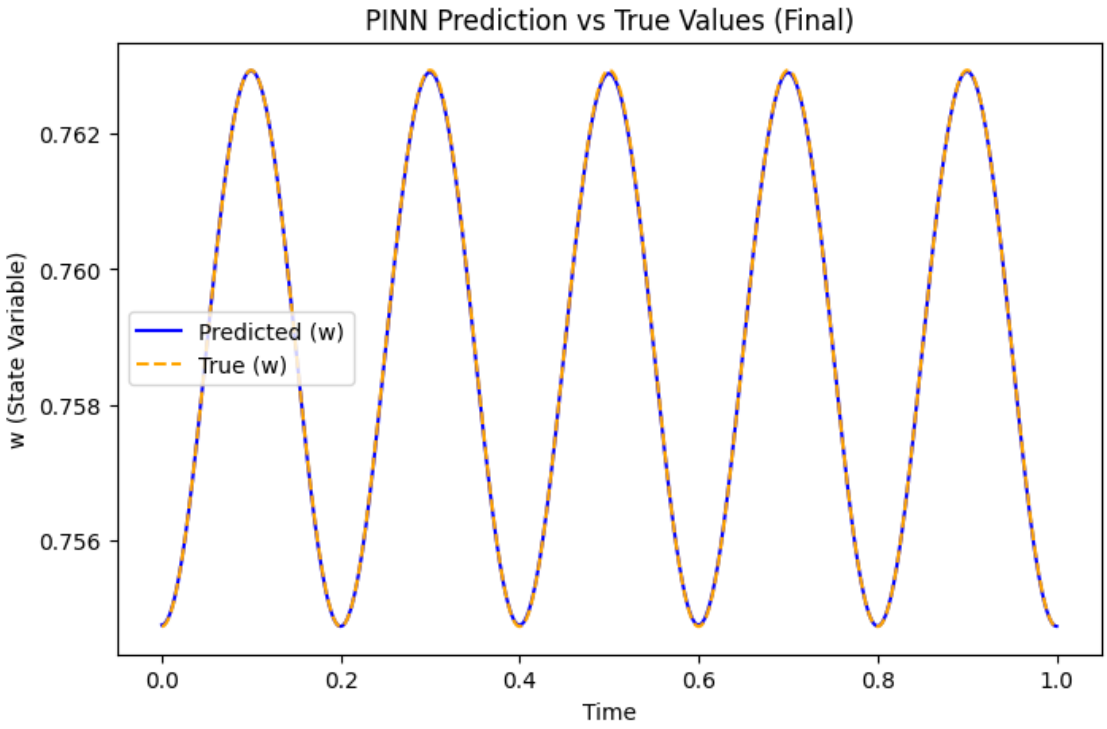
```
Model loaded successfully!
```

[7]:
```python
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import solve_ivp

frequency =5
A_train = 10
W_train = 2 * np.pi * frequency
mu_v = 10**4
D = 60
r_on = 0.1
r_off = 16
r0 = 4
w0 = (r0 - r_off) / (r_on - r_off)

points_per_period = 600
total_points = 10 * points_per_period


def f(t, w, A, W, mu_v, D, r_on, r_off):

    k = mu_v * (r_on / D**2)
    f_w = w * (1 - w)
    r = r_on * w + r_off * (1 - w)
    I = A * np.sin(W * t) / r
    return I * f_w * k


t_all = np.linspace(0, 1, total_points)
sol_all = solve_ivp(
    f,
    (0, 1),
    [w0],
    t_eval=t_all,
    args=(A_train, W_train, mu_v, D, r_on, r_off),
    method='RK45',
    max_step=0.001
)


w_all = sol_all.y[0]
v_all = A_train * np.sin(W_train * t_all)
r_all = r_on * w_all + r_off * (1 - w_all)
I_all = v_all / r_all

X_all = np.column_stack([t_all[:-1], w_all[:-1], I_all[:-1]])
```

```
y_all = w_all[1:]

"""
plt.figure(figsize=(8,4))
plt.plot(t_all, w_all, label="w(t)")
plt.xlabel("Time")
plt.ylabel("w(t)")
plt.title("Memristor ODE Solution with A=10, frequency=2 (0-6s)")
plt.legend()
plt.grid(True)
plt.show()
"""
print("X_all shape:", X_all.shape)
print("y_all shape:", y_all.shape)

########################################################## normalize
##########################
from sklearn.preprocessing import StandardScaler, MinMaxScaler


X_all_scaled = X_all.copy()

scaler_time = StandardScaler()
X_all_scaled[:, 0] = scaler_time.fit_transform(X_all_scaled[:, 0].reshape(-1,
1)).flatten()


scaler_features = MinMaxScaler(feature_range=(-1, 1))
X_all_scaled[:, 1:] = scaler_features.fit_transform(X_all_scaled[:, 1:])

scaler_y = MinMaxScaler(feature_range=(-1,1))
y_all_scaled = scaler_y.fit_transform(y_all.reshape(-1,1))


print("After scaling, X_all_scaled shape:", X_all_scaled.shape)
print("After scaling, y_all_scaled shape:", y_all_scaled.shape)

w_pred_new = pinn_model.predict(X_all_scaled)
w_pred_new_rescaled = scaler_y.inverse_transform(w_pred_new.reshape(-1, 1))

y_all_rescaled = scaler_y.inverse_transform(y_all_scaled.reshape(-1, 1))

plt.figure(figsize=(8, 5))
plt.plot(t_all[:-1], w_pred_new_rescaled, label="Predicted (w)", color="blue")
plt.plot(t_all[:-1], y_all_rescaled, label="True (w)", linestyle="dashed",
color='orange')
```

```
plt.xlabel("Time")
plt.ylabel("w (State Variable)")
plt.legend()
#plt.grid()
plt.title("PINN Prediction vs True Values (Final)")
plt.savefig("test_data4.pdf")
plt.show()
```

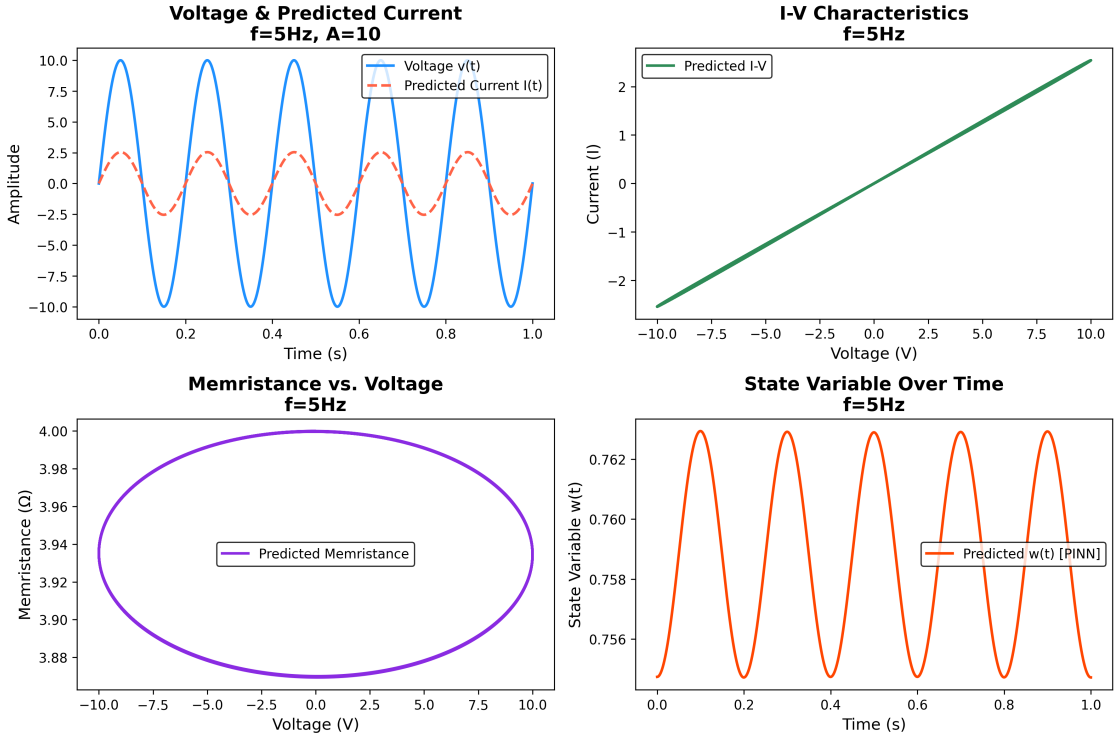
```
X_all shape: (5999, 3)
y_all shape: (5999,)
After scaling, X_all_scaled shape: (5999, 3)
After scaling, y_all_scaled shape: (5999, 1)
188/188             0s 2ms/step
```



[8]:
```
r_pred_new = r_on * w_pred_new_rescaled + r_off * (1.0 - w_pred_new_rescaled)

r_pred_new = r_pred_new.reshape(-1)


I_pred_new = v_all[:-1] / r_pred_new


t_all_plot = t_all[:-1]
```

```
w_pred_new_rescaled = w_pred_new_rescaled.flatten()

fig, axes = plt.subplots(2, 2, figsize=(12, 8), dpi=300)  #


axes[0, 0].plot(t_all, v_all, color='dodgerblue', linewidth=2, label='Voltage␣
 ↪v(t)')  #
axes[0, 0].plot(t_all_plot, I_pred_new, color='tomato', linestyle='dashed',␣
 ↪linewidth=2, label='Predicted Current I(t)')
axes[0, 0].set_xlabel('Time (s)', fontsize=12)
axes[0, 0].set_ylabel('Amplitude', fontsize=12)
axes[0, 0].set_title(f'Voltage & Predicted Current\nf={frequency}Hz,␣
 ↪A={A_train}', fontsize=14, fontweight='bold')
axes[0, 0].legend(fontsize=10, frameon=True, edgecolor='black')
#axes[0, 0].grid(True, linestyle='--', alpha=0.4)


axes[0, 1].plot(v_all[:-1], I_pred_new, color='seagreen', linewidth=2,␣
 ↪linestyle='-', label='Predicted I-V')  #
axes[0, 1].set_xlabel('Voltage (V)', fontsize=12)
axes[0, 1].set_ylabel('Current (I)', fontsize=12)
axes[0, 1].set_title(f'I-V Characteristics\nf={frequency}Hz', fontsize=14,␣
 ↪fontweight='bold')
axes[0, 1].legend(fontsize=10, frameon=True, edgecolor='black')
#axes[0, 1].grid(True, linestyle='--', alpha=0.4)


axes[1, 0].plot(v_all[:-1], r_pred_new, color='blueviolet', linewidth=2,␣
 ↪linestyle='-', label='Predicted Memristance')  #
axes[1, 0].set_xlabel('Voltage (V)', fontsize=12)
axes[1, 0].set_ylabel('Memristance (Ω)', fontsize=12)
axes[1, 0].set_title(f'Memristance vs. Voltage\nf={frequency}Hz', fontsize=14,␣
 ↪fontweight='bold')
axes[1, 0].legend(fontsize=10, frameon=True, edgecolor='black')
#axes[1, 0].grid(True, linestyle='--', alpha=0.4)


axes[1, 1].plot(t_all_plot, w_pred_new_rescaled, color='orangered',␣
 ↪linewidth=2, linestyle='-', label='Predicted w(t) [PINN]')  #
axes[1, 1].set_xlabel('Time (s)', fontsize=12)
axes[1, 1].set_ylabel('State Variable w(t)', fontsize=12)
axes[1, 1].set_title(f'State Variable Over Time\nf={frequency}Hz', fontsize=14,␣
 ↪fontweight='bold')
axes[1, 1].legend(fontsize=10, frameon=True, edgecolor='black')
#axes[1, 1].grid(True, linestyle='--', alpha=0.4)
```

```
plt.tight_layout()

plt.savefig(f"Memristor_Plots_{frequency}Hz2.pdf", format="pdf",␣
 ↪bbox_inches="tight")

plt.show()
```



1 square signal

[10]:
```
import numpy as np
import matplotlib.pyplot as plt
from scipy.signal import square
from scipy.integrate import solve_ivp
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from scipy import signal
from scipy.signal.waveforms import square
# ---------------------- ----------------------
frequency_test = 4
A_test = 10

W_test = 2 * np.pi * frequency_test
```

```
t_test = np.linspace(0, 1, total_points)

def v(t):
    return A_test* signal.square(W_test* t)


v_test =v(t_test)

def f_square(t, w, A, W, mu_v, D, r_on, r_off):
    k = mu_v * (r_on / D**2)
    f_w = w * (1 - w)
    r = r_on * w + r_off * (1 - w)
    I = A * square(W * t) / r
    return I * f_w * k

sol_test = solve_ivp(
    f_square, (0, 1), [w0], t_eval=t_test,
    args=(A_test, W_test, mu_v, D, r_on, r_off),
    method='RK45', max_step=0.001
)

w_test = sol_test.y[0]
r_test = np.maximum(r_on * w_test + r_off * (1 - w_test), 1e-6)
I_test = v_test / r_test

# ---------------------- ----------------------
X_test_square = np.column_stack([t_test[:-1], w_test[:-1], I_test[:-1]])
y_test_square = w_test[1:]

# -------------------- ----------------------
X_test_square_scaled = X_test_square.copy()

X_test_square_scaled[:, 0] = scaler_time.transform(X_test_square_scaled[:, 0].
 ↪reshape(-1, 1)).flatten()
X_test_square_scaled[:, 1:] = scaler_features.transform(X_test_square_scaled[:,␣
 ↪1:])
y_test_square_scaled = scaler_y.transform(y_test_square.reshape(-1, 1))

w_pred_new = pinn_model.predict(X_test_square_scaled)
w_pred_new_rescaled = scaler_y.inverse_transform(w_pred_new.reshape(-1, 1))


# ---------------------- calcute current and  R ----------------------
r_pred_new = np.maximum(r_on * w_pred_new_rescaled + r_off * (1.0 -␣
 ↪w_pred_new_rescaled), 1e-6)
r_pred_new = r_pred_new.reshape(-1)
I_pred_new = v_test[:-1] / r_pred_new
```

```
t_all_plot = t_test[:-1]
w_pred_new_rescaled = w_pred_new_rescaled.flatten()

# ----------------------plotting----------------------
fig, axes = plt.subplots(2, 2, figsize=(12, 8), dpi=300)


axes[0, 0].plot(t_test, v_test, color='dodgerblue', linewidth=2, label='Voltage␣
 ↪v(t)')
axes[0, 0].plot(t_all_plot, I_pred_new, color='tomato', linestyle='dashed',␣
 ↪linewidth=2, label='Predicted Current I(t)')
axes[0, 0].set_xlabel('Time (s)', fontsize=12)
axes[0, 0].set_ylabel('Amplitude', fontsize=12)
axes[0, 0].set_title(f'Voltage & Predicted Current\nf={frequency_test}Hz,␣
 ↪A={A_test}', fontsize=14, fontweight='bold')
axes[0, 0].legend(fontsize=10, frameon=True, edgecolor='black')
#axes[0, 0].grid(True, linestyle='--', alpha=0.4)

axes[0, 1].plot(v_test[:-1], I_pred_new, color='seagreen', linewidth=2,␣
 ↪linestyle='-', label='Predicted I-V')
axes[0, 1].set_xlabel('Voltage (V)', fontsize=12)
axes[0, 1].set_ylabel('Current (I)', fontsize=12)
axes[0, 1].set_title(f'I-V Characteristics\nf={frequency_test}Hz', fontsize=14,␣
 ↪fontweight='bold')
axes[0, 1].legend(fontsize=10, frameon=True, edgecolor='black')
#axes[0, 1].grid(True, linestyle='--', alpha=0.4)


axes[1, 0].plot(v_test[:-1], r_pred_new, color='blueviolet', linewidth=2,␣
 ↪linestyle='-', label='Predicted Memristance')
axes[1, 0].set_xlabel('Voltage (V)', fontsize=12)
axes[1, 0].set_ylabel('Memristance (Ω)', fontsize=12)
axes[1, 0].set_title(f'Memristance vs. Voltage\nf={frequency_test}Hz',␣
 ↪fontsize=14, fontweight='bold')
axes[1, 0].legend(fontsize=10, frameon=True, edgecolor='black')
#axes[1, 0].grid(True, linestyle='--', alpha=0.4)


axes[1, 1].plot(t_all_plot, w_pred_new_rescaled, color='orangered',␣
 ↪linewidth=2, linestyle='-', label='Predicted w(t) [PINN]')  #
axes[1, 1].set_xlabel('Time (s)', fontsize=12)
axes[1, 1].set_ylabel('State Variable w(t)', fontsize=12)
axes[1, 1].set_title(f'State Variable Over Time\nf={frequency}Hz', fontsize=14,␣
 ↪fontweight='bold')
axes[1, 1].legend(fontsize=10, frameon=True, edgecolor='black')
```

```
#axes[1, 1].grid(True, linestyle='--', alpha=0.4)



plt.tight_layout()
plt.savefig(f"Memristor_Plots_square{frequency_test}Hz.pdf", format="pdf",␣
  ↪bbox_inches="tight")
plt.show()
```

```
/tmp/ipykernel_14174/2321244588.py:7: DeprecationWarning: Please import `square`
from the `scipy.signal` namespace; the `scipy.signal.waveforms` namespace is
deprecated and will be removed in SciPy 2.0.0.
  from scipy.signal.waveforms import square
```

```
188/188             0s 2ms/step
```