In [1]:
```
# Load data
data = load_3story_data()
dataset = data['dataset']
states = data['damage_states']

print(f"Dataset shape: {dataset.shape}")
print(f"Shape explanation: ({dataset.shape[0]} time points, {dataset.shape[1
print(f"\nChannels:")
for i, ch in enumerate(data['channels']):
    print(f"  Channel {i}: {ch}")
print(f"\nDamage states: {np.unique(states)}")
print(f"Instances per state: {np.sum(states == 1)}")
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[1], line 2
      1 # Load data
----> 2 data = load_3story_data()
      3 dataset = data['dataset']
      4 states = data['damage_states']

NameError: name 'load_3story_data' is not defined
```

Load Data

Load the 3-story structure dataset. This dataset contains time series data from a base-excited three-story structure with various damage conditions.

In [ ]:
```
# Standard library imports
import sys
from pathlib import Path

# Add shmtools to path for development
current_dir = Path().resolve()
shmtools_path = current_dir.parent.parent.parent  # Go up to shmtools-python
if shmtools_path not in sys.path:
    sys.path.insert(0, str(shmtools_path))
    print(f"Added {shmtools_path} to Python path")

# Scientific computing imports
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.patches import Rectangle

# SHMTools imports
from shmtools.utils.data_loading import load_3story_data
from shmtools.features import ar_model_shm, arx_model_shm
from shmtools.classification import learn_mahalanobis_shm, score_mahalanobis

# Configure plotting
%matplotlib inline
```

```
plt.rcParams['figure.figsize'] = (12, 8)
plt.rcParams['font.size'] = 12
```

Setup and Imports

Damage Localization using AR and ARX Models

This notebook demonstrates damage localization techniques using autoregressive (AR) and autoregressive with exogenous inputs (ARX) model parameters from an array of sensors. The example compares the effectiveness of AR vs ARX models for identifying and localizing structural damage.

Overview

The goal is to locate the source of damage in a structure based on outlier/novelty detection. We extract AR and ARX parameters as damage-sensitive features and use the Mahalanobis distance to create damage indicators (DIs) that are invariant for normal conditions but increase for damaged conditions.

Key Concepts:

1. **AR Model**: Output-only model that captures the system's dynamic behavior
2. **ARX Model**: Input-output model that incorporates force measurements
3. **Damage Localization**: Identifying which sensors are closest to damage
4. **Mahalanobis Distance**: Statistical measure for outlier detection

Dataset:

We use the 3-story structure dataset with:

- 5 channels: 1 input force + 4 output accelerations
- 170 conditions: 90 undamaged + 80 damaged (17 states × 10 tests each)
- Damage is located near channels 4 and 5

Damage Localization using AR/ARX Models

This notebook demonstrates spatial damage analysis using autoregressive (AR) and autoregressive with exogenous inputs (ARX) model parameters across sensor arrays. It shows how to localize damage by comparing damage indicators across different channels.

Overview

The goal is to locate the source of damage in a structure using:

1. **DLAR (Damage Location using AR)**: Channel-wise analysis using AR(15) parameters
2. **DLARX (Damage Location using ARX)**: Input-output analysis using ARX(10,5,0) parameters

Key Concepts:

- Each sensor channel is analyzed independently
- Mahalanobis distance creates damage indicators for each channel
- Channels closest to damage show higher sensitivity
- ARX models incorporate input force information for better localization

References:

- Figueiredo, E., Park, G., Figueiras, J., Farrar, C., & Worden, K. (2009). Structural Health Monitoring Algorithm Comparisons using Standard Data Sets. Los Alamos National Laboratory Report: LA-14393.

Setup and Imports

In [ ]:

```
import numpy as np
import matplotlib.pyplot as plt
from pathlib import Path
import sys

# Add shmtools to path
notebook_dir = Path.cwd()
shmtools_root = notebook_dir.parent.parent.parent
if str(shmtools_root) not in sys.path:
    sys.path.insert(0, str(shmtools_root))
    print(f"Added {shmtools_root} to Python path")

# Import SHMTools functions
from shmtools.utils import (
    load_3story_data,
    compute_channel_wise_damage_indicators,
    plot_damage_indicators,
    analyze_damage_localization,
    compare_ar_arx_localization
)
from shmtools.features import ar_model_shm, arx_model_shm

# Set random seed for reproducibility
np.random.seed(42)
```

```
# Configure plotting
plt.style.use('seaborn-v0_8-darkgrid')
plt.rcParams['figure.figsize'] = (12, 8)
plt.rcParams['font.size'] = 12
```

Load Raw Data

We use the 3-story structure dataset with both force input (channel 1) and acceleration outputs (channels 2-5).

In [ ]:
```
# Load the 3-story structure dataset
try:
    data_dict = load_3story_data()
    dataset = data_dict['dataset']  # Shape: (8192, 5, 170)
    states = data_dict['damage_states']
    print(f"Dataset shape: {dataset.shape}")
    print(f"(time points, channels, instances)")
    print(f"\nChannels: Force (1), Accelerations (2-5)")
    print(f"Damage states: 1-9 (undamaged), 10-17 (damaged)")
    print(f"Instances per state: 10")
except FileNotFoundError as e:
    print(f"Error: {e}")
    print("\nPlease download the example datasets following the instructions
    print("examples/data/README.md")
    raise
```

Plot Sample Time Histories

Let's visualize the time histories from the baseline condition to understand the data structure.

In [ ]:
```
# Plot time histories from baseline condition (channels 2-5)
fig, axes = plt.subplots(2, 2, figsize=(12, 8))
axes = axes.ravel()

for i in range(4):
    channel_data = dataset[:, i+1, 0]  # Channel i+2, first instance
    axes[i].plot(channel_data, 'k', linewidth=0.5)
    axes[i].set_title(f'Channel {i+2}')
    axes[i].set_xlim(0, 8192)
    axes[i].set_ylim(-2.5, 2.5)
    axes[i].set_yticks([-2, -1, 0, 1, 2])

    if i >= 2:
        axes[i].set_xlabel('Observations')
    if i % 2 == 0:
        axes[i].set_ylabel('Acceleration (g)')

plt.tight_layout()
plt.suptitle('Baseline Condition Time Histories (Channels 2-5)', fontsize=14
plt.show()
```

DLAR: Damage Localization using AR Parameters

First, we'll analyze damage localization using AR model parameters from output channels only.

Extract AR Model Features

In [ ]:
```
# Extract data for AR analysis (channels 2-5 only)
ar_data = dataset[:, 1:5, :]  # Exclude force channel
print(f"AR data shape: {ar_data.shape}")

# Extract AR(15) parameters
ar_order = 15
print(f"\nExtracting AR({ar_order}) parameters...")

ar_parameters_fv, rms_residuals_fv, ar_parameters, ar_residuals, ar_predicti
print(f"AR parameters shape: {ar_parameters_fv.shape}")
print(f"Features per channel: {ar_order}")
print(f"Total features: {ar_parameters_fv.shape[1]} (4 channels × {ar_order}
```

Visualize AR Parameters

In [ ]:
```
# Plot AR parameters for all instances
plt.figure(figsize=(14, 8))

# Separate undamaged (1-90) and damaged (91-170) instances
undamaged_mask = states <= 9
damaged_mask = states > 9

# Plot undamaged in black, damaged in red
plt.plot(ar_parameters_fv[undamaged_mask, :].T, 'k-', alpha=0.3, linewidth=0
plt.plot(ar_parameters_fv[damaged_mask, :].T, 'r-', alpha=0.3, linewidth=0.5

# Add channel separators
for i in range(1, 4):
    plt.axvline(i * ar_order, color='k', linestyle='--', alpha=0.7)

# Add channel labels
channel_positions = [ar_order//2 + i*ar_order for i in range(4)]
channel_labels = ['Channel 2', 'Channel 3', 'Channel 4', 'Channel 5']
for pos, label in zip(channel_positions, channel_labels):
    plt.text(pos, plt.ylim()[0] + 0.1*(plt.ylim()[1] - plt.ylim()[0]),
             label, ha='center', va='bottom', fontweight='bold')

plt.xlabel('AR Parameters')
plt.ylabel('Amplitude')
plt.title(f'Concatenated AR({ar_order}) Parameters for all Instances')
plt.legend(['Undamaged', 'Damaged'], loc='upper right')
plt.grid(True, alpha=0.3)
plt.show()
```

Compute Channel-wise Damage Indicators (AR)

In [ ]:
```
# Compute damage indicators for each channel using AR parameters
print("Computing channel-wise damage indicators using AR parameters...")

undamaged_states = list(range(1, 10))  # States 1-9
n_channels = 4
features_per_channel = ar_order

ar_damage_indicators, ar_models = compute_channel_wise_damage_indicators(
    ar_parameters_fv,
    states,
    undamaged_states=undamaged_states,
    n_channels=n_channels,
    features_per_channel=features_per_channel,
    method='mahalanobis'
)

print(f"AR damage indicators shape: {ar_damage_indicators.shape}")
print(f"(states, channels) = ({ar_damage_indicators.shape[0]}, {ar_damage_ir
```

Plot AR Damage Indicators

In [ ]:
```
# Plot damage indicators for AR method
channel_names = ['Channel 2', 'Channel 3', 'Channel 4', 'Channel 5']
state_labels = np.arange(1, 18)
undamaged_indices = list(range(9))  # First 9 states

plot_damage_indicators(
    ar_damage_indicators,
    channel_names=channel_names,
    state_labels=state_labels,
    undamaged_states=undamaged_indices,
    title="AR Method: Channel-wise Damage Indicators"
)
```

DLARX: Damage Localization using ARX Parameters

Now we'll analyze damage localization using ARX model parameters that incorporate the input force information.

Extract ARX Model Features

In [ ]:
```
# Use full dataset for ARX analysis (input + outputs)
arx_data = dataset  # All 5 channels: force (input) + 4 accelerations (outpu
print(f"ARX data shape: {arx_data.shape}")

# Extract ARX(10,5,0) parameters
arx_orders = [10, 5, 0]  # [output_order, input_order, delay]
print(f"\nExtracting ARX({arx_orders[0]},{arx_orders[1]},{arx_orders[2]}) pa
```

```
arx_parameters_fv, rms_residuals_fv, arx_parameters, arx_residuals, arx_prec
print(f"ARX parameters shape: {arx_parameters_fv.shape}")
arx_features_per_channel = arx_orders[0] + arx_orders[1]  # a + b = 15
print(f"Features per channel: {arx_features_per_channel} ({arx_orders[0]} AR
print(f"Total features: {arx_parameters_fv.shape[1]} (4 output channels × {a
```

Visualize ARX Parameters

In [ ]:

```
# Plot ARX parameters for all instances
plt.figure(figsize=(14, 8))

# Plot undamaged in black, damaged in red
plt.plot(arx_parameters_fv[undamaged_mask, :].T, 'k-', alpha=0.3, linewidth=
plt.plot(arx_parameters_fv[damaged_mask, :].T, 'r-', alpha=0.3, linewidth=0.

# Add channel separators
for i in range(1, 4):
    plt.axvline(i * arx_features_per_channel, color='k', linestyle='--', alp

# Add channel labels
channel_positions = [arx_features_per_channel//2 + i*arx_features_per_channe
for pos, label in zip(channel_positions, channel_labels):
    plt.text(pos, plt.ylim()[0] + 0.1*(plt.ylim()[1] - plt.ylim()[0]),
             label, ha='center', va='bottom', fontweight='bold')

plt.xlabel('ARX Parameters')
plt.ylabel('Amplitude')
plt.title(f'Concatenated ARX({arx_orders[0]},{arx_orders[1]}) Parameters fro
plt.legend(['Undamaged', 'Damaged'], loc='upper right')
plt.grid(True, alpha=0.3)
plt.show()
```

Compute Channel-wise Damage Indicators (ARX)

In [ ]:

```
# Compute damage indicators for each channel using ARX parameters
print("Computing channel-wise damage indicators using ARX parameters...")

arx_damage_indicators, arx_models = compute_channel_wise_damage_indicators(
    arx_parameters_fv,
    states,
    undamaged_states=undamaged_states,
    n_channels=n_channels,
    features_per_channel=arx_features_per_channel,
    method='mahalanobis'
)

print(f"ARX damage indicators shape: {arx_damage_indicators.shape}")
print(f"(states, channels) = ({arx_damage_indicators.shape[0]}, {arx_damage_
```

Plot ARX Damage Indicators

In [ ]:
```
# Plot damage indicators for ARX method
plot_damage_indicators(
    arx_damage_indicators,
    channel_names=channel_names,
    state_labels=state_labels,
    undamaged_states=undamaged_indices,
    title="ARX Method: Channel-wise Damage Indicators"
)
```

Damage Localization Analysis

Let's analyze and compare the damage localization results from both methods.

AR Method Analysis

In [ ]:
```
# Analyze AR method results
ar_analysis = analyze_damage_localization(
    ar_damage_indicators,
    channel_names=channel_names,
    undamaged_states=undamaged_indices
)

print("=" * 60)
print("AR METHOD DAMAGE LOCALIZATION ANALYSIS")
print("=" * 60)
print(ar_analysis['interpretation'])

# Show channel sensitivity ranking
print(f"\nChannel sensitivity values:")
for i, (channel, sensitivity) in enumerate(zip(channel_names, ar_analysis['c
    rank = np.where(ar_analysis['damage_ranking'] == i)[0][0] + 1
    print(f"  {channel}: {sensitivity:.3f} (rank {rank})")
```

ARX Method Analysis

In [ ]:
```
# Analyze ARX method results
arx_analysis = analyze_damage_localization(
    arx_damage_indicators,
    channel_names=channel_names,
    undamaged_states=undamaged_indices
)

print("=" * 60)
print("ARX METHOD DAMAGE LOCALIZATION ANALYSIS")
print("=" * 60)
print(arx_analysis['interpretation'])

# Show channel sensitivity ranking
print(f"\nChannel sensitivity values:")
for i, (channel, sensitivity) in enumerate(zip(channel_names, arx_analysis['
```

```python
    rank = np.where(arx_analysis['damage_ranking'] == i)[0][0] + 1
    print(f"  {channel}: {sensitivity:.3f} (rank {rank})")
```

AR vs ARX Comparison

In [ ]:
```python
# Compare AR and ARX methods
comparison = compare_ar_arx_localization(
    ar_damage_indicators,
    arx_damage_indicators,
    channel_names=channel_names,
    undamaged_states=undamaged_indices
)

print("=" * 60)
print("AR vs ARX COMPARISON")
print("=" * 60)
print(comparison['summary'])
```

Side-by-Side Comparison Plot

In [ ]:
```python
# Create side-by-side comparison plot
fig, axes = plt.subplots(2, 4, figsize=(16, 10))

# AR method plots (top row)
for i in range(4):
    ax = axes[0, i]
    colors = ['k' if j < 9 else 'r' for j in range(17)]
    bars = ax.bar(range(17), ar_damage_indicators[:, i], color=colors)
    ax.set_title(f'{channel_names[i]} (AR)', fontsize=12)
    ax.set_xlim(-0.5, 16.5)
    ax.set_xticks(range(17))
    ax.set_xticklabels(range(1, 18))
    ax.grid(True, alpha=0.3)

    if i == 0:
        ax.set_ylabel('AR Damage Indicator')

# ARX method plots (bottom row)
for i in range(4):
    ax = axes[1, i]
    colors = ['k' if j < 9 else 'r' for j in range(17)]
    bars = ax.bar(range(17), arx_damage_indicators[:, i], color=colors)
    ax.set_title(f'{channel_names[i]} (ARX)', fontsize=12)
    ax.set_xlim(-0.5, 16.5)
    ax.set_xticks(range(17))
    ax.set_xticklabels(range(1, 18))
    ax.set_xlabel('State Condition')
    ax.grid(True, alpha=0.3)

    if i == 0:
        ax.set_ylabel('ARX Damage Indicator')

plt.tight_layout()
```

```
plt.suptitle('AR vs ARX Damage Localization Comparison', fontsize=16, y=1.02
plt.show()
```

Sensitivity Improvement Analysis

In [ ]:

```
# Plot sensitivity comparison
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(15, 6))

# Sensitivity values comparison
x_pos = np.arange(len(channel_names))
width = 0.35

bars1 = ax1.bar(x_pos - width/2, ar_analysis['channel_sensitivity'], width,
               label='AR Method', color='lightblue', alpha=0.8)
bars2 = ax1.bar(x_pos + width/2, arx_analysis['channel_sensitivity'], width,
               label='ARX Method', color='lightcoral', alpha=0.8)

ax1.set_xlabel('Channel')
ax1.set_ylabel('Mean Damage Indicator')
ax1.set_title('Channel Sensitivity Comparison')
ax1.set_xticks(x_pos)
ax1.set_xticklabels(channel_names)
ax1.legend()
ax1.grid(True, alpha=0.3)

# Add value labels on bars
for bar1, bar2 in zip(bars1, bars2):
    height1 = bar1.get_height()
    height2 = bar2.get_height()
    ax1.text(bar1.get_x() + bar1.get_width()/2., height1 + 0.01,
             f'{height1:.2f}', ha='center', va='bottom', fontsize=10)
    ax1.text(bar2.get_x() + bar2.get_width()/2., height2 + 0.01,
             f'{height2:.2f}', ha='center', va='bottom', fontsize=10)

# Improvement ratio
improvement_ratio = comparison['sensitivity_ratio']
bars3 = ax2.bar(channel_names, improvement_ratio,
               color=['green' if r > 1 else 'orange' for r in improvement_ra
               alpha=0.7)

ax2.axhline(y=1, color='red', linestyle='--', alpha=0.7, label='No improveme
ax2.set_xlabel('Channel')
ax2.set_ylabel('ARX/AR Sensitivity Ratio')
ax2.set_title('ARX Improvement over AR')
ax2.legend()
ax2.grid(True, alpha=0.3)

# Add value labels
for bar, ratio in zip(bars3, improvement_ratio):
    height = bar.get_height()
    ax2.text(bar.get_x() + bar.get_width()/2., height + 0.02,
             f'{ratio:.2f}×', ha='center', va='bottom', fontsize=10)
```

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

Summary and Conclusions

This analysis demonstrates spatial damage localization using both AR and ARX model parameters:

Key Findings

1. **Damage Localization**: Both methods successfully identify that Channels 4 and 5 are more sensitive to damage than Channels 2 and 3, indicating damage is located closer to the upper floors of the 3-story structure.

2. **AR Method Results**:

    - Uses only output measurements (accelerations)
    - Provides good discrimination between undamaged and damaged states
    - Simple to implement and interpret
3. **ARX Method Results**:

    - Incorporates input force information
    - Can provide improved damage localization in some cases
    - Captures input-output relationships for better physics-based analysis

Method Comparison

AR Model Advantages:

- Simpler implementation (output-only)
- No need for input measurement
- Robust to input measurement noise
- Faster computation

ARX Model Advantages:

- Better physics representation (input-output relationships)
- Potential for improved damage sensitivity
- Input normalization can reduce environmental effects
- Better for systems with known excitation

Practical Implications

- **Channel-wise analysis** enables spatial damage localization across sensor arrays
- **Mahalanobis distance** provides effective outlier detection for each channel
- **Multiple model comparison** increases confidence in damage localization results

- **Structural knowledge** helps interpret which channels correspond to different structural locations

Recommendations

1. **For new applications**: Start with AR analysis for simplicity, then consider ARX if input measurements are available
2. **For critical structures**: Use both methods as complementary approaches
3. **For environmental robustness**: Consider ARX models when input forces can be measured
4. **For sensor network design**: Use these results to optimize sensor placement for damage localization

Part 1: Damage Localization using AR Parameters

First, we'll use the traditional AR model approach, which only uses the output acceleration measurements (channels 2-5).

In [ ]:
```
# Extract AR(15) parameters from channels 2-5
ar_order = 15
output_data = dataset[:, 1:5, :]  # Channels 2-5 only

print("Estimating AR parameters...")
ar_params_fv, ar_rms_fv, ar_params, _, _ = ar_model_shm(output_data, ar_orde

print(f"\nAR parameters shape: {ar_params_fv.shape}")
print(f"Shape explanation: ({ar_params_fv.shape[0]} instances, {ar_params_fv
print(f"Features per channel: {ar_order}")
print(f"Total features: {4} channels × {ar_order} parameters = {ar_params_fv
```

Part 2: Damage Localization using ARX Parameters

Now we'll use the ARX model, which incorporates the input force measurement (channel 1) along with the output accelerations.

In [ ]:
```
# Extract ARX(10,5,0) parameters
arx_orders = [10, 5, 0]  # a=10 (output order), b=5 (input order), tau=0 (no

print("Estimating ARX parameters...")
arx_params_fv, arx_rms_fv, arx_params, _, _, _ = arx_model_shm(dataset, arx_

print(f"\nARX parameters shape: {arx_params_fv.shape}")
print(f"Shape explanation: ({arx_params_fv.shape[0]} instances, {arx_params_
print(f"Features per channel: {arx_orders[0] + arx_orders[1]} = {15}")
print(f"Total features: {4} channels × {15} parameters = {arx_params_fv.shap
```

Conclusions

This notebook demonstrated damage localization using AR and ARX model parameters:

Key Findings:

1. **ARX Model Advantage**: ARX models provide better damage discrimination by incorporating input-output relationships
2. **Damage Location**: Both models correctly identify damage near channels 4 and 5
3. **Practical Application**: ARX models are preferred when force measurements are available

References

- Figueiredo, E., Park, G., Figueiras, J., Farrar, C., & Worden, K. (2009). Structural Health Monitoring Algorithm Comparisons using Standard Data Sets. Los Alamos National Laboratory Report: LA-14393.