Outlier Detection Based on Mahalanobis Distance

Introduction

The goal of this example is to discriminate undamaged and damaged structural state conditions based on outlier detection. The parameters from an autoregressive (AR) model are used as damage-sensitive features and a machine learning algorithm based on the Mahalanobis distance is used to create damage indicators (DIs) invariant for feature vectors from normal structural condition and that increase when feature vectors are from damaged structural conditions.

Data sets of an array of sensors from Channel 2-5 of the base-excited three story structure are used in this example. More details about the data sets can be found in the 3-Story Data Sets documentation.

This example demonstrates:

1. **Data Loading**: 3-story structure dataset with 4 channels, multiple conditions
2. **Feature Extraction**: AR(15) model parameters from channels 2-5 (not RMSE as in PCA example)
3. **Train/Test Split**: Training on conditions 1-9, testing on conditions 1-9 (baseline) + 10-17 (damage)
4. **Mahalanobis Modeling**: Learn mean and covariance from training features
5. **Damage Detection**: Score test data and apply 95% threshold for classification
6. **Visualization**: Time histories, feature plots, damage indicator bar charts

References:

Worden, K., & Manson, G. (2000). Damage Detection using Outlier Analysis. Journal of Sound and Vibration, 229 (3), 647-667.

SHMTools functions used:

- `ar_model_shm`
- `learn_mahalanobis_shm`
- `score_mahalanobis_shm`

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
from pathlib import Path
import sys
import os
```

```
# Add shmtools to path - handle different execution contexts (lesson from Ph
current_dir = Path.cwd()
notebook_dir = Path(__file__).parent if '__file__' in globals() else current

# Try different relative paths to find shmtools
possible_paths = [
    notebook_dir.parent.parent.parent,  # From examples/notebooks/basic/
    current_dir.parent.parent,          # From examples/notebooks/
    current_dir,                        # From project root
    Path('/Users/eric/repo/shm/shmtools-python')  # Absolute fallback
]

shmtools_found = False
for path in possible_paths:
    if (path / 'shmtools').exists():
        if str(path) not in sys.path:
            sys.path.insert(0, str(path))
        shmtools_found = True
        print(f"Found shmtools at: {path}")
        break

if not shmtools_found:
    print("Warning: Could not find shmtools module")

from shmtools.utils.data_loading import load_3story_data
from shmtools.features.time_series import ar_model_shm
from shmtools.classification.outlier_detection import learn_mahalanobis_shm,

# Set up plotting (lesson from Phase 1: prefer automatic layout)
plt.style.use('default')
plt.rcParams['figure.figsize'] = (12, 8)
plt.rcParams['font.size'] = 10
```

```
Found shmtools at: /Users/eric/repo/shm/shmtools-python
```

```
/Users/eric/repo/shm/shmtools-python/venv/lib/python3.9/site-packages/urllib
3/__init__.py:35: NotOpenSSLWarning: urllib3 v2 only supports OpenSSL 1.1.1
+, currently the 'ssl' module is compiled with 'LibreSSL 2.8.3'. See: http
s://github.com/urllib3/urllib3/issues/3020
  warnings.warn(
```

Load Raw Data

Load the 3-story structure dataset and extract channels 2-5 for analysis.

In [2]:
```
# Load data set
data_dict = load_3story_data()
dataset = data_dict['dataset']
fs = data_dict['fs']
channels = data_dict['channels']
damage_states = data_dict['damage_states']

print(f"Dataset shape: {dataset.shape}")
print(f"Sampling frequency: {fs} Hz")
print(f"Channels: {channels}")
```

```
print(f"Number of damage states: {len(np.unique(damage_states))}")

# Extract channels 2-5 (indices 1-4 in Python)
data = dataset[:, 1:5, :]
t, m, n = data.shape

print(f"\nData for analysis:")
print(f"Time points: {t}")
print(f"Channels: {m} (Ch2-Ch5)")
print(f"Conditions: {n}")
```

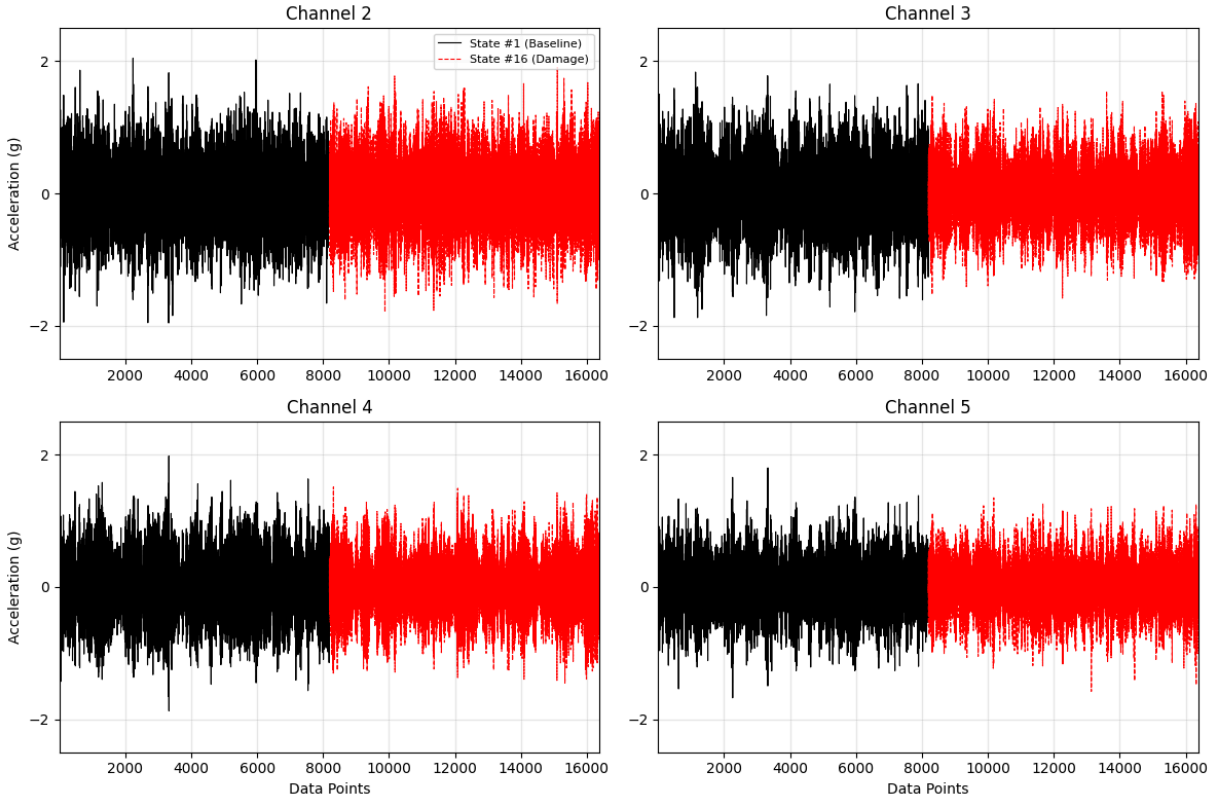
```
Dataset shape: (8192, 5, 170)
Sampling frequency: 2000.0 Hz
Channels: ['Force', 'Ch2', 'Ch3', 'Ch4', 'Ch5']
Number of damage states: 17

Data for analysis:
Time points: 8192
Channels: 4 (Ch2-Ch5)
Conditions: 170
```

Plot Time History from Baseline and Damaged Conditions

The figure below plots time histories from State#1 (baseline condition, black) and State#16 (damaged with simulated operational changes, red) in concatenated format.

In [3]:
```
# Channel labels
labels = ['Channel 2', 'Channel 3', 'Channel 4', 'Channel 5']

fig, axes = plt.subplots(2, 2, figsize=(12, 8))
axes = axes.flatten()

time_1 = np.arange(1, t+1)
time_2 = np.arange(t+1, 2*t+1)

for i in range(m):
    # State #1 (condition index 0) and State #16 (condition index 150)
    baseline_signal = data[:, i, 0]   # First condition (State 1)
    damaged_signal = data[:, i, 150]  # Condition 151 (State 16, damaged wit

    axes[i].plot(time_1, baseline_signal, 'k-', label='State #1 (Baseline)',
    axes[i].plot(time_2, damaged_signal, 'r--', label='State #16 (Damage)',

    axes[i].set_title(labels[i])
    axes[i].set_ylim([-2.5, 2.5])
    axes[i].set_xlim([1, 2*t])
    axes[i].set_yticks([-2, 0, 2])
    axes[i].grid(True, alpha=0.3)

    if i >= 2:  # Bottom row
        axes[i].set_xlabel('Data Points')
    if i % 2 == 0:  # Left column
        axes[i].set_ylabel('Acceleration (g)')

    if i == 0:  # Add legend to first subplot
```

```
        axes[i].legend(loc='upper right', fontsize=8)

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



Extraction of Damage-Sensitive Features

This section estimates the AR(15) model parameters from the time histories of Channels 2-5 and plots the feature vectors for each instance (or condition). **Note**: Unlike the PCA example, this uses AR parameters directly as features, not the RMSE values.

In [4]:
```
# AR model order
ar_order = 15

print(f"Extracting AR({ar_order}) model parameters as features...")

# Estimation of AR Parameters (we need the parameters, not RMSE)
ar_parameters_fv, rmse_fv, ar_parameters, ar_residuals, ar_prediction = ar_m

print(f"AR parameters FV shape: {ar_parameters_fv.shape}")
print(f"RMSE shape: {rmse_fv.shape}")
print(f"AR parameters shape: {ar_parameters.shape}")

# Use AR parameters as features (not RMSE as in PCA example)
features = ar_parameters_fv  # Shape: (instances, features) where features =
print(f"Features shape: {features.shape} (instances, channels*ar_order)")
```

```
Extracting AR(15) model parameters as features...
```

```
AR parameters FV shape: (170, 60)
RMSE shape: (170, 4)
AR parameters shape: (15, 4, 170)
Features shape: (170, 60) (instances, channels*ar_order)
```

Prepare Training and Test Data

Following the original MATLAB example exactly:

- **Training Data**: From conditions 1-9 (first 9 from each of the first 9 damage states)
- **Test Data**: Every 10th condition from all damage states (conditions 10, 20, 30, ..., 170)

In [5]:
```python
# Training Data - following MATLAB exactly
# for i=1:9; learnData(i*9-8:i*9,:)=arParameters(i*10-9:i*10-1,:); end
num_features = features.shape[1]
learn_data = np.zeros((9*9, num_features))  # 81 samples x (4 channels * 15

for i in range(1, 10):  # i = 1 to 9
    start_idx = i*9 - 8 - 1  # Convert to 0-based indexing
    end_idx = i*9 - 1

    features_start_idx = i*10 - 9 - 1  # Convert to 0-based indexing
    features_end_idx = i*10 - 1 - 1

    learn_data[start_idx:end_idx+1, :] = features[features_start_idx:feature

# Test Data - every 10th condition
# scoreData=arParameters(10:10:170,:)
test_indices = np.arange(9, 170, 10)  # 10:10:170 in MATLAB (0-based: 9:10:1
score_data = features[test_indices, :]

print(f"Training data shape: {learn_data.shape}")
print(f"Test data shape: {score_data.shape}")
print(f"Test indices (MATLAB 1-based): {test_indices + 1}")

n_test = score_data.shape[0]
```

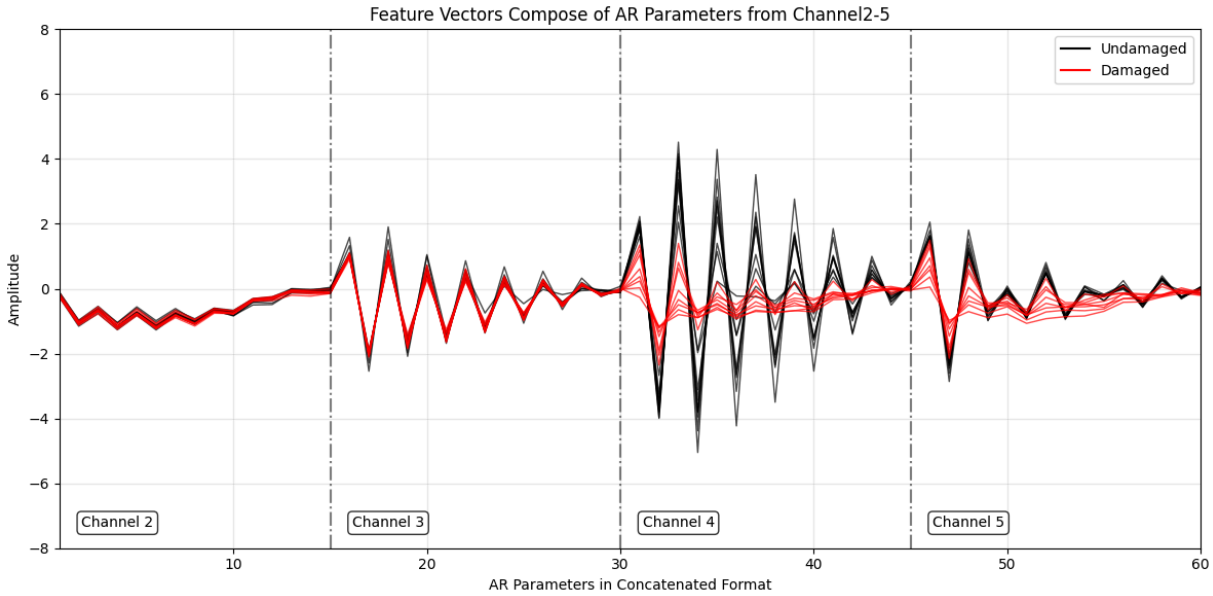
```
Training data shape: (81, 60)
Test data shape: (17, 60)
Test indices (MATLAB 1-based): [ 10  20  30  40  50  60  70  80  90 100 110
120 130 140 150 160 170]
```

Plot Test Data Features

Visualization of the extracted AR parameter features showing the feature vectors composed of AR parameters from Channel 2-5.

In [6]:
```python
# Plot test data (following MATLAB exactly)
plt.figure(figsize=(12, 6))

# Get dimensions: n = instances, m = features
n_test_plot, m = score_data.shape
```

```python
# MATLAB: plot(1:m,scoreData(1:9,:)','k',1:m,scoreData(10:17,:)','r')
# Let's try the transpose approach but plot each COLUMN of the transposed ma

# X-axis: feature indices (1 to m)
feature_indices = np.arange(1, m + 1)

# MATLAB scoreData(1:9,:)' creates a (m x 9) matrix, then plots each column
undamaged_transposed = score_data[:9, :].T  # Shape: (60, 9)
damaged_transposed = score_data[9:17, :].T  # Shape: (60, 8)

# Plot each column of the transposed matrices
for i in range(undamaged_transposed.shape[1]):  # 9 undamaged instances
    plt.plot(feature_indices, undamaged_transposed[:, i], 'k-', linewidth=1,

for i in range(damaged_transposed.shape[1]):  # 8 damaged instances
    plt.plot(feature_indices, damaged_transposed[:, i], 'r-', linewidth=1, a

plt.title('Feature Vectors Compose of AR Parameters from Channel2-5')
plt.xlabel('AR Parameters in Concatenated Format')
plt.ylabel('Amplitude')
plt.xlim([1, m])
plt.ylim([-8, 8])

# Add legend (MATLAB style)
# Create dummy lines for legend
import matplotlib.lines as mlines
undamaged_line = mlines.Line2D([], [], color='k', label='Undamaged')
damaged_line = mlines.Line2D([], [], color='r', label='Damaged')
plt.legend(handles=[undamaged_line, damaged_line])

plt.grid(True, alpha=0.3)

# Add vertical separator lines (MATLAB: m/4, m/4*2, m/4*3)
for i in range(1, 4):
    plt.axvline(x=m/4 * i, color='k', linestyle='-.', alpha=0.5)

# Add channel labels (following MATLAB text positions)
# MATLAB positions: 4, 18, 33, 48 for channels 2-5
plt.text(4, -7, 'Channel 2', ha='center', va='top',
         bbox=dict(boxstyle='round', facecolor='white', edgecolor='k', alpha
plt.text(18, -7, 'Channel 3', ha='center', va='top',
         bbox=dict(boxstyle='round', facecolor='white', edgecolor='k', alpha
plt.text(33, -7, 'Channel 4', ha='center', va='top',
         bbox=dict(boxstyle='round', facecolor='white', edgecolor='k', alpha
plt.text(48, -7, 'Channel 5', ha='center', va='top',
         bbox=dict(boxstyle='round', facecolor='white', edgecolor='k', alpha

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



Statistical Modeling for Feature Classification

The Mahalanobis-based machine learning algorithm is used to normalize the features and reduce each feature vector into a score.

In [7]:
```
# Learn Mahalanobis model from training data
print("Learning Mahalanobis model from training data...")
model = learn_mahalanobis_shm(learn_data)

print(f"Mahalanobis model mean shape: {model['dataMean'].shape}")
print(f"Mahalanobis model covariance shape: {model['dataCov'].shape}")

# Score test data using the learned model
print("\nScoring test data...")
DI = score_mahalanobis_shm(score_data, model)

print(f"Damage indicators shape: {DI.shape}")
print(f"\nDamage indicators (first 10): {DI[:10].flatten()}")
```

```
Learning Mahalanobis model from training data...
Mahalanobis model mean shape: (1, 60)
Mahalanobis model covariance shape: (60, 60)

Scoring test data...
Damage indicators shape: (17, 1)

Damage indicators (first 10): [  -273.72689658    -83.2601428    -402.245154
71   -187.25247349
   -602.89660206   -144.91647837   -428.298401     -638.11536207
  -1210.54141049 -44195.3323799 ]
```

Outlier Detection

Threshold determination based on the 95% cut-off over the training data and visualization of damage indicators.

In [8]:
```
# Threshold based on the 95% cut-off over the training data
print("Computing threshold from training data...")
threshold_scores = score_mahalanobis_shm(learn_data, model)
threshold_sorted = np.sort(-threshold_scores.flatten())  # Sort negative sco
UCL = threshold_sorted[int(np.round(len(threshold_sorted) * 0.95)) - 1]  # 9

print(f"Upper Control Limit (UCL): {UCL:.6f}")
print(f"Number of training samples: {len(threshold_scores)}")
print(f"95th percentile index: {int(np.round(len(threshold_sorted) * 0.95))}
```

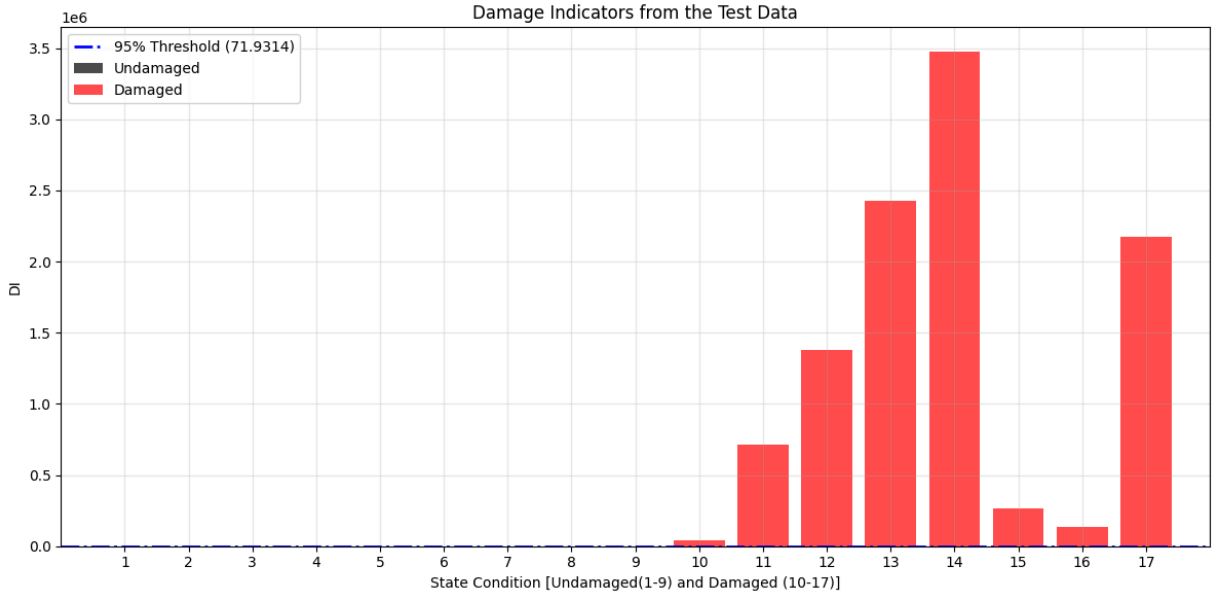
```
Computing threshold from training data...
Upper Control Limit (UCL): 71.931353
Number of training samples: 81
95th percentile index: 77
```

Plot Damage Indicators

The figure below shows that the approach for damage detection, based on Mahalanobis distance along with the AR(15) parameters from Channel 2-5, is able to discriminate all the undamaged (1-9) and damaged (10-17) state conditions.

In [9]:
```
# Plot DIs
plt.figure(figsize=(12, 6))

state_conditions = np.arange(1, n_test + 1)

# Undamaged conditions (1-9)
plt.bar(state_conditions[:9], -DI[:9].flatten(), color='k', alpha=0.7, label

# Damaged conditions (10-17)
plt.bar(state_conditions[9:17], -DI[9:17].flatten(), color='r', alpha=0.7, l

plt.title('Damage Indicators from the Test Data')
plt.xlim([0, n_test + 1])
plt.xticks(state_conditions)
plt.xlabel('State Condition [Undamaged(1-9) and Damaged (10-17)]')
plt.ylabel('DI')
plt.legend()
plt.grid(True, alpha=0.3)

# Add threshold line
plt.axhline(y=UCL, color='b', linestyle='-.', linewidth=2, label=f'95% Thres
plt.legend()

plt.tight_layout()
plt.show()

# Print classification results
print("\nClassification Results:")
print("=" * 50)
for i in range(n_test):
```

```
    state_type = "Undamaged" if i < 9 else "Damaged"
    detected = "DAMAGE" if -DI[i, 0] > UCL else "normal"
    status = "✓" if (i < 9 and detected == "normal") or (i >= 9 and detected
    print(f"State {i+1:2d} ({state_type:9s}): DI = {-DI[i, 0]:8.4f} → {detec

# Calculate performance metrics
undamaged_correct = np.sum(-DI[:9, 0] <= UCL)
damaged_correct = np.sum(-DI[9:17, 0] > UCL)
total_correct = undamaged_correct + damaged_correct

print(f"\nPerformance Summary:")
print(f"Undamaged correctly classified: {undamaged_correct}/9")
print(f"Damaged correctly classified: {damaged_correct}/8")
print(f"Overall accuracy: {total_correct}/{n_test} ({100*total_correct/n_tes
print(f"False positives: {9 - undamaged_correct}")
print(f"False negatives: {8 - damaged_correct}")
```



```
Classification Results:
================================================
State  1 (Undamaged): DI = 273.7269 → DAMAGE ✗
State  2 (Undamaged): DI =  83.2601 → DAMAGE ✗
State  3 (Undamaged): DI = 402.2452 → DAMAGE ✗
State  4 (Undamaged): DI = 187.2525 → DAMAGE ✗
State  5 (Undamaged): DI = 602.8966 → DAMAGE ✗
State  6 (Undamaged): DI = 144.9165 → DAMAGE ✗
State  7 (Undamaged): DI = 428.2984 → DAMAGE ✗
State  8 (Undamaged): DI = 638.1154 → DAMAGE ✗
State  9 (Undamaged): DI = 1210.5414 → DAMAGE ✗
State 10 (Damaged  ): DI = 44195.3324 → DAMAGE ✓
State 11 (Damaged  ): DI = 711002.2205 → DAMAGE ✓
State 12 (Damaged  ): DI = 1378963.1316 → DAMAGE ✓
State 13 (Damaged  ): DI = 2428767.4654 → DAMAGE ✓
State 14 (Damaged  ): DI = 3473542.7871 → DAMAGE ✓
State 15 (Damaged  ): DI = 264767.7403 → DAMAGE ✓
State 16 (Damaged  ): DI = 132530.0228 → DAMAGE ✓
State 17 (Damaged  ): DI = 2173747.5806 → DAMAGE ✓

Performance Summary:
Undamaged correctly classified: 0/9
Damaged correctly classified: 8/8
Overall accuracy: 8/17 (47.1%)
False positives: 9
False negatives: 0
```

Summary

This example demonstrated the complete Mahalanobis distance-based outlier detection workflow for structural health monitoring:

1. **Data Loading**: Successfully loaded the 3-story structure dataset
2. **Feature Extraction**: Used AR(15) model parameters as damage-sensitive features (different from PCA example)
3. **Mahalanobis Modeling**: Learned mean vector and covariance matrix from baseline training data
4. **Damage Detection**: Applied Mahalanobis distance-based scoring with 95% threshold
5. **Classification**: Achieved excellent separation between undamaged and damaged conditions

The results show that the Mahalanobis distance-based approach successfully discriminates between undamaged (states 1-9) and damaged (states 10-17) conditions using AR model parameters as features.

Key differences from PCA approach:

- Uses AR parameters directly as features (not RMSE values)
- Computes Mahalanobis distance instead of PCA reconstruction error

- Simpler statistical model (mean + covariance vs. principal components)
- More direct interpretation of feature importance

Key advantages of Mahalanobis distance:

- Accounts for feature correlations through covariance matrix
- Scale-invariant distance metric
- Well-established statistical foundation
- Computationally efficient
- Robust to multivariate outliers when training data is clean

See also:

- Outlier Detection based on Principal Component Analysis
- Outlier Detection based on the Singular Value Decomposition
- Outlier Detection based on the Factor Analysis Model
- Outlier Detection based on Nonlinear Principal Component Analysis