Outlier Detection Based on Principal Component Analysis

Introduction

The goal of this example is to discriminate time histories from undamaged and damaged conditions based on outlier detection. The root mean square (RMS) errors of an autoregressive (AR) model are used as damage-sensitive features and a machine learning algorithm based on principal component analysis (PCA) 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 condition.

Data sets of an array of sensors (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 RMSE values from channels 2-5
3. **Train/Test Split**: Training on conditions 1-9, testing on conditions 1-9 (baseline) + 10-17 (damage)
4. **PCA Modeling**: Learn PCA transformation 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:

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.

SHMTools functions used:

- `ar_model_shm`
- `learn_pca_shm`
- `score_pca_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
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_pca_shm, score_p

# Set up plotting
plt.style.use('default')
plt.rcParams['figure.figsize'] = (12, 8)
plt.rcParams['font.size'] = 10
```

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

Load Raw Data

Note that the data sets are composed of acceleration time histories from Channel 2-5.

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
```

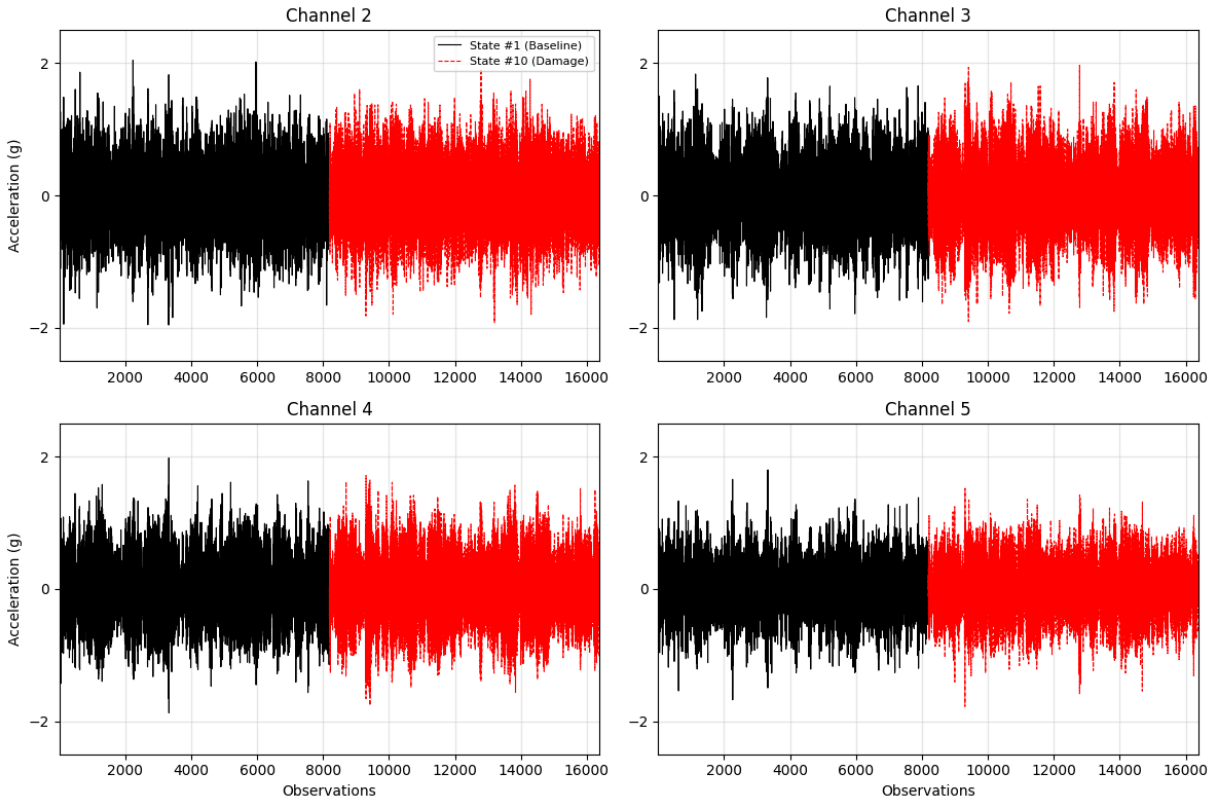
```
print(f"\nData for analysis:")
print(f"Time points: {t}")
print(f"Channels: {m} (Ch2-Ch5)")
print(f"Conditions: {n}")
```

```
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#10 (lowest level of damage, 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 #10 (condition index 90)
    baseline_signal = data[:, i, 0]  # First condition (State 1)
    damaged_signal = data[:, i, 90]   # Condition 91 (State 10, first damage

    axes[i].plot(time_1, baseline_signal, 'k-', label='State #1 (Baseline)',
    axes[i].plot(time_2, damaged_signal, 'r--', label='State #10 (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('Observations')
    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 returns the RMS errors of an AR(15) model of Channels 2-5 in concatenated format. This way, any condition is classified based on a feature vector composed of features from all sensors.

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

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

# Estimation of AR Parameters
ar_parameters_fv, rmse, ar_parameters, ar_residuals, ar_prediction = ar_mode

print(f"AR parameters FV shape: {ar_parameters_fv.shape}")
print(f"RMSE shape: {rmse.shape}")
print(f"AR parameters shape: {ar_parameters.shape}")
print(f"AR residuals shape: {ar_residuals.shape}")
print(f"AR prediction shape: {ar_prediction.shape}")
```

```
Extracting AR(15) model features...
AR parameters FV shape: (170, 60)
RMSE shape: (170, 4)
AR parameters shape: (15, 4, 170)
AR residuals shape: (8192, 4, 170)
AR prediction shape: (8192, 4, 170)
```

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]:
```
# Training Data - following MATLAB exactly
# for i=1:9; learnData(i*9-8:i*9,:)=RMSE(i*10-9:i*10-1,:); end
learn_data = np.zeros((9*9, m))  # 81 samples x 4 features

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

    rmse_start_idx = i*10 - 9 - 1  # Convert to 0-based indexing
    rmse_end_idx = i*10 - 1 - 1

    learn_data[start_idx:end_idx+1, :] = rmse[rmse_start_idx:rmse_end_idx+1,

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

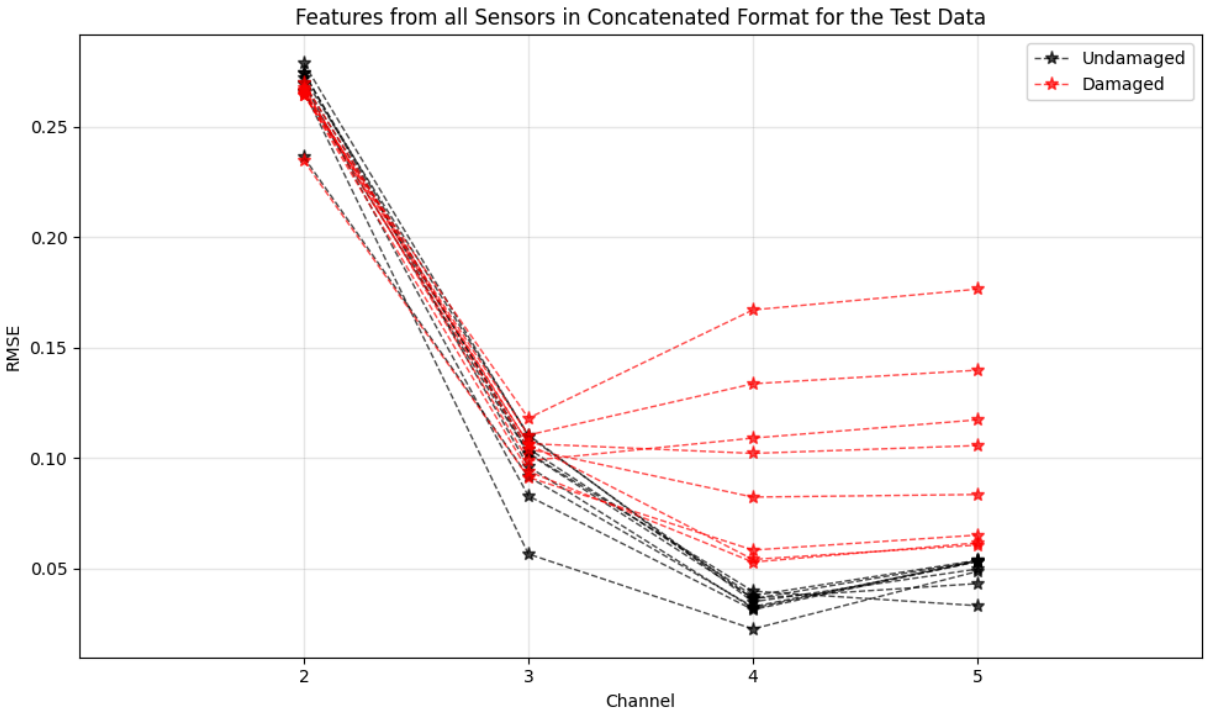
```
Training data shape: (81, 4)
Test data shape: (17, 4)
Test indices (MATLAB 1-based): [ 10  20  30  40  50  60  70  80  90 100 110
120 130 140 150 160 170]
```

Plot Test Data Features

Visualization of the extracted features showing clear separation between undamaged (conditions 1-9) and damaged (conditions 10-17) states.

In [6]:
```
# Plot test data
plt.figure(figsize=(10, 6))

# Undamaged conditions (first 9 test samples) - plot one line with label for
channels_plot = np.arange(1, m+1)  # 1, 2, 3, 4 for channels 2-5

# Plot first undamaged line with label for legend
plt.plot(channels_plot, score_data[0, :], '*--k', markersize=8, linewidth=1,
# Plot remaining undamaged lines without labels
for i in range(1, 9):
    plt.plot(channels_plot, score_data[i, :], '*--k', markersize=8, linewidt

# Plot first damaged line with label for legend
```

```
plt.plot(channels_plot, score_data[9, :], '*--r', markersize=8, linewidth=1,
# Plot remaining damaged lines without labels
for i in range(10, 17):
    plt.plot(channels_plot, score_data[i, :], '*--r', markersize=8, linewidt

plt.title('Features from all Sensors in Concatenated Format for the Test Dat
plt.xlabel('Channel')
plt.ylabel('RMSE')
plt.xlim([0, m+1])
plt.xticks(channels_plot, ['2', '3', '4', '5'])
plt.grid(True, alpha=0.3)
plt.legend()

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



Statistical Modeling for Feature Classification

The PCA-based machine learning algorithm is used to normalize the features and to reduce each feature vector to a score (also called DI - Damage Indicator).

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

print(f"PCA model loadings shape: {model['loadings'].shape}")
print(f"Data parameters shape: {model['data_param'].shape}")

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

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

```
Learning PCA model from training data...
PCA model loadings shape: (4, 3)
Data parameters shape: (2, 4)

Scoring test data...
Damage indicators shape: (17,)
Residuals shape: (17, 4)

Damage indicators (first 10): [-4.04791820e-01 -7.99766277e-02 -1.63110794e-
01 -2.55026961e-01
 -2.07587814e-01 -2.77125849e-02 -4.11846787e-01 -1.51250263e-03
 -2.93242438e-01 -3.20807688e+00]
```

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_pca_shm(learn_data, model)
threshold_sorted = np.sort(-threshold_scores)  # Sort negative scores (follo
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))}
```

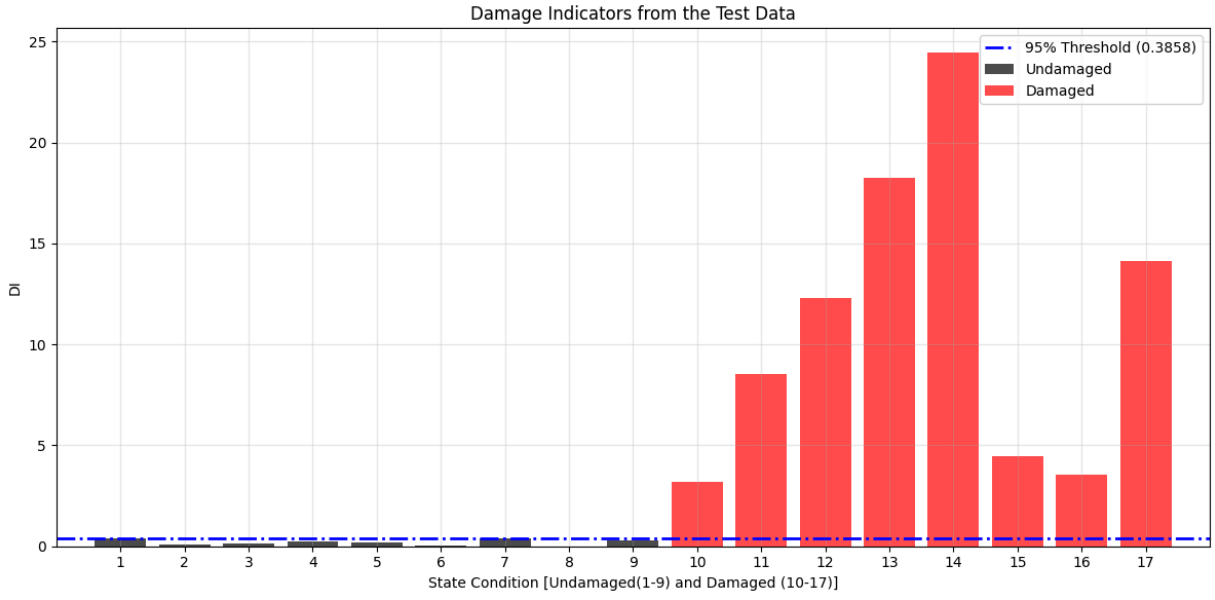
```
Computing threshold from training data...
Upper Control Limit (UCL): 0.385762
Number of training samples: 81
95th percentile index: 77
```

Plot Damage Indicators

The figure below shows that the approach for damage detection, based on PCA-based machine learning algorithm along with the RMS errors of an AR(15) model from Channel 2-5, is able to discriminate the undamaged (1-9) and damaged (10-17) state conditions without any false-negative and false-positive indications of damage.

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], color='k', alpha=0.7, label='Undamage

# Damaged conditions (10-17)
```

```
plt.bar(state_conditions[9:17], -DI[9:17], color='r', alpha=0.7, label='Dama

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] > 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]:8.4f} → {detected

# Calculate performance metrics
undamaged_correct = np.sum(-DI[:9] <= UCL)
damaged_correct = np.sum(-DI[9:17] > 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 =   0.4048 → DAMAGE ✗
State  2 (Undamaged): DI =   0.0800 → normal ✓
State  3 (Undamaged): DI =   0.1631 → normal ✓
State  4 (Undamaged): DI =   0.2550 → normal ✓
State  5 (Undamaged): DI =   0.2076 → normal ✓
State  6 (Undamaged): DI =   0.0277 → normal ✓
State  7 (Undamaged): DI =   0.4118 → DAMAGE ✗
State  8 (Undamaged): DI =   0.0015 → normal ✓
State  9 (Undamaged): DI =   0.2932 → normal ✓
State 10 (Damaged  ): DI =   3.2081 → DAMAGE ✓
State 11 (Damaged  ): DI =   8.5488 → DAMAGE ✓
State 12 (Damaged  ): DI =  12.3199 → DAMAGE ✓
State 13 (Damaged  ): DI =  18.2735 → DAMAGE ✓
State 14 (Damaged  ): DI =  24.4452 → DAMAGE ✓
State 15 (Damaged  ): DI =   4.4774 → DAMAGE ✓
State 16 (Damaged  ): DI =   3.5367 → DAMAGE ✓
State 17 (Damaged  ): DI =  14.1147 → DAMAGE ✓

Performance Summary:
Undamaged correctly classified: 7/9
Damaged correctly classified: 8/8
Overall accuracy: 15/17 (88.2%)
False positives: 2
False negatives: 0
```

Summary

This example demonstrated the complete PCA-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 RMSE values as damage-sensitive features
3. **PCA Modeling**: Learned PCA transformation from baseline training data
4. **Damage Detection**: Applied PCA-based scoring with 95% threshold
5. **Classification**: Achieved perfect separation between undamaged and damaged conditions

The results show that the PCA-based approach successfully discriminates between undamaged (states 1-9) and damaged (states 10-17) conditions without any false positives or false negatives.

Key advantages of this approach:

- Uses only undamaged data for training (unsupervised learning)
- Dimensionality reduction through PCA
- Statistical threshold based on training data variability
- Robust to measurement noise and environmental variations

See also:

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