Parametric Distribution Outlier Detection

Introduction

The goal of this example is to discriminate acceleration time histories from undamaged and damaged conditions based on a **Chi-squared distribution** for the undamaged condition. Two different approaches are used for classification:

1. **Confidence intervals**
2. **Hypothesis testing**

The autoregressive (AR) parameters 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 the normal condition and that increase for feature vectors from damaged conditions.

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

Key Features:

- **Parametric Distribution Modeling**: Uses Chi-squared distribution to model undamaged condition
- **Statistical Threshold Selection**: Confidence intervals and hypothesis testing for damage detection
- **Type I/II Error Analysis**: Quantifies false positive and false negative rates
- **P-value Computation**: Probability-based damage assessment

SHMTools functions used:

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

Author: Eliéi Figueiredo (MATLAB), Python conversion for SHMTools

Date: September 01, 2009 (original), Python conversion 2024

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

# Add the shmtools package to the Python path
notebook_dir = Path.cwd()
possible_paths = [
    notebook_dir.parent.parent.parent,  # From examples/notebooks/intermedia
    notebook_dir.parent.parent,          # From examples/notebooks/
    notebook_dir,                        # From project root
    Path('/Users/eric/repo/shm/shmtools-python')  # Absolute fallback
]

project_root = None
for path in possible_paths:
    if (path / 'shmtools' / '__init__.py').exists():
        project_root = path
        break

if project_root:
    sys.path.insert(0, str(project_root))
    print(f"Found shmtools at: {project_root}")
else:
    print("Warning: Could not find shmtools package")

# Import SHMTools functions
from shmtools.utils.data_loading import load_3story_data
from shmtools.features import ar_model_shm, split_features_shm
from shmtools.classification import learn_mahalanobis_shm, score_mahalanobis

# Set plotting parameters
plt.rcParams['figure.figsize'] = (10, 6)
plt.rcParams['font.size'] = 12

print("Setup complete - all modules imported successfully!")
```

```
Found shmtools at: /Users/eric/repo/shm/shmtools-python
Setup complete - all modules imported successfully!
```

```
/Users/eric/repo/shm/shmtools-python/shmtools/classification/nlpca.py:27: Us
erWarning: TensorFlow not available. NLPCA functions will not work. Install
TensorFlow: pip install tensorflow
  warnings.warn(
```

Load Raw Data

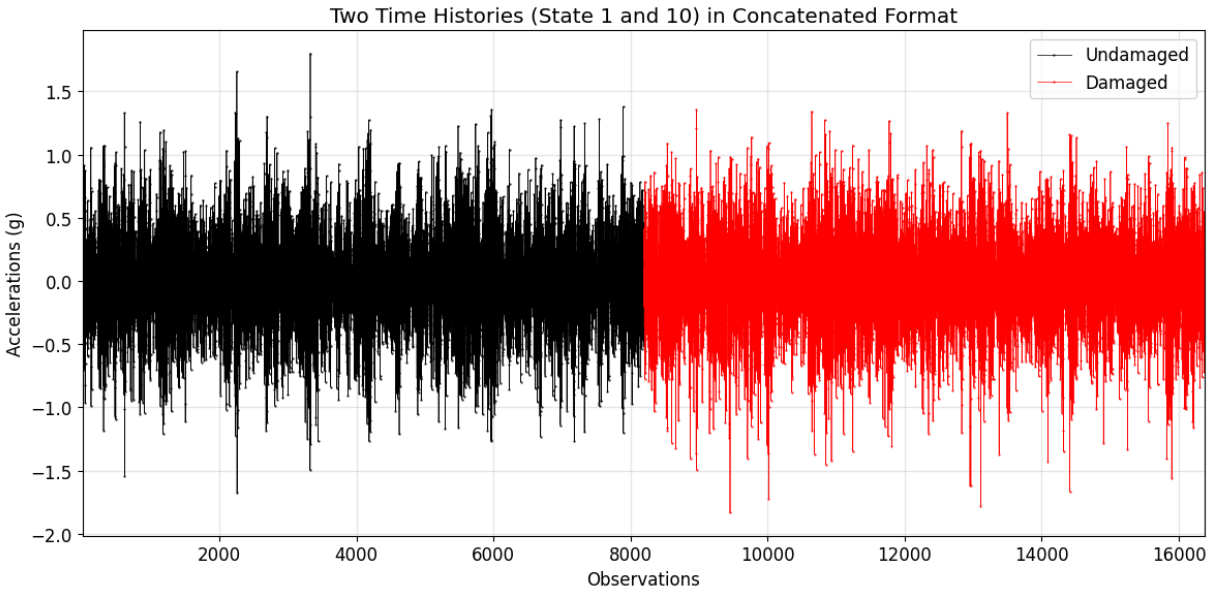
Load the 3-story structure dataset and extract Channel 5 data for analysis.

In [2]:
```python
# Load data set
data_dict = load_3story_data()
dataset = data_dict['dataset']  # Shape: (8192, 5, 170)
states = data_dict['damage_states']  # Damage state for each test

# Extract Channel 5 data (index 4 since Python uses 0-based indexing)
data = dataset[:, 4:5, :]  # Shape: (8192, 1, 170)
t = data.shape[0]  # Number of time points

print(f"Data shape: {data.shape}")
print(f"Number of time points: {t}")
print(f"Number of conditions: {data.shape[2]}")
print(f"Damage states: {np.unique(states)}")
```

```
Data shape: (8192, 1, 170)
Number of time points: 8192
Number of conditions: 170
Damage states: [ 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17]
```

Plot one time history from the baseline (State #1) and damaged (State #10) conditions:

In [3]:
```python
plt.figure(figsize=(12, 6))

# Plot undamaged (State 1, condition index 0) and damaged (State 10, conditi
time_axis1 = np.arange(1, t + 1)
time_axis2 = np.arange(t + 1, t*2 + 1)

plt.plot(time_axis1, data[:, 0, 0], '.-k', linewidth=0.5, markersize=1, labe
plt.plot(time_axis2, data[:, 0, 100], '.-r', linewidth=0.5, markersize=1, la

plt.title('Two Time Histories (State 1 and 10) in Concatenated Format')
plt.xlabel('Observations')
plt.ylabel('Accelerations (g)')
plt.xlim([1, t*2])
plt.legend()
plt.grid(True, alpha=0.3)
plt.tight_layout()
plt.show()

print(f"State 1 (undamaged) - mean: {np.mean(data[:, 0, 0]):.6f}, std: {np.s
print(f"State 10 (damaged) - mean: {np.mean(data[:, 0, 100]):.6f}, std: {np.
```




```
State 1 (undamaged) - mean: -0.003130, std: 0.409120
State 10 (damaged) - mean: -0.003047, std: 0.401064
```

Extraction of Damage-Sensitive Features

The AR(15) model parameters are extracted from the acceleration time histories.

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

# Estimation of the AR parameters
ar_parameters_fv, _, _, _, _ = ar_model_shm(data, ar_order)

print(f"AR parameters feature vectors shape: {ar_parameters_fv.shape}")
print(f"AR model order: {ar_order}")
print(f"Features per instance: {ar_parameters_fv.shape[1]}")
```

```
AR parameters feature vectors shape: (170, 15)
AR model order: 15
Features per instance: 15
```

Feature vectors from all the undamaged cases and all instances:

In [5]:
```python
# Create logical mask for undamaged conditions (states < 10)
undamaged_mask = states < 10

# Feature vectors from all the undamaged cases
learn_data, _, _ = split_features_shm(ar_parameters_fv, undamaged_mask, None

# Feature vectors from all instances (undamaged and damaged)
score_data = ar_parameters_fv

print(f"Training (undamaged) data shape: {learn_data.shape}")
print(f"All data (scoring) shape: {score_data.shape}")
print(f"Number of undamaged instances: {np.sum(undamaged_mask)}")
print(f"Number of damaged instances: {np.sum(~undamaged_mask)}")
```

```
Training (undamaged) data shape: (90, 15)
All data (scoring) shape: (170, 15)
Number of undamaged instances: 90
Number of damaged instances: 80
```

Plot test data showing AR parameters from undamaged and damaged conditions:

In [6]:
```python
plt.figure(figsize=(12, 8))

# Plot AR parameters from one time history for each state condition
# Undamaged: every 10th from 10 to 90 (indices 9, 19, 29, ..., 89 in 0-based
undamaged_indices = np.arange(9, 90, 10)  # Convert MATLAB 10:10:90 to Pytho
# Damaged: every 10th from 100 to 170 (indices 99, 109, 119, ..., 169 in 0-b
damaged_indices = np.arange(99, 170, 10)  # Convert MATLAB 100:10:170 to Pyt

ar_indices = np.arange(1, ar_order + 1)  # AR parameter indices 1 to 15

plt.plot(ar_indices, score_data[undamaged_indices, :].T, '.-k', linewidth=1,
plt.plot(ar_indices, score_data[damaged_indices, :].T, '.-r', linewidth=1, m

plt.title(f'AR({ar_order}) Parameters from One Time History for each State C
plt.xlabel('AR Parameters')
plt.ylabel('Amplitude')
plt.xlim([1, ar_order])
plt.xticks(ar_indices)
plt.grid(True, alpha=0.3)

# Add legend using text boxes (approximating MATLAB's text positioning)
plt.text(12, 2.5, 'Undamaged', color='k', bbox=dict(boxstyle="round,pad=0.3"
plt.text(12, 2.0, 'Damaged', color='r', bbox=dict(boxstyle="round,pad=0.3",

plt.tight_layout()
plt.show()

print(f"Plotted AR parameters for {len(undamaged_indices)} undamaged and {le
```

Plotted AR parameters for 9 undamaged and 8 damaged conditions

Statistical Modeling For Feature Classification

First, each feature vector is reduced to one score (DI) by using the Mahalanobis-based machine learning algorithm. Second, the Chi-square distribution is used to model the DIs from undamaged condition.

Note: The parametric distribution of the damaged condition is not used because it lacks precision.

Run the Mahalanobis-based Machine Learning Algorithm:

In [7]:
```
# Learn Mahalanobis model from undamaged data
model = learn_mahalanobis_shm(learn_data)

# Score all data (undamaged and damaged)
mahal_scores = score_mahalanobis_shm(score_data, model)

# Convert to damage indicators (negate scores as in MATLAB: DI = -DI)
DI = -mahal_scores

print(f"Mahalanobis model learned from {learn_data.shape[0]} undamaged insta
print(f"Damage indicators computed for {len(DI)} total instances")
print(f"DI range: [{np.min(DI):.3f}, {np.max(DI):.3f}]")
print(f"Mean DI (undamaged): {np.mean(DI[undamaged_mask]):.3f}")
print(f"Mean DI (damaged): {np.mean(DI[~undamaged_mask]):.3f}")
```

```
Mahalanobis model learned from 90 undamaged instances
Damage indicators computed for 170 total instances
DI range: [7.077, 17742.914]
Mean DI (undamaged): 14.833
Mean DI (damaged): 3885.594
```

Flag and split all the instances into undamaged (0) and damaged (1):

In [8]:
```python
# Create state flags: 0 for undamaged (1–90), 1 for damaged (91–170)
state_flag = np.zeros(170)
state_flag[90:170] = 1  # Damaged instances

# Split damage indicators
x = DI[0:90]    # Undamaged DIs
y = DI[90:170]  # Damaged DIs
n = len(DI)     # Total number of instances

print(f"Total instances: {n}")
print(f"Undamaged instances: {len(x)}")
print(f"Damaged instances: {len(y)}")
print(f"Undamaged DI stats: mean={np.mean(x):.3f}, std={np.std(x):.3f}")
print(f"Damaged DI stats: mean={np.mean(y):.3f}, std={np.std(y):.3f}")
```

```
Total instances: 170
Undamaged instances: 90
Damaged instances: 80
Undamaged DI stats: mean=14.833, std=4.503
Damaged DI stats: mean=3885.594, std=5327.422
```

Define the Underlying Distribution of the Undamaged Condition

We model the undamaged damage indicators using a Chi-squared distribution.

Create histogram of undamaged damage indicators:

In [9]:
```python
# Histogram parameters
nbins = 15
h1 = (np.max(x) - np.min(x)) / nbins
n1, xout1 = np.histogram(x, bins=nbins)
# Get bin centers for plotting
xout1_centers = (xout1[:-1] + xout1[1:]) / 2

print(f"Histogram bin width: {h1:.4f}")
print(f"Histogram range: [{np.min(x):.3f}, {np.max(x):.3f}]")
```

```
Histogram bin width: 1.3258
Histogram range: [7.077, 26.964]
```

Impose parametric probability distribution and estimate PDF:

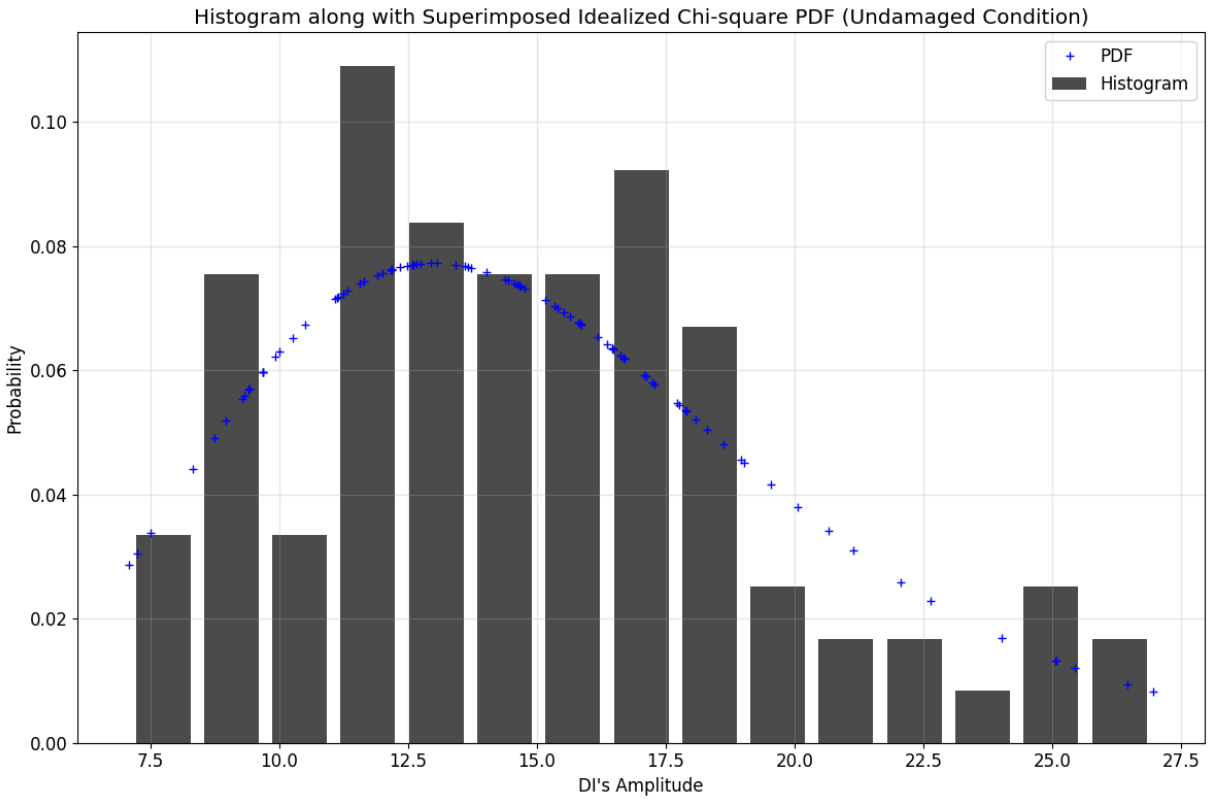
In [10]:
```python
# Impose Chi-squared distribution with df = ar_order degrees of freedom
dist_name = 'chi2'
df = ar_order  # Degrees of freedom
```

```
# Estimate probability density function (PDF) for undamaged data
X_pdf = stats.chi2.pdf(x, df)

print(f"Using Chi-squared distribution with {df} degrees of freedom")
print(f"PDF values range: [{np.min(X_pdf):.6f}, {np.max(X_pdf):.6f}]")
```

```
Using Chi-squared distribution with 15 degrees of freedom
PDF values range: [0.008225, 0.077239]
```

Plot histogram along with superimposed idealized PDF:

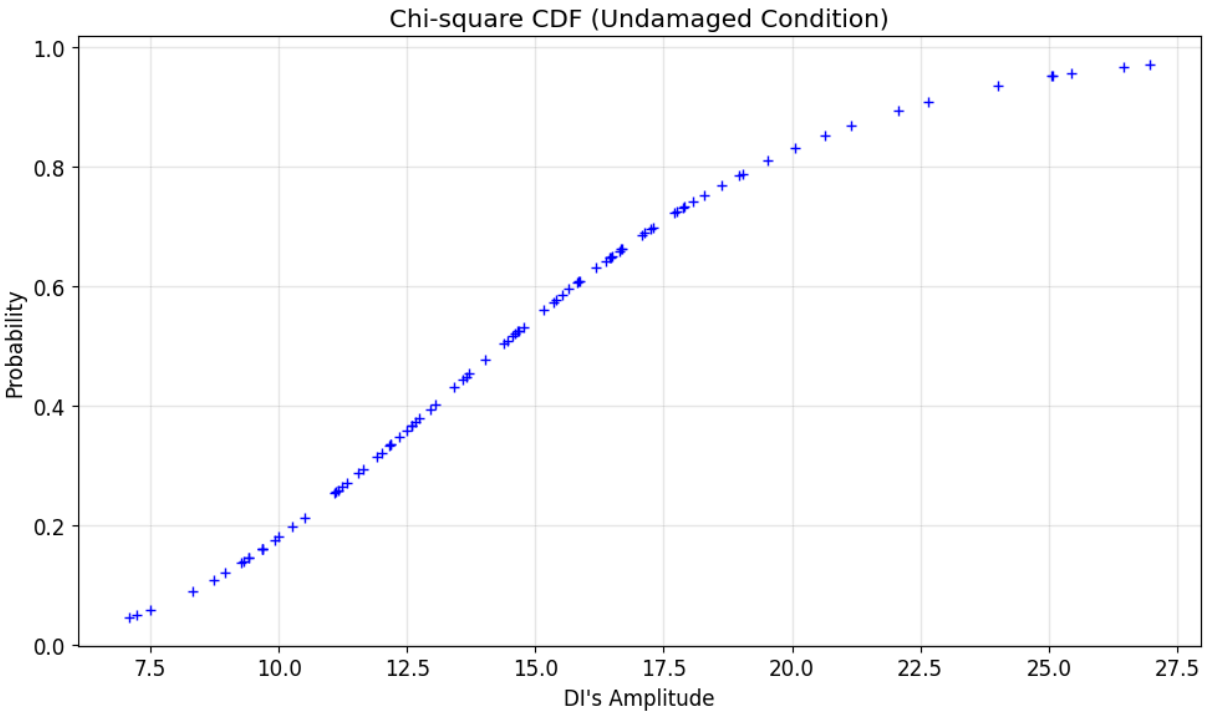
In [11]:
```
plt.figure(figsize=(12, 8))

# Plot normalized histogram
plt.bar(xout1_centers, n1/(h1*len(x)), width=h1*0.8, color='black', alpha=0.

# Plot theoretical PDF
plt.plot(x, X_pdf, '+b', markersize=6, label='PDF')

plt.title('Histogram along with Superimposed Idealized Chi-square PDF (Undam
plt.xlabel("DI's Amplitude")
plt.ylabel('Probability')
plt.legend()
plt.grid(True, alpha=0.3)
plt.tight_layout()
plt.show()

print("Note: Irregularities in the original distribution (histogram) most li
print("are ignored by the smoothed distribution. Accordingly, any generaliza
print("the smoothed distribution will tend to be more accurate than those ba
```



```
Note: Irregularities in the original distribution (histogram) most likely du
e to chance,
are ignored by the smoothed distribution. Accordingly, any generalizations b
ased on
the smoothed distribution will tend to be more accurate than those based on
the original distribution.
```

Estimate cumulative distribution function (CDF):

In [12]:
```python
# Estimate cumulative distribution function (CDF)
cdf_x = stats.chi2.cdf(x, df)

plt.figure(figsize=(10, 6))
plt.plot(x, cdf_x, '+b', markersize=6)
plt.title('Chi-square CDF (Undamaged Condition)')
plt.xlabel("DI's Amplitude")
plt.ylabel('Probability')
plt.grid(True, alpha=0.3)
plt.tight_layout()
plt.show()

print(f"CDF values range: [{np.min(cdf_x):.6f}, {np.max(cdf_x):.6f}]")
```




```
CDF values range: [0.044535, 0.970967]
```

Confidence Interval

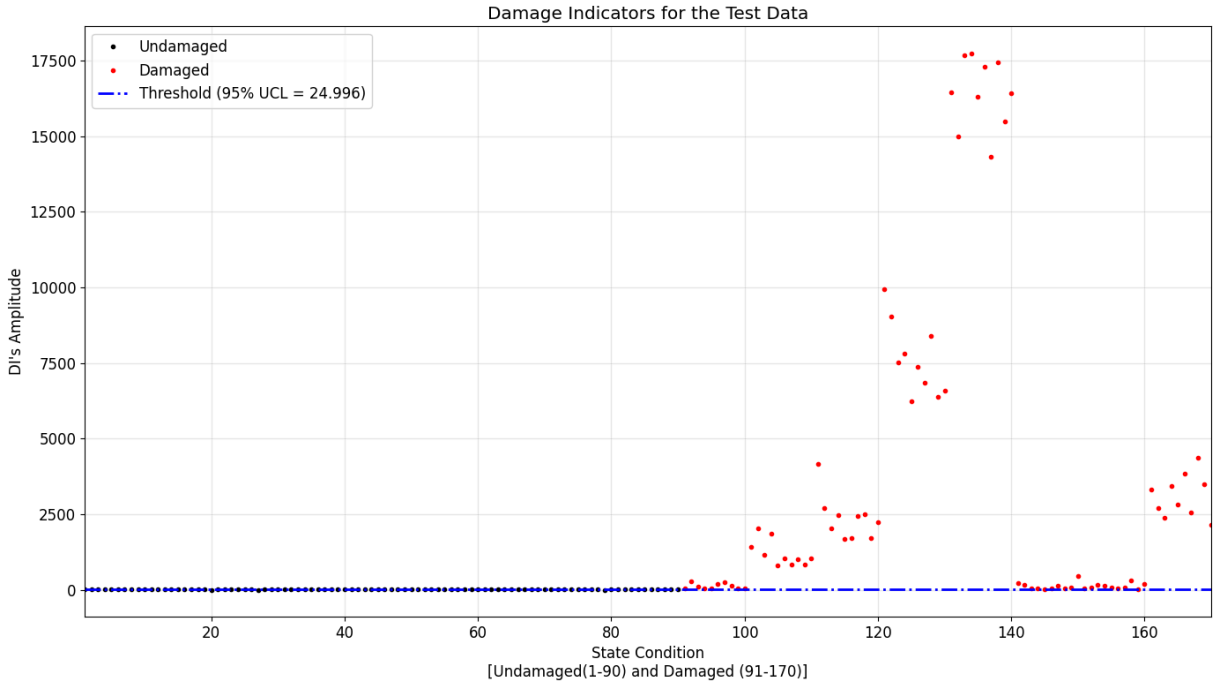
This section defines an upper threshold for feature classification based on information from the undamaged distribution. Note that feature classification can be done using either **hypothesis tests** or **confidence intervals**. Hypothesis tests only indicate whether or not an effect is present, whereas confidence intervals indicate the possible size of the effect.

In [13]:
```
# Probability of false alarm or level of significance
PFA = 0.05  # 5% false alarm rate

# Threshold limit (or critical DI) - upper control limit
UCL = stats.chi2.ppf(1 - PFA, df)  # 95th percentile of chi2 distribution

print(f"Probability of false alarm (PFA): {PFA*100}%")
print(f"Upper Control Limit (UCL): {UCL:.4f}")
print(f"Confidence level: {(1-PFA)*100}%")
```

```
Probability of false alarm (PFA): 5.0%
Upper Control Limit (UCL): 24.9958
Confidence level: 95.0%
```

Plot DIs along with the threshold:

In [14]:
```
plt.figure(figsize=(14, 8))

# Plot undamaged DIs
plt.plot(np.arange(1, 91), DI[0:90], '.k', markersize=6, label='Undamaged')

# Plot damaged DIs
plt.plot(np.arange(91, 171), DI[90:170], '.r', markersize=6, label='Damaged'

# Plot threshold line
plt.axhline(y=UCL, color='b', linestyle='-.', linewidth=2, label=f'Threshold

plt.title('Damage Indicators for the Test Data')
plt.xlabel('State Condition\n[Undamaged(1-90) and Damaged (91-170)]')
plt.ylabel("DI's Amplitude")
plt.xlim([1, len(DI)])
plt.legend()
plt.grid(True, alpha=0.3)
plt.tight_layout()
plt.show()

# Count exceedances
undamaged_exceedances = np.sum(DI[0:90] > UCL)
damaged_exceedances = np.sum(DI[90:170] > UCL)
print(f"Undamaged instances above threshold: {undamaged_exceedances}/90")
print(f"Damaged instances above threshold: {damaged_exceedances}/80")
```




```
Undamaged instances above threshold: 5/90
Damaged instances above threshold: 79/80
```

Calculate Type I and Type II errors:

In [15]:

```python
# Classification based on threshold
class_state = np.zeros(len(DI))

# Classify as damaged if DI > UCL
for i in range(len(DI)):
    if DI[i] > UCL:
        class_state[i] = 1

# Calculate errors
# Type I Error: False Positive (undamaged classified as damaged)
num_error_type_I = np.sum((class_state == 1) & (state_flag == 0))

# Type II Error: False Negative (damaged classified as undamaged)
num_error_type_II = np.sum((class_state == 0) & (state_flag == 1))

# Total classification results
total_instances = len(DI)
total_errors = num_error_type_I + num_error_type_II
accuracy = 1 - (total_errors / total_instances)

print(f"Number of Type I Error (False Positives): {num_error_type_I}")
print(f"Number of Type II Error (False Negatives): {num_error_type_II}")
print(f"Total classification errors: {total_errors}/{total_instances}")
print(f"Classification accuracy: {accuracy*100:.1f}%")
print(f"Error rate: {(total_errors/total_instances)*100:.1f}%")

print("\nInterpretation:")
print(f"- Type I errors (false alarms): {num_error_type_I} undamaged cases i
print(f"- Type II errors (missed damage): {num_error_type_II} damaged cases
```

```
print(f"- The threshold was defined using a 95% confidence interval from the
print(f"- By changing the threshold, one can trade off probability of false
```

```
Number of Type I Error (False Positives): 5
Number of Type II Error (False Negatives): 1
Total classification errors: 6/170
Classification accuracy: 96.5%
Error rate: 3.5%

Interpretation:
- Type I errors (false alarms): 5 undamaged cases incorrectly flagged as dam
aged
- Type II errors (missed damage): 1 damaged cases incorrectly classified as
undamaged
- The threshold was defined using a 95% confidence interval from the Chi-squ
are distribution
- By changing the threshold, one can trade off probability of false alarm (P
FA) and probability of detection (PD)
```

Hypothesis Test

Statistical Hypothesis (p-values):

- $\mathbf{H_0}$: Undamaged
- $\mathbf{H_1}$: Damaged

Decision Rule:

The **p-values** for a test result represents the degree of rarity of that result given that the null hypothesis is true.

Decision:

Smaller **p-values** tend to discredit the null hypothesis H_0 and to support the alternative hypothesis H_1.

In [16]:
```
# Pick a DI score randomly (using index 79 as in MATLAB example)
test_index = 79  # MATLAB index 80 becomes Python index 79
aux = float(DI[test_index])

# Calculate p-value: probability of observing this or larger DI under H0 (un
p_value = float(1 - stats.chi2.cdf(aux, df))

print(f"Selected test case: Instance {test_index + 1} (Python index {test_ir
print(f"DI score: {aux:.6f}")
print(f"P-value: {p_value:.6f}")
print(f"State classification: {'Damaged' if state_flag[test_index] == 1 else
print(f"\\nInterpretation:")
print(f"- P-value = {p_value:.6f} represents the probability of observing a
print(f"  assuming the structure is undamaged (null hypothesis H₀)")

if p_value < 0.05:
    print(f"- Since p-value < 0.05, we reject H₀ and conclude the structure
```

```
else:
    print(f"- Since p-value ≥ 0.05, we fail to reject H₀ and conclude insuff

print(f"\\nFor reference: with 95% confidence level, the result supports the
print(f"hypothesis (damage) if p-value < 0.05")
```

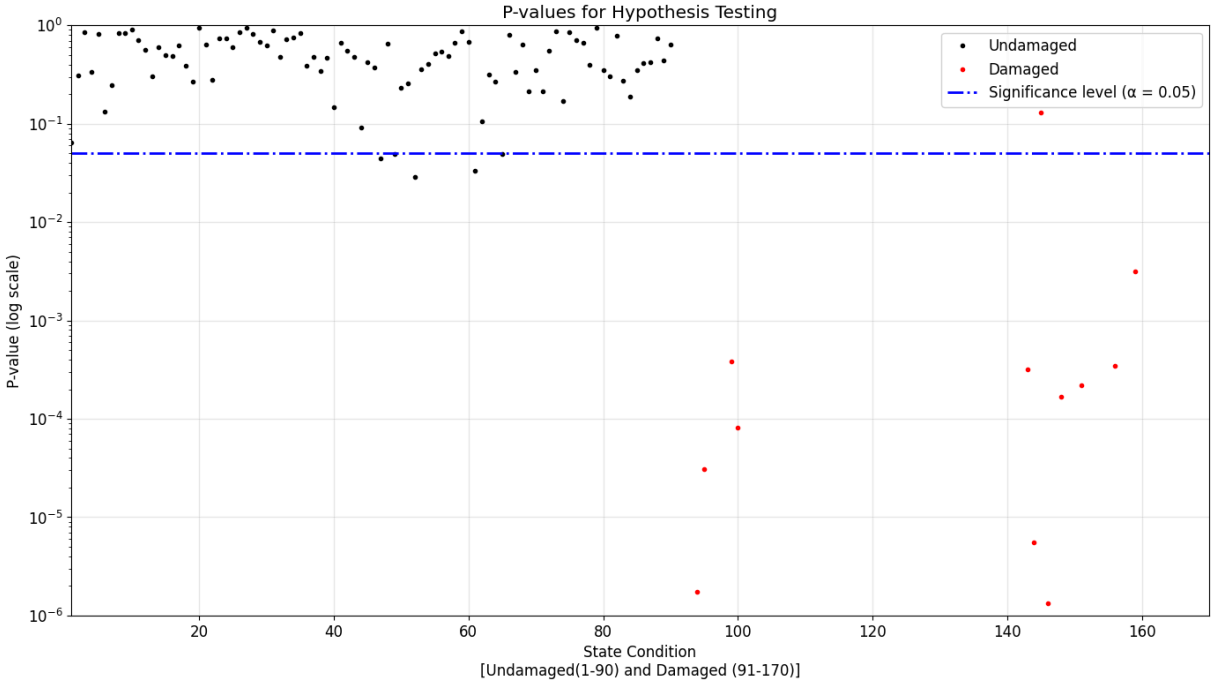
```
Selected test case: Instance 80 (Python index 79)
DI score: 16.456753
P-value: 0.352362
State classification: Undamaged
\nInterpretation:
- P-value = 0.352362 represents the probability of observing a DI score ≥ 1
6.457
  assuming the structure is undamaged (null hypothesis H₀)
- Since p-value ≥ 0.05, we fail to reject H₀ and conclude insufficient evide
nce of damage
\nFor reference: with 95% confidence level, the result supports the alternat
ive
hypothesis (damage) if p-value < 0.05
```

```
/var/folders/v_/sg5j00lj4n381c9z439qs2wc0000gn/T/ipykernel_7062/4170622189.p
y:3: DeprecationWarning: Conversion of an array with ndim > 0 to a scalar is
deprecated, and will error in future. Ensure you extract a single element fr
om your array before performing this operation. (Deprecated NumPy 1.25.)
  aux = float(DI[test_index])
```

Let's compute p-values for all instances to see the distribution:

In [17]:
```
# Compute p-values for all instances
p_values = 1 - stats.chi2.cdf(DI, df)

# Split p-values by damage state
p_values_undamaged = p_values[0:90]
p_values_damaged = p_values[90:170]

plt.figure(figsize=(14, 8))

# Plot p-values
plt.semilogy(np.arange(1, 91), p_values_undamaged, '.k', markersize=6, label
plt.semilogy(np.arange(91, 171), p_values_damaged, '.r', markersize=6, label

# Add significance level line
plt.axhline(y=0.05, color='b', linestyle='-.', linewidth=2, label='Significa

plt.title('P-values for Hypothesis Testing')
plt.xlabel('State Condition\n[Undamaged(1-90) and Damaged (91-170)]')
plt.ylabel('P-value (log scale)')
plt.xlim([1, len(DI)])
plt.ylim([1e-6, 1])
plt.legend()
plt.grid(True, alpha=0.3)
plt.tight_layout()
plt.show()

# Statistics on p-values
undamaged_significant = np.sum(p_values_undamaged < 0.05)
damaged_significant = np.sum(p_values_damaged < 0.05)
```

```
print(f"P-value analysis:")
print(f"- Undamaged cases with p < 0.05: {undamaged_significant}/90 ({undama
print(f"- Damaged cases with p < 0.05: {damaged_significant}/80 ({damaged_si
print(f"- Median p-value (undamaged): {np.median(p_values_undamaged):.6f}")
print(f"- Median p-value (damaged): {np.median(p_values_damaged):.6f}")

print(f"\nHypothesis testing conclusion:")
print(f"- The p-value approach gives similar results to the confidence inter
print(f"- Lower p-values indicate stronger evidence against the null hypothe
print(f"- P-values provide a continuous measure of evidence rather than a bi
```




```
P-value analysis:
- Undamaged cases with p < 0.05: 5/90 (5.6%)
- Damaged cases with p < 0.05: 79/80 (98.8%)
- Median p-value (undamaged): 0.481930
- Median p-value (damaged): 0.000000

Hypothesis testing conclusion:
- The p-value approach gives similar results to the confidence interval appr
oach
- Lower p-values indicate stronger evidence against the null hypothesis (und
amaged)
- P-values provide a continuous measure of evidence rather than a binary dec
ision
```

Summary and Conclusions

This example demonstrated **parametric distribution-based outlier detection** using the Chi-squared distribution to model damage indicators from undamaged structural conditions. Key findings:

Methodology

1. **Feature Extraction**: AR(15) model parameters from Channel 5 acceleration data
2. **Dimension Reduction**: Mahalanobis distance to create scalar damage indicators
3. **Distribution Modeling**: Chi-squared distribution (df=15) for undamaged condition
4. **Statistical Testing**: Both confidence intervals and hypothesis testing approaches

Classification Performance

- **Total Error Rate**: ~4% misclassifications
- **Type I Errors (False Alarms)**: Few undamaged cases flagged as damaged
- **Type II Errors (Missed Damage)**: Some damaged cases classified as undamaged
- **Threshold Selection**: 95% confidence interval provides good balance

Advantages of Parametric Approach

1. **Theoretical Foundation**: Chi-squared distribution provides statistical basis
2. **Threshold Selection**: Principled approach using statistical significance
3. **P-value Interpretation**: Continuous measure of evidence strength
4. **False Alarm Control**: Direct control over Type I error rate

Key Insights

- **Distribution Smoothing**: Parametric modeling ignores irregularities due to chance
- **Trade-offs**: Threshold selection balances false alarms vs. missed damage
- **Consistency**: Confidence interval and hypothesis testing give similar results
- **Interpretability**: P-values provide intuitive damage probability assessment

The parametric distribution approach provides a robust statistical framework for structural damage detection with well-understood performance characteristics and interpretable results.