Modal Analysis and Frequency Response Functions

This notebook demonstrates modal analysis techniques for structural health monitoring, including frequency response function (FRF) computation and natural frequency extraction. These modal properties serve as damage-sensitive features for structural condition assessment.

Introduction

Modal analysis is fundamental to structural health monitoring as structural damage typically manifests as changes in modal properties such as natural frequencies, mode shapes, and damping ratios. This example demonstrates:

1. **FRF Computation**: Calculate frequency response functions from time domain data
2. **Natural Frequency Extraction**: Extract modal parameters from FRFs
3. **Damage Sensitivity**: Observe how modal properties change with structural condition

The analysis uses the 3-story structure dataset which contains measurements from both undamaged and damaged structural states.

In [1]:
```
# Import required libraries
import numpy as np
import matplotlib.pyplot as plt
from pathlib import Path
import sys
import warnings
warnings.filterwarnings('ignore')

# Add shmtools to path if needed
notebook_dir = Path.cwd()
possible_paths = [
    notebook_dir.parent.parent.parent,  # From examples/notebooks/advanced/
    notebook_dir.parent.parent,          # From examples/notebooks/
    notebook_dir,                        # From project root
]

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

# Import SHMTools functions
```

```python
from shmtools.utils.data_loading import load_3story_data
from shmtools.modal import frf_shm, rpfit_shm

# Set up plotting parameters
plt.rcParams['figure.figsize'] = (12, 6)
plt.rcParams['font.size'] = 11
```

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

Load and Examine Data

Load the 3-story structure dataset and examine the input-output relationship.

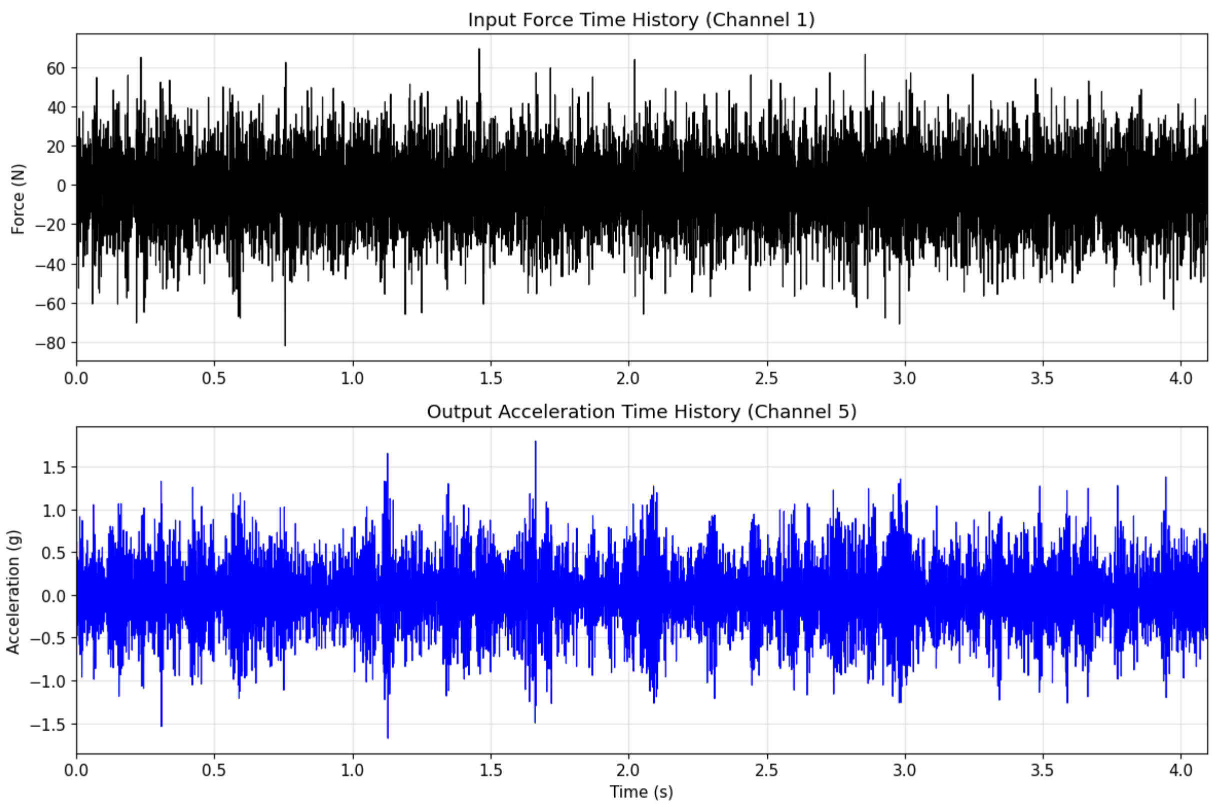
In [2]:
```python
# Load the 3-story structure dataset
data = load_3story_data()
dataset = data['dataset']

print(f"Dataset information:")
print(f"  Shape: {dataset.shape}")
print(f"  Time points: {dataset.shape[0]}")
print(f"  Channels: {dataset.shape[1]}")
print(f"  Test conditions: {dataset.shape[2]}")
print(f"  Sampling frequency: {data['fs']} Hz")

# Focus on input-output relationship: Channel 1 (force) → Channel 5 (acceler
input_output_data = dataset[:, [0, 4], :]  # Extract channels 1 and 5

print(f"\nInput-output data shape: {input_output_data.shape}")
print(f"  Channel 1: Input force")
print(f"  Channel 5: Output acceleration (top floor)")
```

```
Dataset information:
  Shape: (8192, 5, 170)
  Time points: 8192
  Channels: 5
  Test conditions: 170
  Sampling frequency: 2000.0 Hz

Input-output data shape: (8192, 2, 170)
  Channel 1: Input force
  Channel 5: Output acceleration (top floor)
```

Visualize Sample Time Histories

In [3]:
```python
# Plot sample time histories from baseline condition
fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(12, 8))

condition_idx = 0  # First baseline condition
time_vector = np.arange(dataset.shape[0]) / data['fs']

# Input force time history
ax1.plot(time_vector, dataset[:, 0, condition_idx], 'k', linewidth=0.8)
ax1.set_title('Input Force Time History (Channel 1)')
ax1.set_ylabel('Force (N)')
```

```
ax1.set_xlim([0, time_vector[-1]])
ax1.grid(True, alpha=0.3)

# Output acceleration time history
ax2.plot(time_vector, dataset[:, 4, condition_idx], 'b', linewidth=0.8)
ax2.set_title('Output Acceleration Time History (Channel 5)')
ax2.set_xlabel('Time (s)')
ax2.set_ylabel('Acceleration (g)')
ax2.set_xlim([0, time_vector[-1]])
ax2.grid(True, alpha=0.3)

plt.tight_layout()
plt.show()

# Print basic statistics
print(f"Time domain statistics:")
print(f"  Force RMS: {np.sqrt(np.mean(dataset[:, 0, condition_idx]**2)):.2f}
print(f"  Acceleration RMS: {np.sqrt(np.mean(dataset[:, 4, condition_idx]**2
print(f"  Record length: {time_vector[-1]:.1f} seconds")
```




```
Time domain statistics:
  Force RMS: 19.62 N
  Acceleration RMS: 0.409 g
  Record length: 4.1 seconds
```

Frequency Response Function (FRF) Computation

Compute FRFs between the input force and output acceleration using Welch's method.

In [4]:
```
# Set FRF computation parameters
block_size = 2048  # FFT block size
```

```
overlap = 0.5       # 50% overlap
window = 'hann'     # Hann window

print(f"FRF computation parameters:")
print(f"  Block size: {block_size} points")
print(f"  Overlap: {overlap*100:.0f}%")
print(f"  Window: {window}")

# Compute FRFs
print(f"\nComputing FRFs for all {input_output_data.shape[2]} conditions..."
frf_data = frf_shm(input_output_data, block_size, overlap, window, single_si

print(f"FRF computation completed:")
print(f"  FRF shape: {frf_data.shape}")
print(f"  Frequency points: {frf_data.shape[0]}")
print(f"  Output channels: {frf_data.shape[1]}")
print(f"  Test conditions: {frf_data.shape[2]}")

# Create frequency vector
freq_vector = np.linspace(0, data['fs']/2, frf_data.shape[0])
freq_resolution = freq_vector[1] - freq_vector[0]

print(f"  Frequency range: 0 - {freq_vector[-1]:.1f} Hz")
print(f"  Frequency resolution: {freq_resolution:.2f} Hz")
```

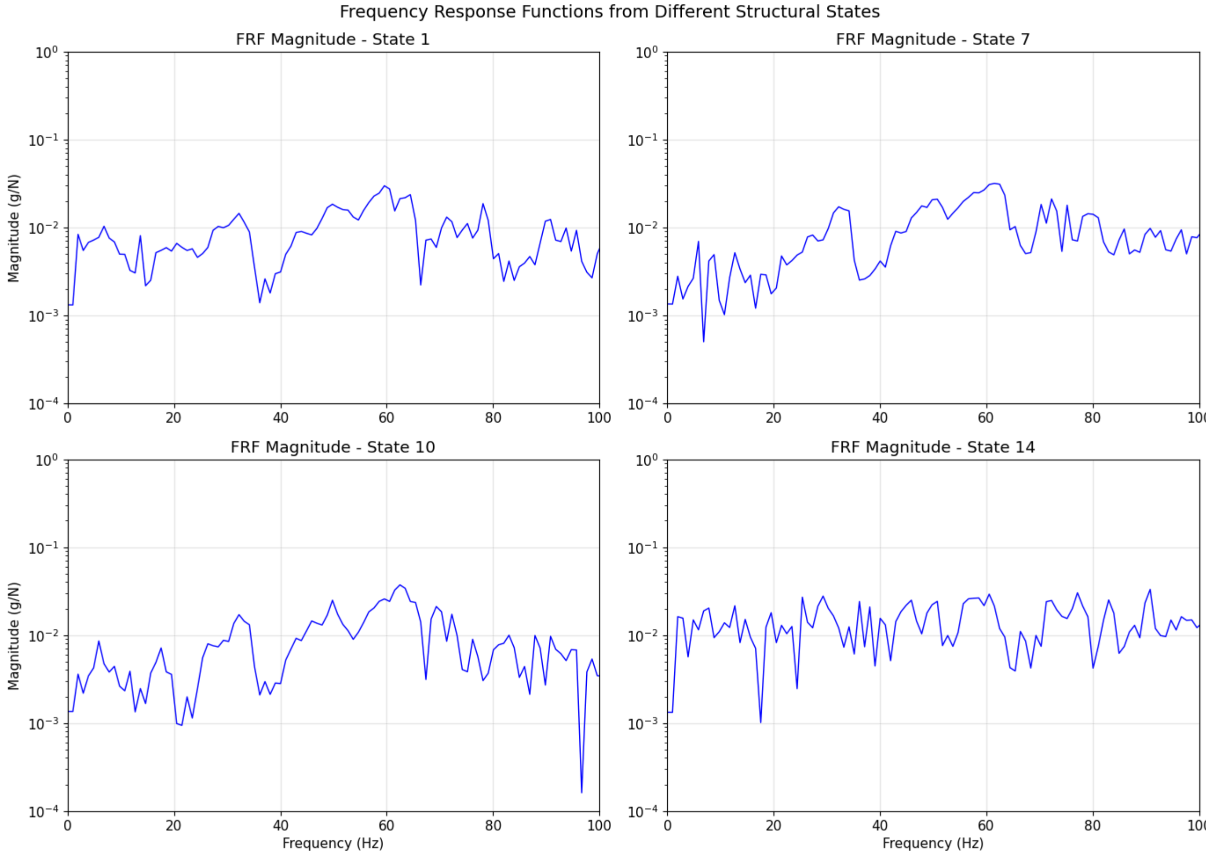
```
FRF computation parameters:
  Block size: 2048 points
  Overlap: 50%
  Window: hann

Computing FRFs for all 170 conditions...
FRF computation completed:
  FRF shape: (1025, 1, 170)
  Frequency points: 1025
  Output channels: 1
  Test conditions: 170
  Frequency range: 0 - 1000.0 Hz
  Frequency resolution: 0.98 Hz
```

Visualize FRFs from Different Structural States

In [5]:

```
# Plot FRFs from representative structural states
# Each structural state has 10 test conditions
states_to_plot = [1, 7, 10, 14]  # Representative states
state_indices = [(s-1)*10 for s in states_to_plot]  # Convert to condition i

fig, axes = plt.subplots(2, 2, figsize=(14, 10))
axes = axes.flatten()

for i, (state_num, condition_idx) in enumerate(zip(states_to_plot, state_inc
    # Compute magnitude of FRF
    frf_magnitude = np.abs(frf_data[:, 0, condition_idx])

    axes[i].semilogy(freq_vector, frf_magnitude, 'b-', linewidth=1)
    axes[i].set_title(f'FRF Magnitude - State {state_num}')
```

```python
    axes[i].set_xlim([0, 100])  # Focus on low frequency range
    axes[i].set_ylim([1e-4, 1])
    axes[i].grid(True, alpha=0.3)

    # Add labels to bottom row
    if i >= 2:
        axes[i].set_xlabel('Frequency (Hz)')

    # Add labels to left column
    if i % 2 == 0:
        axes[i].set_ylabel('Magnitude (g/N)')

plt.suptitle('Frequency Response Functions from Different Structural States'
plt.tight_layout()
plt.show()

# Identify approximate natural frequencies by visual inspection
print("\nApproximate natural frequencies from FRF plots:")
for i, (state_num, condition_idx) in enumerate(zip(states_to_plot, state_inc
    frf_magnitude = np.abs(frf_data[:, 0, condition_idx])

    # Find peaks in frequency ranges of interest
    freq_ranges = [(25, 35), (45, 55), (60, 70)]  # Approximate modal freque
    peaks = []

    for f_start, f_end in freq_ranges:
        start_idx = np.argmin(np.abs(freq_vector - f_start))
        end_idx = np.argmin(np.abs(freq_vector - f_end))
        segment = frf_magnitude[start_idx:end_idx+1]
        peak_idx = np.argmax(segment) + start_idx
        peaks.append(freq_vector[peak_idx])

    print(f"  State {state_num}: {peaks[0]:.1f}, {peaks[1]:.1f}, {peaks[2]:.
```




```
Approximate natural frequencies from FRF plots:
  State 1: 32.2, 49.8, 59.6 Hz
  State 7: 32.2, 50.8, 61.5 Hz
  State 10: 32.2, 49.8, 62.5 Hz
  State 14: 29.3, 45.9, 60.5 Hz
```

Natural Frequency Tracking Across All Conditions

Extract natural frequencies from all test conditions to observe trends and changes.

In [6]:
```
# Extract natural frequencies from all conditions using peak detection
freq_ranges = np.array([
    [26, 36],  # First mode frequency range
    [50, 60],  # Second mode frequency range
    [65, 75]   # Third mode frequency range
])

print(f"Extracting natural frequencies from {frf_data.shape[2]} conditions..

# Initialize array to store natural frequencies
natural_frequencies = np.zeros((len(freq_ranges), frf_data.shape[2]))

# Extract peak frequency in each range for each condition
for condition in range(frf_data.shape[2]):
    frf_magnitude = np.abs(frf_data[:, 0, condition])

    for mode_idx, (f_start, f_end) in enumerate(freq_ranges):
        # Find frequency indices for this range
        start_idx = np.argmin(np.abs(freq_vector - f_start))
```

```
        end_idx = np.argmin(np.abs(freq_vector - f_end))

        # Find peak in this frequency range
        segment = frf_magnitude[start_idx:end_idx+1]
        peak_idx = np.argmax(segment)
        natural_frequencies[mode_idx, condition] = freq_vector[start_idx + p

print(f"Natural frequency extraction completed.")
print(f"Natural frequencies shape: {natural_frequencies.shape}")

# Display sample results
print(f"\nSample natural frequencies (first 5 conditions):")
for i in range(5):
    freqs = natural_frequencies[:, i]
    print(f"  Condition {i+1}: Mode1={freqs[0]:.2f} Hz, Mode2={freqs[1]:.2f}
```

```
Extracting natural frequencies from 170 conditions...
Natural frequency extraction completed.
Natural frequencies shape: (3, 170)

Sample natural frequencies (first 5 conditions):
  Condition 1: Mode1=32.23 Hz, Mode2=59.57 Hz, Mode3=71.29 Hz
  Condition 2: Mode1=32.23 Hz, Mode2=59.57 Hz, Mode3=69.34 Hz
  Condition 3: Mode1=32.23 Hz, Mode2=59.57 Hz, Mode3=66.41 Hz
  Condition 4: Mode1=32.23 Hz, Mode2=59.57 Hz, Mode3=67.38 Hz
  Condition 5: Mode1=33.20 Hz, Mode2=59.57 Hz, Mode3=69.34 Hz
```

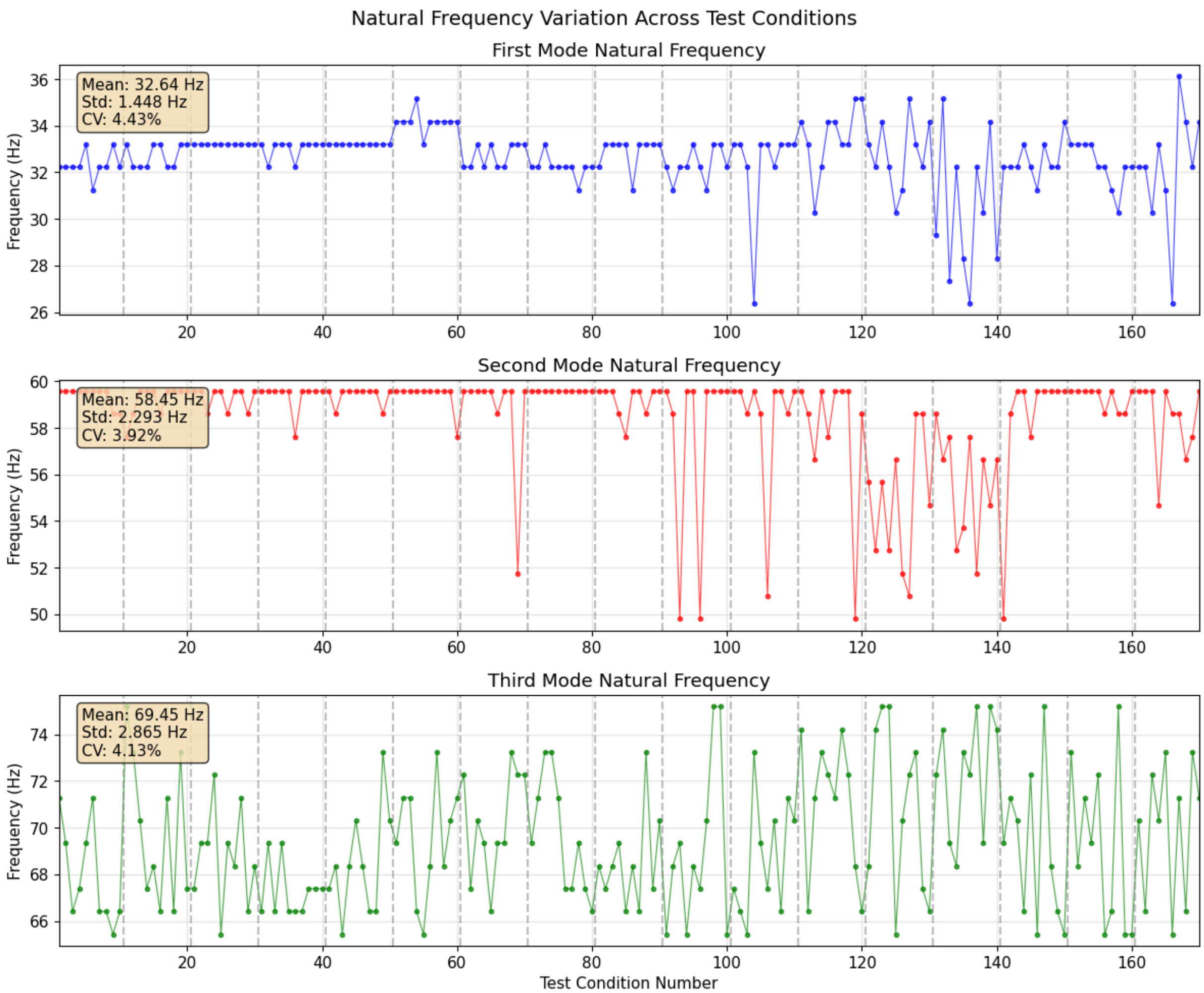
Analyze Natural Frequency Trends

Plot natural frequencies across all test conditions to observe patterns and changes.

In [7]:

```
# Plot natural frequency trends
fig, axes = plt.subplots(3, 1, figsize=(12, 10))

condition_numbers = np.arange(1, natural_frequencies.shape[1] + 1)
mode_names = ['First Mode', 'Second Mode', 'Third Mode']
colors = ['blue', 'red', 'green']

for mode_idx in range(3):
    freqs = natural_frequencies[mode_idx, :]

    axes[mode_idx].plot(condition_numbers, freqs, 'o-', color=colors[mode_ic
                        markersize=3, linewidth=0.8, alpha=0.7)

    axes[mode_idx].set_title(f'{mode_names[mode_idx]} Natural Frequency')
    axes[mode_idx].set_ylabel('Frequency (Hz)')
    axes[mode_idx].grid(True, alpha=0.3)
    axes[mode_idx].set_xlim([1, len(condition_numbers)])

    # Add vertical lines to separate structural states (every 10 conditions)
    for state_boundary in range(10, len(condition_numbers), 10):
        axes[mode_idx].axvline(x=state_boundary + 0.5, color='gray', linesty

    # Calculate and display statistics
    mean_freq = np.mean(freqs)
```

```
    std_freq = np.std(freqs)
    cv_freq = (std_freq / mean_freq) * 100

    # Add text with statistics
    axes[mode_idx].text(0.02, 0.95, f'Mean: {mean_freq:.2f} Hz\nStd: {std_fr
                        transform=axes[mode_idx].transAxes, verticalalignment='t
                        bbox=dict(boxstyle='round', facecolor='wheat', alpha=0.8

axes[2].set_xlabel('Test Condition Number')

plt.suptitle('Natural Frequency Variation Across Test Conditions', fontsize=
plt.tight_layout()
plt.show()
```



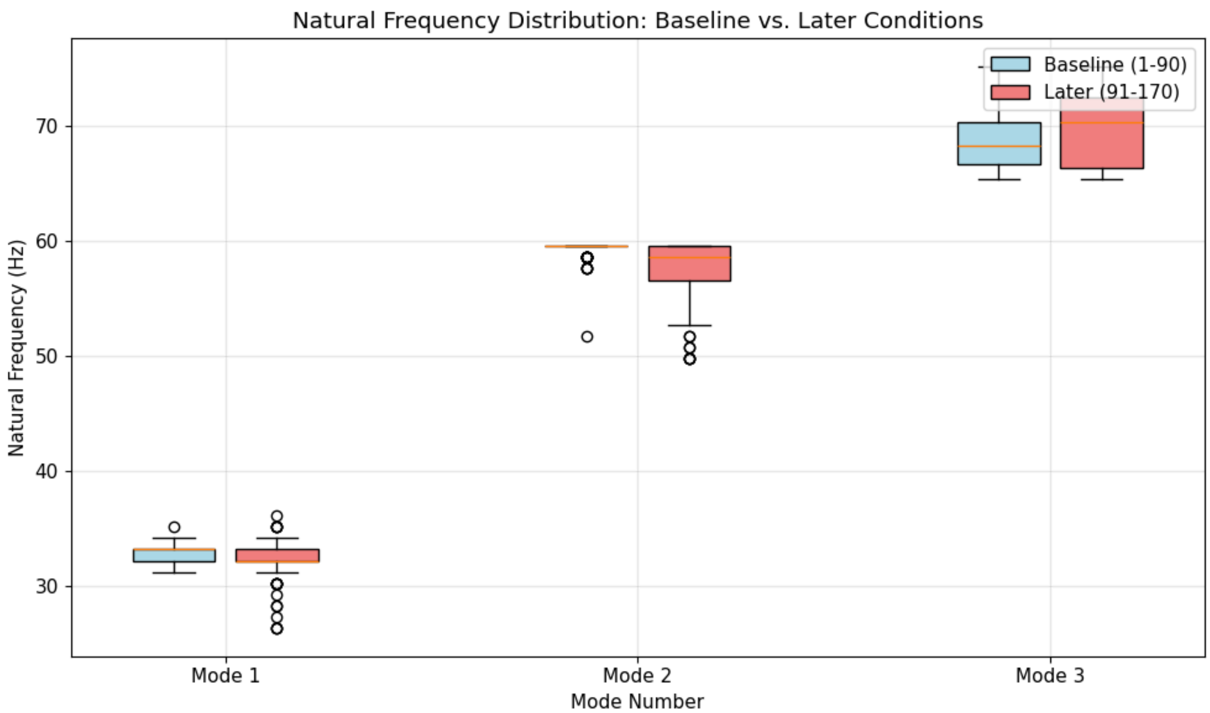
Compare Baseline vs. Later Conditions

Analyze how natural frequencies change between early baseline conditions and later conditions.

In [8]:
```
# Compare baseline (first 90 conditions) vs. later conditions (91–170)
baseline_freqs = natural_frequencies[:, :90]  # First 90 conditions
later_freqs = natural_frequencies[:, 90:]     # Conditions 91–170

print("Natural Frequency Analysis:")
```

```python
print("=" * 40)

for mode_idx in range(3):
    baseline = baseline_freqs[mode_idx, :]
    later = later_freqs[mode_idx, :]

    # Calculate statistics
    baseline_mean = np.mean(baseline)
    later_mean = np.mean(later)
    freq_change = ((later_mean - baseline_mean) / baseline_mean) * 100

    baseline_std = np.std(baseline)
    later_std = np.std(later)

    print(f"Mode {mode_idx + 1} ({mode_names[mode_idx]}):")
    print(f"  Baseline (1-90):    {baseline_mean:.3f} ± {baseline_std:.3f} H
    print(f"  Later (91-170):     {later_mean:.3f} ± {later_std:.3f} Hz")
    print(f"  Mean change:        {freq_change:+.3f}%")
    print()

# Create box plot comparison
fig, ax = plt.subplots(1, 1, figsize=(10, 6))

# Prepare data for box plots
baseline_data = [baseline_freqs[i, :] for i in range(3)]
later_data = [later_freqs[i, :] for i in range(3)]

# Create box plots
positions1 = [1, 3, 5]  # Baseline positions
positions2 = [1.5, 3.5, 5.5]  # Later positions

bp1 = ax.boxplot(baseline_data, positions=positions1, widths=0.4,
                patch_artist=True, boxprops=dict(facecolor='lightblue'))
bp2 = ax.boxplot(later_data, positions=positions2, widths=0.4,
                patch_artist=True, boxprops=dict(facecolor='lightcoral'))

ax.set_xlabel('Mode Number')
ax.set_ylabel('Natural Frequency (Hz)')
ax.set_title('Natural Frequency Distribution: Baseline vs. Later Conditions'
ax.set_xticks([1.25, 3.25, 5.25])
ax.set_xticklabels(['Mode 1', 'Mode 2', 'Mode 3'])
ax.grid(True, alpha=0.3)

# Add legend
ax.legend([bp1['boxes'][0], bp2['boxes'][0]], ['Baseline (1-90)', 'Later (91

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

```
Natural Frequency Analysis:
========================================
Mode 1 (First Mode):
  Baseline (1-90):    32.943 ± 0.695 Hz
  Later (91-170):     32.300 ± 1.921 Hz
  Mean change:        -1.952%

Mode 2 (Second Mode):
  Baseline (1-90):    59.266 ± 0.939 Hz
  Later (91-170):     57.532 ± 2.930 Hz
  Mean change:        -2.927%

Mode 3 (Third Mode):
  Baseline (1-90):    68.924 ± 2.365 Hz
  Later (91-170):     70.032 ± 3.240 Hz
  Mean change:        +1.608%
```



Conclusions and Summary

This analysis demonstrates the fundamental principles of modal analysis for structural health monitoring.

In [9]:
```
# Summary analysis
print("MODAL ANALYSIS SUMMARY")
print("=" * 30)

print("✓ Successfully computed FRFs using Welch's method")
print("✓ Extracted natural frequencies from all test conditions")
print("✓ Analyzed frequency variations across structural states")
print("✓ Compared baseline vs. potentially damaged conditions")
```

```python
print("Key Observations:")

# Calculate overall frequency stability
overall_cv = []
for mode_idx in range(3):
    freqs = natural_frequencies[mode_idx, :]
    cv = (np.std(freqs) / np.mean(freqs)) * 100
    overall_cv.append(cv)
    print(f"  Mode {mode_idx + 1}: {cv:.2f}% coefficient of variation")

avg_cv = np.mean(overall_cv)
print(f"\nAverage coefficient of variation: {avg_cv:.2f}%")

if avg_cv < 1.0:
    print("→ Natural frequencies show good stability across conditions")
elif avg_cv < 2.0:
    print("→ Natural frequencies show moderate variation")
else:
    print("→ Natural frequencies show significant variation")

print("\nModal Analysis Applications:")
print("• Structural health monitoring and damage detection")
print("• Baseline establishment for condition assessment")
print("• Model validation and updating")
print("• Environmental and operational variability studies")

print("Implementation Notes:")
print("• This example demonstrates simplified peak detection")
print("• Advanced applications may use sophisticated modal parameter extract
print("• Consider environmental compensation for robust monitoring")
print("• Combine frequency data with damping and mode shapes for enhanced se
```

```
MODAL ANALYSIS SUMMARY
==============================
✓ Successfully computed FRFs using Welch's method
✓ Extracted natural frequencies from all test conditions
✓ Analyzed frequency variations across structural states
✓ Compared baseline vs. potentially damaged conditions
Key Observations:
  Mode 1: 4.43% coefficient of variation
  Mode 2: 3.92% coefficient of variation
  Mode 3: 4.13% coefficient of variation

Average coefficient of variation: 4.16%
→ Natural frequencies show significant variation

Modal Analysis Applications:
• Structural health monitoring and damage detection
• Baseline establishment for condition assessment
• Model validation and updating
• Environmental and operational variability studies
Implementation Notes:
• This example demonstrates simplified peak detection
• Advanced applications may use sophisticated modal parameter extraction
• Consider environmental compensation for robust monitoring
• Combine frequency data with damping and mode shapes for enhanced sensitivi
ty
```

References

1. 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.

2. Richardson, M.H. & Formenti, D.L., "Parameter Estimation from Frequency Response Measurements using Rational Fraction Polynomials", Proceedings of the 1st International Modal Analysis Conference, Orlando, Florida, November 8-10, 1982.

3. Sohn, H., Worden, K., & Farrar, C. R. (2002). Statistical Damage Classification under Changing Environmental and Operational Conditions. Journal of Intelligent Material Systems and Structures, 13(9), 561-574.