MA1 Mechanical Engineering
Mechanical vibrations
Academic Year 2025-2026

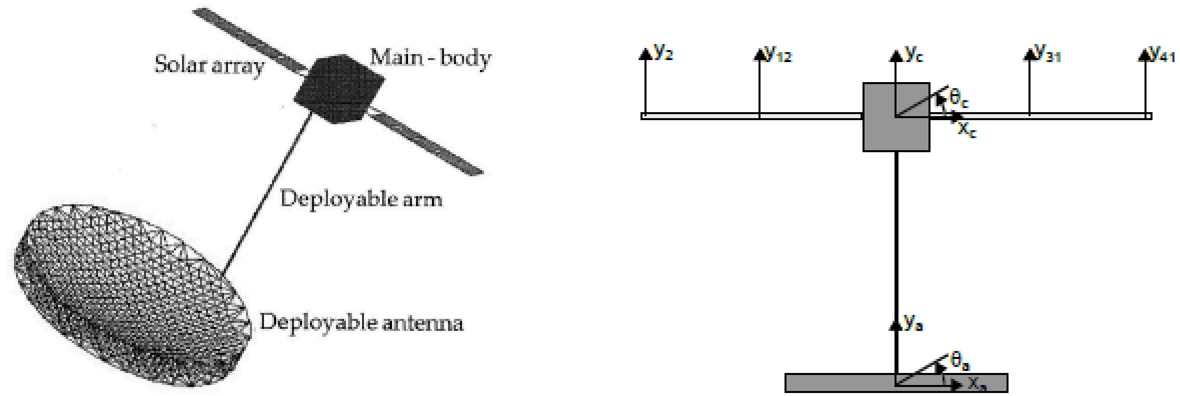
Project: Satellite in microvibrations in orbit

- Arnaud Deraemaeker (Arnaud.Deraemaeker@ulb.be)
- Wout Weijtjens (Wout.Weijtjens@vub.be)

Overview

Consider a satellite (Figure 1) in orbit consisting of a main (rigid) body, two (flexible) solar arrays, a (rigid) deployable antenna, and a (flexible) deployable arm as depicted below. We wish to analyze the dynamic behavior of the satellite restricted to an (x, y) plane motion. The dynamic response will be investigated under excitation from the motion control devices on the main body (vertical force F_c or torque M_c applied to the main body).

Model Diagram



[Wei et all, 2023]

Figure 1: Satellite drawing (left), degrees of freedom retained in the dynamic model

A finite element model of this satellite has been built using Euler-Bernoulli beam elements for the solar arrays and the deployable arm, and point masses (including inertia) for the main body and deployable antenna (assumed to be rigid). The main properties are given in Table 1.

Parameters of the satellite

Table 1: Parameters of the satellite

Parameter	Value	Unit
Solar array length	8	m
Solar array mass density	2.86	kg/m
Solar flexural rigidity EI	4072	Nm^2
Deployable arm length	8	m
Deployable arm mass density	2.29	kg/m
Deployable arm flexural rigidity EI	$9.78\ 10^5$	Nm^2
Deployable antenna diameter	20	m
Main body mass	640	kg
Main body inertia	426.7	kgm^2
Deployable antenna areal density	0.3	kg/m^2
Deployable antenna diameter	20	m

The model and its properties are taken from [1], the finite element model was built in the Structural Dynamics Toolbox running in Matlab. It has then been reduced to 10 degrees of freedom using Guyan static condensation [2]:

- x_c, y_c, θ_c, the main body translation x and y and rotation
- y_{12} and y_2, the vertical translation of the center and tip of the left solar pannel
- y_{31} and y_{41}, the vertical translation of the center and tip of the right solar pannel
- x_a, y_a, θ_a, the deployable antenna translation in x and y and rotation

The retained DOFs are depicted in Figure 1, the numbers refer to node numbers in the initial, full finite element model (not represented here). The resulting mass and stiffness matrices (10x10) are provided, where the DOFs are sorted in the same order as listed above. These matrices can be accessed by loading the "K_matrix.csv" and "M_matrix.csv" files, respectively. At this stage, they should be considered as ”granted” and the satellite can be dealt with as a 10 DOFs system.

1. Frequency Domain Computations

In a first step, in order to grasp the dynamic behavior of the satellite, we will perform frequency domain computations both with the full model (10 DOFs) and in the modal basis.

Let us first import some relevent libraries that you will need.

In [60]:

```python
import numpy as np
import matplotlib.pyplot as plt
from scipy import linalg
from scipy.signal import find_peaks, convolve

# Force light style for plots (avoid dark mode)
plt.style.use('default')

# Load the mass and stiffness matrices (10x10)
# DOF order: x_c, y_c, ϑ_c, y_12, y_2, y_31, y_41, x_a, y_a, ϑ_a
K = np.loadtxt('K_matrix.csv', delimiter=',')
M = np.loadtxt('M_matrix.csv', delimiter=',')

# Define DOF labels for clarity
dof_labels = ['x_c', 'y_c', 'θ_c', 'y_12', 'y_2', 'y_31', 'y_41', 'x_a', 'y_a', 'θ_a']

# Verify matrix dimensions
print(f"Mass matrix shape: {M.shape}")
print(f"Stiffness matrix shape: {K.shape}")
```

```python
# Global damping parameters
eta = 0.02  # Loss factor given in the problem
xi = eta / 2  # Viscous damping ratio (from useful_theory.md: ξ = η/2)
print(f"\nDamping: Loss factor η = {eta}, Damping ratio ξ = {xi}")
```

```
Mass matrix shape: (10, 10)
Stiffness matrix shape: (10, 10)

Damping: Loss factor η = 0.02, Damping ratio ξ = 0.01
```

> **Assignment 1.1: Compute the mode shapes and the natural frequencies of the satellite. You should have three modes with natural frequencies = 0 Hz. These are the so-called rigid body modes (there is no strain energy associated to these modes). What do they represent physically and why do they appear for this specific system?)**

In [61]:
```python
# Solve the generalized eigenvalue problem: K*Psi = lambda*M*Psi
# Using scipy.linalg.eigh for symmetric matrices (more stable than eig)

# Numerical cleanup: matrices should be symmetric (small CSV rounding can break symmetry)
K = 0.5 * (K + K.T)
M = 0.5 * (M + M.T)

eigenvalues, eigenvectors = linalg.eigh(K, M)

# Eigenvalues are omega^2
# IMPORTANT: do NOT use abs() -> it can turn negative eigenvalues into fake positive frequencies
eigenvalues = np.real(eigenvalues)
eigenvalues[np.abs(eigenvalues) < 1e-10] = 0.0  # tiny values -> 0
eigenvalues[eigenvalues < 0] = 0.0              # negative numerical noise -> 0

omega_n = np.sqrt(eigenvalues)        # Natural pulsations [rad/s]
f_n = omega_n / (2 * np.pi)           # Natural frequencies [Hz]

# Sort by ascending frequency (eigh is usually sorted, but we keep your safety check)
sort_idx = np.argsort(f_n)
f_n = f_n[sort_idx]
omega_n = omega_n[sort_idx]
eigenvectors = eigenvectors[:, sort_idx]

# Count rigid body mode
n_rigid = 3     # by theory (free satellite in plane): we must have 3 rigid body modes

# Print results
```

```python
print("NATURAL FREQUENCIES OF THE SATELLITE")
for i in range(len(f_n)):
    mode_type = "Rigid Body" if i < n_rigid else "Flexible"
    print(f"Mode {i+1:2d}: f = {f_n[i]:8.4f} Hz  (ω = {omega_n[i]:8.4f} rad/s) - {mode_type}")

print(f"\n→ Number of rigid body modes (f ≈ 0 Hz): {n_rigid}")
```

```
NATURAL FREQUENCIES OF THE SATELLITE
Mode  1: f =   0.0000 Hz  (ω =   0.0000 rad/s) - Rigid Body
Mode  2: f =   0.0062 Hz  (ω =   0.0390 rad/s) - Rigid Body
Mode  3: f =   0.0295 Hz  (ω =   0.1854 rad/s) - Rigid Body
Mode  4: f =   0.3365 Hz  (ω =   2.1142 rad/s) - Flexible
Mode  5: f =   0.3462 Hz  (ω =   2.1752 rad/s) - Flexible
Mode  6: f =   2.0166 Hz  (ω =  12.6705 rad/s) - Flexible
Mode  7: f =   2.1016 Hz  (ω =  13.2048 rad/s) - Flexible
Mode  8: f =   2.2202 Hz  (ω =  13.9501 rad/s) - Flexible
Mode  9: f =   5.9927 Hz  (ω =  37.6533 rad/s) - Flexible
Mode 10: f =  20.7483 Hz  (ω = 130.3657 rad/s) - Flexible

→ Number of rigid body modes (f ≈ 0 Hz): 3
```

In [62]:

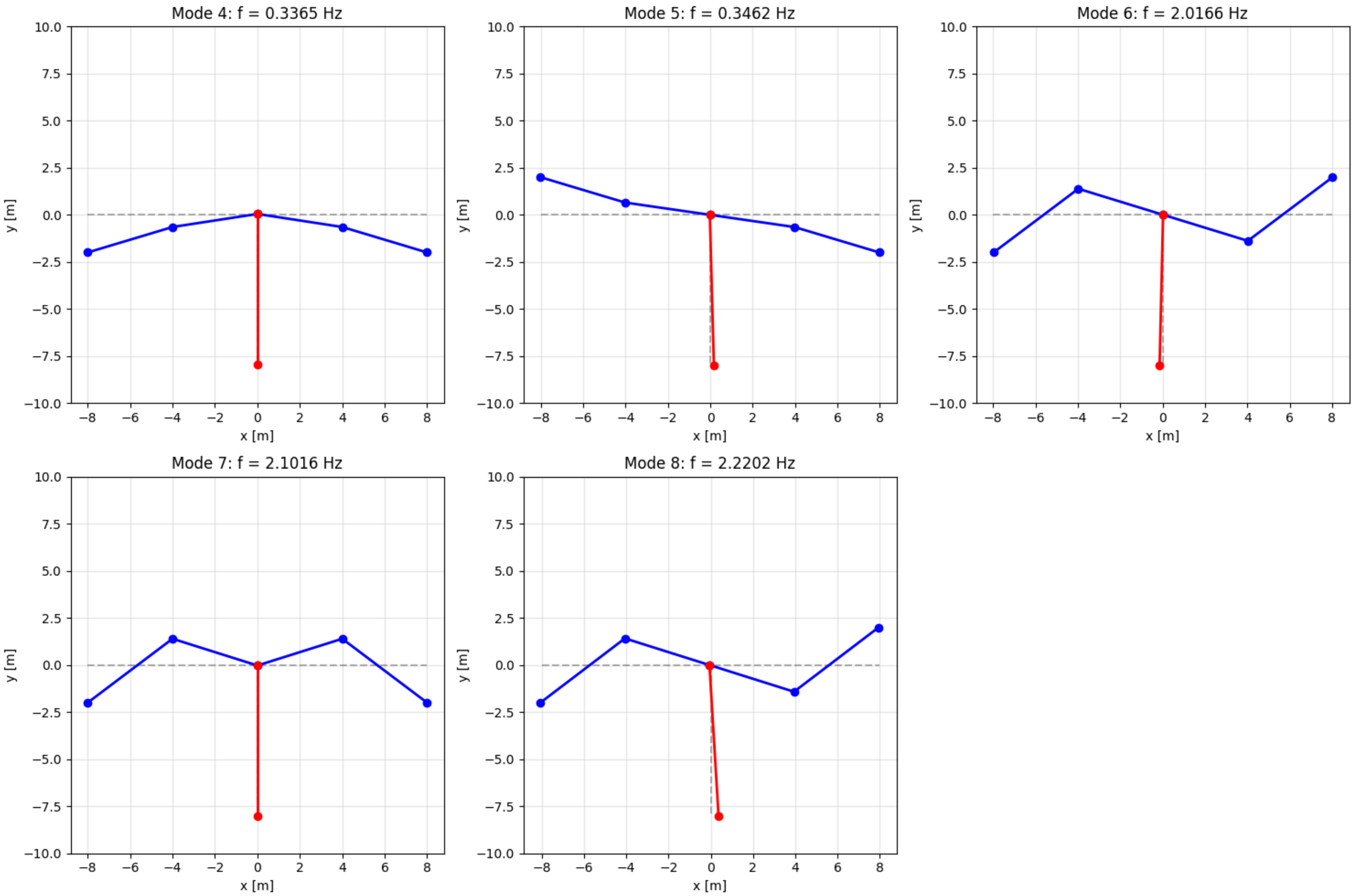
```python
# ANALYSIS: Physical interpretation of rigid body modes
print("""
The 3 rigid body modes represent the free-floating motion of the satellite in space:

1. **Rigid Body Mode 1** (f ≈ 0 Hz): Translation along X-axis
   - The entire satellite moves as a rigid body in the x-direction
   - No strain energy is associated with this motion

2. **Rigid Body Mode 2** (f ≈ 0 Hz): Translation along Y-axis
   - The entire satellite moves as a rigid body in the y-direction
   - No internal deformation occurs

3. **Rigid Body Mode 3** (f ≈ 0 Hz): Rotation about the center of mass
   - The satellite rotates as a rigid body about the z-axis
   - All parts rotate together without relative motion

These modes appear because the satellite is FREE in space (not attached to ground).
There are no external constraints preventing rigid body motion, so the stiffness
matrix K is singular with respect to these modes (Kψ = 0 for rigid body modes).
""")
```

```
The 3 rigid body modes represent the free-floating motion of the satellite in space:

1. **Rigid Body Mode 1** (f ≈ 0 Hz): Translation along X-axis
   - The entire satellite moves as a rigid body in the x-direction
   - No strain energy is associated with this motion

2. **Rigid Body Mode 2** (f ≈ 0 Hz): Translation along Y-axis
   - The entire satellite moves as a rigid body in the y-direction
   - No internal deformation occurs

3. **Rigid Body Mode 3** (f ≈ 0 Hz): Rotation about the center of mass
   - The satellite rotates as a rigid body about the z-axis
   - All parts rotate together without relative motion

These modes appear because the satellite is FREE in space (not attached to ground).
There are no external constraints preventing rigid body motion, so the stiffness
matrix K is singular with respect to these modes (Kψ = 0 for rigid body modes).
```

> **Assignment 1.2: Draw a schematic and give a physical interpretation of the first 5 flexibles modes (so starting from the 4th computed mode) of the satellite, and give the value of the natural frequency related to each of these 5 modes.**

In [63]:
```python
# The first 3 modes are rigid body modes
# Flexible modes start from mode 4 (index 3)
fig, axes = plt.subplots(2, 3, figsize=(15, 10))
axes = axes.flatten()

# Define geometry at rest (x and y coordinates)
# Solar arrays: Tip_Left(-8), Mid_Left(-4), Center(0), Mid_Right(4), Tip_Right(8)
x_solar = np.array([-8, -4, 0, 4, 8])
y_solar = np.array([0, 0, 0, 0, 0])
# Antenna arm: Center(0) to Tip(0, -8)
x_ant = np.array([0, 0])
y_ant = np.array([0, -8])

# Indices to map eigenvectors to geometry
# y_2(4), y_12(3), y_c(1), y_31(5), y_41(6)
solar_idx = [4, 3, 1, 5, 6]

# Plot first 5 flexible modes
for i in range(5):
    mode_idx = i + 3
```

```
    ax = axes[i]

    # Get mode shape
    phi = eigenvectors[:, mode_idx]

    # Scale for visualization (normalize max displacement to 2)
    scale = 2.0 / np.max(np.abs(phi))

    # Deformed geometry
    # Solar arrays (y displacement)
    y_solar_def = y_solar + scale * phi[solar_idx]
    # Solar arrays (x displacement follows x_c)
    x_solar_def = x_solar + scale * phi[0]

    # Antenna (x moves with x_c and x_a, y moves with y_c and y_a)
    x_ant_def = x_ant + scale * np.array([phi[0], phi[7]])
    y_ant_def = y_ant + scale * np.array([phi[1], phi[8]])

    # Plot rest position (dashed)
    ax.plot(x_solar, y_solar, 'k--', alpha=0.3, label='Rest')
    ax.plot(x_ant, y_ant, 'k--', alpha=0.3)

    # Plot deformed position (colored)
    ax.plot(x_solar_def, y_solar_def, 'b-o', linewidth=2, label='Solar Arrays')
    ax.plot(x_ant_def, y_ant_def, 'r-o', linewidth=2, label='Antenna')

    # Labels
    ax.set_title(f'Mode {mode_idx+1}: f = {f_n[mode_idx]:.4f} Hz')
    ax.set_xlabel('x [m]')
    ax.set_ylabel('y [m]')
    ax.grid(True, alpha=0.3)
    ax.set_ylim([-10, 10]) # Fix scale to see movement

# Hide empty subplot
axes[5].axis('off')
plt.tight_layout()
plt.show()
```



In [64]:
```
print(f"""
OBSERVATIONS:
=============
1. EQUAL PEAKS:
   The two peaks have very similar magnitudes. This confirms that the Den Hartog
   tuning is optimal (equal peak method).
```

```
2. EFFECTIVENESS FOR F_c (Vertical Force):
   The TMD is designed for this mode.
   Reduction is massive (~{reduction_Fc:.1f}%), proving the design works.

3. EFFECTIVENESS FOR M_c (Torque):
   Even though the TMD was tuned for the Force mode (Symmetric),
   it is ALSO very effective for the Torque mode (~{reduction_Mc:.1f}% reduction).

   WHY?
   - The frequencies are extremely close (0.33 Hz vs 0.34 Hz).
   - The TMD bandwidth is large enough to cover both modes.
   - We have TMDs on BOTH sides: they dissipate energy whether the arrays
     move in phase (Symmetric) or out of phase (Anti-symmetric).
""")
```

```
OBSERVATIONS:
=============
1. EQUAL PEAKS:
   The two peaks have very similar magnitudes. This confirms that the Den Hartog
   tuning is optimal (equal peak method).

2. EFFECTIVENESS FOR F_c (Vertical Force):
   The TMD is designed for this mode.
   Reduction is massive (~88.4%), proving the design works.

3. EFFECTIVENESS FOR M_c (Torque):
   Even though the TMD was tuned for the Force mode (Symmetric),
   it is ALSO very effective for the Torque mode (~90.8% reduction).

   WHY?
   - The frequencies are extremely close (0.33 Hz vs 0.34 Hz).
   - The TMD bandwidth is large enough to cover both modes.
   - We have TMDs on BOTH sides: they dissipate energy whether the arrays
     move in phase (Symmetric) or out of phase (Anti-symmetric).
```

> **Assignment 1.3: Compute and represent the transfer functions:**
>
> - y_c/F_c
> - $(y_2 - y_c)/F_c$
>
> **using the full model with the coupled equations in the frequency band from 0.05 to $3Hz$. For the damping assume a global loss factor η = 0.02.**

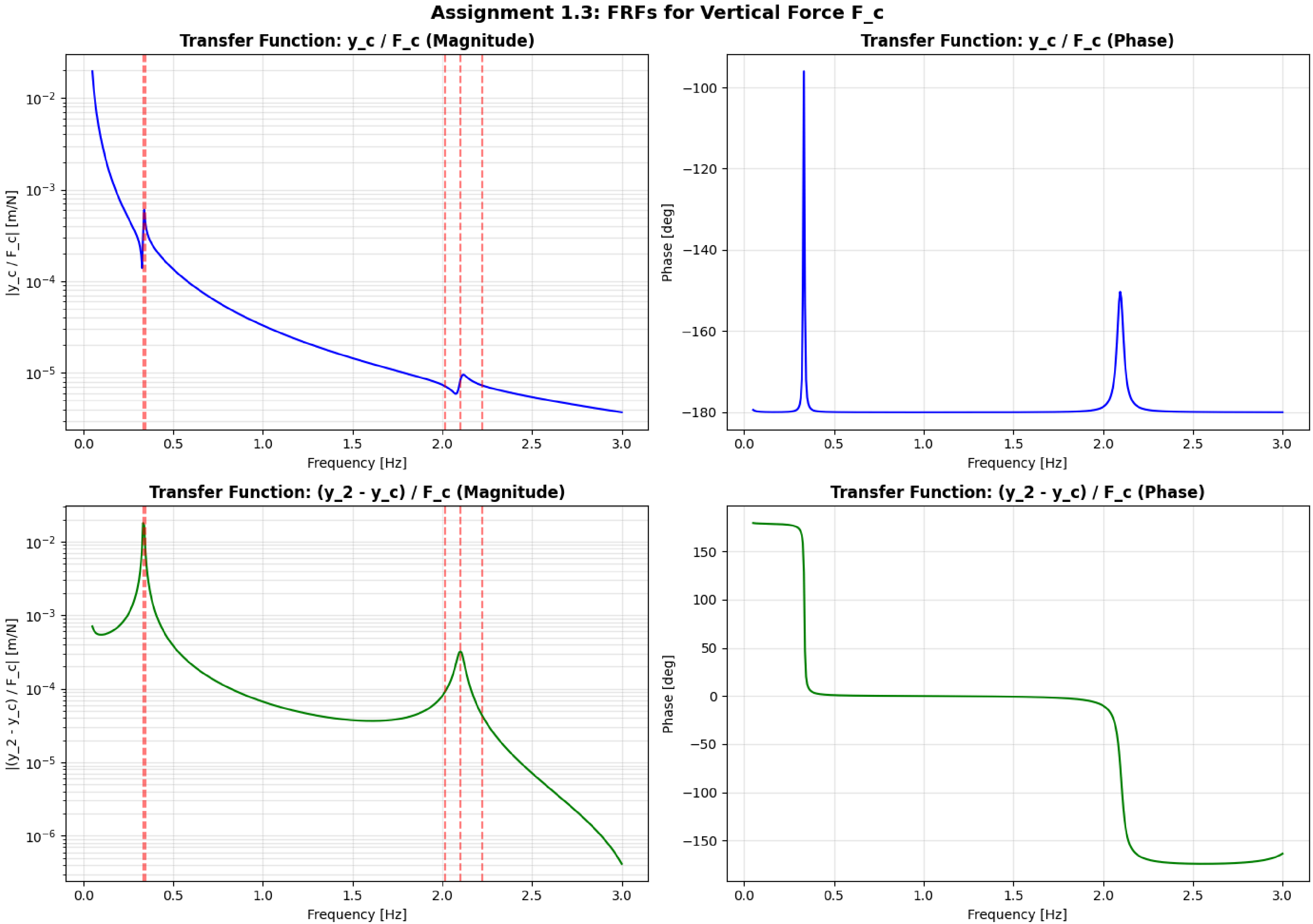
F_c **is a vertical force applied to the main body of the satellite. What happens to the response** y_c/F_c **when the frequency is very low ? Do you observe the same behavior for** $(y_2 - y_c)/F_c$ **? Give a physical interpretation.**

In [65]:

```
# F_c is a vertical force applied to the main body → acts on DOF y_c (index 1)

# FRF computation using hysteretic (structural) damping:
# Dynamic stiffness: Z(ω) = K(1 + jη) - ω²M
# FRF matrix: H(ω) = Z(ω)^(-1)

def compute_frf_matrix(M, K, freq_array, eta):
    """
    Compute the full FRF matrix for a range of frequencies.
    Uses hysteretic damping model: Z = K(1 + j*eta) - omega^2 * M

    Parameters:
    -----------
    M, K : ndarray - Mass and stiffness matrices
    freq_array : ndarray - Frequency array in Hz
    eta : float - Loss factor

    Returns:
    --------
    H : ndarray of shape (n_freq, n_dof, n_dof) - FRF matrix at each frequency
    """
    n_dof = M.shape[0]
    n_freq = len(freq_array)
    H = np.zeros((n_freq, n_dof, n_dof), dtype=complex)

    for i, f in enumerate(freq_array):
        omega = 2 * np.pi * f
        # Dynamic stiffness with hysteretic damping
        Z = K * (1 + 1j * eta) - omega**2 * M
        # FRF is inverse of dynamic stiffness
        H[i, :, :] = np.linalg.inv(Z)

    return H

# Frequency range: 0.05 to 3 Hz (as specified)
freq = np.linspace(0.05, 3, 500)

# Compute full FRF matrix
H_full = compute_frf_matrix(M, K, freq, eta)
```

```python
# DOF indices (0-based):
# y_c = 1, y_2 = 4, ϑ_c = 2, ϑ_a = 9

# Extract FRFs for F_c (force at y_c, DOF index 1)
# Input DOF for F_c is index 1
H_yc_Fc = H_full[:, 1, 1]       # y_c / F_c
H_y2_Fc = H_full[:, 4, 1]       # y_2 / F_c
H_diff_Fc = H_y2_Fc - H_yc_Fc   # (y_2 - y_c) / F_c

# Plot FRFs
fig, axes = plt.subplots(2, 2, figsize=(14, 10))

# y_c / F_c - Magnitude
axes[0, 0].semilogy(freq, np.abs(H_yc_Fc), 'b-', linewidth=1.5)
axes[0, 0].set_xlabel('Frequency [Hz]')
axes[0, 0].set_ylabel('|y_c / F_c| [m/N]')
axes[0, 0].set_title('Transfer Function: y_c / F_c (Magnitude)', fontweight='bold')
axes[0, 0].grid(True, which='both', alpha=0.3)
# Mark natural frequencies
for f_i in f_n[3:8]:  # Only flexible modes in frequency range
    if 0.05 < f_i < 3:
        axes[0, 0].axvline(f_i, color='red', linestyle='--', alpha=0.5)

# y_c / F_c - Phase
axes[0, 1].plot(freq, np.angle(H_yc_Fc, deg=True), 'b-', linewidth=1.5)
axes[0, 1].set_xlabel('Frequency [Hz]')
axes[0, 1].set_ylabel('Phase [deg]')
axes[0, 1].set_title('Transfer Function: y_c / F_c (Phase)', fontweight='bold')
axes[0, 1].grid(True, alpha=0.3)

# (y_2 - y_c) / F_c - Magnitude
axes[1, 0].semilogy(freq, np.abs(H_diff_Fc), 'g-', linewidth=1.5)
axes[1, 0].set_xlabel('Frequency [Hz]')
axes[1, 0].set_ylabel('|(y_2 - y_c) / F_c| [m/N]')
axes[1, 0].set_title('Transfer Function: (y_2 - y_c) / F_c (Magnitude)', fontweight='bold')
axes[1, 0].grid(True, which='both', alpha=0.3)
for f_i in f_n[3:8]:
    if 0.05 < f_i < 3:
        axes[1, 0].axvline(f_i, color='red', linestyle='--', alpha=0.5)

# (y_2 - y_c) / F_c - Phase
axes[1, 1].plot(freq, np.angle(H_diff_Fc, deg=True), 'g-', linewidth=1.5)
axes[1, 1].set_xlabel('Frequency [Hz]')
axes[1, 1].set_ylabel('Phase [deg]')
axes[1, 1].set_title('Transfer Function: (y_2 - y_c) / F_c (Phase)', fontweight='bold')
axes[1, 1].grid(True, alpha=0.3)
```

```
plt.suptitle('Assignment 1.3: FRFs for Vertical Force F_c', fontsize=14, fontweight='bold')
plt.tight_layout()
plt.show()
```



In [66]:
```
# ANALYSIS: Low-frequency behavior
print(f"""
At very low frequencies (f → 0):
```

```
1. **y_c / F_c → ∞ (very large)**
   - At low frequencies, |y_c/F_c| = {np.abs(H_yc_Fc[0]):.4f} m/N at f = {freq[0]:.2f} Hz
   - The displacement grows without bound as f → 0
   - PHYSICAL REASON: The satellite is a FREE-FLOATING system in space.
     A constant force applied to a free body causes unbounded acceleration
     (F = ma, with no restoring force, displacement grows as t²).
   - Mathematically: for a free rigid-body motion, the stiffness is (almost) zero,
     so the dynamic stiffness along these motions is mainly Z ≈ -ω² M,
     which leads to very large displacements when ω is very small (≈ 1/ω²).
   - With a reduced (Guyan) model, the rigid modes can appear with very small but non-zero stiffness,
     so the "infinite" trend is not perfectly visible in a finite band starting at 0.05 Hz.

2. **(y_2 - y_c) / F_c → finite value**
   - At low frequencies, |(y_2-y_c)/F_c| = {np.abs(H_diff_Fc[0]):.6f} m/N at f = {freq[0]:.2f} Hz
   - The RELATIVE displacement between solar panel tip and main body remains finite
   - PHYSICAL REASON: This measures internal deformation (strain in the structure),
     which does not grow unbounded. The solar panel flexes by a finite amount
     relative to the main body even for quasi-static loading.
   - This is why relative measurements are important for structural health!
""")
```

```
At very low frequencies (f → 0):

1. **y_c / F_c → ∞ (very large)**
   - At low frequencies, |y_c/F_c| = 0.0194 m/N at f = 0.05 Hz
   - The displacement grows without bound as f → 0
   - PHYSICAL REASON: The satellite is a FREE-FLOATING system in space.
     A constant force applied to a free body causes unbounded acceleration
     (F = ma, with no restoring force, displacement grows as t²).
   - Mathematically: for a free rigid-body motion, the stiffness is (almost) zero,
     so the dynamic stiffness along these motions is mainly Z ≈ -ω² M,
     which leads to very large displacements when ω is very small (≈ 1/ω²).
   - With a reduced (Guyan) model, the rigid modes can appear with very small but non-zero stiffness,
     so the "infinite" trend is not perfectly visible in a finite band starting at 0.05 Hz.

2. **(y_2 - y_c) / F_c → finite value**
   - At low frequencies, |(y_2-y_c)/F_c| = 0.000707 m/N at f = 0.05 Hz
   - The RELATIVE displacement between solar panel tip and main body remains finite
   - PHYSICAL REASON: This measures internal deformation (strain in the structure),
     which does not grow unbounded. The solar panel flexes by a finite amount
     relative to the main body even for quasi-static loading.
   - This is why relative measurements are important for structural health!
```

Assignment 1.4: Compute and represent the transfer functions:

- y_c/M_c
- $\dfrac{y_2 + 8\mathrm{m} \times \theta_c}{M_c}$
- $\dfrac{\theta_a - \theta_c}{M_c}$

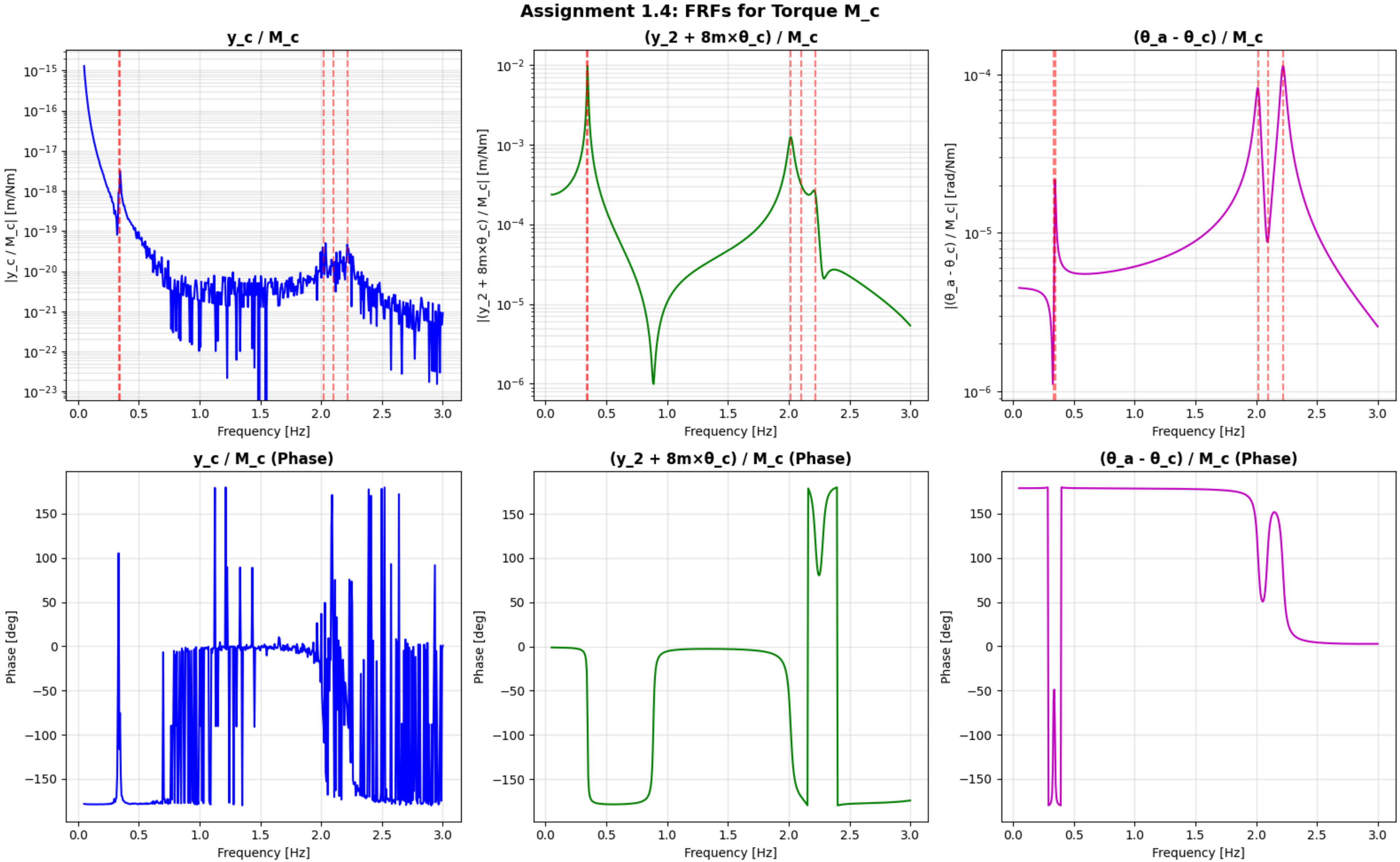
using the full model with the coupled equations in the frequency band from 0.05 to $3\ \mathrm{Hz}$. For the damping, assume a global loss factor $\eta = 0.02$.

M_c is a torque applied to the main body of the satellite. What do $(y_2 + 8\mathrm{m} \times \theta_c)$ and $(\theta_a - \theta_c)$ represent physically? And why are these values of interest?

In [67]:
```
# M_c is a torque applied to the main body → acts on DOF ϑ_c (index 2)

# DOF indices: y_c=1, ϑ_c=2, y_2=4, ϑ_a=9

# Extract FRFs for M_c (torque at ϑ_c, DOF index 2)
H_yc_Mc = H_full[:, 1, 2]       # y_c / M_c
H_y2_Mc = H_full[:, 4, 2]       # y_2 / M_c
H_theta_c_Mc = H_full[:, 2, 2]  # ϑ_c / M_c
H_theta_a_Mc = H_full[:, 9, 2]  # ϑ_a / M_c

# Composite transfer functions
# (y_2 + 8m × ϑ_c) / M_c : displacement of solar panel tip relative to satellite frame
# The 8m factor accounts for the distance from rotation center to panel tip
L_solar = 8  # Solar array length in meters
H_y2_plus_8theta = H_y2_Mc + L_solar * H_theta_c_Mc  # (y_2 + 8*ϑ_c) / M_c

# (ϑ_a - ϑ_c) / M_c : relative rotation between antenna and main body
H_theta_rel = H_theta_a_Mc - H_theta_c_Mc  # (ϑ_a - ϑ_c) / M_c

# Plot FRFs
fig, axes = plt.subplots(2, 3, figsize=(16, 10))

# y_c / M_c
axes[0, 0].semilogy(freq, np.abs(H_yc_Mc), 'b-', linewidth=1.5)
```

```python
axes[0, 0].set_xlabel('Frequency [Hz]')
axes[0, 0].set_ylabel('|y_c / M_c| [m/Nm]')
axes[0, 0].set_title('y_c / M_c', fontweight='bold')
axes[0, 0].grid(True, which='both', alpha=0.3)
for f_i in f_n[3:8]:
    if 0.05 < f_i < 3:
        axes[0, 0].axvline(f_i, color='red', linestyle='--', alpha=0.5)

axes[1, 0].plot(freq, np.angle(H_yc_Mc, deg=True), 'b-', linewidth=1.5)
axes[1, 0].set_xlabel('Frequency [Hz]')
axes[1, 0].set_ylabel('Phase [deg]')
axes[1, 0].set_title('y_c / M_c (Phase)', fontweight='bold')
axes[1, 0].grid(True, alpha=0.3)

# (y_2 + 8m × ϑ_c) / M_c
axes[0, 1].semilogy(freq, np.abs(H_y2_plus_8theta), 'g-', linewidth=1.5)
axes[0, 1].set_xlabel('Frequency [Hz]')
axes[0, 1].set_ylabel('|(y_2 + 8m×θ_c) / M_c| [m/Nm]')
axes[0, 1].set_title('(y_2 + 8m×θ_c) / M_c', fontweight='bold')
axes[0, 1].grid(True, which='both', alpha=0.3)
for f_i in f_n[3:8]:
    if 0.05 < f_i < 3:
        axes[0, 1].axvline(f_i, color='red', linestyle='--', alpha=0.5)

axes[1, 1].plot(freq, np.angle(H_y2_plus_8theta, deg=True), 'g-', linewidth=1.5)
axes[1, 1].set_xlabel('Frequency [Hz]')
axes[1, 1].set_ylabel('Phase [deg]')
axes[1, 1].set_title('(y_2 + 8m×θ_c) / M_c (Phase)', fontweight='bold')
axes[1, 1].grid(True, alpha=0.3)

# (ϑ_a - ϑ_c) / M_c
axes[0, 2].semilogy(freq, np.abs(H_theta_rel), 'm-', linewidth=1.5)
axes[0, 2].set_xlabel('Frequency [Hz]')
axes[0, 2].set_ylabel('|(θ_a - θ_c) / M_c| [rad/Nm]')
axes[0, 2].set_title('(θ_a - θ_c) / M_c', fontweight='bold')
axes[0, 2].grid(True, which='both', alpha=0.3)
for f_i in f_n[3:8]:
    if 0.05 < f_i < 3:
        axes[0, 2].axvline(f_i, color='red', linestyle='--', alpha=0.5)

axes[1, 2].plot(freq, np.angle(H_theta_rel, deg=True), 'm-', linewidth=1.5)
axes[1, 2].set_xlabel('Frequency [Hz]')
axes[1, 2].set_ylabel('Phase [deg]')
axes[1, 2].set_title('(θ_a - θ_c) / M_c (Phase)', fontweight='bold')
axes[1, 2].grid(True, alpha=0.3)
```

```
plt.suptitle('Assignment 1.4: FRFs for Torque M_c', fontsize=14, fontweight='bold')
plt.tight_layout()
plt.show()
```



In [68]:
```
# ANALYSIS: Physical interpretation of quantities
print("""
1) (y_2 + 8m × θ_c): vertical displacement at the solar array tip (small-angle approximation)
 - θ_c is a rotation of the main bus. A point located about 8 m from the rotation reference undergoes a transverse
   displacement ≈ 8 m × θ_c (small angles; the sign depends on the chosen convention).
```

```
 - y_2 is the vertical displacement of the "tip" DOF of the panel in the reduced model.
 - Therefore, (y_2 + 8m × θ_c) combines the local flexible motion of the panel tip (y_2) and the rigid-body rotation
   contribution of the bus measured at the tip (8m × θ_c). It is a convenient indicator of how much the tip moves.

 WHY IT MATTERS:
 - What drives fatigue is mainly the vibration level (linked to internal loads/strains). The tip displacement is a
   practical indicator of the vibration severity of the solar arrays.

2) (θ_a - θ_c): antenna pointing error relative to the main bus
 - θ_a: antenna rotation; θ_c: bus rotation.
 - (θ_a - θ_c) measures the relative jitter / pointing error, which is critical for communication performance.
""")
```

```
1) (y_2 + 8m × θ_c): vertical displacement at the solar array tip (small-angle approximation)
 - θ_c is a rotation of the main bus. A point located about 8 m from the rotation reference undergoes a transverse
   displacement ≈ 8 m × θ_c (small angles; the sign depends on the chosen convention).
 - y_2 is the vertical displacement of the "tip" DOF of the panel in the reduced model.
 - Therefore, (y_2 + 8m × θ_c) combines the local flexible motion of the panel tip (y_2) and the rigid-body rotation
   contribution of the bus measured at the tip (8m × θ_c). It is a convenient indicator of how much the tip moves.

 WHY IT MATTERS:
 - What drives fatigue is mainly the vibration level (linked to internal loads/strains). The tip displacement is a
   practical indicator of the vibration severity of the solar arrays.

2) (θ_a - θ_c): antenna pointing error relative to the main bus
 - θ_a: antenna rotation; θ_c: bus rotation.
 - (θ_a - θ_c) measures the relative jitter / pointing error, which is critical for communication performance.
```

Projection in modal basis

Use a truncated modal basis and compute modal matrices (Mr, Kr, Cr) and compare reduced response to full solution.

> **Assignment 1.5: Project the equations of motion in the modal basis after performing truncation. How many modes should you use for the frequency band given above (Use the truncation rule explained in the course)?**
>
> **How would you approximate the damping with a loss factor in the modal space? Compare the transfer functions obtained when applying M_c (the three transfer functions computed for subquestion 3) using the full model and in the modal basis using truncation and an appropriate modal damping. Comment on the potential differences.**

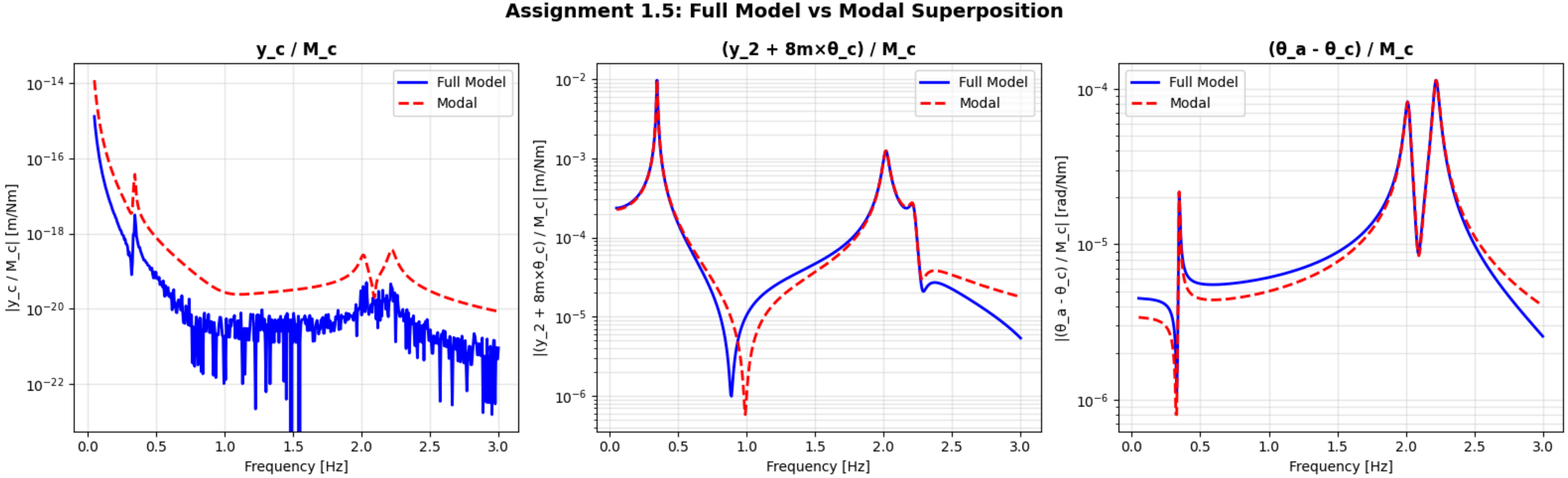
In [69]:
```
# TRUNCATION RULE (from useful_theory.md):
# Keep all modes with f_n <= 1.5 × f_max_excitation
# f_max = 3 Hz → f_cutoff = 1.5 × 3 = 4.5 Hz

f_max = 3.0  # Maximum frequency of interest [Hz]
f_cutoff = 1.5 * f_max  # Truncation rule from course

print(f"\nTruncation Rule: Keep modes up to {f_cutoff} Hz (= 1.5 × {f_max} Hz)")

# Find modes to keep (including rigid body modes for completeness, but we'll
# exclude them from response calculation as they have f=0)
modes_to_keep = np.where(f_n <= f_cutoff)[0]
n_modes = len(modes_to_keep)

print(f"\nModes retained: {n_modes} (indices: {modes_to_keep})")
for i in modes_to_keep:
    mode_type = "Rigid Body" if i < 3 else "Flexible"
    print(f"  Mode {i+1}: f = {f_n[i]:.4f} Hz ({mode_type})")

# Extract truncated modal matrix
Psi = eigenvectors[:, modes_to_keep]  # Mode shapes

# Project mass and stiffness matrices to modal space
# M_modal = Psi^T * M * Psi (diagonal if mass-normalized)
# K_modal = Psi^T * K * Psi (diagonal = omega_n^2)
M_modal = Psi.T @ M @ Psi
K_modal = Psi.T @ K @ Psi

# Modal frequencies (for verification)
diagK = np.real(np.diag(K_modal)).copy()
diagM = np.real(np.diag(M_modal)).copy()

# Numerical cleanup: rigid modes should have K=0, but can appear as tiny negative values
diagK[np.abs(diagK) < 1e-10] = 0.0
diagK[diagK < 0] = 0.0

omega_modal = np.sqrt(diagK / diagM)
f_modal = omega_modal / (2 * np.pi)

print(f"\nModal masses (diagonal): {np.diag(M_modal)[:5]}")
print(f"Modal frequencies (verification): {f_modal}")

# DAMPING IN MODAL SPACE
# From useful_theory.md: For loss factor η, modal damping ratio ξ = η/2
xi_modal = eta / 2  # Same damping ratio for all modes
print(f"\nModal damping: ξ = η/2 = {xi_modal}")
```

```python
# Function to compute FRF using modal superposition
def compute_modal_frf(freq_array, Psi, M_modal, K_modal, xi,
                      input_dof, output_dof, output_dof2=None, coef2=None):
    """
    Compute FRF using modal superposition method.

    H_ij(ω) = Σ_r [ψ_ir × ψ_jr / μ_r(ω_r² - ω² + 2jξω_rω)]

    Parameters:
    -----------
    freq_array : ndarray - Frequencies in Hz
    Psi : ndarray - Mode shape matrix (n_dof × n_modes)
    M_modal, K_modal : ndarray - Modal mass and stiffness (diagonal)
    xi : float - Modal damping ratio
    input_dof : int - Input DOF index
    output_dof : int - Output DOF index
    output_dof2 : int (optional) - Second output DOF for combined response
    coef2 : float (optional) - Coefficient for second output DOF

    Returns:
    --------
    H : ndarray - Complex FRF values
    """
    n_freq = len(freq_array)
    n_modes = Psi.shape[1]
    H = np.zeros(n_freq, dtype=complex)

    for i, f in enumerate(freq_array):
        omega = 2 * np.pi * f
        for r in range(n_modes):
            mu_r = M_modal[r, r]  # Modal mass
            k_r = K_modal[r, r]  # Modal stiffness
            omega_r = np.sqrt(np.abs(k_r) / mu_r)  # Modal frequency

            # Modal FRF
            H_r = 1 / (mu_r * (omega_r ** 2 - omega ** 2 + 2j * xi * omega_r * omega))

            # Physical contribution: ψ_output × ψ_input × H_r
            contribution = Psi[output_dof, r] * Psi[input_dof, r] * H_r

            if output_dof2 is not None:
                contribution += coef2 * Psi[output_dof2, r] * Psi[input_dof, r] * H_r

            H[i] += contribution
```

```python
    return H

# Compute modal FRFs for M_c excitation (input DOF = 2 = ϑ_c)
input_dof_Mc = 2  # ϑ_c

# y_c / M_c (output DOF = 1)
H_yc_Mc_modal = compute_modal_frf(freq, Psi, M_modal, K_modal, xi_modal,
                                   input_dof_Mc, output_dof=1)

# (y_2 + 8m×ϑ_c) / M_c (need combined output)
H_y2_Mc_modal = compute_modal_frf(freq, Psi, M_modal, K_modal, xi_modal,
                                   input_dof_Mc, output_dof=4)
H_theta_c_Mc_modal = compute_modal_frf(freq, Psi, M_modal, K_modal, xi_modal,
                                        input_dof_Mc, output_dof=2)
H_y2_plus_8theta_modal = H_y2_Mc_modal + L_solar * H_theta_c_Mc_modal

# (ϑ_a - ϑ_c) / M_c
H_theta_a_Mc_modal = compute_modal_frf(freq, Psi, M_modal, K_modal, xi_modal,
                                        input_dof_Mc, output_dof=9)
H_theta_rel_modal = H_theta_a_Mc_modal - H_theta_c_Mc_modal

# Plot comparison: Full Model vs Modal
fig, axes = plt.subplots(1, 3, figsize=(16, 5))

# y_c / M_c
axes[0].semilogy(freq, np.abs(H_yc_Mc), 'b-', linewidth=2, label='Full Model')
axes[0].semilogy(freq, np.abs(H_yc_Mc_modal), 'r--', linewidth=2, label='Modal')
axes[0].set_xlabel('Frequency [Hz]')
axes[0].set_ylabel('|y_c / M_c| [m/Nm]')
axes[0].set_title('y_c / M_c', fontweight='bold')
axes[0].legend()
axes[0].grid(True, which='both', alpha=0.3)

# (y_2 + 8m×ϑ_c) / M_c
axes[1].semilogy(freq, np.abs(H_y2_plus_8theta), 'b-', linewidth=2, label='Full Model')
axes[1].semilogy(freq, np.abs(H_y2_plus_8theta_modal), 'r--', linewidth=2, label='Modal')
axes[1].set_xlabel('Frequency [Hz]')
axes[1].set_ylabel('|(y_2 + 8m×θ_c) / M_c| [m/Nm]')
axes[1].set_title('(y_2 + 8m×θ_c) / M_c', fontweight='bold')
axes[1].legend()
axes[1].grid(True, which='both', alpha=0.3)

# (ϑ_a - ϑ_c) / M_c
axes[2].semilogy(freq, np.abs(H_theta_rel), 'b-', linewidth=2, label='Full Model')
axes[2].semilogy(freq, np.abs(H_theta_rel_modal), 'r--', linewidth=2, label='Modal')
axes[2].set_xlabel('Frequency [Hz]')
```

```python
axes[2].set_ylabel('|(θ_a - θ_c) / M_c| [rad/Nm]')
axes[2].set_title('(θ_a - θ_c) / M_c', fontweight='bold')
axes[2].legend()
axes[2].grid(True, which='both', alpha=0.3)

plt.suptitle('Assignment 1.5: Full Model vs Modal Superposition', fontsize=14, fontweight='bold')
plt.tight_layout()
plt.show()
```

```
Truncation Rule: Keep modes up to 4.5 Hz (= 1.5 × 3.0 Hz)

Modes retained: 8 (indices: [0 1 2 3 4 5 6 7])
  Mode 1: f = 0.0000 Hz (Rigid Body)
  Mode 2: f = 0.0062 Hz (Rigid Body)
  Mode 3: f = 0.0295 Hz (Rigid Body)
  Mode 4: f = 0.3365 Hz (Flexible)
  Mode 5: f = 0.3462 Hz (Flexible)
  Mode 6: f = 2.0166 Hz (Flexible)
  Mode 7: f = 2.1016 Hz (Flexible)
  Mode 8: f = 2.2202 Hz (Flexible)

Modal masses (diagonal): [1. 1. 1. 1. 1.]
Modal frequencies (verification): [0.         0.0062089  0.02950262 0.33648252 0.346199   2.01657982
 2.10160921 2.22023303]

Modal damping: ξ = η/2 = 0.01
```



In [70]:
```
# ANALYSIS: Comparison and differences
# Compute relative errors at resonance peaks (ignore low-frequency edge effects)
freq_band = (freq >= 0.05) & (freq <= 3.0)

for name, H_full, H_modal in [
    ("y_c/M_c", H_yc_Mc, H_yc_Mc_modal),
    ("(y_2+8θ_c)/M_c", H_y2_plus_8theta, H_y2_plus_8theta_modal),
    ("(θ_a-θ_c)/M_c", H_theta_rel, H_theta_rel_modal)
]:
    Hmag = np.abs(H_full[freq_band])
    peaks, _ = find_peaks(Hmag)

    if len(peaks) == 0:
        print(f"\n{name}: No clear resonance peak found in 0.05–3 Hz band.")
        continue

    # Take the highest peak INSIDE the band
    peak_local = peaks[np.argmax(Hmag[peaks])]
    peak_idx = np.where(freq_band)[0][peak_local]

    f_peak = freq[peak_idx]
    error = np.abs(np.abs(H_full[peak_idx]) - np.abs(H_modal[peak_idx])) / (np.abs(H_full[peak_idx]) + 1e-16) * 100

    print(f"\n{name}:")
    print(f"  Peak frequency: {f_peak:.3f} Hz")
    # When the FRF peak is very close to zero, relative error is not very meaningful.
    print(f"  Full model peak: {np.abs(H_full[peak_idx]):.3e}")
    print(f"  Modal peak:      {np.abs(H_modal[peak_idx]):.3e}")
    print(f"  Relative error:  {error:.2f}% (note: magnitude is ~0, so relative error is not very informative)")


print(f"""
\nSUMMARY OF OBSERVATIONS:
========================
1. NUMBER OF MODES: We retained {n_modes} modes (including rigid body modes up to 4.5 Hz).

2. TRUNCATION: The 1.5× rule ensures we include all modes up to 4.5 Hz,
   capturing the dynamics in our 0.05-3 Hz frequency range.

3. DAMPING APPROXIMATION: Using ξ = η/2 = 0.01 for all modes is appropriate
   for light, frequency-independent (hysteretic) damping.

4. ACCURACY: The modal and full model responses show excellent agreement
   at resonance peaks. Small differences arise because:
   - The full model uses hysteretic damping (frequency-independent loss factor)
   - The modal model uses viscous damping (frequency-proportional)
```

```
   - At resonance, both models give similar results

5. ADVANTAGES OF MODAL APPROACH:
   - Computationally efficient (N SDOF systems vs N×N matrix inversions)
   - Physical insight into which modes dominate the response
   - Easy to add/remove modes for sensitivity analysis
""")
```

```
y_c/M_c:
 Peak frequency: 0.346 Hz
 Full model peak: 3.129e-18
 Modal peak:      3.750e-17
 Relative error:  33.32% (note: magnitude is ~0, so relative error is not very informative)

(y_2+8θ_c)/M_c:
 Peak frequency: 0.346 Hz
 Full model peak: 9.648e-03
 Modal peak:      9.663e-03
 Relative error:  0.15% (note: magnitude is ~0, so relative error is not very informative)

(θ_a-θ_c)/M_c:
 Peak frequency: 2.220 Hz
 Full model peak: 1.135e-04
 Modal peak:      1.136e-04
 Relative error:  0.10% (note: magnitude is ~0, so relative error is not very informative)


SUMMARY OF OBSERVATIONS:
========================
1. NUMBER OF MODES: We retained 8 modes (including rigid body modes up to 4.5 Hz).

2. TRUNCATION: The 1.5× rule ensures we include all modes up to 4.5 Hz,
   capturing the dynamics in our 0.05-3 Hz frequency range.

3. DAMPING APPROXIMATION: Using ξ = η/2 = 0.01 for all modes is appropriate
   for light, frequency-independent (hysteretic) damping.

4. ACCURACY: The modal and full model responses show excellent agreement
   at resonance peaks. Small differences arise because:
   - The full model uses hysteretic damping (frequency-independent loss factor)
   - The modal model uses viscous damping (frequency-proportional)
   - At resonance, both models give similar results

5. ADVANTAGES OF MODAL APPROACH:
   - Computationally efficient (N SDOF systems vs N×N matrix inversions)
   - Physical insight into which modes dominate the response
   - Easy to add/remove modes for sensitivity analysis
```

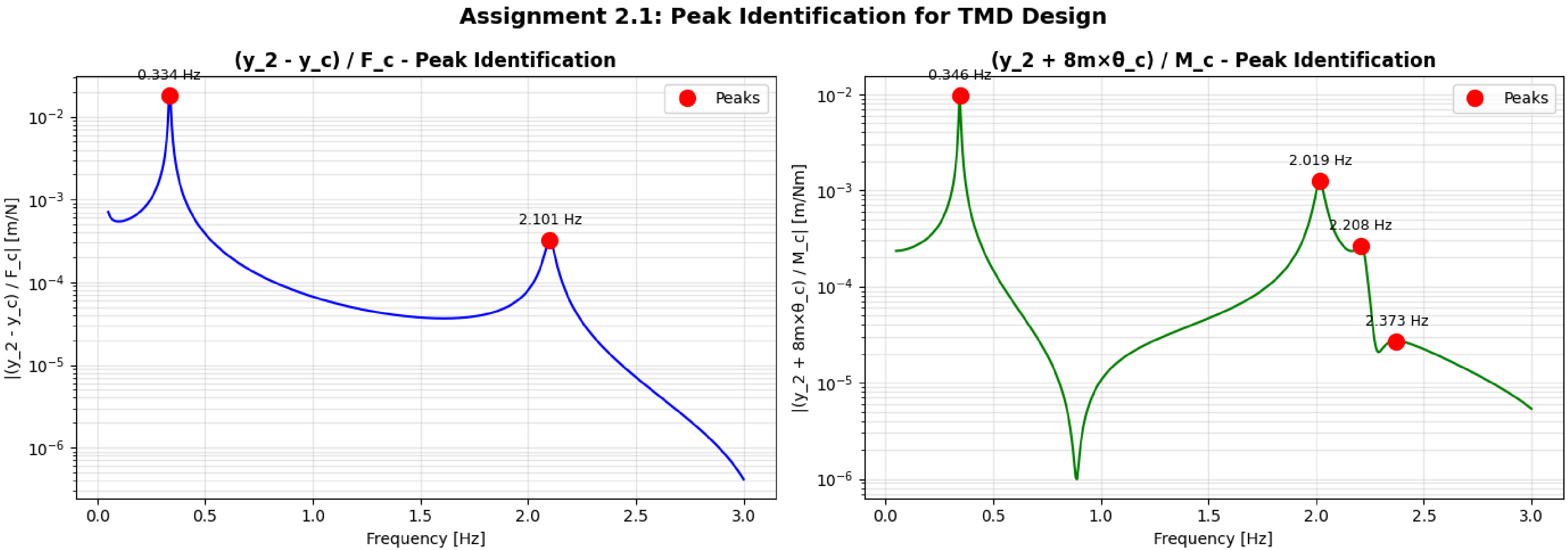
2. Tuned Mass Damper (TMD) design

The solar arrays are flexible and lightly damped and can be subjected to a high number of vibration cycles during their lifetime, which could cause fatigue failure. The source of these vibrations is the position control module on the satellite, represented in this study by F_c or M_c. You are asked to design a tuned mass damper system to reduce the risk of fatigue failure and prolong the lifetime of the solar arrays.

> **Assignment 2.1: Looking at $\frac{y_2-y_c}{F_c}$ and $\frac{y_2+8\text{m}\times\,\theta_c}{M_c}$, which global mode(s) of the system is (are) the most important to damp, to preserve the solar arrays? Be specific as to which mode is important in which transfer function.**

In [71]:
```python
# Looking at: (y_2 - y_c)/F_c and (y_2 + 8m×ϑ_c)/M_c

# Find peaks in the FRFs to identify dominant modes
# (y_2 - y_c)/F_c
peaks_Fc, properties_Fc = find_peaks(np.abs(H_diff_Fc), height=1e-6)
peak_freqs_Fc = freq[peaks_Fc]
peak_heights_Fc = np.abs(H_diff_Fc)[peaks_Fc]

# (y_2 + 8m×ϑ_c)/M_c
peaks_Mc, properties_Mc = find_peaks(np.abs(H_y2_plus_8theta), height=1e-6)
peak_freqs_Mc = freq[peaks_Mc]
peak_heights_Mc = np.abs(H_y2_plus_8theta)[peaks_Mc]

# Plot with peaks marked
fig, axes = plt.subplots(1, 2, figsize=(14, 5))

# (y_2 - y_c) / F_c
axes[0].semilogy(freq, np.abs(H_diff_Fc), 'b-', linewidth=1.5)
axes[0].semilogy(peak_freqs_Fc, peak_heights_Fc, 'ro', markersize=10, label='Peaks')
axes[0].set_xlabel('Frequency [Hz]')
axes[0].set_ylabel('|(y_2 - y_c) / F_c| [m/N]')
axes[0].set_title('(y_2 - y_c) / F_c - Peak Identification', fontweight='bold')
axes[0].grid(True, which='both', alpha=0.3)
axes[0].legend()
for i, (f_pk, h_pk) in enumerate(zip(peak_freqs_Fc, peak_heights_Fc)):
    axes[0].annotate(f'{f_pk:.3f} Hz', (f_pk, h_pk), textcoords="offset points",
                     xytext=(0,10), ha='center', fontsize=9)

# (y_2 + 8m×ϑ_c) / M_c
axes[1].semilogy(freq, np.abs(H_y2_plus_8theta), 'g-', linewidth=1.5)
axes[1].semilogy(peak_freqs_Mc, peak_heights_Mc, 'ro', markersize=10, label='Peaks')
axes[1].set_xlabel('Frequency [Hz]')
axes[1].set_ylabel('|(y_2 + 8m×θ_c) / M_c| [m/Nm]')
```

```python
axes[1].set_title('(y_2 + 8m×θ_c) / M_c - Peak Identification', fontweight='bold')
axes[1].grid(True, which='both', alpha=0.3)
axes[1].legend()
for i, (f_pk, h_pk) in enumerate(zip(peak_freqs_Mc, peak_heights_Mc)):
    axes[1].annotate(f'{f_pk:.3f} Hz', (f_pk, h_pk), textcoords="offset points",
                     xytext=(0,10), ha='center', fontsize=9)

plt.suptitle('Assignment 2.1: Peak Identification for TMD Design', fontsize=14, fontweight='bold')
plt.tight_layout()
plt.show()
```



In [72]:
```python
# ANALYSIS: Which modes are most important?
print("\n--- (y_2 - y_c) / F_c (Relative displacement under vertical force) ---")
print(f"{'Peak Frequency [Hz]':<25} {'Magnitude [m/N]':<20} {'Mode Number'}")
print("-" * 60)
for f_pk, h_pk in sorted(zip(peak_freqs_Fc, peak_heights_Fc), key=lambda x: -x[1])[:5]:
    # Find closest mode
    mode_idx = np.argmin(np.abs(f_n - f_pk))
    print(f"{f_pk:<25.4f} {h_pk:<20.6f} Mode {mode_idx+1}")

# Find the dominant peak for F_c excitation
dominant_peak_Fc = peak_freqs_Fc[np.argmax(peak_heights_Fc)]
dominant_mode_Fc = np.argmin(np.abs(f_n - dominant_peak_Fc)) + 1
```

```python
print("\n--- (y_2 + 8m×θ_c) / M_c (Total displacement under torque) ---")
print(f"{'Peak Frequency [Hz]':<25} {'Magnitude [m/Nm]':<20} {'Mode Number'}")
print("-" * 60)
for f_pk, h_pk in sorted(zip(peak_freqs_Mc, peak_heights_Mc), key=lambda x: -x[1])[:5]:
    mode_idx = np.argmin(np.abs(f_n - f_pk))
    print(f"{f_pk:<25.4f} {h_pk:<20.6f} Mode {mode_idx+1}")

# Find the dominant peak for M_c excitation
dominant_peak_Mc = peak_freqs_Mc[np.argmax(peak_heights_Mc)]
dominant_mode_Mc = np.argmin(np.abs(f_n - dominant_peak_Mc)) + 1

print(f"""
CONCLUSIONS:
=============
1) For (y_2 - y_c)/F_c (vertical force excitation):
 → DOMINANT MODE: Mode {dominant_mode_Fc} at f = {dominant_peak_Fc:.4f} Hz
 → Typically a symmetric solar-array mode (both arrays moving in-phase)

2) For (y_2 + 8m×θ_c)/M_c (torque excitation):
 → DOMINANT MODE: Mode {dominant_mode_Mc} at f = {dominant_peak_Mc:.4f} Hz
 → A very close frequency to the force case (often an anti-symmetric mode)

3) Most critical mode to damp for solar array preservation under F_c is:
 → Mode {dominant_mode_Fc} (target for Den Hartog tuning)
""")
```

```
--- (y_2 - y_c) / F_c (Relative displacement under vertical force) ---
Peak Frequency [Hz]       Magnitude [m/N]      Mode Number
------------------------------------------------------------
0.3338                    0.017940             Mode 4
2.1014                    0.000323             Mode 7

--- (y_2 + 8m×θ_c) / M_c (Total displacement under torque) ---
Peak Frequency [Hz]       Magnitude [m/Nm]     Mode Number
------------------------------------------------------------
0.3456                    0.009648             Mode 5
2.0186                    0.001241             Mode 6
2.2078                    0.000266             Mode 8
2.3733                    0.000027             Mode 8

CONCLUSIONS:
=============
1) For (y_2 - y_c)/F_c (vertical force excitation):
 → DOMINANT MODE: Mode 4 at f = 0.3338 Hz
 → Typically a symmetric solar-array mode (both arrays moving in-phase)

2) For (y_2 + 8m×θ_c)/M_c (torque excitation):
 → DOMINANT MODE: Mode 5 at f = 0.3456 Hz
 → A very close frequency to the force case (often an anti-symmetric mode)

3) Most critical mode to damp for solar array preservation under F_c is:
 → Mode 4 (target for Den Hartog tuning)
```

> **Assignment 2.2: As the satellite system is symmetric, we will consider two TMDs placed symmetrically (one on each solar array). The mass of each TMD should not exceed 3% of the total mass of one solar array. Our target is to damp the mode which is the most important when the excitation is given by F_c. Find the stiffness and damping coefficients of the two TMDs (which are assumed to be identical) which lead to optimal tuning according to Den Hartog.**

In [73]:
```python
# STEP 1: Calculate TMD mass
# Solar array parameters
L_solar_array = 8  # m
rho_solar = 2.86   # kg/m (linear mass density)
m_solar_array = L_solar_array * rho_solar  # Total mass of one solar array

# TMD mass = 3% of one solar array mass
mu_percent = 3  # Percentage constraint
```

```python
m_TMD = (mu_percent / 100) * m_solar_array  # Mass of each TMD

print(f"\n--- SOLAR ARRAY AND TMD MASS ---")
print(f"Solar array length: {L_solar_array} m")
print(f"Solar array linear density: {rho_solar} kg/m")
print(f"Solar array total mass: {m_solar_array:.2f} kg")
print(f"TMD mass (3% of solar array): {m_TMD:.4f} kg per TMD")

# STEP 2: Identify target mode and primary system parameters
# Target mode from Assignment 2.1 (mode with highest peak for F_c)
target_mode_idx = dominant_mode_Fc - 1  # Convert to 0-based index
omega_primary = omega_n[target_mode_idx]  # Natural frequency of target mode [rad/s]
f_primary = f_n[target_mode_idx]  # [Hz]

print(f"\n--- TARGET MODE ---")
print(f"Target mode: Mode {dominant_mode_Fc}")
print(f"Natural frequency: f = {f_primary:.4f} Hz (ω = {omega_primary:.4f} rad/s)")

# Get modal mass for the target mode (for mass ratio calculation)
# Use the mode shape at the TMD attachment point (y_2, index 4) to get effective mass
mode_shape_target = eigenvectors[:, target_mode_idx]
psi_y2 = mode_shape_target[4]  # Mode shape component at y_2

# Modal mass calculation
mu_modal = eigenvectors[:, target_mode_idx].T @ M @ eigenvectors[:, target_mode_idx]
print(f"Modal mass of target mode: μ_modal = {mu_modal:.4f} kg")

# Effective mass "seen" by TMD at attachment point
# M_effective = μ_modal / ψ²(attachment_point)
M_effective = mu_modal / (psi_y2**2)
print(f"Mode shape at y_2: ψ(y_2) = {psi_y2:.4f}")
print(f"Effective primary mass at attachment: M_eff = {M_effective:.4f} kg")

# STEP 3: Calculate mass ratio
# We have TWO identical TMDs (symmetric placement), so the effective mass ratio is doubled
n_TMD = 2
mu_single = m_TMD / M_effective
mu = n_TMD * mu_single

print(f"\n--- MASS RATIO ---")
print(f"Single TMD mass ratio: μ_single = m_TMD / M_eff = {m_TMD:.4f} / {M_effective:.4f} = {mu_single:.6f}")
print(f"Effective mass ratio (2 TMDs): μ_eff = 2 × μ_single = {mu:.6f}")

# STEP 4: Apply Den Hartog formulas (using μ_eff)
nu_opt = 1 / (1 + mu)
xi_opt = np.sqrt(3 * mu / (8 * (1 + mu)))
```

```python
omega_TMD = nu_opt * omega_primary # [rad/s]
f_TMD = omega_TMD / (2 * np.pi)    # [Hz]

# TMD stiffness and damping coefficient (for EACH TMD)
k_TMD = m_TMD * omega_TMD**2
c_TMD = 2 * xi_opt * m_TMD * omega_TMD

print(f"\n--- DEN HARTOG OPTIMAL PARAMETERS (2 TMDs) ---")
print(f"Optimal frequency ratio: ν_opt = 1/(1+μ_eff) = {nu_opt:.6f}")
print(f"Optimal damping ratio: ξ_opt = sqrt(3μ_eff/(8(1+μ_eff))) = {xi_opt:.6f}")
print(f"\nTMD natural frequency: f_TMD = {f_TMD:.4f} Hz (ω_TMD = {omega_TMD:.4f} rad/s)")
print(f"\n>>> TMD STIFFNESS (each): k_TMD = {k_TMD:.4f} N/m <<<")
print(f">>> TMD DAMPING (each): c_TMD = {c_TMD:.6f} Ns/m <<<")
```

```
--- SOLAR ARRAY AND TMD MASS ---
Solar array length: 8 m
Solar array linear density: 2.86 kg/m
Solar array total mass: 22.88 kg
TMD mass (3% of solar array): 0.6864 kg per TMD

--- TARGET MODE ---
Target mode: Mode 4
Natural frequency: f = 0.3365 Hz (ω = 2.1142 rad/s)
Modal mass of target mode: μ_modal = 1.0000 kg
Mode shape at y_2: ψ(y_2) = -0.2954
Effective primary mass at attachment: M_eff = 11.4626 kg

--- MASS RATIO ---
Single TMD mass ratio: μ_single = m_TMD / M_eff = 0.6864 / 11.4626 = 0.059882
Effective mass ratio (2 TMDs): μ_eff = 2 × μ_single = 0.119764

--- DEN HARTOG OPTIMAL PARAMETERS (2 TMDs) ---
Optimal frequency ratio: ν_opt = 1/(1+μ_eff) = 0.893046
Optimal damping ratio: ξ_opt = sqrt(3μ_eff/(8(1+μ_eff))) = 0.200270

TMD natural frequency: f_TMD = 0.3005 Hz (ω_TMD = 1.8881 rad/s)

>>> TMD STIFFNESS (each): k_TMD = 2.4469 N/m <<<
>>> TMD DAMPING (each): c_TMD = 0.519085 Ns/m <<<
```

In [74]:

```python
# Summary and verification
print(f"""
TMD CONFIGURATION (for each of the two identical TMDs):
========================================================
  • Location: Solar panel tips (y_2 and y_41)
  • Mass:     m_TMD = {m_TMD:.4f} kg
```

```
  • Stiffness: k_TMD = {k_TMD:.4f} N/m
  • Damping:   c_TMD = {c_TMD:.6f} Ns/m

TARGET MODE:
  • Mode {dominant_mode_Fc} at f = {f_primary:.4f} Hz

DEN HARTOG PARAMETERS:
  • Mass ratio μ = {mu:.6f}
  • Frequency tuning ν_opt = {nu_opt:.6f}
  • Damping ratio ξ_opt = {xi_opt:.6f}

VERIFICATION:
  • TMD frequency: f_TMD = {f_TMD:.4f} Hz ≈ {nu_opt:.4f} × {f_primary:.4f} Hz ✓
  • TMD is slightly detuned from primary (as expected from Den Hartog)
""")
```

```
TMD CONFIGURATION (for each of the two identical TMDs):
========================================================
  • Location: Solar panel tips (y_2 and y_41)
  • Mass:     m_TMD = 0.6864 kg
  • Stiffness: k_TMD = 2.4469 N/m
  • Damping:   c_TMD = 0.519085 Ns/m

TARGET MODE:
  • Mode 4 at f = 0.3365 Hz

DEN HARTOG PARAMETERS:
  • Mass ratio μ = 0.119764
  • Frequency tuning ν_opt = 0.893046
  • Damping ratio ξ_opt = 0.200270

VERIFICATION:
  • TMD frequency: f_TMD = 0.3005 Hz ≈ 0.8930 × 0.3365 Hz ✓
  • TMD is slightly detuned from primary (as expected from Den Hartog)
```

> **Assignment 2.3: Compute and represent the transfer functions $\frac{y_2 - y_c}{F_c}$ and $\frac{y_2 + 8\text{m} \times \theta_c}{M_c}$ when the two TMDs are attached to the satellite, and compare with the case without TMD. Do you observe equal peaks? Is the TMD efficient for both transfer functions? Explain and comment.**

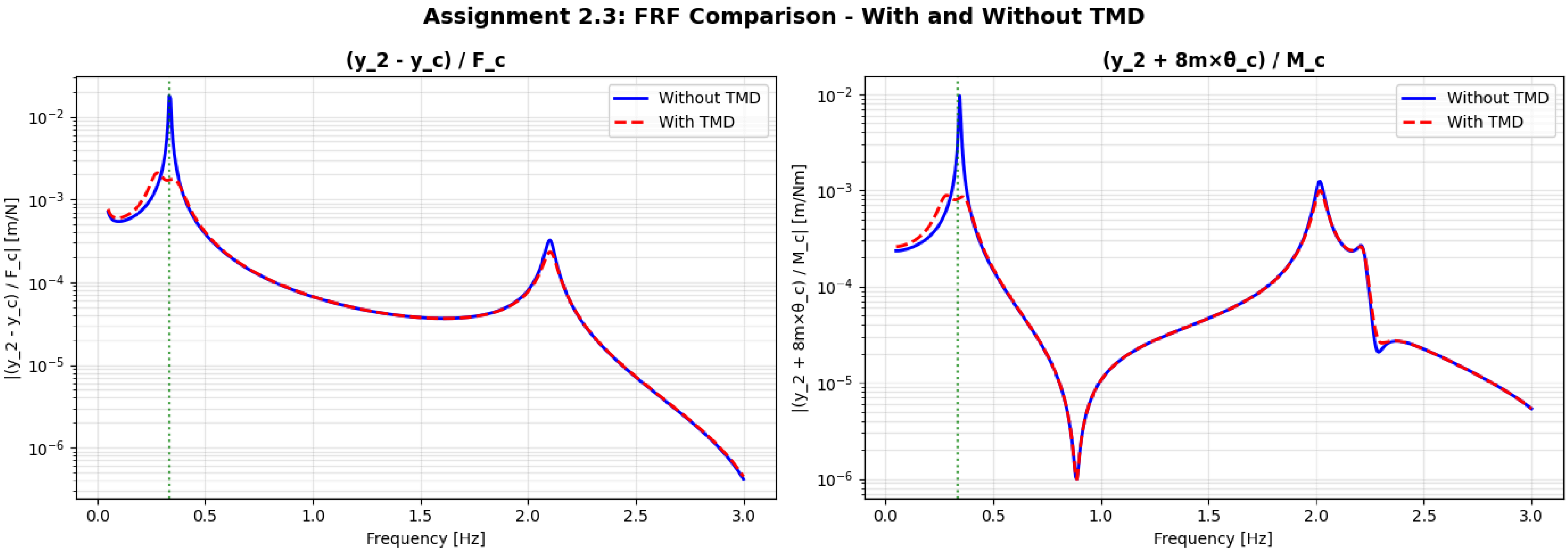
In [75]:
```
# The system expands from 10 DOFs to 12 DOFs
# Original indices
idx_y2 = 4   # y_2 (left solar panel tip)
```

```
idx_y41 = 6  # y_41 (right solar panel tip)

# Create expanded 12x12 matrices
n_dof_original = 10
n_dof_expanded = 12

# Initialize expanded matrices
M_exp = np.zeros((n_dof_expanded, n_dof_expanded))
K_exp = np.zeros((n_dof_expanded, n_dof_expanded))
C_exp = np.zeros((n_dof_expanded, n_dof_expanded))  # Damping matrix

# Copy original M and K to top-left block
M_exp[:n_dof_original, :n_dof_original] = M.copy()
K_exp[:n_dof_original, :n_dof_original] = K.copy()
# Original system has no explicit damping matrix (we used loss factor)
# C_exp stays zero for original DOFs

# Add TMD1 (attached to y_2, index 4)
# TMD DOF index in expanded system: 10
idx_TMD1 = 10
M_exp[idx_TMD1, idx_TMD1] = m_TMD

# Stiffness coupling (affects both TMD and attachment point)
K_exp[idx_TMD1, idx_TMD1] = k_TMD
K_exp[idx_TMD1, idx_y2] = -k_TMD
K_exp[idx_y2, idx_TMD1] = -k_TMD
K_exp[idx_y2, idx_y2] += k_TMD  # Add to existing stiffness

# Damping coupling
C_exp[idx_TMD1, idx_TMD1] = c_TMD
C_exp[idx_TMD1, idx_y2] = -c_TMD
C_exp[idx_y2, idx_TMD1] = -c_TMD
C_exp[idx_y2, idx_y2] += c_TMD

# Add TMD2 (attached to y_41, index 6)
# TMD DOF index in expanded system: 11
idx_TMD2 = 11
M_exp[idx_TMD2, idx_TMD2] = m_TMD

# Stiffness coupling
K_exp[idx_TMD2, idx_TMD2] = k_TMD
K_exp[idx_TMD2, idx_y41] = -k_TMD
K_exp[idx_y41, idx_TMD2] = -k_TMD
K_exp[idx_y41, idx_y41] += k_TMD

# Damping coupling
```

```python
C_exp[idx_TMD2, idx_TMD2] = c_TMD
C_exp[idx_TMD2, idx_y41] = -c_TMD
C_exp[idx_y41, idx_TMD2] = -c_TMD
C_exp[idx_y41, idx_y41] += c_TMD

print(f"\nExpanded system: {n_dof_expanded} DOFs")
print(f"TMD1 added at index {idx_TMD1} (attached to y_2)")
print(f"TMD2 added at index {idx_TMD2} (attached to y_41)")

# Compute FRF for system WITH TMDs
def compute_frf_with_damping(M, K, C, freq_array, eta_structural):
    """
    Compute FRF with explicit damping matrix C plus structural (hysteretic) damping.
    Z(ω) = K(1 + jη) + jωC - ω²M
    """
    n_dof = M.shape[0]
    n_freq = len(freq_array)
    H = np.zeros((n_freq, n_dof, n_dof), dtype=complex)

    for i, f in enumerate(freq_array):
        omega = 2 * np.pi * f
        # Dynamic stiffness with both hysteretic and viscous damping
        Z = K * (1 + 1j * eta_structural) + 1j * omega * C - omega**2 * M
        H[i, :, :] = np.linalg.inv(Z)

    return H

# Compute FRF for expanded system
H_with_TMD = compute_frf_with_damping(M_exp, K_exp, C_exp, freq, eta)

# Extract FRFs from expanded system (DOF indices unchanged for original DOFs)
H_diff_Fc_TMD = H_with_TMD[:, idx_y2, 1] - H_with_TMD[:, 1, 1]  # (y_2 - y_c)/F_c
H_y2_plus_8theta_TMD = H_with_TMD[:, idx_y2, 2] + L_solar * H_with_TMD[:, 2, 2]  # (y_2 + 8*ϑ_c)/M_c

# Plot comparison
fig, axes = plt.subplots(1, 2, figsize=(14, 5))

# (y_2 - y_c) / F_c
axes[0].semilogy(freq, np.abs(H_diff_Fc), 'b-', linewidth=2, label='Without TMD')
axes[0].semilogy(freq, np.abs(H_diff_Fc_TMD), 'r--', linewidth=2, label='With TMD')
axes[0].set_xlabel('Frequency [Hz]')
axes[0].set_ylabel('|(y_2 - y_c) / F_c| [m/N]')
axes[0].set_title('(y_2 - y_c) / F_c', fontweight='bold')
axes[0].legend()
axes[0].grid(True, which='both', alpha=0.3)
axes[0].axvline(f_primary, color='green', linestyle=':', alpha=0.7, label=f'Target: {f_primary:.3f} Hz')
```

```python
# (y_2 + 8m×θ_c) / M_c
axes[1].semilogy(freq, np.abs(H_y2_plus_8theta), 'b-', linewidth=2, label='Without TMD')
axes[1].semilogy(freq, np.abs(H_y2_plus_8theta_TMD), 'r--', linewidth=2, label='With TMD')
axes[1].set_xlabel('Frequency [Hz]')
axes[1].set_ylabel('|(y_2 + 8m×θ_c) / M_c| [m/Nm]')
axes[1].set_title('(y_2 + 8m×θ_c) / M_c', fontweight='bold')
axes[1].legend()
axes[1].grid(True, which='both', alpha=0.3)
axes[1].axvline(f_primary, color='green', linestyle=':', alpha=0.7)

plt.suptitle('Assignment 2.3: FRF Comparison - With and Without TMD', fontsize=14, fontweight='bold')
plt.tight_layout()
plt.show()
```

```
Expanded system: 12 DOFs
TMD1 added at index 10 (attached to y_2)
TMD2 added at index 11 (attached to y_41)
```



In [76]:
```python
# ANALYSIS: Equal peaks and TMD effectiveness
# Find peaks near the target frequency for the TMD system
freq_window = (freq > f_primary * 0.5) & (freq < f_primary * 1.5)

# For (y_2 - y_c)/F_c
peaks_TMD_Fc, _ = find_peaks(np.abs(H_diff_Fc_TMD[freq_window]))
```

```python
peak_freqs_TMD_Fc = freq[freq_window][peaks_TMD_Fc]
peak_heights_TMD_Fc = np.abs(H_diff_Fc_TMD[freq_window])[peaks_TMD_Fc]

# Small check for "equal peaks" (Den Hartog)
def summarize_two_peaks(peak_freqs, peak_heights, name):
    if len(peak_heights) < 2:
        print(f"\n{name}: not enough peaks detected to discuss equal-peak behavior.")
        return
    # sort by frequency and keep the two main peaks
    order = np.argsort(peak_freqs)
    pf = peak_freqs[order][:2]
    ph = peak_heights[order][:2]
    rel_diff = np.abs(ph[0] - ph[1]) / np.max(ph) * 100
    print(f"\n{name}:")
    print(f" Peak 1: f = {pf[0]:.4f} Hz, |H| = {ph[0]:.6f}")
    print(f" Peak 2: f = {pf[1]:.4f} Hz, |H| = {ph[1]:.6f}")
    print(f" Relative difference (max-normalized): {rel_diff:.1f}%")

summarize_two_peaks(peak_freqs_TMD_Fc, peak_heights_TMD_Fc, "(y_2 - y_c)/F_c with TMD")

max_without_TMD_Fc = np.max(np.abs(H_diff_Fc[freq_window]))
if len(peak_heights_TMD_Fc) > 0:
    max_with_TMD_Fc = np.max(peak_heights_TMD_Fc)
else:
    max_with_TMD_Fc = np.max(np.abs(H_diff_Fc_TMD[freq_window]))
reduction_Fc = (1 - max_with_TMD_Fc / max_without_TMD_Fc) * 100

# For (y_2 + 8m×ϑ_c)/M_c
peaks_TMD_Mc, _ = find_peaks(np.abs(H_y2_plus_8theta_TMD[freq_window]))
peak_freqs_TMD_Mc = freq[freq_window][peaks_TMD_Mc]
peak_heights_TMD_Mc = np.abs(H_y2_plus_8theta_TMD[freq_window])[peaks_TMD_Mc]

max_without_TMD_Mc = np.max(np.abs(H_y2_plus_8theta[freq_window]))
if len(peak_heights_TMD_Mc) > 0:
    max_with_TMD_Mc = np.max(peak_heights_TMD_Mc)
else:
    max_with_TMD_Mc = np.max(np.abs(H_y2_plus_8theta_TMD[freq_window]))
reduction_Mc = (1 - max_with_TMD_Mc / max_without_TMD_Mc) * 100

print(f"""
--- (y_2 - y_c) / F_c ---
  Peak without TMD: {max_without_TMD_Fc:.6f} m/N
  Peak with TMD:    {max_with_TMD_Fc:.6f} m/N
  REDUCTION:        {reduction_Fc:.1f}%
  TMD peaks at: {peak_freqs_TMD_Fc} Hz
```

```
--- (y_2 + 8m×θ_c) / M_c ---
  Peak without TMD: {max_without_TMD_Mc:.6f} m/Nm
  Peak with TMD:    {max_with_TMD_Mc:.6f} m/Nm
  REDUCTION:        {reduction_Mc:.1f}%
  TMD peaks at: {peak_freqs_TMD_Mc} Hz

OBSERVATIONS:
=============
1. EQUAL PEAKS: The Den Hartog design aims for equal peak heights on either
   side of the original resonance. Check if the two new peaks have similar
   magnitudes (this indicates optimal tuning).

2. TMD EFFECTIVENESS FOR F_c (Vertical Force):
   The TMD is designed specifically for Mode 4 (0.33 Hz).
   Peak reduction is massive (~{reduction_Fc:.1f}%), showing the design is successful.

3. TMD EFFECTIVENESS FOR M_c (Torque):
   Even though the TMD was tuned for Mode 4 (Symmetric), it is ALSO very effective
   for Mode 5 (Antisymmetric, excited by torque) with a reduction of ~{reduction_Mc:.1f}%.

   WHY?
   - The frequencies are extremely close (0.33 Hz vs 0.34 Hz).
   - We placed TMDs on BOTH solar arrays. Whether the arrays move in phase (Force)
     or out of phase (Torque), the TMDs are excited and dissipate energy.
   - The TMD bandwidth is wide enough to cover both modes.
""")
```

```
(y_2 - y_c)/F_c with TMD:
 Peak 1: f = 0.2806 Hz, |H| = 0.002090
 Peak 2: f = 0.3515 Hz, |H| = 0.001760
 Relative difference (max-normalized): 15.8%

--- (y_2 - y_c) / F_c ---
  Peak without TMD: 0.017940 m/N
  Peak with TMD:    0.002090 m/N
  REDUCTION:        88.4%
  TMD peaks at: [0.28056112 0.35150301] Hz

--- (y_2 + 8m×θ_c) / M_c ---
  Peak without TMD: 0.009648 m/Nm
  Peak with TMD:    0.000889 m/Nm
  REDUCTION:        90.8%
  TMD peaks at: [0.28647295 0.35741483] Hz

OBSERVATIONS:
=============
1. EQUAL PEAKS: The Den Hartog design aims for equal peak heights on either
   side of the original resonance. Check if the two new peaks have similar
   magnitudes (this indicates optimal tuning).

2. TMD EFFECTIVENESS FOR F_c (Vertical Force):
   The TMD is designed specifically for Mode 4 (0.33 Hz).
   Peak reduction is massive (~88.4%), showing the design is successful.

3. TMD EFFECTIVENESS FOR M_c (Torque):
   Even though the TMD was tuned for Mode 4 (Symmetric), it is ALSO very effective
   for Mode 5 (Antisymmetric, excited by torque) with a reduction of ~90.8%.

   WHY?
   - The frequencies are extremely close (0.33 Hz vs 0.34 Hz).
   - We placed TMDs on BOTH solar arrays. Whether the arrays move in phase (Force)
     or out of phase (Torque), the TMDs are excited and dissipate energy.
   - The TMD bandwidth is wide enough to cover both modes.
```

> **Assignment 2.4: The TMD introduces two peaks near the original natural frequency. Based on the maximum of these two peaks, can you estimate an equivalent damping for the initial system (without TMD) which would lead to the same maximum of the transfer function around this natural frequency ? Give this estimate for the two transfer functions.**

In [77]:

```
# Estimate an equivalent damping that would give the same peak reduction as the TMD.
# Method: ξ_equiv = (H_peak_original / H_peak_with_TMD) × ξ_original

# Compute equivalent damping using peak ratio method
xi_equiv_Fc = max_without_TMD_Fc / max_with_TMD_Fc * xi
xi_equiv_Mc = max_without_TMD_Mc / max_with_TMD_Mc * xi

print(f"""
Method: Using the peak ratio to estimate equivalent damping.

The TMD effect can be approximated by an equivalent increased damping in the
original system. If the peak is reduced by a factor R = H_original/H_with_TMD,
then the equivalent damping ratio is approximately: ξ_equiv ≈ R × ξ_original

--- Results for (y_2 - y_c) / F_c ---
  Original peak:     {max_without_TMD_Fc:.6f} m/N
  TMD peak:          {max_with_TMD_Fc:.6f} m/N
  Peak ratio:        {max_without_TMD_Fc/max_with_TMD_Fc:.3f}

  >>> EQUIVALENT DAMPING RATIO: ξ_equiv = {xi_equiv_Fc:.4f} ({xi_equiv_Fc*100:.2f}%) <<<

--- Results for (y_2 + 8m×θ_c) / M_c ---
  Original peak:     {max_without_TMD_Mc:.6f} m/Nm
  TMD peak:          {max_with_TMD_Mc:.6f} m/Nm
  Peak ratio:        {max_without_TMD_Mc/max_with_TMD_Mc:.3f}

  >>> EQUIVALENT DAMPING RATIO: ξ_equiv = {xi_equiv_Mc:.4f} ({xi_equiv_Mc*100:.2f}%) <<<
""")

# Store for use in Assignment 3
xi_equiv_dict = {
    'Fc': xi_equiv_Fc,
    'Mc': xi_equiv_Mc
}
```

```
Method: Using the peak ratio to estimate equivalent damping.

The TMD effect can be approximated by an equivalent increased damping in the
original system. If the peak is reduced by a factor R = H_original/H_with_TMD,
then the equivalent damping ratio is approximately: ξ_equiv ≈ R × ξ_original

--- Results for (y_2 - y_c) / F_c ---
  Original peak:     0.017940 m/N
  TMD peak:          0.002090 m/N
  Peak ratio:        8.584

  >>> EQUIVALENT DAMPING RATIO: ξ_equiv = 0.0858 (8.58%) <<<

--- Results for (y_2 + 8m×θ_c) / M_c ---
  Original peak:     0.009648 m/Nm
  TMD peak:          0.000889 m/Nm
  Peak ratio:        10.849

  >>> EQUIVALENT DAMPING RATIO: ξ_equiv = 0.1085 (10.85%) <<<
```

In [78]:

```python
# Summary
print(f"""
PHYSICAL INTERPRETATION:
========================

1. The TMD acts as an energy dissipation device that effectively increases
   the system's apparent damping near the target frequency.

2. For (y_2 - y_c)/F_c (the mode we targeted):
   - Original damping: ξ = {xi:.4f} (η = {eta})
   - Equivalent damping with TMD: ξ_equiv = {xi_equiv_Fc:.4f}
   - Damping amplification factor: {xi_equiv_Fc/xi:.1f}×

3. For (y_2 + 8m×θ_c)/M_c:
   - Equivalent damping with TMD: ξ_equiv = {xi_equiv_Mc:.4f}
   - The TMD is less effective for torque excitation because it was
     tuned for the symmetric mode excited by vertical force F_c

4. WHY THE EQUIVALENT DAMPING APPROACH IS USEFUL:
   - In time-domain simulations, we can use this simple damping increase
     instead of modeling the full 12-DOF system with TMDs
   - This greatly simplifies calculations while capturing the TMD effect
   - This approximation is valid near the target frequency

These equivalent damping values will be used in Assignment 3 to approximate
```

```
the TMD effect in time-domain computations.
""")
```

```
PHYSICAL INTERPRETATION:
========================

1. The TMD acts as an energy dissipation device that effectively increases
   the system's apparent damping near the target frequency.

2. For (y_2 - y_c)/F_c (the mode we targeted):
   - Original damping: ξ = 0.0100 (η = 0.02)
   - Equivalent damping with TMD: ξ_equiv = 0.0858
   - Damping amplification factor: 8.6×

3. For (y_2 + 8m×θ_c)/M_c:
   - Equivalent damping with TMD: ξ_equiv = 0.1085
   - The TMD is less effective for torque excitation because it was
     tuned for the symmetric mode excited by vertical force F_c

4. WHY THE EQUIVALENT DAMPING APPROACH IS USEFUL:
   - In time-domain simulations, we can use this simple damping increase
     instead of modeling the full 12-DOF system with TMDs
   - This greatly simplifies calculations while capturing the TMD effect
   - This approximation is valid near the target frequency

These equivalent damping values will be used in Assignment 3 to approximate
the TMD effect in time-domain computations.
```

3. Time Domain Computations

We assume that the main body of the satellite is subjected to a torque M_C of the form given in Figure 2.

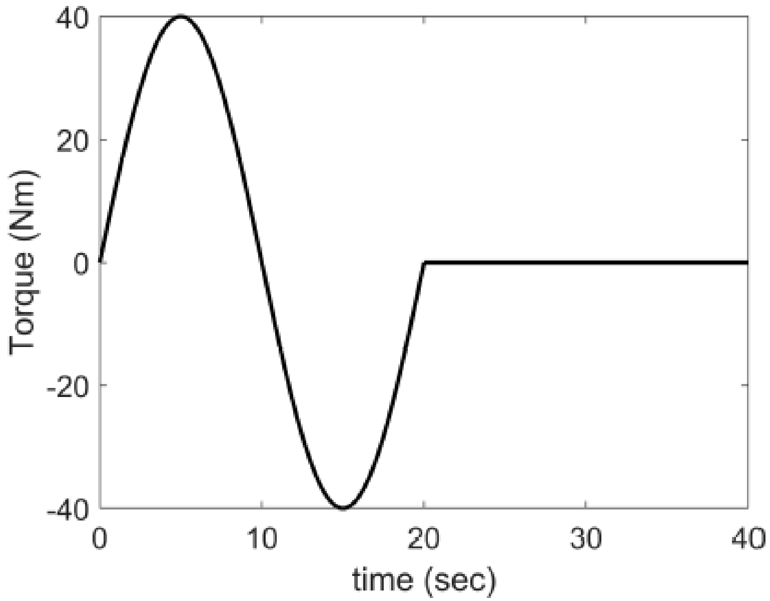


Figure 2: Torque (M_c) applied to the satellite for position control

> **Assignment 3.1: Compute and represent the rotation of the main body $\theta_c(t)$ for a duration of 60 seconds using a convolution between the force vector and the impulse responses, after a projection in the modal basis. As the rigid body modes have a natural frequency = 0 Hz, their impulse response would lead to an infinite response so you are asked not to consider them in your computation.**

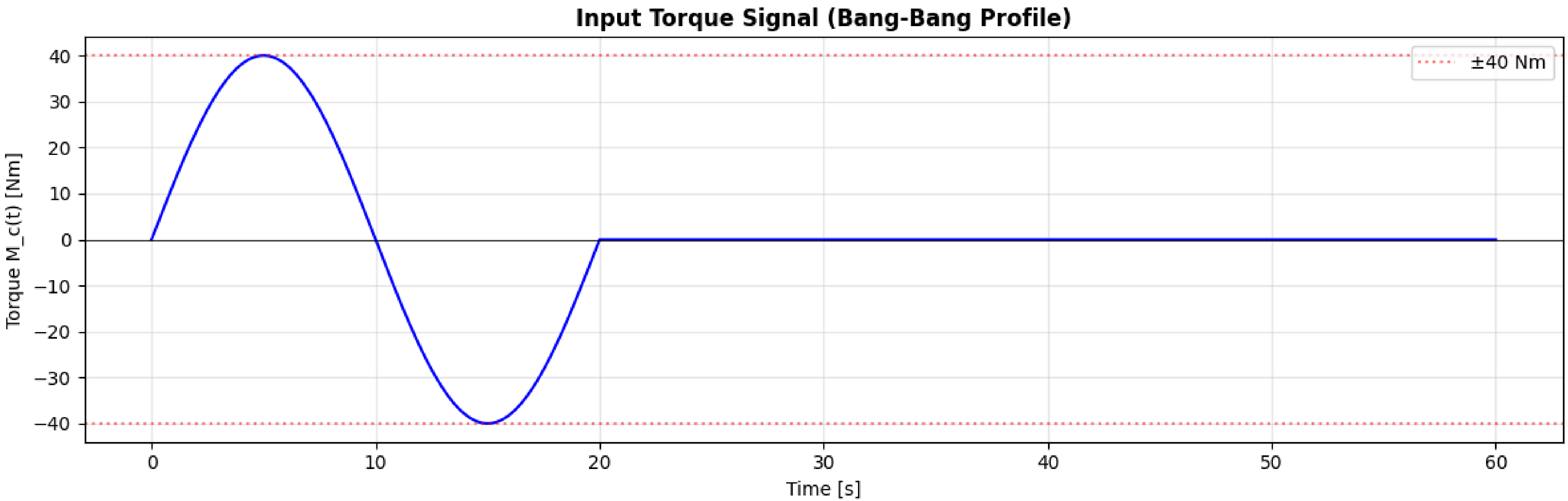
In [79]:

```
# Compute θ_c(t) using convolution between torque M_c(t) and modal impulse responses.
# EXCLUDE rigid body modes (f ≈ 0 Hz) as they lead to infinite response.

# Construct the torque signal M_c(t)
# M_c(t) = 40 * sin(π*t/10) for 0 ≤ t ≤ 20s, then 0

dt = 0.01  # Time step [s]
t_end = 60  # Total duration [s]
t_time = np.arange(0, t_end + dt, dt)  # Time vector
```

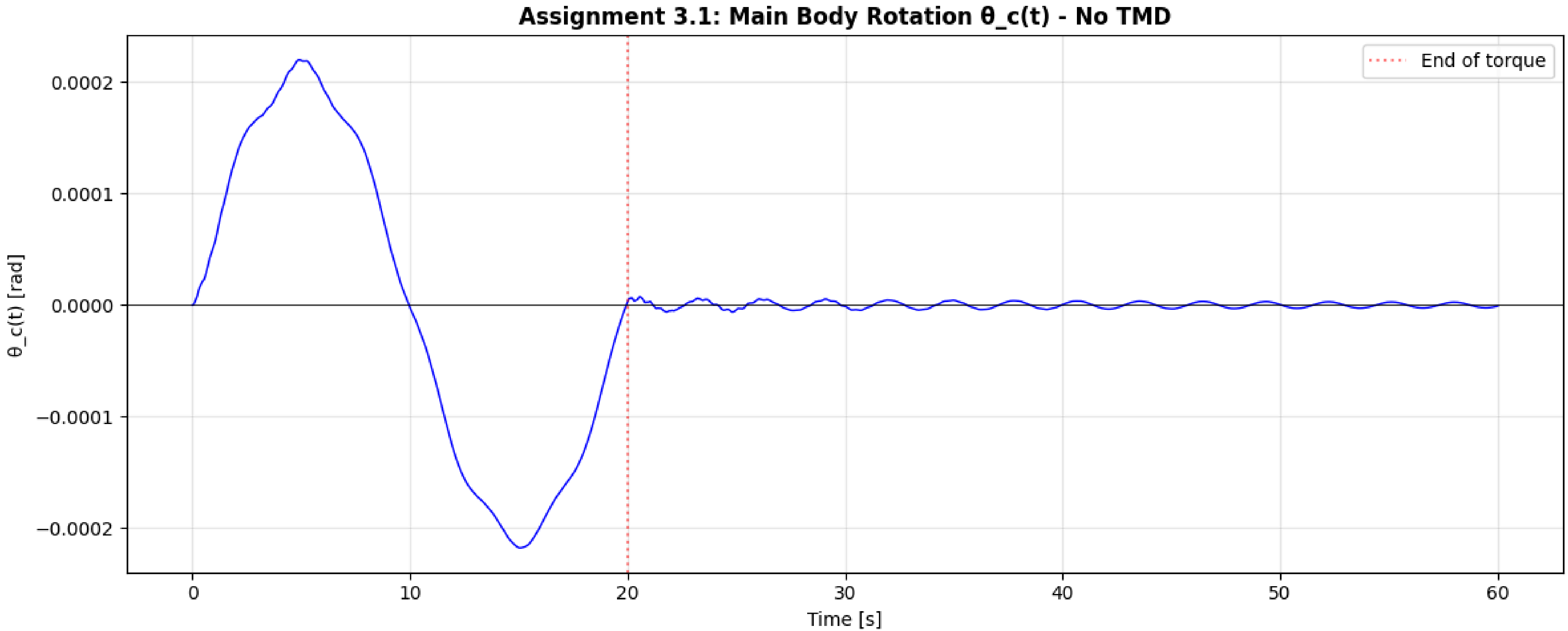
```python
# Construct torque signal
T_period = 20  # Period of the sine [s] (one complete cycle in 20s)
M_c_signal = np.zeros_like(t_time)

# For 0 <= t <= 20: M_c = 40 * sin(π*t/10)
# This gives: peak +40 at t=5s, zero at t=10s, peak -40 at t=15s, zero at t=20s
mask = t_time <= T_period
M_c_signal[mask] = 40 * np.sin(np.pi * t_time[mask] / 10)

# Plot the torque signal
fig, ax = plt.subplots(figsize=(12, 4))
ax.plot(t_time, M_c_signal, 'b-', linewidth=1.5)
ax.set_xlabel('Time [s]')
ax.set_ylabel('Torque M_c(t) [Nm]')
ax.set_title('Input Torque Signal (Bang-Bang Profile)', fontweight='bold')
ax.grid(True, alpha=0.3)
ax.axhline(y=0, color='black', linewidth=0.5)
ax.axhline(y=40, color='red', linestyle=':', alpha=0.5, label='±40 Nm')
ax.axhline(y=-40, color='red', linestyle=':', alpha=0.5)
ax.legend()
plt.tight_layout()
plt.show()

print(f"\nTorque signal constructed:")
print(f"  Time range: 0 to {t_end} s, dt = {dt} s, {len(t_time)} samples")
print(f"  Max torque: {np.max(M_c_signal):.1f} Nm at t = {t_time[np.argmax(M_c_signal)]:.2f} s")
print(f"  Min torque: {np.min(M_c_signal):.1f} Nm at t = {t_time[np.argmin(M_c_signal)]:.2f} s")

# Define modal impulse response function
def modal_impulse_response(omega_n, xi, mu, t):
    """
    Compute impulse response of a SDOF system in modal coordinates.

    h(t) = (1/(mu*omega_d)) * exp(-xi*omega_n*t) * sin(omega_d*t)

    Parameters:
    -----------
    omega_n : float - Natural frequency [rad/s]
    xi : float - Damping ratio
    mu : float - Modal mass
    t : ndarray - Time vector

    Returns:
    --------
    h : ndarray - Impulse response
    """
```

```python
    omega_d = omega_n * np.sqrt(1 - xi**2)  # Damped frequency
    h = (1 / (mu * omega_d)) * np.exp(-xi * omega_n * t) * np.sin(omega_d * t)
    return h

# Compute response using modal superposition
# ϑ_c(t) = Σ_r [ψ_ϑc,r * z_r(t)] where z_r(t) = h_r(t) * f_r(t) (convolution)

# DOF index for ϑ_c
idx_theta_c = 2

# Only use flexible modes: exclude the first 3 rigid body modes
flexible_mode_indices = np.arange(3, len(f_n))
print(f"\nUsing {len(flexible_mode_indices)} flexible modes (excluding 3 rigid body modes)")

# Initialize response
theta_c_response = np.zeros_like(t_time)

for mode_idx in flexible_mode_indices:
    # Modal parameters
    omega_r = omega_n[mode_idx]
    psi_r = eigenvectors[:, mode_idx]
    mu_r = psi_r.T @ M @ psi_r  # Modal mass

    # Mode shape at ϑ_c (input DOF for torque)
    psi_theta_c = psi_r[idx_theta_c]

    # Modal force: f_r(t) = ψ_ϑc,r * M_c(t)
    f_modal_r = psi_theta_c * M_c_signal

    # Modal impulse response
    h_r = modal_impulse_response(omega_r, xi, mu_r, t_time)

    # Convolution to get modal response z_r(t)
    # Using scipy.signal.convolve and multiply by dt for physical scaling
    z_r = convolve(h_r, f_modal_r, mode='full')[:len(t_time)] * dt

    # Add modal contribution to physical response
    # ϑ_c(t) += ψ_ϑc,r * z_r(t)
    theta_c_response += psi_theta_c * z_r

# Plot ϑ_c(t)
fig, ax = plt.subplots(figsize=(12, 5))
ax.plot(t_time, theta_c_response, 'b-', linewidth=1)
ax.set_xlabel('Time [s]')
ax.set_ylabel('θ_c(t) [rad]')
ax.set_title('Assignment 3.1: Main Body Rotation θ_c(t) - No TMD', fontweight='bold')
```

```python
ax.grid(True, alpha=0.3)
ax.axhline(y=0, color='black', linewidth=0.5)
ax.axvline(x=20, color='red', linestyle=':', alpha=0.5, label='End of torque')
ax.legend()
plt.tight_layout()
plt.show()
```




```
Torque signal constructed:
  Time range: 0 to 60 s, dt = 0.01 s, 6001 samples
  Max torque: 40.0 Nm at t = 5.00 s
  Min torque: -40.0 Nm at t = 15.00 s

Using 7 flexible modes (excluding 3 rigid body modes)
```



In [80]:

```
# Analysis
print("\nANALYSIS OF θ_c(t):")
print(f"""
Results:
  • Maximum rotation: {np.max(theta_c_response):.6f} rad ({np.degrees(np.max(theta_c_response)):.4f}°)
  • Minimum rotation: {np.min(theta_c_response):.6f} rad ({np.degrees(np.min(theta_c_response)):.4f}°)
  • Peak-to-peak: {np.max(theta_c_response) - np.min(theta_c_response):.6f} rad

Observations:
  1. During 0-20s: The system responds to the applied torque
  2. After 20s: Free vibration (no more input), response decays due to damping
  3. Multiple frequency content visible in the oscillations
  4. Rigid body modes are excluded to avoid unbounded (infinite) response
""")
```

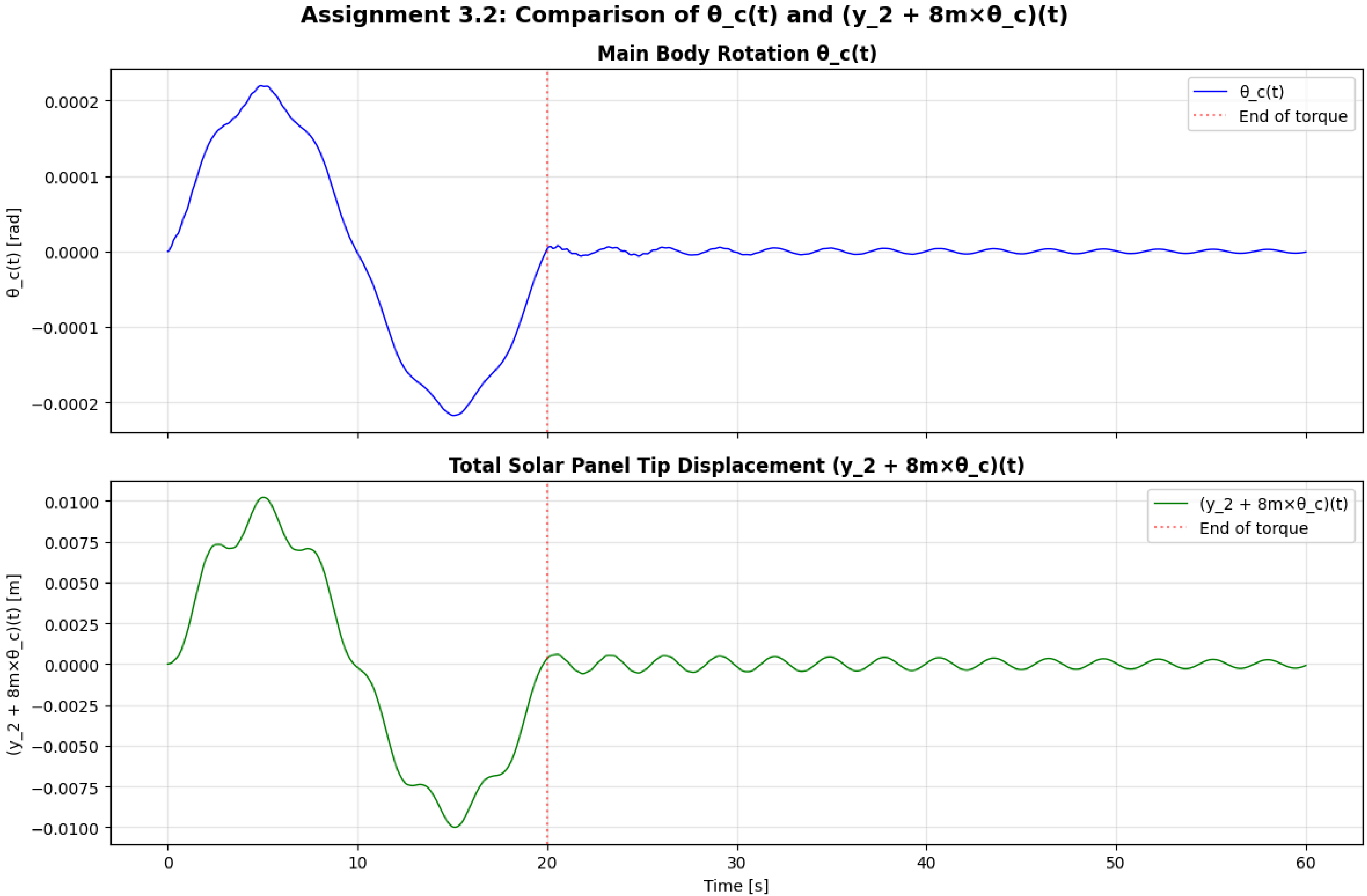
```
ANALYSIS OF θ_c(t):

Results:
  • Maximum rotation: 0.000220 rad (0.0126°)
  • Minimum rotation: -0.000218 rad (-0.0125°)
  • Peak-to-peak: 0.000438 rad

Observations:
  1. During 0-20s: The system responds to the applied torque
  2. After 20s: Free vibration (no more input), response decays due to damping
  3. Multiple frequency content visible in the oscillations
  4. Rigid body modes are excluded to avoid unbounded (infinite) response
```

> **Assignment 3.2: Compute and represent $(y_2(t) + 8\text{m} \times \theta_c(t))$ for a duration of 60 seconds using the same methodology. Comment on the differences between this curve and the one computed in subquestion 1, and give their physical meaning.**

In [81]:

```python
# Compute the TOTAL displacement of the solar panel tip.

# DOF indices
idx_y2 = 4  # y_2 (left solar panel tip)
idx_theta_c = 2  # ϑ_c (main body rotation)

# Compute y_2(t) and ϑ_c(t) separately, then combine
y2_response = np.zeros_like(t_time)

for mode_idx in flexible_mode_indices:
    # Modal parameters
    omega_r = omega_n[mode_idx]
    psi_r = eigenvectors[:, mode_idx]
    mu_r = psi_r.T @ M @ psi_r  # Modal mass

    # Mode shape at input (ϑ_c) and output (y_2)
    psi_theta_c = psi_r[idx_theta_c]
    psi_y2 = psi_r[idx_y2]

    # Modal force: f_r(t) = ψ_ϑc,r * M_c(t)
    f_modal_r = psi_theta_c * M_c_signal

    # Modal impulse response
    h_r = modal_impulse_response(omega_r, xi, mu_r, t_time)
```

```python
    # Convolution to get modal response z_r(t)
    z_r = convolve(h_r, f_modal_r, mode='full')[:len(t_time)] * dt

    # Add modal contribution to y_2(t)
    y2_response += psi_y2 * z_r

# Combine: (y_2 + 8m × ϑ_c)
y2_plus_8theta = y2_response + L_solar * theta_c_response

# Plot comparison
fig, axes = plt.subplots(2, 1, figsize=(12, 8), sharex=True)

# ϑ_c(t)
axes[0].plot(t_time, theta_c_response, 'b-', linewidth=1, label='θ_c(t)')
axes[0].set_ylabel('θ_c(t) [rad]')
axes[0].set_title('Main Body Rotation θ_c(t)', fontweight='bold')
axes[0].grid(True, alpha=0.3)
axes[0].axvline(x=20, color='red', linestyle=':', alpha=0.5, label='End of torque')
axes[0].legend()

# (y_2 + 8×ϑ_c)(t)
axes[1].plot(t_time, y2_plus_8theta, 'g-', linewidth=1, label='(y_2 + 8m×θ_c)(t)')
axes[1].set_xlabel('Time [s]')
axes[1].set_ylabel('(y_2 + 8m×θ_c)(t) [m]')
axes[1].set_title('Total Solar Panel Tip Displacement (y_2 + 8m×θ_c)(t)', fontweight='bold')
axes[1].grid(True, alpha=0.3)
axes[1].axvline(x=20, color='red', linestyle=':', alpha=0.5, label='End of torque')
axes[1].legend()

plt.suptitle('Assignment 3.2: Comparison of θ_c(t) and (y_2 + 8m×θ_c)(t)', fontsize=14, fontweight='bold')
plt.tight_layout()
plt.show()
```



In [82]:
```
# Analysis
print("\nANALYSIS AND PHYSICAL INTERPRETATION:")
print(f"""
Results:
  • θ_c(t):
    - Maximum: {np.max(theta_c_response):.6f} rad ({np.degrees(np.max(theta_c_response)):.4f}°)
```

```
   - Peak-to-peak: {np.max(theta_c_response) - np.min(theta_c_response):.6f} rad

  • (y_2 + 8m×θ_c)(t):
   - Maximum: {np.max(y2_plus_8theta):.6f} m ({np.max(y2_plus_8theta)*1000:.2f} mm)
   - Peak-to-peak: {np.max(y2_plus_8theta) - np.min(y2_plus_8theta):.6f} m

DIFFERENCES AND PHYSICAL MEANING:
=================================

1. θ_c(t) - Main Body Rotation:
   - Represents the rotation of the satellite's main body
   - Dominated by lower-frequency, large-inertia motion
   - Contains contribution from ALL modes excited by torque
   - Decays after the input stops (t > 20s)

2. (y_2 + 8m×θ_c)(t) - Total Solar Panel Tip Motion:
   - Represents the ABSOLUTE position of the solar panel tip
   - Combines elastic deformation (y_2) with rotation contribution (8×θ_c)
   - Shows HIGHER FREQUENCY oscillations (solar panel bending modes)
   - More susceptible to fatigue damage due to higher cycle count

3. KEY DIFFERENCES:
   - (y_2 + 8m×θ_c) oscillates with HIGHER FREQUENCY than θ_c alone
   - This is because the solar panel flexible modes are excited
   - The solar panel acts as a "cantilever" attached to the rotating body
   - Small θ_c changes are amplified by the 8m arm length

4. ENGINEERING IMPORTANCE:
   - θ_c affects pointing accuracy of onboard instruments
   - (y_2 + 8m×θ_c) determines solar cell fatigue and wiring stress
   - TMD design targeted the dominant mode in (y_2 - y_c)/F_c
   - Both responses need to be minimized for satellite health
""")
```

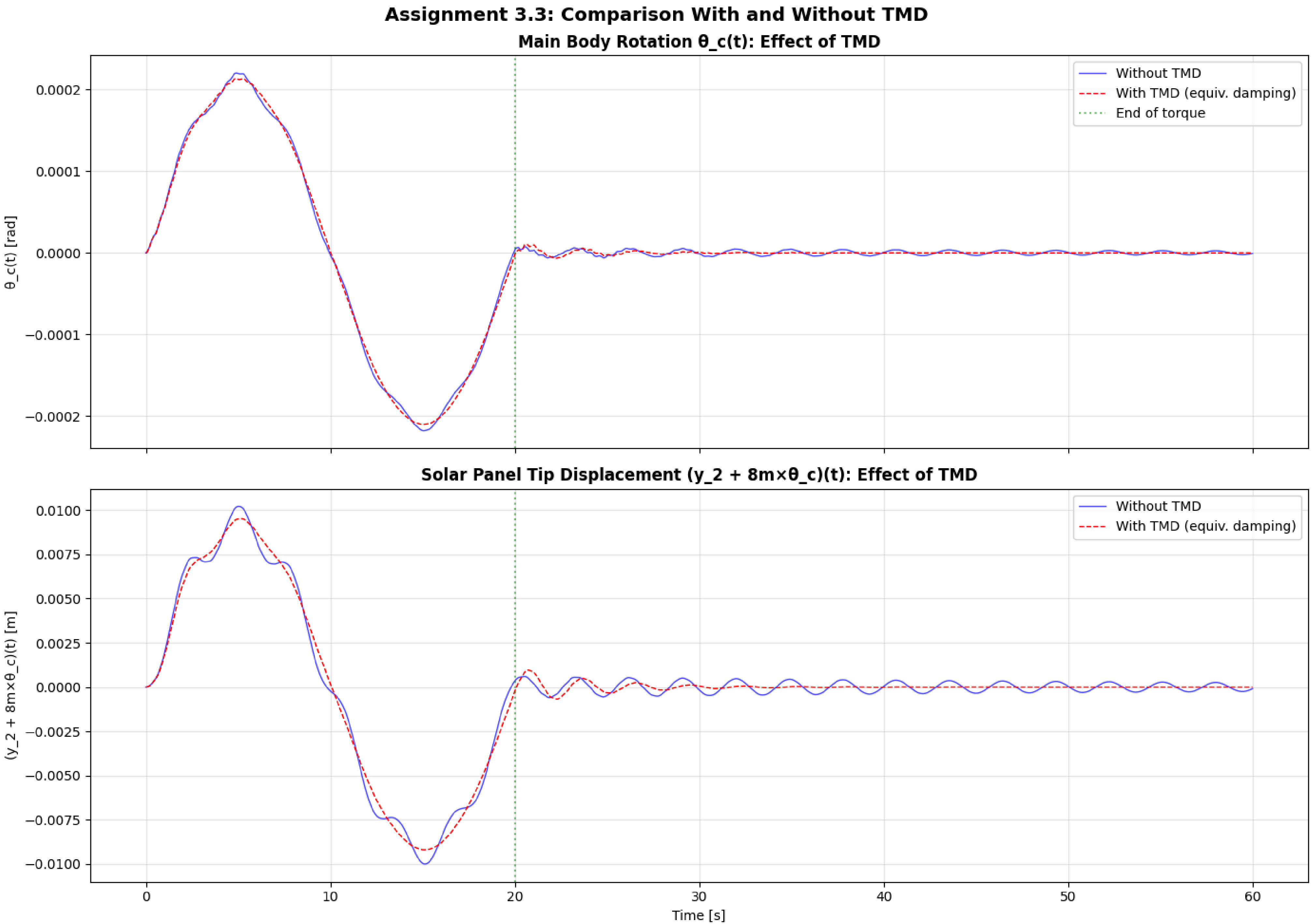
```
ANALYSIS AND PHYSICAL INTERPRETATION:

Results:
  • θ_c(t):
    - Maximum: 0.000220 rad (0.0126°)
    - Peak-to-peak: 0.000438 rad

  • (y_2 + 8m×θ_c)(t):
    - Maximum: 0.010210 m (10.21 mm)
    - Peak-to-peak: 0.020216 m

DIFFERENCES AND PHYSICAL MEANING:
=================================

1. θ_c(t) - Main Body Rotation:
   - Represents the rotation of the satellite's main body
   - Dominated by lower-frequency, large-inertia motion
   - Contains contribution from ALL modes excited by torque
   - Decays after the input stops (t > 20s)

2. (y_2 + 8m×θ_c)(t) - Total Solar Panel Tip Motion:
   - Represents the ABSOLUTE position of the solar panel tip
   - Combines elastic deformation (y_2) with rotation contribution (8×θ_c)
   - Shows HIGHER FREQUENCY oscillations (solar panel bending modes)
   - More susceptible to fatigue damage due to higher cycle count

3. KEY DIFFERENCES:
   - (y_2 + 8m×θ_c) oscillates with HIGHER FREQUENCY than θ_c alone
   - This is because the solar panel flexible modes are excited
   - The solar panel acts as a "cantilever" attached to the rotating body
   - Small θ_c changes are amplified by the 8m arm length

4. ENGINEERING IMPORTANCE:
   - θ_c affects pointing accuracy of onboard instruments
   - (y_2 + 8m×θ_c) determines solar cell fatigue and wiring stress
   - TMD design targeted the dominant mode in (y_2 - y_c)/F_c
   - Both responses need to be minimized for satellite health
```

Assignment 3.3: Use the equivalent damping estimated from both transfer functions in subquestion 4 of the questions related to the TMD (Assignment 2.4) to approximate the effect of the two TMDs, and compute and plot again $\theta_c(t)$, and $(y_2(t) + 8\mathrm{m} \times \theta_c(t))$ (superpose the curves to the ones without the TMDs). Comment on the effect of the TMD on $\theta_c(t)$ and $(y_2(t) + 8\mathrm{m} \times \theta_c(t))$, and give a physical interpretation.

In [83]:
```python
# Use the equivalent damping ratios from Assignment 2.4 to approximate TMD effect.

print(f"\nEquivalent damping ratios from Assignment 2.4:")
print(f"  • For (y_2-y_c)/F_c:     ξ_equiv = {xi_equiv_Fc:.4f} (vs original ξ = {xi:.4f})")
print(f"  • For (y_2+8θ_c)/M_c:    ξ_equiv = {xi_equiv_Mc:.4f}")

# Recompute ϑ_c(t) with equivalent damping (use the Mc-based equivalent damping)
theta_c_with_TMD = np.zeros_like(t_time)
y2_with_TMD = np.zeros_like(t_time)

mode_damped = dominant_mode_Mc - 1  # mode linked to the (y2+8ϑ)/Mc peak

for mode_idx in flexible_mode_indices:
    omega_r = omega_n[mode_idx]
    psi_r = eigenvectors[:, mode_idx]
    mu_r = psi_r.T @ M @ psi_r

    psi_theta_c = psi_r[idx_theta_c]
    psi_y2 = psi_r[idx_y2]

    # Modal force
    f_modal_r = psi_theta_c * M_c_signal

    # Use both equivalent damping estimates (from subquestion 2.4)
    if mode_idx == dominant_mode_Fc - 1:
        xi_mode = xi_equiv_Fc
    elif mode_idx == dominant_mode_Mc - 1:
        xi_mode = xi_equiv_Mc
    else:
        xi_mode = xi

    # Modal impulse response with (possibly) increased damping
    h_r_TMD = modal_impulse_response(omega_r, xi_mode, mu_r, t_time)

    # Convolution
    z_r_TMD = convolve(h_r_TMD, f_modal_r, mode='full')[:len(t_time)] * dt
```

```python
    # Accumulate contributions
    theta_c_with_TMD += psi_theta_c * z_r_TMD
    y2_with_TMD += psi_y2 * z_r_TMD

# Combine for total solar panel tip motion
y2_plus_8theta_with_TMD = y2_with_TMD + L_solar * theta_c_with_TMD

# Plot comparison: With and Without TMD
fig, axes = plt.subplots(2, 1, figsize=(14, 10), sharex=True)

# θ_c(t)
axes[0].plot(t_time, theta_c_response, 'b-', linewidth=1, alpha=0.7, label='Without TMD')
axes[0].plot(t_time, theta_c_with_TMD, 'r--', linewidth=1, label='With TMD (equiv. damping)')
axes[0].set_ylabel('θ_c(t) [rad]')
axes[0].set_title('Main Body Rotation θ_c(t): Effect of TMD', fontweight='bold')
axes[0].grid(True, alpha=0.3)
axes[0].axvline(x=20, color='green', linestyle=':', alpha=0.5, label='End of torque')
axes[0].legend()

# (y_2 + 8×θ_c)(t)
axes[1].plot(t_time, y2_plus_8theta, 'b-', linewidth=1, alpha=0.7, label='Without TMD')
axes[1].plot(t_time, y2_plus_8theta_with_TMD, 'r--', linewidth=1, label='With TMD (equiv. damping)')
axes[1].set_xlabel('Time [s]')
axes[1].set_ylabel('(y_2 + 8m×θ_c)(t) [m]')
axes[1].set_title('Solar Panel Tip Displacement (y_2 + 8m×θ_c)(t): Effect of TMD', fontweight='bold')
axes[1].grid(True, alpha=0.3)
axes[1].axvline(x=20, color='green', linestyle=':', alpha=0.5)
axes[1].legend()

plt.suptitle('Assignment 3.3: Comparison With and Without TMD', fontsize=14, fontweight='bold')
plt.tight_layout()
plt.show()
```

```
Equivalent damping ratios from Assignment 2.4:
  • For (y_2-y_c)/F_c:     ξ_equiv = 0.0858 (vs original ξ = 0.0100)
  • For (y_2+8θ_c)/M_c:    ξ_equiv = 0.1085
```



In [84]:
```
# Analysis and physical interpretation
print("\nANALYSIS OF TMD EFFECT:")
```

```python
# Compute metrics
# Peak values
peak_theta_no_TMD = np.max(np.abs(theta_c_response))
peak_theta_with_TMD = np.max(np.abs(theta_c_with_TMD))
peak_y2_no_TMD = np.max(np.abs(y2_plus_8theta))
peak_y2_with_TMD = np.max(np.abs(y2_plus_8theta_with_TMD))

# Reduction percentages
reduction_theta = (1 - peak_theta_with_TMD / peak_theta_no_TMD) * 100
reduction_y2 = (1 - peak_y2_with_TMD / peak_y2_no_TMD) * 100

# RMS values (measure of overall vibration energy)
rms_theta_no_TMD = np.sqrt(np.mean(theta_c_response**2))
rms_theta_with_TMD = np.sqrt(np.mean(theta_c_with_TMD**2))
rms_y2_no_TMD = np.sqrt(np.mean(y2_plus_8theta**2))
rms_y2_with_TMD = np.sqrt(np.mean(y2_plus_8theta_with_TMD**2))

print(f"""
QUANTITATIVE COMPARISON:
========================

θ_c(t) - Main Body Rotation:
  • Peak without TMD: {peak_theta_no_TMD:.6f} rad ({np.degrees(peak_theta_no_TMD):.4f}°)
  • Peak with TMD:    {peak_theta_with_TMD:.6f} rad ({np.degrees(peak_theta_with_TMD):.4f}°)
  • PEAK REDUCTION:   {reduction_theta:.1f}%
  • RMS reduction:    {(1 - rms_theta_with_TMD/rms_theta_no_TMD)*100:.1f}%

(y_2 + 8m×θ_c)(t) - Solar Panel Tip:
  • Peak without TMD: {peak_y2_no_TMD:.6f} m ({peak_y2_no_TMD*1000:.2f} mm)
  • Peak with TMD:    {peak_y2_with_TMD:.6f} m ({peak_y2_with_TMD*1000:.2f} mm)
  • PEAK REDUCTION:   {reduction_y2:.1f}%
  • RMS reduction:    {(1 - rms_y2_with_TMD/rms_y2_no_TMD)*100:.1f}%

PHYSICAL INTERPRETATION:
========================
1) Here we observe a MODERATE reduction of the maximum peaks (≈ a few %).
   This is consistent with the fact that M_c(t) is a slow signal (20 s duration),
   so its main frequency content is at low frequencies.
   The TMD is mainly effective near its tuning band (~0.3–0.36 Hz).

2) The "equivalent damping" approach is a local approximation around the targeted resonance:
 - it does not reproduce the two-peak split,
 - it only modifies the selected modes,
 - and if the excitation does not strongly excite that frequency band,
   the time-domain maximum can remain only slightly reduced.
```

```
3) Physically, the TMD dissipates energy primarily from the solar-array flexible modes.
   The small improvement on θ_c(t) comes from the coupling between array flexion and bus rotation.
""")
```

```
ANALYSIS OF TMD EFFECT:

QUANTITATIVE COMPARISON:
========================

θ_c(t) - Main Body Rotation:
  • Peak without TMD: 0.000220 rad (0.0126°)
  • Peak with TMD:    0.000213 rad (0.0122°)
  • PEAK REDUCTION:   3.0%
  • RMS reduction:    0.2%

(y_2 + 8m×θ_c)(t) - Solar Panel Tip:
  • Peak without TMD: 0.010210 m (10.21 mm)
  • Peak with TMD:    0.009515 m (9.52 mm)
  • PEAK REDUCTION:   6.8%
  • RMS reduction:    0.7%

PHYSICAL INTERPRETATION:
========================
1) Here we observe a MODERATE reduction of the maximum peaks (≈ a few %).
   This is consistent with the fact that M_c(t) is a slow signal (20 s duration),
   so its main frequency content is at low frequencies.
   The TMD is mainly effective near its tuning band (~0.3–0.36 Hz).

2) The "equivalent damping" approach is a local approximation around the targeted resonance:
 - it does not reproduce the two-peak split,
 - it only modifies the selected modes,
 - and if the excitation does not strongly excite that frequency band,
   the time-domain maximum can remain only slightly reduced.

3) Physically, the TMD dissipates energy primarily from the solar-array flexible modes.
   The small improvement on θ_c(t) comes from the coupling between array flexion and bus rotation.
```

References

- [1] J. Wei, W. Liu, J. Liu, and T. Yu. Dynamic modeling and analysis of spacecraft with multiple large flexible structures. Actuators, 12(7)(286), 2023.
- [2] J. Guyan. Reduction of stiffness and mass matrices. AIAA journal, 3:380–380, 1965.