```
"""
GRACE-FO Dynamics Assignment
Satellite Orbit Determination - Assignment 1

This script computes perturbing accelerations and performs orbit propagation
for the GRACE-FO satellite using different dynamical models.
"""

import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import solve_ivp

# ============================================================================
# Constants and Parameters
# ============================================================================

# Spacecraft parameters
m = 600.0                    # Spacecraft mass [kg]
A = 1.0                      # Reference surface area [m^2]
C_D = 2.6                    # Aerodynamic drag coefficient [-]

# Environmental parameters
omega_E = 7.292115e-05       # Earth rotation rate (z-axis) [rad/s]
C_20 = -4.841692151273e-04   # Gravity field coefficient (J2) [-]
rho = 1.0e-11                # Atmospheric density [kg/m^3]

# Physical constants
c = 299792458                # Speed of light [m/s]
GM = 3.986004415e5           # Earth's gravitational constant [km^3/s^2]
R_E = 6378.1366              # Earth's reference radius [km]

# Earth rotation vector (ECEF frame)
omega_vec = np.array([0, 0, omega_E])  # [rad/s]

# ============================================================================
# Load Data
# ============================================================================

print("Loading orbital data...")
t = np.loadtxt('data/t.txt')      # epochs [s]
r = np.loadtxt('data/r.txt')      # precise positions [km]
v = np.loadtxt('data/v.txt')      # precise velocities [km/s]

print(f"Data loaded: {len(t)} epochs")
print(f"Time span: {t[0]} to {t[-1]} seconds ({(t[-1]-t[0])/3600:.2f} hours)")
print(f"First position: {r[0]} km")
print(f"First velocity: {v[0]} km/s")

# ============================================================================
# Part 1: Acceleration Calculations
# ============================================================================

def centrifugal_coriolis_acceleration(r_ecef, v_ecef, omega_vec):
    """
    Calculate centrifugal and Coriolis accelerations in ECEF frame.

    In a rotating frame, the acceleration is:
    a = -2*omega * v_ecef - omega**2 * r_ecef

    where:
    - First term is Coriolis acceleration
    - Second term is centrifugal acceleration

    Args:
        r_ecef: Position vector in ECEF frame [km]
        v_ecef: Velocity vector in ECEF frame [km/s]
        omega_vec: Earth rotation vector [rad/s]
```

```
    Returns:
        Total acceleration [km/s^2]
    """
    # Coriolis: -2 * (Omega x v_rot)
    coriolis = -2 * np.cross(omega_vec, v_ecef)

    # Centrifugal: - Omega x (Omega x r)
    centrifugal = - np.cross(omega_vec, np.cross(omega_vec, r_ecef))

    return coriolis + centrifugal

def j2_acceleration(r_ecef, GM, R_E, C_20):
    """
    Calculate acceleration due to Earth's oblateness (J2 effect).

    Using the formula from course notes:
    a = -(GM/R^3) * C_20 * sqrt(45) * (R/|r|)^5 *
        (r * (5/2 * (z/|r|)^2 - 1/2) - [0, 0, z])

    Args:
        r_ecef: Position vector in ECEF frame [km]
        GM: Earth's gravitational constant [km^3/s^2]
        R_E: Earth's reference radius [km]
        C_20: Gravity field coefficient [-]

    Returns:
        J2 acceleration [km/s^2]
    """
    x, y, z = r_ecef
    r_norm = np.linalg.norm(r_ecef)

    # Common factor: -(GM/R^3) * C_20 * sqrt(45) * (R/|r|)^5
    factor = -(GM / R_E**3) * C_20 * np.sqrt(45) * (R_E / r_norm)**5

    # (z/|r|)^2 term
    z_norm_sq = (z / r_norm)**2

    # Scalar term: (5/2 * (z/|r|)^2 - 1/2)
    scalar = 5/2 * z_norm_sq - 1/2

    # Vector part: r * scalar - [0, 0, z]
    a_vec = r_ecef * scalar - np.array([0, 0, z])

    return factor * a_vec


def drag_acceleration(r_ecef, v_ecef, omega_vec, m, A, C_D, rho):
    """
    Calculate acceleration due to atmospheric drag.

    The drag acceleration is:
    a_drag = -0.5 * (rho * C_D * A / m) * v_rel * |v_rel|

    where v_rel is the velocity relative to the atmosphere (rotating with Earth).
    v_rel = v_ecef

    Args:
        r_ecef: Position vector in ECEF frame [km]
        v_ecef: Velocity vector in ECEF frame [km/s]
        omega_vec: Earth rotation vector [rad/s]
        m: Spacecraft mass [kg]
        A: Reference surface area [m^2]
        C_D: Drag coefficient [-]
        rho: Atmospheric density [kg/m^3]

    Returns:
        Drag acceleration [km/s^2]
    """
```

```
    # Velocity relative to atmosphere
    v_rel = v_ecef  # [km/s]
    v_rel_norm = np.linalg.norm(v_rel)

    # Convert units: km -> m for consistency
    # rho is in kg/m^3, A in m^2, v_rel in km/s
    # Result needs to be in km/s^2

    # Drag force coefficient
    # Note: 1 km = 1000 m, so v_rel in m/s = v_rel * 1000
    drag_coeff = 0.5 * rho * C_D * A / m  # [m^2/kg]

    # Drag acceleration = -drag_coeff * v_rel * |v_rel|
    # v_rel is in km/s, but rho*A/m is in SI, so we need conversion
    # drag_coeff has units [1/m], v_rel [km/s] = [1000 m/s]
    # So: drag_coeff * (v_rel*1000)^2 gives [m/s^2]
    # We want [km/s^2], so divide by 1000

    a_drag = -drag_coeff * (v_rel * 1000) * (v_rel_norm * 1000) / 1000  # [km/s^2]

    return a_drag


# Calculate accelerations at first epoch
print("\n" + "="*70)
print("PART 1: PERTURBING ACCELERATION CALCULATIONS AT FIRST EPOCH")
print("="*70)

r0 = r[0]  # Position at first epoch [km]
v0 = v[0]  # Velocity at first epoch [km/s]

# Question 1: Centrifugal and Coriolis
print("\nQuestion 1: Centrifugal and Coriolis Effects")
print("-" * 70)
a_centrifugal_coriolis = centrifugal_coriolis_acceleration(r0, v0, omega_vec)
# Convert to mm/s^2
a_cc_mm = a_centrifugal_coriolis * 1e6  # km/s^2 to mm/s^2
print(f"X-axis acceleration: {a_cc_mm[0]:.4g} mm/s^2")
print(f"Y-axis acceleration: {a_cc_mm[1]:.4g} mm/s^2")
print(f"Z-axis acceleration: {a_cc_mm[2]:.4g} mm/s^2")

# Question 2: Earth's flattening (J2)
print("\nQuestion 2: Earth's Flattening (J2 Effect)")
print("-" * 70)
a_j2 = j2_acceleration(r0, GM, R_E, C_20)
# Convert to mm/s^2
a_j2_mm = a_j2 * 1e6  # km/s^2 to mm/s^2
print(f"X-axis acceleration: {a_j2_mm[0]:.4g} mm/s^2")
print(f"Y-axis acceleration: {a_j2_mm[1]:.4g} mm/s^2")
print(f"Z-axis acceleration: {a_j2_mm[2]:.4g} mm/s^2")

# Question 3: Atmospheric drag
print("\nQuestion 3: Atmospheric Drag")
print("-" * 70)
a_drag = drag_acceleration(r0, v0, omega_vec, m, A, C_D, rho)
# Convert to mm/s^2
a_drag_mm = a_drag * 1e6  # km/s^2 to mm/s^2
print(f"X-axis acceleration: {a_drag_mm[0]:.4g} mm/s^2")
print(f"Y-axis acceleration: {a_drag_mm[1]:.4g} mm/s^2")
print(f"Z-axis acceleration: {a_drag_mm[2]:.4g} mm/s^2")

# Summary
print("\nSummary of Acceleration Magnitudes:")
print("-" * 70)
print(f"Centrifugal & Coriolis: {np.linalg.norm(a_cc_mm):.4g} mm/s^2")
print(f"J2 (Earth flattening):  {np.linalg.norm(a_j2_mm):.4g} mm/s^2")
print(f"Atmospheric drag:       {np.linalg.norm(a_drag_mm):.4g} mm/s^2")
```

```
# ============================================================================
# Part 2: Orbit Propagation
# ============================================================================

print("\n" + "="*70)
print("PART 2: ORBIT PROPAGATION")
print("="*70)

def two_body_acceleration(r, GM):
    """Calculate two-body gravitational acceleration."""
    r_norm = np.linalg.norm(r)
    return -GM * r / r_norm**3


def dynamics_model_1(t, state, GM):
    """
    Dynamical Model 1: No perturbing forces, ignore frame rotation.
    Pure two-body problem in inertial frame.
    """
    r = state[0:3]
    v = state[3:6]

    a = two_body_acceleration(r, GM)

    return np.concatenate([v, a])


def dynamics_model_2(t, state, GM, omega_vec):
    """
    Dynamical Model 2: No perturbing forces, account for frame rotation.

    Complete rotating frame equation:
    a_rot = -omega**2 * r_rot - 2omega x v_rot + a_in^rot
    where a_in^rot = gravitational acceleration (inertial) observed in rotating frame
    """
    r = state[0:3]
    v = state[3:6]

    # Inertial acceleration (gravity) observed in rotating frame
    a_gravity = two_body_acceleration(r, GM)

    # Centrifugal and Coriolis effects
    a_rotation = centrifugal_coriolis_acceleration(r, v, omega_vec)

    # Total acceleration in rotating frame
    a_total = a_gravity + a_rotation

    return np.concatenate([v, a_total])


def dynamics_model_3(t, state, GM, omega_vec, R_E, C_20):
    """
    Dynamical Model 3: Account for Earth's flattening and frame rotation.

    Complete rotating frame equation:
    a_rot = -omega**2 * r_rot - 2*omega x v_rot + a_in^rot
    where a_in^rot includes gravity + J2 perturbation
    """
    r = state[0:3]
    v = state[3:6]

    # Inertial accelerations observed in rotating frame
    a_gravity = two_body_acceleration(r, GM)
    a_j2_pert = j2_acceleration(r, GM, R_E, C_20)

    # Centrifugal and Coriolis effects
    a_rotation = centrifugal_coriolis_acceleration(r, v, omega_vec)
```

```
    # Total acceleration in rotating frame
    a_total = a_gravity + a_j2_pert + a_rotation

    return np.concatenate([v, a_total])


def dynamics_model_4(t, state, GM, omega_vec, R_E, C_20, m, A, C_D, rho):
    """
    Dynamical Model 4: Account for Earth's flattening, frame rotation, and drag.

    Complete rotating frame equation:
    a_rot = -omega**2 * r_rot - 2*omega x v_rot + a_in^rot
    where a_in^rot includes gravity + J2 + drag
    """
    r = state[0:3]
    v = state[3:6]

    # Inertial accelerations observed in rotating frame
    a_gravity = two_body_acceleration(r, GM)
    a_j2_pert = j2_acceleration(r, GM, R_E, C_20)
    a_drag_pert = drag_acceleration(r, v, omega_vec, m, A, C_D, rho)

    # Centrifugal and Coriolis effects
    a_rotation = centrifugal_coriolis_acceleration(r, v, omega_vec)

    # Total acceleration in rotating frame
    a_total = a_gravity + a_j2_pert + a_drag_pert + a_rotation

    return np.concatenate([v, a_total])


# Initial conditions
state0 = np.concatenate([r[0], v[0]])
t_span = (t[0], t[-1])
t_eval = t

print("\nPropagating orbits with different dynamical models...")

# Model 1: No perturbations, no rotation
print("\nModel 1: No perturbing forces, ignore frame rotation...")
sol1 = solve_ivp(dynamics_model_1, t_span, state0, method='DOP853',
                 t_eval=t_eval, args=(GM,), rtol=1e-12, atol=1e-12)

# Model 2: No perturbations, with rotation
print("Model 2: No perturbing forces, account for frame rotation...")
sol2 = solve_ivp(dynamics_model_2, t_span, state0, method='DOP853',
                 t_eval=t_eval, args=(GM, omega_vec), rtol=1e-12, atol=1e-12)

# Model 3: With J2 and rotation
print("Model 3: Earth's flattening and frame rotation...")
sol3 = solve_ivp(dynamics_model_3, t_span, state0, method='DOP853',
                 t_eval=t_eval, args=(GM, omega_vec, R_E, C_20), rtol=1e-12, atol=1e-12)

# Model 4: With J2, rotation, and drag
print("Model 4: Earth's flattening, frame rotation, and atmospheric drag...")
sol4 = solve_ivp(dynamics_model_4, t_span, state0, method='DOP853',
                 t_eval=t_eval, args=(GM, omega_vec, R_E, C_20, m, A, C_D, rho),
                 rtol=1e-12, atol=1e-12)

print("All propagations complete!")

# ============================================================================
# Calculate Errors
# ============================================================================

print("\nCalculating position and velocity errors...")

# Extract propagated states
```

```
r1 = sol1.y[0:3, :].T
v1 = sol1.y[3:6, :].T

r2 = sol2.y[0:3, :].T
v2 = sol2.y[3:6, :].T

r3 = sol3.y[0:3, :].T
v3 = sol3.y[3:6, :].T

r4 = sol4.y[0:3, :].T
v4 = sol4.y[3:6, :].T

# Calculate position errors (L2 norm)
pos_err1 = np.linalg.norm(r1 - r, axis=1)
pos_err2 = np.linalg.norm(r2 - r, axis=1)
pos_err3 = np.linalg.norm(r3 - r, axis=1)
pos_err4 = np.linalg.norm(r4 - r, axis=1)

# Calculate velocity errors (L2 norm)
vel_err1 = np.linalg.norm(v1 - v, axis=1)
vel_err2 = np.linalg.norm(v2 - v, axis=1)
vel_err3 = np.linalg.norm(v3 - v, axis=1)
vel_err4 = np.linalg.norm(v4 - v, axis=1)

# Convert time to hours
t_hours = t / 3600.0

# ============================================================================
# Question 5: Plot Errors
# ============================================================================

print("\nGenerating error plots...")

fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(12, 10))

# Position errors
ax1.semilogy(t_hours, pos_err1 * 1000, 'bo', label='Model 1: No perturbations, no rotation'
, markersize=8)
ax1.semilogy(t_hours, pos_err2 * 1000, 'rs', label='Model 2: No perturbations, with rotatio
n', markersize=8)
ax1.semilogy(t_hours, pos_err3 * 1000, 'g^', label='Model 3: With J2 and rotation', markers
ize=8)
ax1.semilogy(t_hours, pos_err4 * 1000, 'md', label='Model 4: With J2, rotation, and drag',
markersize=8)
ax1.set_xlabel('Time [hours]', fontsize=12)
ax1.set_ylabel('Position Error [m]', fontsize=12)
ax1.set_title('Absolute Position Error vs Time', fontsize=14, fontweight='bold')
ax1.legend(fontsize=10, loc='best')
ax1.grid(True, alpha=0.3)

# Velocity errors
ax2.semilogy(t_hours, vel_err1 * 1000, 'bo', label='Model 1: No perturbations, no rotation'
, markersize=8)
ax2.semilogy(t_hours, vel_err2 * 1000, 'rs', label='Model 2: No perturbations, with rotatio
n', markersize=8)
ax2.semilogy(t_hours, vel_err3 * 1000, 'g^', label='Model 3: With J2 and rotation', markers
ize=8)
ax2.semilogy(t_hours, vel_err4 * 1000, 'md', label='Model 4: With J2, rotation, and drag',
markersize=8)
ax2.set_xlabel('Time [hours]', fontsize=12)
ax2.set_ylabel('Velocity Error [m/s]', fontsize=12)
ax2.set_title('Absolute Velocity Error vs Time', fontsize=14, fontweight='bold')
ax2.legend(fontsize=10, loc='best')
ax2.grid(True, alpha=0.3)

plt.tight_layout()
plt.savefig('output/orbit_propagation_errors.png', dpi=300, bbox_inches='tight')
print("Plot saved as 'output/orbit_propagation_errors.png'")
```

```
# ============================================================================
# Question 7: Report Final Errors
# ============================================================================

print("\n" + "="*70)
print("QUESTION 7: ABSOLUTE POSITION ERROR AT LAST EPOCH")
print("="*70)

# Position errors at last epoch in meters
pos_err1_final = pos_err1[-1] * 1000  # km to m
pos_err2_final = pos_err2[-1] * 1000
pos_err3_final = pos_err3[-1] * 1000
pos_err4_final = pos_err4[-1] * 1000

print(f"\nCase 1 (no perturbations, no rotation):      {pos_err1_final:.4g} m")
print(f"Case 2 (no perturbations, with rotation):    {pos_err2_final:.4g} m")
print(f"Case 3 (J2 + rotation):                      {pos_err3_final:.4g} m")
print(f"Case 4 (J2 + rotation + drag):               {pos_err4_final:.4g} m")

# ============================================================================
# Additional Analysis
# ============================================================================

print("\n" + "="*70)
print("ADDITIONAL ANALYSIS")
print("="*70)

# Calculate mean accelerations over the orbit
print("\nMean acceleration magnitudes over entire orbit:")
a_cc_all = np.array([np.linalg.norm(centrifugal_coriolis_acceleration(r[i], v[i], omega_vec
))
                     for i in range(len(t))])
a_j2_all = np.array([np.linalg.norm(j2_acceleration(r[i], GM, R_E, C_20))
                     for i in range(len(t))])
a_drag_all = np.array([np.linalg.norm(drag_acceleration(r[i], v[i], omega_vec, m, A, C_D, r
ho))
                       for i in range(len(t))])

print(f"Centrifugal & Coriolis: {np.mean(a_cc_all)*1e6:.4g} mm/s^2 (mean)")
print(f"J2 (Earth flattening):  {np.mean(a_j2_all)*1e6:.4g} mm/s^2 (mean)")
print(f"Atmospheric drag:       {np.mean(a_drag_all)*1e6:.4g} mm/s^2 (mean)")

# Estimate position error from acceleration
dt = t[-1] - t[0]  # Total time [s]
print(f"\nSimple error estimation (\\Delta r ~= 0.5 * a * \\Delta t^2):")
print(f"From centrifugal/Coriolis: {0.5 * np.mean(a_cc_all) * dt**2:.4g} km = {0.5 * np.mea
n(a_cc_all) * dt**2 * 1000:.4g} m")
print(f"From J2:                   {0.5 * np.mean(a_j2_all) * dt**2:.4g} km = {0.5 * np.mea
n(a_j2_all) * dt**2 * 1000:.4g} m")
print(f"From drag:                 {0.5 * np.mean(a_drag_all) * dt**2:.4g} km = {0.5 * np.m
ean(a_drag_all) * dt**2 * 1000:.4g} m")

print("\n" + "="*70)
print("ANALYSIS COMPLETE!")
print("="*70)
print("\nSummary of outputs:")
print("  - orbit_propagation_errors.png (and .pdf)")
print("  - All numerical answers printed above")
print("\nYou can now answer all questions in the assignment!")
```