```
from functinos import *
from math import asin, pi
import matplotlib.pyplot as plt

# Constants
CAT_NUM = 16

DIAMETER_LOWER = 0.6
DENSITY = 600
BETA_MAX = pi / 2
SPINDLE_PITCH = 3

DELTA_TIME = 0.001    # Defines accuracy

# Parameters
DIAMETER_UPPER = 0.2 - CAT_NUM / 300
HEIGHT_TREE = 33 + CAT_NUM / 3
LIFTING_TIME = 33 + CAT_NUM * 3
ENVIRONMENT_TEMP = 14 - CAT_NUM

START_HEIGHT = 1

STARTING_TEMP = ENVIRONMENT_TEMP
ALPHA = -0.04
BETA = 0.005
EFFICIENCY = 0.92

# General
spec_pitch = SPINDLE_PITCH / 1000 / pi / 2

# Introduction
print()
print("Lifting a tree - Numerical calculation script")
print("=============================================")
print("")
print("Student-Number:", CAT_NUM)
print("")
print("Parameters:")
print(" -> d_t:", DIAMETER_UPPER, "m")
print(" -> D_t:", DIAMETER_LOWER, "m")
print(" -> h_t:", HEIGHT_TREE, "m")
print(" -> t_l:", LIFTING_TIME, "s")
print(" -> p_t:", DENSITY, "kg/m³")
print(" -> h_s:", START_HEIGHT, "m")
print(" -> T_E:", ENVIRONMENT_TEMP, "°C")
print("")

# Cone
v_col = trun_cone_vol(DIAMETER_UPPER / 2, DIAMETER_LOWER / 2,
HEIGHT_TREE)
m_tree = v_col * DENSITY

print("Truncated code:")
print(" -> Volume: \t\t\t", v_col, "m³")
print(" -> Mass: \t\t\t", m_tree, "kg")

h_com = trun_cone_com(DIAMETER_UPPER / 2, DIAMETER_LOWER / 2,
HEIGHT_TREE)
```

```python
print(" -> Center of mass height:\t", h_com, "m")
print()

beta_0 = asin(1 / h_com)
delta_z_s = h_com - START_HEIGHT

e_req = m_tree * G * delta_z_s
p_req = e_req / LIFTING_TIME

print("Approximate motor calculations:")
print(" -> Average theoretical power:\t", p_req, "W")

delta_beta = BETA_MAX - beta_0
omega_av = delta_beta / LIFTING_TIME
omega_max = omega_av * 2

print(" -> Average angluar velocity: \t", omega_av, "rad/s")
print()

# Inertias
i_tree = inertia_tree(HEIGHT_TREE, m_tree)
i_rod = inertia_rod(i_tree, h_com)
i_motor = inertia_motor(i_rod, spec_pitch)

print("Loads:")
print(" -> Tree inertia: \t\t", i_tree, "kgm²")
print(" -> Rod inertia: \t\t", i_rod, "kg")
print(" -> Motor inertia: \t\t", i_motor, "kgm²")

# Load
t_tree_max = tree_load_torque(h_com, m_tree, beta_0)
f_cyl_max = cylinder_load_force(t_tree_max, h_com,
gamma_t(alpha_t(beta_0)))
t_mot_max = motor_load_force(f_cyl_max, spec_pitch)

print(" -> Max motor torque: \t\t", t_mot_max, "Nm")

# Plotting
distance_0 = cyl_length_from_angle(h_com, alpha_t(beta_0))

velocities = [ 0.0 ]
phis = [ 0.0 ]
distances = [ distance_0 ]
accelerations = [ 0.0 ]
torques = [ 0.0 ]
torques_motor = [ ]
torques_load = [ ]

betas = [ beta_0 ]
alphas = [ alpha_t(beta_0) ]
gammas = [ gamma_t(alphas[-1]) ]

amps = [ ]
powers_mech = [ ]
powers_el = [ ]

temps = [ STARTING_TEMP ]
```

```
counter = 0

while True:
    try:
        velocity = velocities[-1]
        torque_load =
motor_load_force(cylinder_load_force(tree_load_torque(h_com, m_tree,
betas[-1]), h_com, gammas[-1]), spec_pitch)
        torque_motor = motor_torque(velocity)
        torque = torque_motor - torque_load

        if torque < 0:
            raise Exception(f"OVERLOAD! Select a different motor or
spindle ... \nReq: {torque_load},  Current: {torque_motor}\nVel:
{velocities[-1]}")

        acceleration = torque / i_motor
        velocity += acceleration * DELTA_TIME
        phi = phis[-1] + velocity * DELTA_TIME
        distance = distance_0 - phi * spec_pitch

        alpha = cyl_angle_from_length(h_com, distance)
        beta = beta_t(alpha)
        gamma = gamma_t(alpha)

        if distance < 0:
            alpha = 0
            beta = pi / 2
            gamma = 0

        amp = motor_amperes(torque_motor)
        power_mech = torque_motor * velocity
        power_el = amp * MOTOR_VOLTAGE

        power_el_help = max(power_el, power_mech)

        temp = temps[-1] + ((temps[-1] - ENVIRONMENT_TEMP) * ALPHA +
(power_el_help - power_mech) * BETA + power_mech * (1 - EFFICIENCY) *
BETA) * DELTA_TIME

        # if power_mech > power_el:
        #     power_el = power_mech

        velocities.append(velocity)
        torques.append(torque)
        torques_motor.append(torque_motor)
        torques_load.append(torque_load)
        accelerations.append(acceleration)
        phis.append(phi)
        distances.append(distance)
        alphas.append(alpha)
        betas.append(beta)
        gammas.append(gamma)

        amps.append(amp)
        powers_mech.append(power_mech)
        powers_el.append(power_el)
```

```
        temps.append(temp)

        if beta >= (pi / 2):
            break

        counter += 1
    except Exception as e:
        print("Error in iteration number:", counter)
        print(e)
        break


def plot(ydata : list, color = "black", label=""):
    low_len = len(ydata)

    print(f" -> Ploting data with color '{color}' with {low_len} nodes")

    plt.plot([ DELTA_TIME*i for i in range(0, low_len) ], ydata, color, 
label=label)


# Create plot
# plot(velocities, "blue", "Velocity") # Velocity plot

# plot(torques_motor, "green", "Motor torque") # Motor and load torque 
plot
# plot(torques_load, "red", "Load torque")

# plot(amps, "red", "Ampere") # Ampere

# plot(powers_mech, "red", "Mechanical power")
# plot(powers_el, "green", "Electrical power")

plot(temps, "blue", "Temperature")

plt.legend()
plt.show()
```