```python
import math
import numpy as np
import scipy.optimize

# -----------------------------
# Configuration Dictionary
# -----------------------------
CONFIG = {
    "gamma_air": 1.4,          # Specific heat ratio for air
    "gamma_gas": 1.33,         # Specific heat ratio for combustion products
    "cp_air": 1005,            # Specific heat of air (J/kg·K)
    "cp_gas": 1200,            # Specific heat of combustion products (J/kg·K)
    "QR": 43e6,                # Heat of reaction (J/kg fuel)
    "R_gas": 287,              # Gas constant (J/kg·K)
    "eta_b": 0.98,             # Combustor efficiency
    "eta_n": 0.98,             # Nozzle efficiency
    "eta_combustor": 0.95,     # Combustor total pressure loss factor
    "P_ambient": 1e5           # Ambient pressure (Pa)
}

# -----------------------------
# Input Validation
# -----------------------------
def validate_inputs(M, P, T, theta_deg=None):
    """Validate physical inputs for flow calculations."""
    if M <= 0:
        raise ValueError("Mach number must be positive.")
    if P <= 0 or T <= 0:
        raise ValueError("Pressure and temperature must be positive.")
    if theta_deg is not None and (theta_deg < 0 or theta_deg > 90):
        raise ValueError("Deflection angle must be between 0° and 90°.")

# -----------------------------
# Total Conditions
# -----------------------------
def T0(M, T, gamma):
    """Calculate total temperature."""
    validate_inputs(M, 1, T)
    return T * (1 + ((gamma - 1) / 2) * M**2)

def P0(M, P, T, gamma):
    """Calculate total pressure."""
    validate_inputs(M, P, T)
    return P * (T0(M, T, gamma) / T) ** (gamma / (gamma - 1))

# -----------------------------
# θ-β-M Relation Solver
# -----------------------------
def beta_from_theta_mach(M, theta_deg, gamma):
    """Calculate shock angle β for given Mach number and deflection angle."""
    validate_inputs(M, 1, 1, theta_deg)
    if M <= 1:
        raise ValueError("Oblique shock requires supersonic flow (M > 1).")
    theta = math.radians(theta_deg)

    def func(beta):
        left = math.tan(theta)
        right = (2 / math.tan(beta)) * ((M**2 * math.sin(beta)**2 - 1) /
                                        (M**2 * (gamma + math.cos(2 * beta)) + 2))
        return left - right

    beta_min = np.arcsin(1 / M) + 1e-5
    beta_max = np.radians(89.9)
    beta_vals = np.linspace(beta_min, beta_max, 1000)
    for i in range(len(beta_vals) - 1):
        if func(beta_vals[i]) * func(beta_vals[i + 1]) < 0:
            return math.degrees(scipy.optimize.brentq(func, beta_vals[i], beta_vals[i + 1]))
    raise ValueError(f"No shock angle found for M = {M:.3f}, θ = {theta_deg:.3f}°.")

# -----------------------------
# Max Deflection Angle
# -----------------------------
def max_deflection_angle(M, gamma):
    """Calculate maximum deflection angle for a given Mach number."""
    validate_inputs(M, 1, 1)
    if M <= 1:
        return 0.0
    beta_vals = np.radians(np.linspace(0.1, 89.9, 1000))
    theta_max = 0.0
    for beta in beta_vals:
        num = 2 * (M**2 * math.sin(beta)**2 - 1)
```

```python
        den = math.tan(beta) * (M**2 * (gamma + math.cos(2 * beta)) + 2)
        if den != 0:
            theta = math.atan(num / den)
            theta_deg = math.degrees(theta)
            if theta_deg > theta_max:
                theta_max = theta_deg
    return theta_max

# -----------------------------
# Total Pressure Ratio (Normal Shock)
# -----------------------------
def total_pressure_ratio_via_static_jump(Mn, gamma):
    """Calculate total pressure ratio across a normal shock."""
    validate_inputs(Mn, 1, 1)
    p0_p1 = (1 + ((gamma - 1) / 2) * Mn**2) ** (gamma / (gamma - 1))
    p2_p1 = 1 + (2 * gamma / (gamma + 1)) * (Mn**2 - 1)
    Mn2 = math.sqrt((2 + (gamma - 1) * Mn**2) / (2 * gamma * Mn**2 - (gamma - 1)))
    p0_p2 = (1 + ((gamma - 1) / 2) * Mn2**2) ** (gamma / (gamma - 1))
    return (p2_p1 * p0_p2) / p0_p1

# -----------------------------
# Oblique Shock
# -----------------------------
def oblique_shock(M_upstream, theta_deg, P_upstream, T_upstream, P0_upstream, gamma, station_name):
    """Calculate flow properties after an oblique shock."""
    validate_inputs(M_upstream, P_upstream, T_upstream, theta_deg)
    if M_upstream <= 1:
        raise ValueError(f"Oblique shock not possible at {station_name}: Flow is subsonic (M = {M_upstream:.3f}).")

    theta_max = max_deflection_angle(M_upstream, gamma)
    if theta_deg > theta_max:
        raise ValueError(f"Deflection angle {theta_deg:.3f}° exceeds θ_max = {theta_max:.3f}° at {station_name}.")

    beta_deg = beta_from_theta_mach(M_upstream, theta_deg, gamma)
    beta_rad = math.radians(beta_deg)
    theta_rad = math.radians(theta_deg)

    Mn1 = M_upstream * math.sin(beta_rad)
    Mn2 = math.sqrt((2 + (gamma - 1) * Mn1**2) / (2 * gamma * Mn1**2 - (gamma - 1)))
    M_downstream = Mn2 / math.sin(beta_rad - theta_rad)

    P_downstream = P_upstream * (1 + (2 * gamma / (gamma + 1)) * (Mn1**2 - 1))
    T_downstream = T_upstream * (T0(M_upstream, T_upstream, gamma) / T0(M_downstream, T_upstream, gamma))
    P0_ratio = total_pressure_ratio_via_static_jump(Mn1, gamma)
    P0_downstream = P0_upstream * P0_ratio

    print(f"\n---- Oblique Shock Results ({station_name}) ----")
    print(f"Shock angle β:            {beta_deg:.3f}°")
    print(f"Normal Mach before shock: {Mn1:.3f}")
    print(f"Normal Mach after shock:  {Mn2:.3f}")
    print(f"Mach after shock:         {M_downstream:.3f}")
    print(f"Static Pressure:          {P_downstream:.2f} Pa")
    print(f"Static Temperature:       {T_downstream:.2f} K")
    print(f"Total Pressure:           {P0_downstream:.2f} Pa")
    print(f"Total Pressure Ratio:     {P0_downstream / P0_upstream:.4f}")

    return M_downstream, P_downstream, T_downstream, P0_downstream

# -----------------------------
# Normal Shock
# -----------------------------
def normal_shock(M1, P1, T1, P01, gamma, station_name):
    """Calculate flow properties after a normal shock."""
    validate_inputs(M1, P1, T1)
    if M1 <= 1:
        print(f"\nWarning: Normal shock not needed at {station_name}: Flow is already subsonic (M = {M1:.3f}).")
        return M1, P1, T1, P01

    Mn1 = M1
    Mn2 = math.sqrt((2 + (gamma - 1) * Mn1**2) / (2 * gamma * Mn1**2 - (gamma - 1)))
    M2 = Mn2

    P2 = P1 * (1 + (2 * gamma / (gamma + 1)) * (Mn1**2 - 1))
    T2 = T1 * (T0(M1, T1, gamma) / T0(M2, T1, gamma))
    P0_ratio = total_pressure_ratio_via_static_jump(Mn1, gamma)
    P02 = P01 * P0_ratio

    print(f"\n---- Normal Shock Results ({station_name}) ----")
    print(f"Mach before shock:         {M1:.3f}")
    print(f"Mach after shock:          {M2:.3f}")
    print(f"Static Pressure:           {P2:.2f} Pa")
    print(f"Static Temperature:        {T2:.2f} K")
    print(f"Total Pressure:            {P02:.2f} Pa")
```

```
        print(f"Total Pressure Ratio:      {P02 / P01:.4f}")

        return M2, P2, T2, P02

    # -----------------------------
    # Combustion Chamber Class
    # -----------------------------
    class Combustor:
        def __init__(self, cp_air, cp_gas, QR, eta_combustor):
            self.cp_air = cp_air
            self.cp_gas = cp_gas
            self.QR = QR
            self.eta_combustor = eta_combustor

        def compute(self, Tt3, Tt4, eta_b):
            """Calculate fuel-air ratio and combustor properties."""
            if Tt4 <= Tt3:
                raise ValueError("Exit total temperature Tt4 must be greater than inlet Tt3.")
            numerator = self.cp_gas * Tt4 - self.cp_air * Tt3
            denominator = eta_b * self.QR - self.cp_gas * Tt4
            if denominator <= 0:
                raise ValueError("Invalid combustor parameters: denominator must be positive.")

            f = numerator / denominator
            return {"f": f, "eta_b": eta_b, "Tt4": Tt4}

    # -----------------------------
    # Choked C-D Nozzle Class
    # -----------------------------
    class Nozzle:
        def __init__(self, cp_gas, gamma_gas, R_gas):
            self.cp_gas = cp_gas
            self.gamma = gamma_gas
            self.R = R_gas

        def compute(self, Tt4, P0, eta_n, P_ambient):
            """Calculate properties for a choked C-D nozzle expanding to ambient pressure."""
            if Tt4 <= 0 or P0 <= 0:
                raise ValueError("Total temperature and pressure must be positive.")

            # Throat conditions (choked, M = 1)
            P_throat = P0 * (2 / (self.gamma + 1)) ** (self.gamma / (self.gamma - 1))
            T_throat = Tt4 * (2 / (self.gamma + 1))

            # Solve for exit Mach number (Me) such that Pe = P_ambient
            def pressure_ratio(Me):
                return (1 + (self.gamma - 1) / 2 * Me**2) ** (self.gamma / (self.gamma - 1)) - P0 / P_ambient

            try:
                Me = scipy.optimize.brentq(pressure_ratio, 1.0, 10.0)  # Start from M=1 for supersonic expansion
            except ValueError:
                raise ValueError("Could not find exit Mach number for given P0 and P_ambient.")

            # Calculate exit temperature
            Te = Tt4 / (1 + (self.gamma - 1) / 2 * Me**2)

            # Calculate exit velocity using speed of sound
            Ve = Me * math.sqrt(self.gamma * self.R * Te) * math.sqrt(eta_n)

            # Exit pressure
            Pe = P0 / (1 + (self.gamma - 1) / 2 * Me**2) ** (self.gamma / (self.gamma - 1))

            # Check nozzle expansion state
            expansion_state = "perfectly expanded"
            if Pe > P_ambient * 1.01:
                expansion_state = "under-expanded"
            elif Pe < P_ambient * 0.99:
                expansion_state = "over-expanded"

            return {
                "Pe": Pe,
                "Te": Te,
                "Ve": Ve,
                "Me": Me,
                "P_throat": P_throat,
                "T_throat": T_throat,
                "expansion_state": expansion_state
            }

    # -----------------------------
    # Simulation Begins
    # -----------------------------
    # Freestream conditions
```

```
# Freestream Conditions
M1 = 4.0
P1 = 1e5         # Pa
T1 = 288         # K
T01_val = T0(M1, T1, CONFIG["gamma_air"])
P01_val = P0(M1, P1, T1, CONFIG["gamma_air"])

print("---- Freestream Conditions (Station 1) ----")
print(f"Total Temperature T01: {T01_val:.2f} K")
print(f"Total Pressure P01:    {P01_val:.2f} Pa")

# User-defined ramps (degrees)
ramp_angles = [10, 12, 15, 25, 30]

# Initial state
M_current = M1
P_current = P1
T_current = T1
P0_current = P01_val

for i, theta in enumerate(ramp_angles):
    station_name = f"Station 1.{i+1}"
    if M_current > 1:
        try:
            M_new, P_new, T_new, P0_new = oblique_shock(
                M_current, theta, P_current, T_current, P0_current, CONFIG["gamma_air"], station_name
            )
            M_current, P_current, T_current, P0_current = M_new, P_new, T_new, P0_new
        except ValueError as e:
            print(f"\nWarning at {station_name}: {e}")
            print(f"Inserting normal shock at {station_name}.")
            M_new, P_new, T_new, P0_new = normal_shock(
                M_current, P_current, T_current, P0_current, CONFIG["gamma_air"], station_name
            )
            M_current, P_current, T_current, P0_current = M_new, P_new, T_new, P0_new
    else:
        print(f"\n---- Subsonic Flow at {station_name} ----")
        print(f"Flow is subsonic (M = {M_current:.3f}). No shock applied for θ = {theta:.3f}°.")
        print(f"Static Pressure:          {P_current:.2f} Pa")
        print(f"Static Temperature:       {T_current:.2f} K")
        print(f"Total Pressure:           {P0_current:.2f} Pa")

# Optional final normal shock
if M_current > 1:
    M_current, P_current, T_current, P0_current = normal_shock(
        M_current, P_current, T_current, P0_current, CONFIG["gamma_air"], "Station Final (Normal Shock)"
    )

# -----------------------------
# Combustion Chamber Execution
# -----------------------------
Tt3 = T0(M_current, T_current, CONFIG["gamma_air"])
Tt4 = 2000  # Desired combustor exit total temp (K)

combustor = Combustor(CONFIG["cp_air"], CONFIG["cp_gas"], CONFIG["QR"], CONFIG["eta_combustor"])
try:
    combustor_results = combustor.compute(Tt3, Tt4, CONFIG["eta_b"])
    P0_combustor_exit = P0_current * CONFIG["eta_combustor"]

    print(f"\n---- Combustion Chamber ----")
    print(f"Combustor Inlet Total Temp (Tt3): {Tt3:.2f} K")
    print(f"Combustor Exit Total Temp (Tt4):  {combustor_results['Tt4']:.2f} K")
    print(f"Fuel-to-Air Ratio (f):            {combustor_results['f']:.5f}")
    print(f"Total Pressure (with loss):       {P0_combustor_exit:.2f} Pa")
except ValueError as e:
    print(f"\nCombustor Error: {e}")
    raise

# -----------------------------
# Choked C-D Nozzle
# -----------------------------
nozzle = Nozzle(CONFIG["cp_gas"], CONFIG["gamma_gas"], CONFIG["R_gas"])
try:
    nozzle_results = nozzle.compute(Tt4, P0_combustor_exit, CONFIG["eta_n"], CONFIG["P_ambient"])

    print(f"\n---- Choked C-D Nozzle ----")
    print(f"Throat Pressure (P_throat):       {nozzle_results['P_throat']:.2f} Pa")
    print(f"Throat Temperature (T_throat):    {nozzle_results['T_throat']:.2f} K")
    print(f"Nozzle Exit Mach Number (Me):     {nozzle_results['Me']:.3f}")
    print(f"Nozzle Exit Pressure (Pe):        {nozzle_results['Pe']:.2f} Pa")
    print(f"Nozzle Exit Temperature (Te):     {nozzle_results['Te']:.2f} K")
    print(f"Nozzle Exit Velocity (Ve):        {nozzle_results['Ve']:.2f} m/s")
    print(f"Nozzle Expansion State:           {nozzle_results['expansion_state']}")
```

```
    except ValueError as e:
        print(f"\nNozzle Error: {e}")
        raise
```

```
Normal Mach after shock:  0.696
Mach after shock:         3.286
Static Pressure:          250604.31 Pa
Static Temperature:       382.83 K
Total Pressure:           14051261.88 Pa
Total Pressure Ratio:     0.9254

---- Oblique Shock Results (Station 1.2) ----
Shock angle β:            27.391°
Normal Mach before shock: 1.512
Normal Mach after shock:  0.697
Mach after shock:         2.626
Static Pressure:          626466.92 Pa
Static Temperature:       508.43 K
Total Pressure:           13011475.59 Pa
Total Pressure Ratio:     0.9260

---- Oblique Shock Results (Station 1.3) ----
Shock angle β:            35.534°
Normal Mach before shock: 1.526
Normal Mach after shock:  0.692
Mach after shock:         1.973
Static Pressure:          1597882.70 Pa
Static Temperature:       680.13 K
Total Pressure:           11987365.53 Pa
Total Pressure Ratio:     0.9213

Warning at Station 1.4: Deflection angle 25.000° exceeds θ_max = 22.501° at Station 1.4.
Inserting normal shock at Station 1.4.

---- Normal Shock Results (Station 1.4) ----
Mach before shock:         1.973
Mach after shock:          0.582
Static Pressure:           6989965.36 Pa
Static Temperature:        1132.84 K
Total Pressure:            8792896.75 Pa
Total Pressure Ratio:      0.7335

---- Subsonic Flow at Station 1.5 ----
Flow is subsonic (M = 0.582). No shock applied for θ = 30.000°.
Static Pressure:          6989965.36 Pa
Static Temperature:       1132.84 K
Total Pressure:           8792896.75 Pa

---- Combustion Chamber ----
Combustor Inlet Total Temp (Tt3): 1209.60 K
Combustor Exit Total Temp (Tt4):  2000.00 K
Fuel-to-Air Ratio (f):            0.02980
Total Pressure (with loss):       8353251.91 Pa

---- Choked C-D Nozzle ----
Throat Pressure (P_throat):       4513796.76 Pa
Throat Temperature (T_throat):    1716.74 K
Nozzle Exit Mach Number (Me):     3.480
Nozzle Exit Pressure (Pe):        100000.00 Pa
Nozzle Exit Temperature (Te):     667.08 K
Nozzle Exit Velocity (Ve):        1738.36 m/s
Nozzle Expansion State:           perfectly expanded
```