Shaft Design Project Code

Machine Design MAE 4300
Professor Larry Gardner

Tyler Black
Daniel Estes
Jonah Frederick
Mason Frederick
Michael Fretz
Travis Slade

Due: 28 January 2025

1 Code

```python
from math import sqrt, pi


def get_inputs():
    """
    Prompts the user to input the moment, torque, and diameter of the shaft.
    Also allows selection of a stress concentration case.

    Returns:
        m (float): Bending moment in kip-in (kilo pound-inch).
        t (float): Torque in kip-in.
        d (float): Shaft diameter in inches.
        case (int): Selected stress concentration case (1-4).
    """
    m = float(input("Input moment (lbf*in): ")) / 1000  # Convert to kip-in
    t = float(input("Input torque (lbf*in): ")) / 1000  # Convert to kip-in
    d = float(input("Input diameter (in): "))  # Input shaft diameter in inches

    def select_case():
        """
        Displays the available stress concentration cases and gets user input.

        Returns:
            int: Selected case number (1-4).
        """
```

```
        print("""Stress Concentration Cases:
        1. Wide Radius
        2. Sharp Radius
        3. Keyway
        4. R.R.G""")

        case = int(input("Select Case: "))
        return case if case in {1, 2, 3, 4} else select_case()

    case = select_case()
    return m, t, d, case


def get_kf_kfs(d, case):
    """
    Computes the fatigue stress concentration factors (kf and kfs) based on
    the selected case.

    Args:
        d (float): Shaft diameter in inches.
        case (int): Selected stress concentration case.

    Returns:
        kf (float): Fatigue stress concentration factor for bending.
        kfs (float): Fatigue stress concentration factor for torsion.
    """
    a = 0.009601465  # Notch sensitivity coefficient for bending
    a_s = 0.00538003  # Notch sensitivity coefficient for shear

    # Stress concentration factors (k, ks) and notch radii (r) based on case
    scf = {
        1: (1.7, 1.5, 0.1 * d),
        2: (2.7, 2.2, 0.02 * d),
        3: (2.14, 3, 0.02 * d),
        4: (5, 3, 0.01 * d)
    }

    k, ks, r = scf[case]
    r = max(r, 1e-9)  # Prevent division by zero

    # Calculate fatigue stress concentration factors
    kf = 1 + (k - 1) / (1 + sqrt(a) / sqrt(r))
    kfs = 1 + (ks - 1) / (1 + sqrt(a_s) / sqrt(r))

    return kf, kfs


def fatigue_sf(m, t, d, case):
    """
    Computes the fatigue safety factor for the shaft.

    Args:
        m (float): Bending moment in kip-in.
        t (float): Torque in kip-in.
```

```python
        d (float): Shaft diameter in inches.
        case (int): Selected stress concentration case.

    Returns:
        n_f (float): Fatigue safety factor.
    """
    uts = 68  # Ultimate tensile strength in ksi (kilo pounds per square inch)
    kf, kfs = get_kf_kfs(d, case)

    # Calculate bending and torsional stress components
    sigma_b = 2 * kf * m  # Bending stress component
    tau = sqrt(3) * kfs * t  # Torsional stress component

    # Compute endurance limit (Se) using size factor correction
    Se_prime = uts / 2  # Initial endurance limit assumption
    Se = 2 * uts ** (-0.217) * 0.879 * d ** (-0.107) * Se_prime  # Modified endurance limit

    # Compute fatigue safety factor
    n_f = (pi * d ** 3 / 16) / ((sigma_b / Se) + (tau / uts))

    return n_f


def yield_sf(m, t, d, case):
    """
    Computes the yield safety factor using von Mises stress.

    Args:
        m (float): Bending moment in kip-in.
        t (float): Torque in kip-in.
        d (float): Shaft diameter in inches.
        case (int): Selected stress concentration case.

    Returns:
        n_y (float): Yield safety factor.
    """
    sy = 37.5  # Yield strength in ksi
    kf, kfs = get_kf_kfs(d, case)

    # Compute bending and torsional stresses
    sigma = 32 * m * kf / (pi * d ** 3)  # Bending stress in ksi
    tau = 16 * t * kfs / (pi * d ** 3)  # Torsional stress in ksi

    # Compute von Mises equivalent stress
    sigma_vm = sqrt(sigma ** 2 + 3 * tau ** 2)

    # Compute yield safety factor
    n_y = sy / sigma_vm

    return n_y

def con_yield_sf(m, t, d, case):
    """
```

```python
    Computes the yield safety factor a quick conservative check.

    Args:
        m (float): Bending moment in kip-in.
        t (float): Torque in kip-in.
        d (float): Shaft diameter in inches.
        case (int): Selected stress concentration case.

    Returns:
        n_y (float): Yield safety factor.
    """
    sy = 37.5  # Yield strength in ksi
    kf, kfs = get_kf_kfs(d, case)

    # Compute bending and torsional stresses
    sigma_a = 32 * m * kf / (pi * d ** 3)
    sigma_m = sqrt(3) * 16 * t * kfs / (pi * d ** 3)

    # Compute yield safety factor
    n_y_c = sy / (sigma_a + sigma_m)

    return n_y_c


def calc_diameter(m, t, case):
    """
    Computes the required shaft diameter to meet a target safety factor of 1.5.

    Args:
        m (float): Bending moment in kip-in.
        t (float): Torque in kip-in.
        case (int): Selected stress concentration case.

    Returns:
        d (float): Recommended shaft diameter in inches.
    """

    def f(d):
        """Returns the fatigue safety factor for a given diameter."""
        return fatigue_sf(m, t, d, case)

    def g(d):
        """Returns the yield safety factor for a given diameter."""
        return con_yield_sf(m, t, d, case)

    d = 0.01  # Initial guess for shaft diameter in inches
    n = 1.5  # Target safety factor

    # Iteratively increase diameter until safety factor conditions are met
    while f(d) < n or g(d) < n:
        d += 0.001  # Increment diameter in small steps

    return d
```

```python


if __name__ == '__main__':
    while True:
        # Get user input
        m, t, d, case = get_inputs()

        # Compute safety factors
        n_f = fatigue_sf(m, t, d, case)
        n_y = yield_sf(m, t, d, case)
        n_y_c = con_yield_sf(m, t, d, case)
        d_rec = calc_diameter(m, t, case)

        # Print results
        print(f"Fatigue Safety Factor: {n_f:.3f}")
        print(f"Yield Safety Factor: {n_y:.3f}")
        print(f"Conservative Yield Safety Factor: {n_y_c:.3f}")
        print(f"Recommended Shaft Diameter (for SF=1.5): {d_rec:.3f} in")
```

shaft_design.py