Assignment 2: Bending and torsion in a wind-turbine tower

Overview

Analysis of static and dynamic loads in structures such as a tower of a wind turbine is imperative to ensure the integrity of these machines in adverse conditions in service. For instance, consider the effects of ice-accumulation on the blades (see Figure 1a), which not only results in increased loads, but in case of ice break-off from one of the blades can create imbalance in the entire structure. Figure 1b shows a failed tower of a wind turbine structure revealing the susceptibility of the typical tubular composite structure in bending and torsional loads.

Figure 1: a) Wind turbine blades gathering ice and snow resulting in increased loads and potential imbalance b) A failed wind turbine tower

You are tasked with estimating the impact of the potential imbalance caused by the ice-accumulation utilizing a rather simplified static analysis (with concepts from IEA) by (only) considering a few of the loads that are resisted by the actual structure.

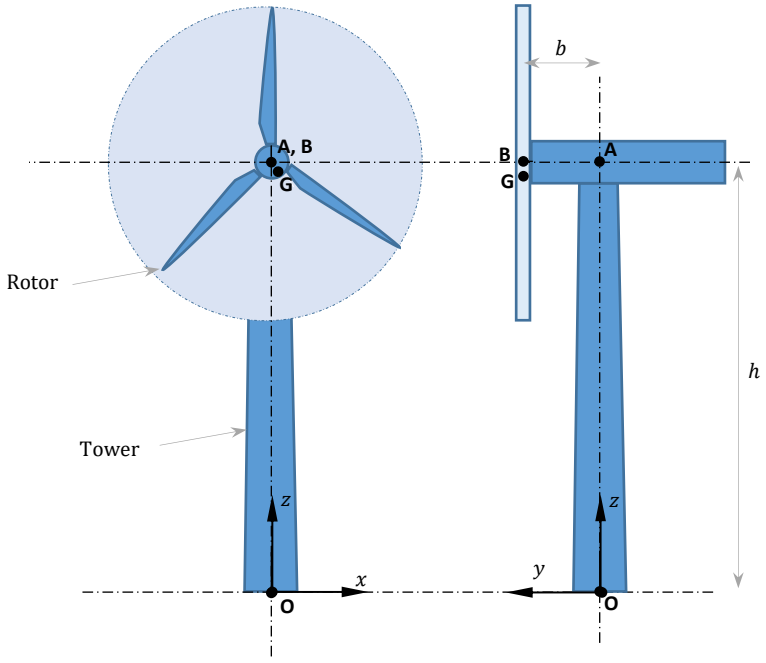

Figure 2: Schematic of a wind turbine structure for a simplified analysis of loads due to ice-accumulation

As part of this assignment, you will develop a Python program to evaluate the bending moment and torque on the tower due to a potential imbalance in the accumulated mass. Consider a scenario where the center of gravity of the rotor (consisting of the blades and the rotor hub) has

shifted to the point (G) shown in Figure 2. Here, we are interested in computing the bending moment and torque at the base of the tower (point O), caused by the forces (i) weight of the rotor ($W = mg$) and (ii) the centrifugal force ($F_C = m\omega^2 r$), where m is the mass of the rotor, g is acceleration due to gravity, r is the radial distance between the center of the rotor (point B) and the shifted center of gravity at point G. Use SI units.

Recall that, using vector mechanics, the moment caused by a force $\boldsymbol{F}$ at point $\boldsymbol{G}$ about a point $\boldsymbol{O}$ is given by

$$\boldsymbol{M} = \boldsymbol{r}_{\boldsymbol{OG}} \times \boldsymbol{F} \tag{1}$$

Note that $\boldsymbol{F}$ here is the vector resultant of the afore-mentioned force magnitudes given in (i) and (ii). That is, be sure to express the forces (i) and (ii) as vectors with the proper magnitudes and directions to compute the resultant force vector.

The moment (to be utilized for torsion) about an axis denoted by unit vector $\hat{\boldsymbol{u}}$ is given by

$$\boldsymbol{M}_{\boldsymbol{OA}} = \hat{\boldsymbol{u}} \cdot (\boldsymbol{r} \times \boldsymbol{F}) \tag{2}$$

where r is the vector from any point along the axis to any point on the line of action of the force $\boldsymbol{F}$.

Deliverables

Create a file named **hw02.py** that contains the functions listed below.

- Your functions must be named exactly as written below
- Include an informative standard-conforming docstring for each function
- All of the vectors and matrices must be **`numpy`** arrays (ndarray)
- Do not use any library functions to perform the actual computational tasks of these functions; for example, do not use numpy.cross to compute the cross product – code the formula for the cross product based on what you have learned so far in this course and IEA and Physics. (You may continue to use methods, functions, and constants such as size, shape, cos, tan, pi, etc.)

`dot_product`

Returns a scalar that is the dot product of two equal size input vectors

- You may assume that the vectors are of equal size; no error checking is required
- You must use a loop to compute the dot product

`cross_product`

Returns a vector that is the cross product of two input vectors

- You may assume that the vectors are 3-element 1D arrays
- It must return a 3-element 1D array per the definition of the cross product

`tower_moment_torque`

Returns the moment (vector) and magnitude of moment about the tower axis (torque) at the base of the tower given by Equation (1) and Equation (2), provided the following input parameters:

mass (m),
the offset vector ($\boldsymbol{r}_{BG}$),
the height of the tower (h),
the (perpendicular) distance between center of rotor to the tower axis (b),

the angular velocity of the rotor (ω)

The following function signature must be used:

```
bend_moment, torque = tower_moment_torque (m, rBG, h, b, omega):
```

- You are required to utilize the functions you wrote as described above to compute and return the required quantities as appropriate.
- Utilize vector addition to compute rOG from rBG and other scalar distances
- Use the acceleration due to gravity $g = 9.81 \frac{m}{s^2}$

Computation of moment and torque at the base of the tower:

Utilize the **tower_moment_torque** function (in a program in the same python file) to compute the bending moment ($\boldsymbol{M}$) and the torque (T) at the rotor base using the following inputs:

$m = 1{,}000$ kg
$\boldsymbol{r}_{BG} = \{0.03, 0.00, -0.04\}^T$ m
$h = 135$ m
$b = 1.2$ m
$\omega = 10\pi$ rad/s

Store the values of the moment vector and torque in variables **bend_moment** and **torque** respectively. Note that it is important to name these correctly for *Submitty* to process your answers to give you credit.

Output

When you run **hw02.py** for the last time before submitting to Submitty, there should be no printed output at all. Use Hint 3 below to suppress your test code.

Hints

1. An "informative" docstring will consist of at least four lines. It will a) summarize the function in one line, followed by a blank line, and then b) describe the inputs and outputs and briefly summarize the numerical method (as applicable) used to compute the function result in a minimum of two lines (and usually more).

For example, if the function were to implement the quadratic formula to find roots you might write something like

```
"""Returns the two roots of a quadratic polynomial as scalars.

Implements the standard quadratic formula to compute and
return scalar roots `x1` and `x2`, which may be real or
complex, of the quadratic polynomial a*x**2 + b*x + c = 0.
"""
```

and not

```
"""Turns a, b, c into x1, x2

Squares b and subtracts from it four times a times c, then
takes the square root of that and....
```

```
    """
```

2. You should test your functions individually as you code them and verify the results with hand calculations.

3. As an alternative to commenting out print statements you can use if statements with a "flag" variable to signal whether to execute a section of test code or not. For example,

```
#####################
# SET DEBUGGING FLAGS – set all flags to False before submitting
#
test_dot_product = True
test_cross_product = False
#####################
if (test_dot_product):
    # test_dot_product = True means these statements will run
    # set input data with known result
    # call the dot_product function
    print("dot_product function gives ", result)
if (test_cross_product):
    # test_cross_product = False means statements will _not_ run
    # set input data with known result
    # call the cross_product function
    print("cross_product function gives ",result)
```