BME Fac. of Mech. Eng.	Advanced Mechanics	Name: Gergő Kelle
Dept. of Applied Mechanics	HOME WORK 1	Neptun: GBBNUL
2022/23 II.	Deadline: 2023.05.08. 18:00	Late submission: ☐ Correction: ☐
Statement: I hereby confirm that this homework is my own work and the submitted document shows my way of understanding.		Signature:

All the results have to be correct and checked online. The formal requirements must be also fulfilled. Online HW result checking: `http://www.mm.bme.hu/targyak/msc/mwsm/MW01_amech/hw1check.htm`

Assignment

The turbine shown in the figure is rotating with the rotational speed n. The instantaneous state of acceleration of the turbine is known. In the calculation, the blades can be modelled as beams with rectangular cross sections. The dynamic load of the blades is dominant; all the other loads can be neglected. The density of the blade material is denoted by ρ.

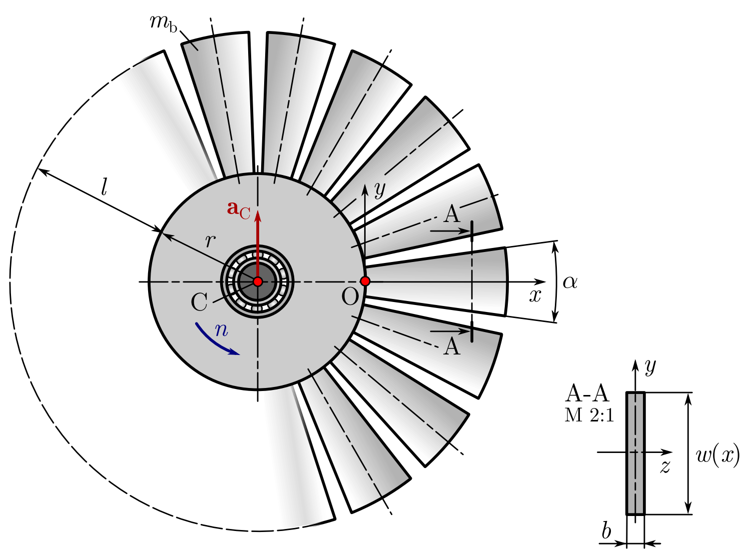

Figure 1: Mechanical model

Data
$\rho = 7800\,\mathrm{kg/m^3}$
$r = 145$ mm
$l = 105$ mm
$\alpha = 18\,°$
$w(x) = (r + x)\tan\alpha$
$b = 7$ mm
$n = 2800$ rpm
$a_C = 1000\ \mathrm{m/s^2}$
$\varepsilon_z = 0\ \mathrm{rad/s^2}$

Task

1. Calculate the reaction forces acting on the blades from the hub.
2. Determine the stress resultant functions for the chosen blade and plot them with the help of an appropriate computer software. Write the results in the table below according to the sign convention used during the lecture.
3. Calculate the maximum of the normal stress and its location (the critical cross section and the critical points of this cross section).

Results

N_{max} [N]	$M_{h,max}$ [Nm]	$\sigma_{x,max}$ [MPa]
6394.053	21.026	27.507

Budapest University of Technology and Economics
Faculty of Mechanical Engineering

Advanced Mechanics
I. Homework

Gergő Kelle

May 8, 2023

Contents

1 Calculation of Reaction Forces

Calculate the reaction forces acting on the blades from the hub.

During the following tasks i will calculate according to the values shown in Table 1.

Table 1: Table of data

Meaning	**Symbol**	**Value**	**Dimension**
Density	ρ	7800	$\frac{kg}{m^3}$
Inner radius	r	0.145	m
Blade radial length	l	0.105	m
Blade edge angle	α	18	$^\circ$
Blade thickness	b	0.007	m
Revolution	n	2800	rpm
Acceleration	a_C	1000	$\frac{m}{s^2}$
Acceleration of rotation	ε	0	$\frac{rad}{s^2}$

The primary objective is to compute the reaction forces acting on the blades from the hub, given that there are no external forces. Instead, there exists an acceleration state that leads to reaction forces. Initially, it is necessary to determine the acceleration at each point along a single blade, which can be accomplished using the free body diagram of the mechanical model shown in Figure 1, where the blade located on the x-axis is used.

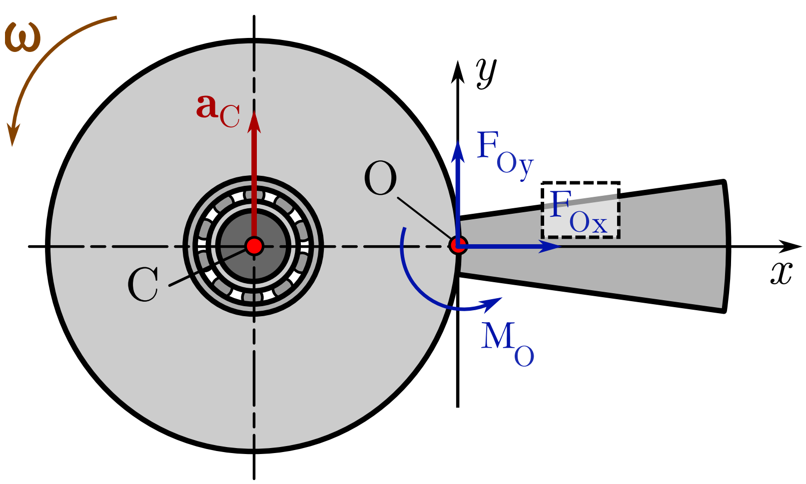

Figure 1: Free body diagram

Utilizing the planar problem, the acceleration reduction formula can be expressed as a function of x:

$$\mathbf{a}(x) = \mathbf{a}_C + \varepsilon \times \mathbf{r}_{Cx} - \omega^2\, \mathbf{r}_{Cx} \tag{1}$$

Where $\mathbf{r}_{Cx}$ is the vector which goes from the beginning of the blade till its end as a function of $x \in [r\ ,\ r+l] \rightarrow x \in [0\ ,\ l]$. We can ellimiate the $\varepsilon \times \mathbf{r}_{Cx}$ part since $\varepsilon = \mathbf{0}$ leaving us with the following result

$$\mathbf{a}(x) = \begin{bmatrix} -\omega^2\ (r+x) \\ 1000 \\ 0 \end{bmatrix} \tag{2}$$

D'Alambert's principle can be utilized to simplify a dynamical problem by reducing the acceleration state to a force system. This is particularly useful since solving problems in statics is generally easier than in dynamics. The principle states that, for a system of particles or rigid bodies in motion, the equation of motion can be modified by adding a term that is proportional to the acceleration of the system. This term, known as the fictitious force, cancels out the effect of the acceleration and transforms the problem into a static one.
Therefore, by applying D'Alambert's principle, the dynamic problem can be converted to a static one, which simplifies the solution process.

$$\mathbf{p}(x) = -\rho\ A\ \mathbf{a}(x) = \begin{bmatrix} \rho\ b\ w(x)\ \omega^2\ (r+x) \\ -\rho\ b\ w(x) \cdot 1000 \\ 0 \end{bmatrix} \tag{3}$$

We can determine the reaction forces by performing an integration along the blades.

$$F_{Ox} = -\int_0^l p_x(x)\ \mathrm{d}x = -6394.053\ [\mathrm{N}] \tag{4}$$

$$F_{Oy} = -\int_0^l p_y(x)\ \mathrm{d}x = -367.896\ [\mathrm{N}] \tag{5}$$

$$M_O = -\int_0^l p_y(x)\ x\ \mathrm{d}x = -21.026\ [\mathrm{Nm}] \tag{6}$$

2 Calculation of Stress Resultant Functions

Determine the stress resultant functions for the chosen blade and plot them with the help of an appropriate computer software. Write the results in the table below according to the sign convention used during the lecture.

$$N(x) = -F_{Ox} - \int_0^x p_x(x)\ \mathrm{d}x \tag{7}$$

$$V(x) = F_{Oy} + \int_0^x p_x(x)\ \mathrm{d}x \tag{8}$$

$$M_b(x) = M_O - \int_0^x V(x)\ \mathrm{d}x \tag{9}$$

At the cross-sections along the blade, the distribution functions exhibit their maximum values at the root section ($x = 0$). As a result, the critical cross-sections are also located at $x = 0$. The maximal values of these functions are obtained at this critical cross-section, indicating the point of highest stress or load on the blade which values are the following:

Table 2: Maximum force and moment values along the blade

Parameter	Value	Unit
N_{max}	6394.053	N
V_{max}	367.896	N
$M_{b\ max}$	21.026	Nm

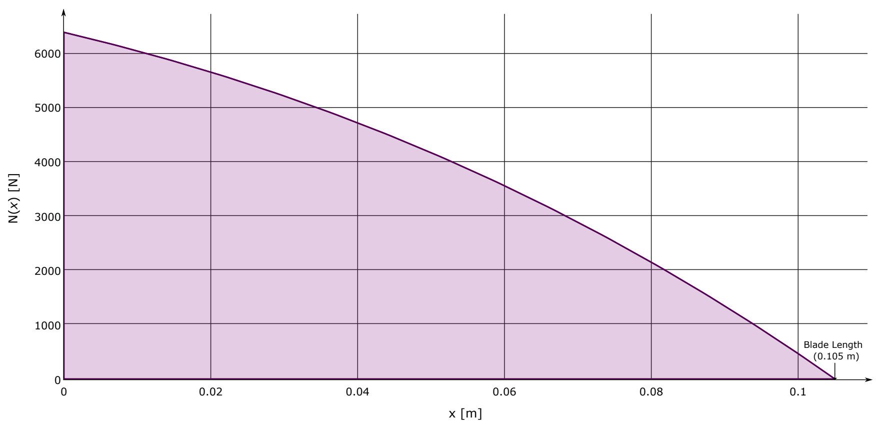

Figure 2: Normal force distribution along x axis

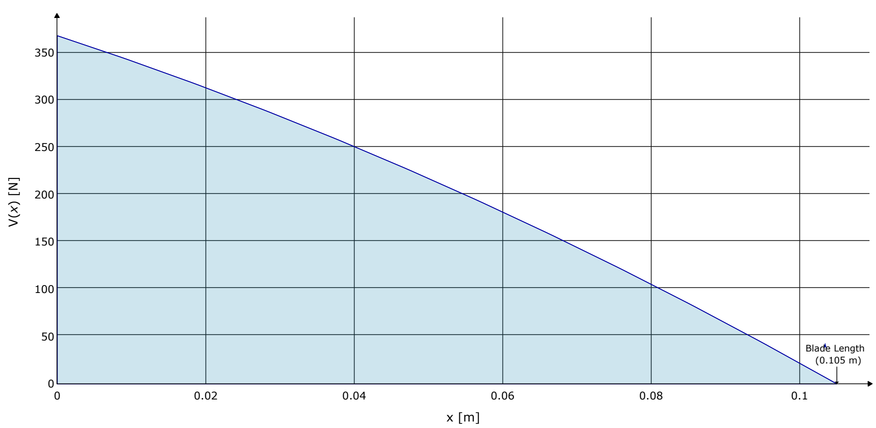

Figure 3: Sheer force distribution along x axis

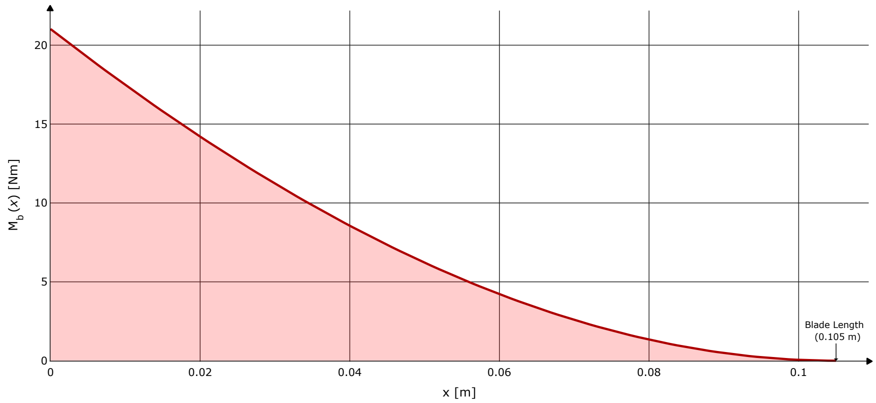

Figure 4: Bending moment distribution along x axis

3 Calculation of Normal Stress

Calculate the maximum of the normal stress and its location (the critical cross section and the critical points of this cross section).

According to the stress analysis of the blade, the maximum normal stress and its location can be calculated. The normal stress in the blade is a result of two components: the normal force and the bending. The maximal normal stress occurs at the critical cross-section of the blade, which is located at the foot of the blade where $x = 0$ and $y = 0$. At this critical cross-section, the cross-sectional area of the blade is minimal.
To calculate the maximum normal stress, the equations for the area (A) and the moment of inertia (I_z) at the critical cross-section need to be determined which is done below.

$$A(x = 0) = b\ w(x = 0) = 3.3 \cdot 10^{-4}\ \left[\mathrm{m}^2\right] \tag{10}$$

$$I_z(x = 0) = \frac{b\ w(x = 0)^3}{12} = 6.1 \cdot 10^{-8}\ \left[\mathrm{m}^4\right] \tag{11}$$

These equations can be used to calculate the maximum normal stress ($\sigma_{N\ max}$)

$$\sigma_{N\ max} = \frac{N_{max}}{A(x = 0)} = 19.388\ [\mathrm{MPa}] \tag{12}$$

and the maximum bending stress ($\sigma_{b\ max}$)

$$\sigma_{b\ max} = \frac{M_{b\ max}\ w(x = 0)}{2\ I_z(x = 0)} = 8.119\ [\mathrm{MPa}]\,. \tag{13}$$

The total maximum normal stress ($\sigma_{x\ max}$) can be determined by adding these two stress components together.

$$\boxed{\sigma_{x\ max} = \sigma_{N\ max} + \sigma_{b\ max} = 27.507\ [\mathrm{MPa}]} \tag{14}$$

4 Code

4.1 Calculation

```python
import numpy as np
from scipy.integrate import quad
import sympy as sym
import plotly.graph_objects as go
from plotly.subplots import make_subplots

# Define the constants
rho = 7800        # kg/m^3
r = 0.145         # m
l = 0.105         # m
alpha = 18        # degrees
b = 0.007         # m
n = 2800          # rpm
a_C = 1000        # m/s^2
epsilon_z = 0     # rad/s

## TASK 1 ##

# Define the function to be integrated
def integrand(x):
    return rho * b * (r + x) * np.tan(alpha*np.pi/180) * (n/60*2*np.pi)**2 * (r
        + x)
def integrand2(x):
    return rho * b * (r + x) * np.tan(alpha*np.pi/180) * 1000
def integrand3(x):
    return rho * b * (r + x) * np.tan(alpha*np.pi/180) * 1000 * x

# Integrate the function from 0 to l
resultX, error = quad(integrand, 0, l)
resultY, error = quad(integrand2, 0, l)
resultMb, error = quad(integrand3, 0, l)
resultX = -1 * resultX
resultY = -1 * resultY
resultMb = -1 * resultMb


# Results
print("TASK 1")
print("X: The result of the integration is:", resultX, "N")
print("Y: The result of the integration is:", resultY, "N")
print("Mb: The result of the integration is:", resultMb, "Nm")

## TASK 2 ##

# Define the function to be integrated
x = sym.symbols('x')
f = rho * b * (r + x) * sym.tan(alpha*np.pi/180) * (n/60*2*np.pi)**2 * (r + x)
f2 = rho * b * (r + x) * np.tan(alpha*np.pi/180) * 1000
f3 = rho * b * (r + x) * np.tan(alpha*np.pi/180) * 1000 * x
```

```python
resultX, error = quad(sym.lambdify(x, f), 0, 1)
resultY, error = quad(sym.lambdify(x, f2), 0, 1)
resultMb, error = quad(sym.lambdify(x, f3), 0, 1)

# Define the function N(x), V(x) and Mb (x)
N = resultX - sym.integrate(f, (x, 0, x))
V = resultY - sym.integrate(f2, (x, 0, x))
Mb = resultMb - sym.integrate(V, (x, 0, x))

# Print the resulting polynomial function for N(x), V(x) and Mb (x)
print("------")
print("TASK 2")
print("The resulting polynomial function for N(x) is:")
print(sym.expand(N))
print("The resulting polynomial function for V(x) is:")
print(sym.expand(V))
print("The resulting polynomial function for Mb(x) is:")
print(sym.expand(Mb))

## TASK 3 ##

def w(x):
    return (r + x) * np.tan(alpha*np.pi/180)
A = b * w(0);
Iz= b*w(0)**3/12;
sigma_N_max = N.subs(x,0)/A;
sigma_b_max = Mb.subs(x,0)*w(0)/(2*Iz);
sigma_x_max = sigma_N_max + sigma_b_max;
print("------")
print("TASK 3")
print("A(x=0) [m^2]:")
print(A)
print("I_z(x=0) [m^4]:")
print(Iz)
print("Sigma_{Nmax} [MPa]:")
print(sigma_N_max/10**6)
print("Sigma_{bmax} [MPa]:")
print(sigma_b_max/10**6)
print("Sigma_{Nmax} [MPa]:")
print(sigma_x_max/10**6)
```

4.2 Plot

```python
## N(x) ##
# Create a list of x values
x_vals = np.linspace(0, 1, 100)

# Create a list of corresponding y values
y_vals = np.array([N.subs(x, val) for val in x_vals]).astype(float)

# Create the plotly figure
fig = go.Figure()

```

```python
# Add the N(x) curve to the figure
fig.add_trace(go.Scatter(x=x_vals, y=y_vals, mode='lines', name='N(x)', line=dict(width=3)))

# Update the figure layout
fig.update_layout(
    xaxis=dict(
        title='x [m]',
        linecolor='black',
        linewidth=1,
        tickfont=dict(size=14, color='black'),
        title_font=dict(size=16, color='black')
    ),
    yaxis=dict(
        title='N(<i>x</i>) [N]',
        linecolor='black',
        linewidth=1,
        tickfont=dict(size=14, color='black'),
        title_font=dict(size=16, color='black')
    ),
    width=1200,
    height=600,
    title='',
    plot_bgcolor='rgba(0,0,0,0)',
    margin=dict(l=50, r=50, t=50, b=50),
    xaxis_gridcolor='lightgrey',
    yaxis_gridcolor='lightgrey',
    annotations=[dict(x=0.105, y=0, text='Blade Length <br> (0.105 m)', showarrow=True, arrowhead=1, ax=0, ay=-40)]
)

# Update the fill color for the area under the curve
fig.update_traces(fill='tozeroy', fillcolor='rgba(128, 0, 128, 0.2)')

# Save the plot in .svg format
fig.write_image("N(x)_plot.svg")
# Show the plotly figure
fig.show()

## V(x) ##
# Create a list of x values
x_vals = np.linspace(0, 1, 100)

# Create a list of corresponding y values
y_vals = np.array([V.subs(x, val) for val in x_vals]).astype(float)

# Create the plotly figure
fig = go.Figure()

# Add the N(x) curve to the figure
fig.add_trace(go.Scatter(x=x_vals, y=y_vals, mode='lines', name='V(x)', line=dict(width=3)))

# Update the figure layout
```

```
fig.update_layout(
    xaxis=dict(
        title='x [m]',
        linecolor='black',
        linewidth=1,
        tickfont=dict(size=14, color='black'),
        title_font=dict(size=16, color='black')
    ),
    yaxis=dict(
        title='V(<i>x</i>) [N]',
        linecolor='black',
        linewidth=1,
        tickfont=dict(size=14, color='black'),
        title_font=dict(size=16, color='black')
    ),
    width=1200,
    height=600,
    title='',
    plot_bgcolor='rgba(0,0,0,0)',
    margin=dict(l=50, r=50, t=50, b=50),
    xaxis_gridcolor='lightgrey',
    yaxis_gridcolor='lightgrey',
    annotations=[dict(x=0.105, y=0, text='Blade Length <br> (0.105 m)',
        showarrow=True, arrowhead=1, ax=0, ay=-40)]
)

# Update the fill color for the area under the curve
fig.update_traces(fill='tozeroy', fillcolor='rgba(128, 0, 128, 0.2)')

# Save the plot in .svg format
fig.write_image("V(x)_plot.svg")
# Show the plotly figure
fig.show()

## Mb(x) ##
# Create a list of x values
x_vals = np.linspace(0, l, 100)

# Create a list of corresponding y values
y_vals = np.array([Mb.subs(x, val) for val in x_vals]).astype(float)

# Create the plotly figure
fig = go.Figure()

# Add the N(x) curve to the figure
fig.add_trace(go.Scatter(x=x_vals, y=y_vals, mode='lines', name='V(x)', line=
    dict(width=3)))

# Update the figure layout
fig.update_layout(
    xaxis=dict(
        title='x [m]',
        linecolor='black',
        linewidth=1,
```

```
        tickfont=dict(size=14, color='black'),
        title_font=dict(size=16, color='black')
    ),
    yaxis=dict(
        title='M_b(<i>x</i>) [Nm]',
        linecolor='black',
        linewidth=1,
        tickfont=dict(size=14, color='black'),
        title_font=dict(size=16, color='black')
    ),
    width=1200,
    height=600,
    title='',
    plot_bgcolor='rgba(0,0,0,0)',
    margin=dict(l=50, r=50, t=50, b=50),
    xaxis_gridcolor='lightgrey',
    yaxis_gridcolor='lightgrey',
    annotations=[dict(x=0.105, y=0, text='Blade Length <br> (0.105 m)',
        showarrow=True, arrowhead=1, ax=0, ay=-40)]
)

# Update the fill color for the area under the curve
fig.update_traces(fill='tozeroy', fillcolor='rgba(128, 0, 128, 0.2)')

# Save the plot in .svg format
fig.write_image("Mb(x)_plot.svg")
# Show the plotly figure
fig.show()
```