In [1]:
```python
import numpy as np

wheel_base = 1200       # [mm]
L = 3600                # [mm]
M = 200                 # [kg]
h = 75                  # [mm]
g = 9.81                # [m/s^2]
F_point = M * g / 4.0   # [N] (per wheel)

a = (L - wheel_base) / 2.0 # [mm] distance frome edge to point load (wheel)
```

Deformation functions

In [2]:
```python
def disp(x, P, a, L, EI):
    """ Fra formelsamling i mek 2. Bjelkeformel 12 """
    b = L - a
    if x <= a:
        return P*b*x/(6*EI*L) * (L**2 - b**2 - x**2)
    else:
        return P*a*(L-x)/(6*EI*L) * (2*L*x - a**2 - x**2)


def calculate_critical_disp(wheel_base, F1, F2, L, BendingStiffness):

    a1 = (L - wheel_base) / 2.0
    a2 = a1 + wheel_base

    max_disp_pos = L/2.0

    # Calculate deflection caused by each point load
    w = disp(max_disp_pos, F1, a1, L, BendingStiffness) + disp(max_disp_pos, F2, a2, L, BendingStiffness)

    print(f"Max displacement: {w}")
```

Deviations of second moment of area caused by mid plane offset in Abaqus

In [3]:
```python
t = 1
PROFILE_WIDTH = 200

h = 75
max_spars = 8

I_approx_list = np.zeros(max_spars+1)
I_analytical_list = np.zeros(max_spars+1)

# Calculate box profile
I_horizontal_approx = (1.0/12.0) * (PROFILE_WIDTH - t) * t**3 + ((h - t)/2.0)**2 * (PROFILE_WIDTH - t) * t # Steiners sats bunn eller topp
I_vertical_approx = (1.0/12.0) * (t) * (h - t)**3 # Steiners sats høyre eller venstre side
I_box_approx = 2 * I_horizontal_approx + 2 * I_vertical_approx
I_approx_list[0] = I_box_approx

# Calculate analytical
I_box_analytical = (1.0/12.0) * (PROFILE_WIDTH * h**3 - (PROFILE_WIDTH-2*t) * (h-2*t)**3)
I_analytical_list[0] = I_box_analytical
I_spar_analytical = (1.0/12.0) * t * (h-2*t)**3

# add spars
for i in range(max_spars):
    # add spar contribution to I and append to list
    I_box_approx += I_vertical_approx
    I_approx_list[i+1] = I_box_approx

    I_box_analytical += I_spar_analytical
    I_analytical_list[i+1] = I_box_analytical


print(np.round(I_approx_list/I_analytical_list * 100, 2))
```

```
[ 99.99 100.2   100.39 100.57 100.72 100.87 101.    101.12 101.24]
```

Calculate composite bending stiffness

The class `Profile` consists of rectangular sections from the class `MaterialSection`. `Profile` calculates the bending stiffness of a composite cross section.

In [ ]:
```python
import math

def rotated_inertia(b, h, theta_deg):
    """ Transformasjonsformler fra Mekanikk 2 Formelhefte side 16 """
    theta = math.radians(theta_deg)
    # For a Rectangle
    Iy = (b * h**3) / 12
    Iz = (h * b**3) / 12
    Iyz = 0
    Iyp = 0.5 * (Iy + Iz) + 0.5 * (Iy - Iz) * math.cos(2 * theta) - Iyz * math.sin(2 * theta)
    return Iyp
```

```python
class MaterialSection:
    def __init__(self, z_center, b, h, E_modulus, rotation_deg=0.0):
        self.z_center = z_center
        self.b = b
        self.h = h
        self.area = b*h
        self.E_modulus = E_modulus
        self.rotation_deg = rotation_deg

    @property
    def EA(self):
        return self.E_modulus * self.area

    @property
    def I_local(self):
        theta = math.radians(self.rotation_deg)
        # For a Rectangle
        Iy = (self.b * self.h**3) / 12
        Iz = (self.h * self.b**3) / 12
        Iyz = 0
        Iyp = 0.5 * (Iy + Iz) + 0.5 * (Iy - Iz) * math.cos(2 * theta) - Iyz * math.sin(2 * theta)
        return Iyp

    def __repr__(self):
        return f"MaterialSection(z_center={self.z_center}, area={self.area}, E_modulus={self.E_modulus})"


class Profile:
    def __init__(self):
        self.sections = []

    def add_section(self, z_center, b, h, E_modulus, rotation_deg=0.0):
        section = MaterialSection(z_center, b, h, E_modulus, rotation_deg=rotation_deg)
        self.sections.append(section)

    def compute_neutral_axis(self):
        """
        Beregn nøytralaksen (z-coordinate) for en komposittbjelkeprofil, tilsvarende formel i mekanikk 2 formelhefte.
        Nøytralaksen er definert ved:

            z_neutral = sum(EA * z_center) / sum(EA)

        Der EA = E_modulus * area for hver seksjon.
        """
        total_EA = sum(section.EA for section in self.sections)
        if total_EA == 0:
            raise ValueError("Total EA kan ikke være 0")
        numerator = sum(section.EA * section.z_center for section in self.sections)
        neutral_axis =  numerator / total_EA
        self.neutral_axis = neutral_axis
        return neutral_axis


    def compute_bending_stiffness(self):
        # First, compute the neutral axis
        self.compute_neutral_axis()

        EI = 0
        for section in self.sections:
            EI += section.E_modulus * (section.I_local + section.area * (section.z_center - self.neutral_axis)**2)
        self.bending_stiffness = EI
        return EI

    def calculate_cross_section_ares(self):
        cs_area = 0
        for section in self.sections:
            cs_area += section.area
        return cs_area


    def __repr__(self):
        sections_str = "\n".join(str(section) for section in self.sections)
        return f"Profile med {len(self.sections)} seksjoner:\n{sections_str}"
```

Inlcude laminates

In [5]:

```python
from laminatelib import laminateStiffnessMatrix, laminateThickness, repeatLayers, symmetricLayup
import matlib

def effective_laminate_Ex(layup):
    ABD = laminateStiffnessMatrix(layup)
    h=laminateThickness(layup)
    Ex = (1/h)*( ABD[0,0] - (ABD[0,1]**2)/ABD[1,1]  )
    return Ex
```

In [6]:

```python
def print_weigh_estimate(cross_section_area):
    rho_s_glass = 2.0E-6 # [kg/mm^3] Density of S-glass

    L = 3600

    # Total weight
```

```
    total_weight = rho_s_glass * cross_section_area * L
    print(f"Total weight per bridge component: {total_weight:.2f} kg")
```

Slanted Profile with 1 spar (Test Profile)

In [7]:
```
m1 = matlib.get('S-glass/Epoxy')

# Define top layup
layup_top = [
    {'mat':m1 , 'ori':  0  , 'thi':0.25},
    {'mat':m1 , 'ori': 90  , 'thi':0.25}
]
repeatLayers(layup_top, 2)
symmetricLayup(layup_top)

# Define bottom Layup
layup_bottom = layup_top

# Define side layups
layup_sides = [
    {'mat':m1 , 'ori':  45  , 'thi':0.25},
    {'mat':m1 , 'ori': -45  , 'thi':0.25}
]
repeatLayers(layup_sides, 2)
symmetricLayup(layup_sides)

# Get effective E-modulus
E_eff_top = effective_laminate_Ex(layup_top)
E_eff_bottom = effective_laminate_Ex(layup_bottom)
E_eff_sides = effective_laminate_Ex(layup_sides)

# Define custom profile
outer_height = 75
outer_width = 200
o1 = 10
o2 = 40


profile1 = Profile()

# TOP
t_top = laminateThickness(layup_top)
profile1.add_section(z_center=(outer_height-0.5*t_top),
                    b=outer_width,
                    h=t_top,
                    E_modulus=E_eff_top)
# BOTTOM
t_bottom = laminateThickness(layup_bottom)
profile1.add_section(z_center=(0.5*t_bottom),
                    b=(outer_width-2*o2),
                    h=t_bottom,
                    E_modulus=E_eff_bottom)

# SLANTED SIDE 1
t_side = laminateThickness(layup_sides)
length_slanted_side = np.sqrt((o2-o1)**2 + outer_height**2)
angle = math.degrees(np.arctan((o2-o1) / outer_height))
profile1.add_section(z_center=0.5*outer_height,
                    b=t_side,
                    h=length_slanted_side,
                    E_modulus=E_eff_sides,
                    rotation_deg=angle)

# SLANTED SIDE 2
profile1.add_section(z_center=0.5*outer_height,
                    b=t_side,
                    h=length_slanted_side,
                    E_modulus=E_eff_sides,
                    rotation_deg=angle)

# SPAR
t_spar = laminateThickness(layup_sides)
profile1.add_section(z_center=0.5*outer_height,
                    b=t_spar,
                    h=outer_height,
                    E_modulus=E_eff_sides)

# EFFECTIVE BENDING STIFFNESS
Db = profile1.compute_bending_stiffness()

calculate_critical_disp(wheel_base=1200, F1=(50*9.81), F2=(50*9.81), L=3600 , BendingStiffness=Db)
print_weigh_estimate(cross_section_area=profile1.calculate_cross_section_ares())
```

```
Max displacement: 20.427895342741124
Total weight per bridge component: 12.02 kg
```

Design 1

In [8]:
```
m1 = matlib.get('S-glass/Epoxy')
```

```python
# Define top layup
layup_top = [
    {'mat':m1 , 'ori':  0  , 'thi':0.25},
    {'mat':m1 , 'ori': 90  , 'thi':0.25},
    {'mat':m1 , 'ori':  0  , 'thi':0.25},
    {'mat':m1 , 'ori': 90  , 'thi':0.25},
    {'mat':m1 , 'ori': 90  , 'thi':0.25},
    {'mat':m1 , 'ori':  0  , 'thi':0.25},
    {'mat':m1 , 'ori': 90  , 'thi':0.25},
    {'mat':m1 , 'ori':  0  , 'thi':0.25},
]

# Define bottom Layup
layup_bottom = [
    {'mat':m1 , 'ori':  0  , 'thi':0.25},
    {'mat':m1 , 'ori': 90  , 'thi':0.25},
    {'mat':m1 , 'ori': 90  , 'thi':0.25},
    {'mat':m1 , 'ori':  0  , 'thi':0.25},
]

# Define side layups
layup_sides = [
    {'mat':m1 , 'ori':  45  , 'thi':0.25},
    {'mat':m1 , 'ori': -45  , 'thi':0.25},
    {'mat':m1 , 'ori':  45  , 'thi':0.25},
    {'mat':m1 , 'ori':  45  , 'thi':0.25},
    {'mat':m1 , 'ori': -45  , 'thi':0.25},
    {'mat':m1 , 'ori':  45  , 'thi':0.25},
]

# Define spar layup
layup_spar = [
    {'mat':m1 , 'ori':  45  , 'thi':0.25},
    {'mat':m1 , 'ori': -45  , 'thi':0.25},
    {'mat':m1 , 'ori':  45  , 'thi':0.25},
    {'mat':m1 , 'ori':  45  , 'thi':0.25},
    {'mat':m1 , 'ori': -45  , 'thi':0.25},
    {'mat':m1 , 'ori':  45  , 'thi':0.25},
]

# Get effective E-modulus
E_eff_top = effective_laminate_Ex(layup_top)
E_eff_bottom = effective_laminate_Ex(layup_bottom)
E_eff_sides = effective_laminate_Ex(layup_sides)
E_eff_spar = effective_laminate_Ex(layup_spar)

# Define custom profile
outer_height = 75
outer_width = 200
o1 = 10
o2 = 40


profile1 = Profile()

# TOP
t_top = laminateThickness(layup_top)
profile1.add_section(z_center=(outer_height-0.5*t_top),
                    b=outer_width,
                    h=t_top,
                    E_modulus=E_eff_top)
# BOTTOM
t_bottom = laminateThickness(layup_bottom)
profile1.add_section(z_center=(0.5*t_bottom),
                    b=(outer_width-2*o2),
                    h=t_bottom,
                    E_modulus=E_eff_bottom)

# SLANTED SIDE 1
t_side = laminateThickness(layup_sides)
length_slanted_side = np.sqrt((o2-o1)**2 + outer_height**2)
angle = math.degrees(np.arctan((o2-o1) / outer_height))
profile1.add_section(z_center=0.5*outer_height,
                    b=t_side,
                    h=length_slanted_side,
                    E_modulus=E_eff_sides,
                    rotation_deg=angle)

# SLANTED SIDE 2
profile1.add_section(z_center=0.5*outer_height,
                    b=t_side,
                    h=length_slanted_side,
                    E_modulus=E_eff_sides,
                    rotation_deg=angle)

# SPAR
t_spar = laminateThickness(layup_sides)
profile1.add_section(z_center=0.5*outer_height,
                    b=t_spar,
                    h=outer_height,
                    E_modulus=E_eff_spar)

# EFFECTIVE BENDING STIFFNESS
Db = profile1.compute_bending_stiffness()
```

```
calculate_critical_disp(wheel_base=1200, F1=(50*9.81), F2=(50*9.81), L=3600 , BendingStiffness=Db)
print_weigh_estimate(cross_section_area=profile1.calculate_cross_section_ares())
```

```
Max displacement: 43.99739524011364
Total weight per bridge component: 6.30 kg
```