Solution to Exercise 12 – Plate Buckling - Part 1 - Effective widths and Class 4 cross section resistance

May 26, 2022

1 Introduction

[1]:
```
import numpy as np
import pandas as pd
```

Class 4 cross-sections must ensure the following cross-section check, according to SIA263-§5.3.5:

$$\sigma_{Ed} = \frac{N_{Ed}}{A_{eff}} + \frac{M_{y,Ed} + N_{Ed} e_z}{W_{y,eff}} + \frac{M_{y,Ed} + N_{Ed} e_z}{W_{y,eff}} \leq \frac{f_y}{\gamma_{M1}} \tag{1}$$

In this exercise we are asked to verify cross-setion resistance solely to axial load, meaning that the above equation, worst-case scenario, will look like this:

$$\sigma_{Ed} = \frac{N_{Ed}}{A_{eff}} + \frac{N_{Ed} e_z}{W_{y,eff}} \leq \frac{f_y}{\gamma_{M1}} \tag{2}$$

Now, each effective geometric property in the above expression corresponds to different load conditions (pure compression and ppure bending), meaning that each of their effective properties need to be computed **separately**.

Before we continue let's define our geometric variables:

- The top flange will be called plate 1, with $t_{f1} = 10$mm and gross width $b_{f1} = 590$mm;
- The bottom flange will be called plate 2, with $t_{f2} = 10$mm and gross width $b_{f2} = 590$mm;
- The left plate will be called plate 3, with $t_{w3} = 10$mm and gross width $b_{w3} = 585$mm;
- The right plate will be called plate 4, with $t_{w4} = 10$mm and gross width $b_{w5} = 585$mm;

[2]:
```
t_f1=10.
b_f1=590.

t_f2=20.
b_f2=590.

t_w3=10.
b_w3=585.

t_w4=10.
```

```
b_w4=585.
```

2 Gross section properties

Let's pick as reference point the lower left corner of the intersecting mid-lines of our box-section

[3]:
```
grossSectionPlates={1:{'length_y':b_f1,
                       'length_z':t_f1,
                       'y':b_f1/2.,
                       'z':b_w3},
                    2:{'length_y':b_f2,
                       'length_z':t_f2,
                       'y':b_f1/2.,
                       'z':0.0},
                    3:{'length_y':t_w3,
                       'length_z':b_w3,
                       'y':0,
                       'z':b_w3/2.},
                    4:{'length_y':t_w4,
                       'length_z':b_w4,
                       'y':b_f1,
                       'z':b_w4/2.}}
# Cross-section Area

A_gross=0.0
for plateNo, plateChar in grossSectionPlates.items():
    A_gross+=plateChar['length_y']*plateChar['length_z']

print('The gross section area is equal to ',A_gross, ' mm2')

# Center of gravity
y_G=0.0
for plateNo, plateChar in grossSectionPlates.items():
    y_G+=plateChar['length_y']*plateChar['length_z']*plateChar['y']

y_G=y_G/A_gross


z_G=0.0
for plateNo, plateChar in grossSectionPlates.items():
    z_G+=plateChar['length_y']*plateChar['length_z']*plateChar['z']

z_G=z_G/A_gross

print('The gross section center of gravity is at y=',round(y_G,2), ' mm and at␣
 ↪z= ',round(z_G,2),' mm')
```

```
# Moments of Inertia

I_z_gross=0.0

for plateNo, plateChar in grossSectionPlates.items():
    I_z_gross+=plateChar['length_y']**3*plateChar['length_z']/12. +\
              ␣
  ↪plateChar['length_y']*plateChar['length_z']*(plateChar['y']-y_G)**2 ##␣
  ↪Steiner

print('The gross section moment of inertia about z is equal to ', I_z_gross, '␣
  ↪mm4')

I_y_gross=0.0

for plateNo, plateChar in grossSectionPlates.items():
    I_y_gross+=plateChar['length_y']*plateChar['length_z']**3/12. +\
              ␣
  ↪plateChar['length_y']*plateChar['length_z']*(plateChar['z']-z_G)**2 ##␣
  ↪Steiner

print('The gross section moment of inertia about y is equal to ', I_y_gross, '␣
  ↪mm4')
```

```
The gross section area is equal to  29400.0  mm2
The gross section center of gravity is at y= 295.0  mm and at z=  233.8  mm
The gross section moment of inertia about z is equal to  1531737500.0  mm4
The gross section moment of inertia about y is equal to  1747157735.9693878  mm4
```

Because we will be using this over and over let's create a function for the critical load (Eq. 12 in Part 1 of plate lecture series):

$$\sigma_{cr} = k\frac{\pi^2 E}{12(1-\nu^2)(b/t)^2)} \tag{3}$$

[4]:
```
def sigma_cr(k,E,nu,b,t):
    return k*np.pi**2*E/(12*(1-nu**2)*(b/t)**2)
```

3 Uniform compression load scenario

[5]:
```
compressionEffectivePlates={}
```

3.1 Section classification and effective width computations

A plate under pure compression has a slenderness limit for class 3 section of 42ε

The steel type requires us to use $\varepsilon = \sqrt{\frac{235}{fy}} = \sqrt{\frac{235}{275}} = 0.924$

[6]:
```
fy=275
epsilon=(235/275.)**0.5
epsilon
```

[6]:
```
0.9244162777371754
```

Meaning that the slenderness limit is

[7]:
```
42*0.924
```

[7]:
```
38.808
```

3.1.1 Plate 1

[8]:
```
b_f1/t_f1
```

[8]:
```
59.0
```

59.0>38.8 —-> Class 4

From slide 42, and SIA §5.6.4.3 we have that

$$b_{eff} = \frac{\bar{\lambda}_p - 0.055(3+\psi)}{\bar{\lambda}_p^2} b \tag{4}$$

also from Eq. 18 in Part 1 of plate lecture series,

$$\bar{\lambda}_p = \sqrt{\frac{f_y}{\sigma_{cr}}} \tag{5}$$

[9]:
```
k=4.0 #<--- uniform compression
sig_cr=sigma_cr(k,210e3,0.3,b_f1,t_f1)
lambda_p=(fy/sig_cr)**0.5
psi=1.0 #uniform compression

b_f1_eff=(lambda_p-0.055*(3+psi))/lambda_p**2*b_f1
b_f1_eff
```

[9]:
```
422.48393226647954
```

[10]:
```
compressionEffectivePlates.update({'1_left':{'length_y':b_f1_eff/2.,
                                             'length_z':t_f1,
                                              'y':b_f1_eff/4.,
                                              'z':b_w3}})
```

[11]:
```
compressionEffectivePlates.update({'1_right':{'length_y':b_f1_eff/2.,
                                              'length_z':t_f1,
                                               'y':b_f1-b_f1_eff/4.,
```

```
                                                'z':b_w3}})
```

3.1.2 Plate 2

[12]:
```
b_f2/t_f2
```

[12]: 29.5

29.5<38.8 —-> Class 3 or smaller

[13]:
```
compressionEffectivePlates.update({2:{'length_y':b_f2,
                                      'length_z':t_f2,
                                      'y':b_f1/2.,
                                      'z':0.0}})
```

3.1.3 Plates 3 and 4

[14]:
```
b_w3/t_w3
```

[14]: 58.5

[15]:
```
b_w4/t_w4
```

[15]: 58.5

58.0>38.8 —-> Class 4

[16]:
```
k=4.0 #<--- uniform compression
sig_cr=sigma_cr(k,210e3,0.3,b_w3,t_w3)
lambda_p=(fy/sig_cr)**0.5
psi=1.0 #uniform compression

b_w3_eff=(lambda_p-0.055*(3+psi))/lambda_p**2*b_w3
b_w3_eff
```

[16]: 421.60408253326756

[17]:
```
compressionEffectivePlates.update({'3_bottom':{'length_y':t_w3,
                                               'length_z':b_w3_eff/2.,
                                                'y':0.,
                                                'z':b_w3_eff/4.}})
```

[18]:
```
compressionEffectivePlates.update({'3_top':{'length_y':t_w3,
                                               'length_z':b_w3_eff/2.,
                                                'y':0.,
                                                'z':b_w3-b_w3_eff/4.}})
```

[19]:
```
k=4.0 #<--- uniform compression
sig_cr=sigma_cr(k,210e3,0.3,b_w4,t_w4)
```

```
lambda_p=(fy/sig_cr)**0.5
psi=1.0 #uniform compression

b_w4_eff=(lambda_p-0.055*(3+psi))/lambda_p**2*b_w4
b_w4_eff
```

[19]:
```
421.60408253326756
```

[20]:
```
compressionEffectivePlates.update({'4_bottom':{'length_y':t_w4,
                                               'length_z':b_w4_eff/2.,
                                                'y':b_f1,
                                                'z':b_w4_eff/4.}})
```

[21]:
```
compressionEffectivePlates.update({'4_top':{'length_y':t_w4,
                                               'length_z':b_w4_eff/2.,
                                                'y':b_f1,
                                                'z':b_w4-b_w4_eff/4.}})
```

3.2 Effective properties

[22]:
```
# Effective Area - Compression

A_eff_comp=0.0
for plateNo, plateChar in compressionEffectivePlates.items():
    A_eff_comp+=plateChar['length_y']*plateChar['length_z']

print('The effective (compression) area is equal to ',A_eff_comp, ' mm2')

# Center of gravity
y_G_eff_comp=0.0
for plateNo, plateChar in compressionEffectivePlates.items():
    y_G_eff_comp+=plateChar['length_y']*plateChar['length_z']*plateChar['y']

y_G_eff_comp=y_G_eff_comp/A_eff_comp


z_G_eff_comp=0.0
for plateNo, plateChar in compressionEffectivePlates.items():
    z_G_eff_comp+=plateChar['length_y']*plateChar['length_z']*plateChar['z']

z_G_eff_comp=z_G_eff_comp/A_eff_comp

print('The effective (compression) center of gravity is at␣
 ↪y=',round(y_G_eff_comp,2), ' mm and at z= ',round(z_G_eff_comp,2),' mm')

# Moments of Inertia
```

```
I_z_eff_comp=0.0

for plateNo, plateChar in compressionEffectivePlates.items():
    I_z_eff_comp+=plateChar['length_y']**3*plateChar['length_z']/12. +\
                 ␣
 ↪plateChar['length_y']*plateChar['length_z']*(plateChar['y']-y_G_eff_comp)**2␣
 ↪## Steiner

print('The effective (compression) moment of inertia about z is equal to ',␣
 ↪I_z_eff_comp, ' mm4')

I_y_eff_comp=0.0

for plateNo, plateChar in compressionEffectivePlates.items():
    I_y_eff_comp+=plateChar['length_y']*plateChar['length_z']**3/12. +\
                 ␣
 ↪plateChar['length_y']*plateChar['length_z']*(plateChar['z']-z_G_eff_comp)**2␣
 ↪## Steiner

print('The effective (compression) moment of inertia about y is equal to ',␣
 ↪I_y_eff_comp, ' mm4')
```

```
The effective (compression) area is equal to  24456.920973330147  mm2
The effective (compression) center of gravity is at y= 295.0  mm and at z=
201.9  mm
The effective (compression) moment of inertia about z is equal to
1243402360.9914234  mm4
The effective (compression) moment of inertia about y is equal to
1497112560.0709546  mm4
```

Then the eccentricity is,

[23]:
```
e_z=z_G-z_G_eff_comp
e_z
```

[23]:
```
31.898463174436557
```

4 Pure bending load scenario

[24]:
```
bendingEffectivePlates={}
```

4.1 Section classification and effective width computations

A plate under pure compression has a slenderness limit for class 3 section of 42ε

The steel type requires us to use $\varepsilon = \sqrt{\frac{235}{fy}} = \sqrt{\frac{235}{275}} = 0.924$

[25]:
```
fy=275
epsilon=(235/275.)**0.5
epsilon
```

[25]:
```
0.9244162777371754
```

Meaning that the slenderness limit is

[26]:
```
42*0.924
```

[26]:
```
38.808
```

4.1.1 Plate 1 - under pure compression (same as before)

[27]:
```
b_f1/t_f1
```

[27]:
```
59.0
```

59.0>38.8 —-> Class 4

From slide 42, and SIA §5.6.4.3 we have that

$$b_{eff} = \frac{\bar{\lambda}_p - 0.055(3+\psi)}{\bar{\lambda}_p^2} b \tag{6}$$

also from Eq. 18 in Part 1 of plate lecture series,

$$\bar{\lambda}_p = \sqrt{\frac{f_y}{\sigma_{cr}}} \tag{7}$$

[28]:
```
k=4.0 #<--- uniform compression
sig_cr=sigma_cr(k,210e3,0.3,b_f1,t_f1)
lambda_p=(fy/sig_cr)**0.5
psi=1.0 #uniform compression

b_f1_eff=(lambda_p-0.055*(3+psi))/lambda_p**2*b_f1
b_f1_eff
```

[28]:
```
422.48393226647954
```

[29]:
```
bendingEffectivePlates.update({'1_left':{'length_y':b_f1_eff/2.,
                                         'length_z':t_f1,
                                          'y':b_f1_eff/4.,
                                          'z':b_w3}})
```

[30]:
```
bendingEffectivePlates.update({'1_right':{'length_y':b_f1_eff/2.,
                                         'length_z':t_f1,
                                          'y':b_f1-b_f1_eff/4.,
```

```
                                                'z':b_w3}})
```

4.1.2 Plate 2 - under tension

[31]:
```
b_f2/t_f2
```

[31]: 29.5

29.5<38.8 —-> Class 3 or smaller

[32]:
```
bendingEffectivePlates.update({2:{'length_y':b_f2,
                                  'length_z':t_f2,
                                  'y':b_f1/2.,
                                  'z':0.0}})
```

4.1.3 Plates 3 and 4 under pure bending

[33]:
```
b_w3/t_w3
```

[33]: 58.5

[34]:
```
b_w4/t_w4
```

[34]: 58.5

The slenderness limits here are somewhat a chicken and the egg situation: the classification depends on the uniaxial stress distribution that depends on effective center of gravity, but the effective center of gravity can only be computed if we know the section classification and its effective width.

The solution for this is to iterate. First, we need to arbitrate a stress distribution. Then we calculate the effective properties. After the effective properties are calculated, we re-do the effective properties with that stress distribution. And so on until we converge.

As a first guess I'll use a stress distrbution in the web for the gross section.

[35]:
```
psi= z_G/(z_G-b_w3)
psi
```

[35]: -0.6657223796033994

The slenderness limit from SIA263 is given by

$$\frac{42\varepsilon}{0.67+0.33\psi} \tag{8}$$

[36]:
```
42*0.924/(0.67+0.33*psi)
```

[36]: 86.18032209360845

58.5<86.18 —-> Class 3 or smaller

And so, in this case, we don't even have to compute effective widths and can just add the webs to our cross-section

[37]:
```
bendingEffectivePlates.update({3:{'length_y':t_w3,
                                  'length_z':b_w3,
                                  'y':0,
                                  'z':b_w3/2.},
                               4:{'length_y':t_w4,
                                  'length_z':b_w4,
                                  'y':b_f1,
                                  'z':b_w4/2.}})
```

4.2 Effective properties

[38]:
```
# Effective Area - Bending

A_eff_bend=0.0
for plateNo, plateChar in bendingEffectivePlates.items():
    A_eff_bend+=plateChar['length_y']*plateChar['length_z']

print('The effective (bending) area is equal to ',A_eff_bend, ' mm2')

# Center of gravity
y_G_eff_bend=0.0
for plateNo, plateChar in bendingEffectivePlates.items():
    y_G_eff_bend+=plateChar['length_y']*plateChar['length_z']*plateChar['y']

y_G_eff_bend=y_G_eff_bend/A_eff_bend


z_G_eff_bend=0.0
for plateNo, plateChar in bendingEffectivePlates.items():
    z_G_eff_bend+=plateChar['length_y']*plateChar['length_z']*plateChar['z']

z_G_eff_bend=z_G_eff_bend/A_eff_bend

print('The effective (bending) center of gravity is at␣
 ↪y=',round(y_G_eff_bend,2), ' mm and at z= ',round(z_G_eff_bend,2),' mm')

# Moments of Inertia

I_z_eff_bend=0.0

for plateNo, plateChar in bendingEffectivePlates.items():
    I_z_eff_bend+=plateChar['length_y']**3*plateChar['length_z']/12. +\
```

```
                ␣
 ↪plateChar['length_y']*plateChar['length_z']*(plateChar['y']-y_G_eff_bend)**2␣
 ↪## Steiner

print('The effective (bending) moment of inertia about z is equal to ',␣
 ↪I_z_eff_bend, ' mm4')

I_y_eff_bend=0.0

for plateNo, plateChar in bendingEffectivePlates.items():
    I_y_eff_bend+=plateChar['length_y']*plateChar['length_z']**3/12. +\
                ␣
 ↪plateChar['length_y']*plateChar['length_z']*(plateChar['z']-z_G_eff_bend)**2␣
 ↪## Steiner

print('The effective (bending) moment of inertia about y is equal to ',␣
 ↪I_y_eff_bend, ' mm4')
```

```
The effective (bending) area is equal to  27724.839322664797  mm2
The effective (bending) center of gravity is at y= 295.0  mm and at z=  212.58
mm
The effective (bending) moment of inertia about z is equal to
1527820187.9951823  mm4
The effective (bending) moment of inertia about y is equal to
1528044343.5652056  mm4
```

Let's re-check if the webs with this new position are still class 3

[39]:
```
psi= z_G_eff_bend/(z_G_eff_bend-b_w3)
psi
```

[39]:
```
-0.5708124262133029
```

The slenderness limit from SIA263 is given by

$$\frac{42\varepsilon}{0.67+0.33\psi} \tag{9}$$

[40]:
```
42*0.924/(0.67+0.33*psi)
```

[40]:
```
80.57605829764569
```

58.5<80.7 —-> Class 3 or smaller

The webs are still class 3, and so we can proceed with the code checks

For the code check we will need the section modulus about 'y' for compression from the incremental bending moment (top fiber):

[41]:
```
W_y_eff_bend=I_y_eff_bend/(b_w3-z_G_eff_bend)
W_y_eff_bend
```

[41]:
```
4103027.423379785
```

5 Code verification

[42]:
```
N_Ed=5250e3#N

sigma_Ed=N_Ed/A_eff_comp+(N_Ed*e_z)/W_y_eff_bend
sigma_Ed
```

[42]:
```
255.4786175756846
```

[43]:
```
gamma_M1=1.05
fy/gamma_M1
```

[43]:
```
261.90476190476187
```

[44]:
```
sigma_Ed<fy/gamma_M1
```

[44]:
```
True
```

The cross section passes in sectional resistance.

[ ]:
```

```