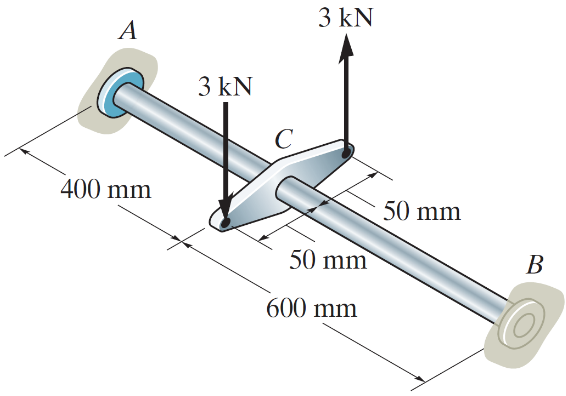
1 1.20

Az ábrán látható befogott, tömör kör keresztmetszetű alumínium tengelyre két koncentrált erő hat.

- Határozzuk meg a befogásokban ébredő reakciónyomatékokat
- Méretezzük a tengelyt, ha $\tau_{\mathrm{meg}} = 100\,\mathrm{MPa}$
- Határozzuk meg a C keresztmetszet A-hoz képesti elcsavarodási szögét



1.1 Megoldás

Első lépésként importáljuk a szimbolikus számításhoz szükséges modult, és felvesszük a megadott adatokat.

In [1]:

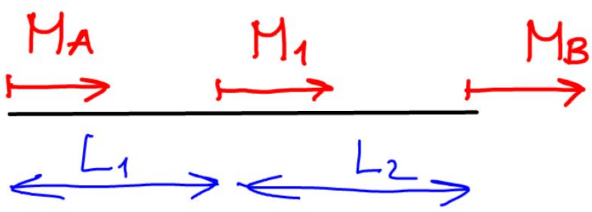
```
import sympy as sp
sp.init_printing()

M_A, M_B, I_p, d_min = sp.symbols('M_A,M_B,I_p,d_min')

# Adatok
E = 70e3     #MPa
ν = 0.34     #[-] (\nu)
L_1 = 400    #mm
L_2 = 600    #mm
F = 3e3      #N
τ_meg = 100 #MPa

# A csavaró rugalmassági moduluszra is szükségünk lesz
G = E/(2*(1+ν))

# 'round()' magyarázatát lásd a notebook végén
round(G) #MPa
```

executed in 1.39s, finished 13:22:22 2020-03-05

Out[1]:

$$26119$$

A reakciónyomatékok meghatározásához rajzoljuk fel a szerkezetre ható nyomatékokat.



In [2]:

```
M_1 = 50*F + 50*F
M_1 #Nmm
```

executed in 495ms, finished 13:22:23 2020-03-05

Out[2]:

300000.0

A reakciónyomatékok meghatározásához ismerjük fel, hogy a befogásoknál a keresztmetszetek elcsavarodási szöge zérus. Ezt felhasználva felírhatunk egy alakváltozási feltételt a B keresztmetszetre:

$$\frac{M_\mathrm{A} L_1}{I_\mathrm{p} G} + \frac{(M_\mathrm{A} + M_1) L_2}{I_\mathrm{p} G} = 0.$$

In [3]:

```
# Írjuk fel a fenti egyenlet bal oldalát az 'egyenlet' változóban.
egyenlet = ((M_A*L_1)/(I_p*G) + (M_A+M_1)*L_2/(I_p*G))

# Tároljuk el az egyenlet megoldását a 'sol' változóban
sol = sp.solve(egyenlet, M_A)
# Az eredményt egy 'list' objektumban kapjuk, aminek több eleme lenne, ha az eg

# 'M_An' legyen egyenlő a 'sol' nevű list első elemével:
M_An = sol[0]

M_An.evalf(4) #Nmm
```

executed in 582ms, finished 13:22:23 2020-03-05

Out[3]:

$-1.8 \cdot 10^5$

Mivel a szerkezet egyensúlyban van, ezért:

$$M_\mathrm{B} + M_\mathrm{A} + M_1 = 0.$$

In [4]:

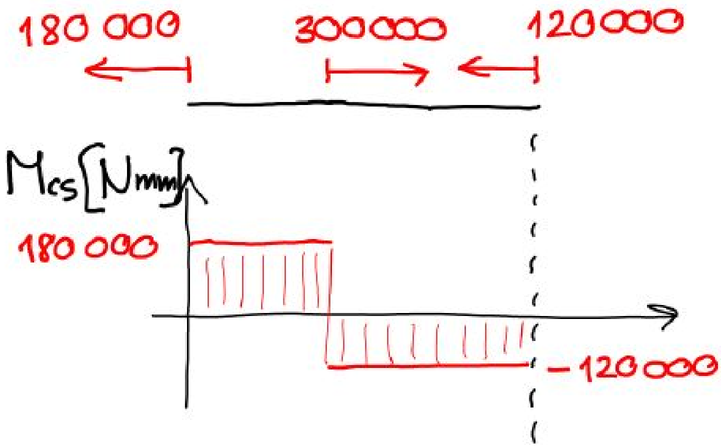
```python
# Írjuk fel a fenti egyenlet bal oldalát az 'egyenlet2' változóban.
egyenlet2 = M_B + M_A + M_1

sol2 = sp.solve(egyenlet2, M_B) # Megoldjuk 'M_B'-re

M_Bn = sol2[0].subs(M_A,M_An)

M_Bn.evalf(4) #Nmm
```

executed in 636ms, finished 13:22:24 2020-03-05

Out[4]:

$$-1.2 \cdot 10^5$$

Az igénybevételi ábra:



A minimális tengelyvastagság meghatározásához felírhatjuk a kritikus keresztmetszet megfelelőségi kritériumát, miszerint:

$$\frac{M_{\mathrm{cs,max}}}{K_{\mathrm{p}}} = \tau_{\mathrm{meg}},$$

ahol:

$$K_{\mathrm{p}} = \frac{d_{\mathrm{min}}^3 \pi}{16}.$$

In [5]:

```
# A maximális csavaró nyomaték a rúdban:
M_cs_max = abs(M_An)

# Írjuk fel a fenti 2 egyenletet:
K_p = d_min**3*sp.pi/16
egyenlet3 = M_cs_max/K_p - τ_meg

# Megj.: Vegyük észre, hogy a szimbolikus megoldó tudja kezelni az egymásba lán
#        Nem kellett a 'K_p'-t kézzel behelyettesítenünk a megfelelőségi egyenl
#        kell, hogy a deklarálási sorrendet betartsuk. Az 'egyenlet3' előtt kel
#        szimbolikus kifejezés, különben hibát kapunk.

sol3 = sp.solve(egyenlet3, d_min)

# Itt láthatjuk, hogy a 'solve' az összes lehetséges gyököt megtalálja!
display([sol.evalf(5) for sol in sol3])

d_min_n = sol3[0] # Természetesen minket a valós gyök érdekel (a képzetes egysé

display(round(d_min_n,5)) #mm
```

executed in 1.32s, finished 13:22:25 2020-03-05

$$[20.929, \ -10.464 - 18.125i, \ -10.464 + 18.125i]$$

$$20.92895$$

A minimális átmérő ismeretében kiszámolhatjuk a rúd poláris másodrendű nyomatékát, és felírhatjuk a kritikus keresztmetszet szögelcsavarodását.

In [6]:

```
I_p = d_min**4*sp.pi/32
display(I_p.evalf(5)) #mm^4

ϕ_CA = M_cs_max*L_1/(I_p*G)
display(ϕ_CA.evalf(5)) #rad
display((ϕ_CA*180/sp.pi).evalf(5)) #fok
```

executed in 1.67s, finished 13:22:27 2020-03-05

$$0.098175 d_{min}^4$$

$$\frac{28078.0}{d_{min}^4}$$

$$\frac{1.6088 \cdot 10^6}{d_{min}^4}$$

1.2 + Extra a kíváncsiaknak

Felmerülhet az a kérdés, hogy miért használtuk a `round()` metódust a kerekítésekhez, mikor már az előző feladatokban megismerkedtünk az `.evalf()` metódussal. Ha látszólag ugyan hasonló feladatot is lát el a két függvény, vannak közöttük fontos különbségek.
A változóink deklarálásakor a Python egy osztály elemeként tárolja a változókat (pl: `int` , `float` , `list` ...), ami meghatározza azt is, hogy milyen metódusokat ('parancsokat', 'függvényeket') tudunk hívni az adott

változón. Ez egy természetes megszorítás, hiszen szeretnénk elkerülni, hogy a programunk logikailag értelmetlen kérdésekre eredményeket adjon. Pl: `round('sziltan') -> !?!?` . Sajnos ennek a problémának a kevésbé egyértelmű esetei változatos hibákat szülhetnek a kódunkban.
Nézzük meg, hogy az eddig használt változóink milyen osztályba tartoznak a program szerint:

In [7]:

```python
n = 10
pi_1 = 3.141592653
lista = [1, 2, 3]
konyvtar = {'sziltan': 5, 'matekG2': 4, 'anyagtech': 3}
pi_2 = sp.pi
i = sp.symbols('i')

# Nézzük meg a fenti változók milyen osztályba tartoznak
display(type(n))
display(type(pi_1))
display(type(lista))
display(type(konyvtar))
display(type(pi_2))
display(type(i))
```

executed in 18ms, finished 13:22:27 2020-03-05

```
int

float

list

dict

sympy.core.numbers.Pi

sympy.core.symbol.Symbol
```

Látható, hogy a `sympy` könyvtár függvényeivel létrehozott változóink a `sympy` saját osztályainak az elemei. Ebből már meg lehet sejteni, hogy a Python gyári metódusai nem biztos, hogy ezeket a kiterjesztett osztályokat hiba nélkül tudják kezelni. Nézzünk néhány példát arra, amikor jól működnek a dolgok:

In [8]:

```python
pi_1 = 3.141592653
pi_1_kerek = round(pi_1)
display('Eredeti: '+str(type(pi_1))+' Új: '+str(type(pi_1_kerek)))

pi_2 = sp.pi
pi_2_kerek = pi_2.evalf(5)
display('Eredeti: '+str(type(pi_2))+' Új: '+str(type(pi_2_kerek)))
```

executed in 30ms, finished 13:22:27 2020-03-05

```
"Eredeti: <class 'float'> Új: <class 'int'>"

"Eredeti: <class 'sympy.core.numbers.Pi'> Új: <class 'sympy.core.numb
ers.Float'>"
```

Látszik, hogy mikor műveleteket végzünk a változóinkon (pl.: kerekítjük őket), akkor azoknak nem csak a numerikus értéke, de a programban tárolt osztálya is változik. Ezt a Python dinamikus változó kezelése teszi lehetővé, ami elég sok átalakítást képes automatikusan elvégezni számunkra. Lássunk egy példát arra amikor ez mégsem működik:

In [9]:

```
pi_1 = 3.141592653
pi_1.evalf(5)
```

executed in 134ms, finished 13:22:27 2020-03-05

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-9-69b262bfe70e> in <module>
      1 pi_1 = 3.141592653
----> 2 pi_1.evalf(5)

AttributeError: 'float' object has no attribute 'evalf'
```

Hibaüzentet kaptunk amikor a saját `float` változónkat próbáltuk -a csak `sympy` osztályok álltal ismert- `.evalf()` metódussal kerekíteni. Az ehhez hasonló hibák összezavaróak lehetnek, hiszen egy logikailag és szemantikailag értelmes parancsunk nem futott le. Ezeknek a kezeléséhez tartsuk észben, hogy a változóink osztálya meghatározza a rajtuk végezhető műveleteket is!