# STÆ405G Töluleg greining
Verkefni 1

Andri Freyr Viðarsson og Valgeir Einarsson

Birgir Hrafnkelsson

21.febrúar 2020

## 1.

Write a Matlab(Python) program to define the structure matrix $A$ in (2.34). Then, using the Matlab \command or code of your own design, solve the system for the displacements $y_i$ using $n = 10$ grid steps.
**Lausn:**

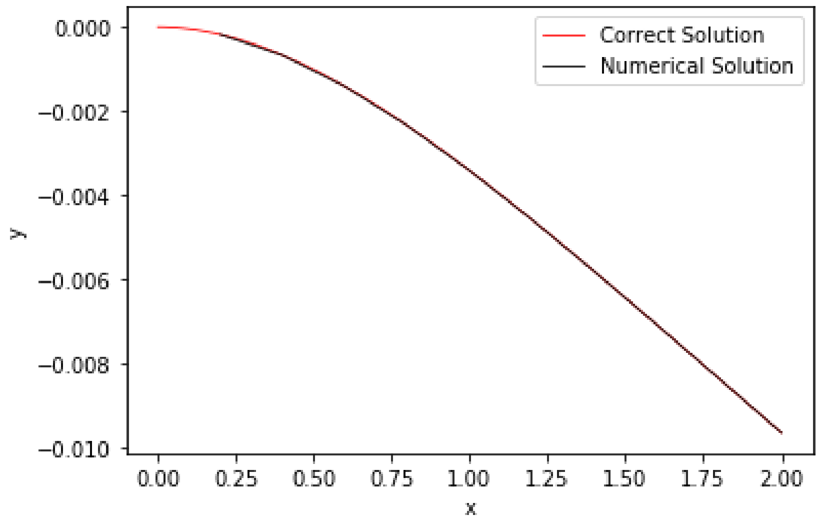
```python
import numpy as np
import scipy as sci
from scipy.constants import g
import sympy as sp
import matplotlib.pyplot as plt
def struct(n):
    l_1 = [16, -9, 8/3, -1/4]
    l_2 = [-4, 6, -4, 1]
    b_2 = [16/17, -60/17, 72/17, -28/17]
    b_1 = [-12/17, 96/17, -156/17, 72/17]
    const = [1, -4, 6, -4, 1]
    A = np.zeros((n,n))
    A[0, 0:4] = l_1
    A[1, 0:4] = l_2
    A[n-2, -4:] = b_2
    A[n-1, -4:] = b_1
    col_index = 0
    for i in range(2, n-2): # fyllum út aðra dálka
        A[i, col_index:col_index+5] = const
        col_index +=1
    return A
n = 10
A_struct = struct(n)
L = 2
h = L /n
w = 0.3
d = 0.03
I = (w*d**3)/12
g = sci.constants.g
E = 1.3e10
f_const = -480*w*d*g
f_mat = ((h**4)/(E*I))*f_const*np.ones(n)
Y_1 = np.linalg.solve(A_struct, f_mat) #leysum kerfið
```

## 2.

Plot the solution from Step 1 against the correct solution $y(x) = (f/24EI)x^2(x^2 - 4Lx + 6L^2)$, where $f = f(x)$ is the constant defined above. Check the error at the end of the beam, $x = L$ meters. In this simple case the derivative approximations are exact, so your error should be near machine roundoff.

**Lausn:**

```python
x = np.linspace(0, L, 100)
def correct_y(x, f, E, I, L):
    y = (f/(24*E*I))*(x**2)*((x**2)-4*L*x+6*L**2)
    return y
y_corr = correct_y(x, f_const, E, I, L)
x_int = np.linspace(L/n, L, n)
plt.plot(x, y_corr, c='r', label= 'Correct Solution',linewidth=0.8)
plt.plot(x_int, Y_1, c='black', label ='Numerical Solution', linewidth=0.8)
plt.xlabel('x')
plt.ylabel('y')
plt.legend()
plt.show()
end_error = np.abs(y_corr[-1]-Y_1[-1])
print('Error at end of beam:', end_error)
```



```
>    Error at end of beam: 1.8561541192951836e-16
```

## 3.

Rerun the calculation in Step 1 for $n = 102^k$, where $k = 1, ..., 11$. Make a table of the errors at $x = L$ for each $n$. For which $n$ is the error smallest? Why does the error begin to increase with $n$ after a certain point? You may want to make an accompanying table of the condition number of $A$ as a function of $n$ to help answer the last question. To carry out this step for large $k$, you may need to ask Matlab to store the matrix $A$ as a sparse matrix to avoid running out of memory. To do this, just initialize $A$ with the command $A = \text{sparse}(n, n)$, and proceed as before. We will discuss sparse matrices in more detail in the next section.

**Lausn:**

Ath. að þegar cond(A) er reiknað er notast við maximum absolute row sum eins og skilgreint er í bók, en ekki maximum absolute colmun sum eins og er sjálfgefið í matlab.

```
from scipy.sparse.linalg import spsolve
from scipy.sparse import csc_matrix
# skilgreinum fall sem reiknar cond fyrir sparse fylki
def sp_cond(A):
    norm_A = sci.sparse.linalg.norm(A, ord = np.inf)
    norm_A_inv = sci.sparse.linalg.norm(sci.sparse.linalg.inv(A), ord = np.inf)
    cond_A = norm_A*norm_A_inv
    return cond_A

end_errors = []
cond_nums = []
for k in range(1,12):
    n = 10*(2**k)
    h = L/n
    f_iter = ((h**4)/(E*I))*f_const*np.ones(n)
    A = struct(n)
    A_sparse = csc_matrix(A)
    cond_A = sp_cond(A_sparse)
    y = spsolve(A_sparse, f_iter)
    end_error = np.abs(y_corr[-1] - y[-1]) #error at x = L
    end_errors.append(end_error)
    cond_nums.append(cond_A)
```

```
from tabulate import tabulate
nums =[10*2**k for k in range(1,12)]
header = ['n', 'end_errors', 'cond(A)']
table_mat = np.array([nums, end_errors, cond_nums])
table_mat = table_mat.T
table_1 = tabulate(table_mat, header, tablefmt = 'latex',
                   numalign = 'right')
print(table_1)
```

| n | end_errors | cond(A) |
|---:|---:|---:|
| 20 | 8.5279e-15 | 558333 |
| 40 | 5.00659e-14 | 8.93333e+06 |
| 80 | 1.46773e-12 | 1.42933e+08 |
| 160 | 1.06958e-11 | 2.28693e+09 |
| 320 | 2.00621e-10 | 3.65909e+10 |
| 640 | 2.74567e-09 | 5.85455e+11 |
| 1280 | 1.63032e-08 | 9.36726e+12 |
| 2560 | 5.41588e-07 | 1.49868e+14 |
| 5120 | 3.52549e-07 | 2.39794e+15 |
| 10240 | 3.37427e-06 | 3.83818e+16 |
| 20480 | 4.09825e-05 | 6.165e+17 |

Af töflunni má sjá að frávikið frá réttri lausn í $x = L$ eykst með stærð fylkisins $A$. Sjáum að ástandstala fylkisins eykst einnig og því má álykta að hækkandi frávik sé til komið vegna óstöðugleika í fylkjunum þegar stærð þeirra eykst.

## 4.

Add a sinusoidal pile to the beam. This means adding a function of form $s(x) = -pg \sin \frac{\pi}{L}x$ to the force term $f(x)$. Prove that the solution

$$y(x) = \frac{f}{24EI}x^2\left(x^2 - 4Lx + 6L^2\right) - \frac{pgL}{EI\pi}\left(\frac{L^3}{\pi^3}\sin\frac{\pi}{L}x - \frac{x^3}{6} + \frac{L}{2}x^2 - \frac{L}{\pi^2}x\right)$$

satisfies the Euler–Bernoulli beam equation and the clamped-free boundary conditions.

**Lausn:**

Euler-Bernoulli jafnan segir að lóðrétt tilfærsla, $y(x)$, fasts innspennts bita í annan endann undir álagi, $f(x)$, uppfyllir: $EIy''''(x) = f(x)$, þar sem I er flatarvægi bita og E er Young's stuðull efnis. Diffrum því með aðstoð sympy til að meta afleiðujöfnuna og jaðarskilyrðin $y(0) = y'(0) = y''(L) = y'''(L) = 0$. Úttak if setningarinnar mun svo segja til um hvort öll skilyrði séu uppfyllt fyrir lausnina sem gefin er hér fyrir ofan.

```
x, E, I ,L, p, g, w, d, f = sp.symbols('x E I L p g w d f')
y_x = (f/(24*E*I))*x**2*(x**2-4*L*x+6*L**2)-((p*g*L)/(E*I*sp.pi))*((L**3/sp.pi**3)*sp.sin(sp.p
f_x = -p*g*sp.sin((sp.pi/L)*x)+f
left_side_eq = E*I*sp.diff(y_x, x, 4)  #vinstri hlið
ydot_x = sp.diff(y_x, x, 1)
y2dot_x = sp.diff(y_x, x, 2)
y3dot_x = sp.diff(y_x, x, 3)
# gerum næst sympy expression-in callable með lambdify
y = sp.lambdify([x, E, I ,L, p, g, w, d, f, sp.pi], y_x, 'numpy')
ydot = sp.lambdify([x, E, I ,L, p, g, w, d, f, sp.pi], ydot_x, 'numpy')
y2dot = sp.lambdify([x, E, I ,L, p, g, w, d, f, sp.pi], y2dot_x, 'numpy')
y3dot = sp.lambdify([x, E, I ,L, p, g, w, d, f, sp.pi], y3dot_x, 'numpy')
L = 2
w = 0.3
d = 0.03
I = (w*d**3)/12
g = sci.constants.g
E = 1.3e10
f = -480*w*d*g
p=100
#skoðum hér upphafsskilyrði og stóru afleiðujöfnuna
if (left_side_eq == f_x and y(0, E, I ,L, p, g, w, d, f, np.pi) == 0 and
    ydot(0, E, I ,L, p, g, w, d, f, np.pi) == 0 and
    y2dot(L, E, I ,L, p, g, w, d, f, np.pi) == 0 and
    y3dot(L, E, I ,L, p, g, w, d, f, np.pi) == 0):
    print('Euler-Bernoulli equation satisfied')
```

```
>   Euler-Bernoulli equation satisfied
```

## 5.

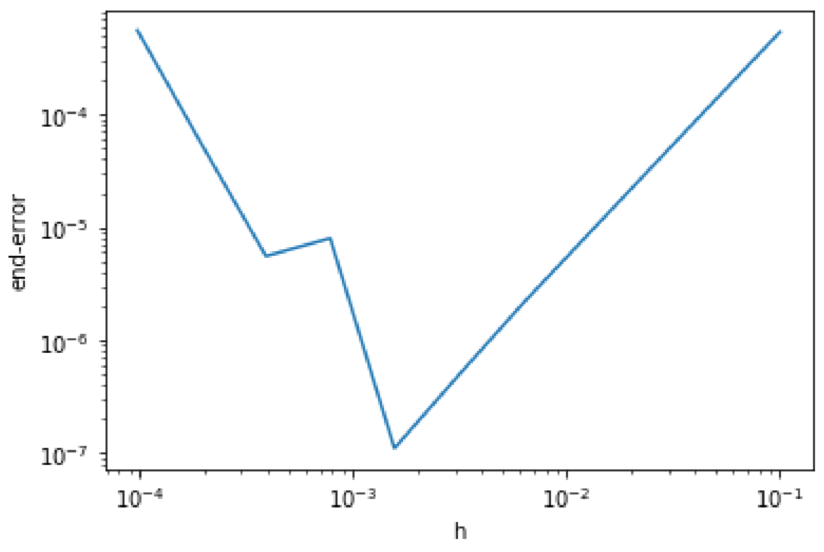
Rerun the calculation as in Step 3 for the sinusoidal load. (Be sure to include the weight of the beam itself.) Set $p = 100$ kg/m and plot your computed solutions against the correct solution. Answer the questions from Step 3, and in addition the following one: Is the error at $x = L$ proportional to $h^2$ as claimed above? You may want to plot the error versus $h$ on a log–log graph to investigate this question. Does the condition number come into play?
**Lausn:**

```
def y_x(x):
    y_x_1 = (f/(24*E*I))*x**2*(x**2-4*L*x+6*L**2)
    y_x_2 = ((p*g*L)/(E*I*np.pi))*((L**3/np.pi**3)*np.sin(np.pi/L*x)-(x**3/6)+(L/2)*x**2-(L**2
    y_x = y_x_1 - y_x_2
    return y_x

y_end = y_x(L)
end_errors = []
h_list = []
for k in range(1,12):
    n = 10*(2**k)
    h = L/n
    h_list.append(h)
    x = np.linspace(L/n, L, n)
    f_iter = ((h**4)/(E*I))*(f_const-g*p*np.sin((np.pi*x)/L))
    A = struct(n)
    A_sparse = csc_matrix(A)
    y = spsolve(A_sparse, f_iter)
    end_error = np.abs(y[-1] - y_end)
    end_errors.append(end_error)
```

Teiknum graf á loglog skala sem sýnir samband fráviksins í $x = L$ og $h$. Taflan setur myndina í samhengi við ástandstölu fylkisins í hverri ítrun.

```
plt.loglog(h_list, end_errors)
plt.xlabel('h')
plt.ylabel('end-error')
plt.show()
header =['n', 'end-error','cond(A)']
table_mat = np.array([nums, end_errors, cond_nums])
table_mat = table_mat.T
table_2 = tabulate(table_mat, header, tablefmt = 'fancy_grid',
                  numalign = 'right')
print(table_2)
```



| n | end-error | cond(A) |
|---:|---:|---:|
| 20 | 0.000537512 | 558333 |
| 40 | 0.000135419 | 8.93333e+06 |
| 80 | 3.39193e-05 | 1.42933e+08 |
| 160 | 8.48371e-06 | 2.28693e+09 |
| 320 | 2.11826e-06 | 3.65909e+10 |
| 640 | 4.89978e-07 | 5.85455e+11 |
| 1280 | 1.1021e-07 | 9.36726e+12 |
| 2560 | 7.98603e-06 | 1.49868e+14 |
| 5120 | 5.55426e-06 | 2.39794e+15 |
| 10240 | 5.42951e-05 | 3.83818e+16 |
| 20480 | 0.000555509 | 6.165e+17 |

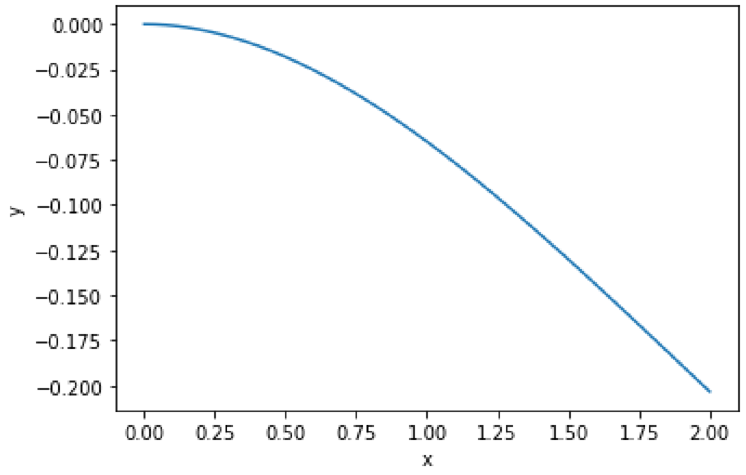
Þegar horft er á grafið hér að ofan frá hægri má sjá að hallatala á log skala virðist vera u.þ.b 2 sem gefur til kynna að frávikið sé í hlutfalli við $h^2$. Þó má sjá að frávikið eykst síðan aftur eftir ákveðið gildi á $n$. Það er sennilega vegna þess að $A$ fylkið er orðið mjög óstöðugt eins og sjá má í töflu.

## 6.

Now remove the sinusoidal load and add a 70 kg diver to the beam, balancing on the last 20 cm of the beam. You must add a force per unit length of $-g$ times 70/0.2 kg/m to $f(x_i)$ for all $1.8 \leq x_i \leq 2$, and solve the problem again with the optimal value of n found in Step 5. Plot the solution and find the deflection of the diving board at the free end.

**Lausn:**

```
min_error = end_errors.index(min(end_errors)) # finnum besta gildi á n
n_opt = int(L/h_list[min_error])
h = L/n_opt
x = np.linspace(L/n_opt, L, n_opt)
f_iter = np.zeros(n_opt)
for i in range(0, n_opt):
    if x[i]<1.8:
        f_iter[i]=((h**4)/(E*I))*(f_const)
    else:
        f_iter[i]=((h**4)/(E*I))*(f_const-g*(70/0.2)) # bætum þyngd á öftustu 20cm
A = struct(n_opt)
A_sp = csc_matrix(A)
y = spsolve(A_sp, f_iter)
deflect_0 = y[-1]
plt.plot(x, y)
plt.xlabel('x')
plt.ylabel('y')
plt.show()
print('Deflection at the free end:', deflect_0)
```



```
>    Deflection at the free end: -0.20334130816830054
```

## 7.

If we also fix the free end of the diving board, we have a "clamped-clamped" beam, obeying identical boundary conditions at each end: $y(0) = y'(0) = y(L) = y'(L) = 0$. This version is used to model the sag in a structure, like a bridge. Begin with the slightly different evenly spaced grid $0 = x_0 < x_1 < ... < x_n < x_{n+1} = L$, where $h = x_i - x_{i-1}$ for $i = 1, ..., n$, and find the system of $n$ equations in n unknowns that determine $y_1, ..., y_n$. (It should be similar to the clamped-free version, except that the last two rows of the coefficient matrix A should be the first two rows reversed.) Solve for a sinusoidal load and answer the questions of Step 5 for the center $x = L/2$ of the beam. The exact solution for the clamped-clamped beam under a sinusoidal load is

$$y(x) = \frac{f}{24EI}x^2(L-x)^2 - \frac{pgL^2}{\pi^4 EI}\left(L^2 \sin\frac{\pi}{L}x + \pi x(x-L)\right)$$

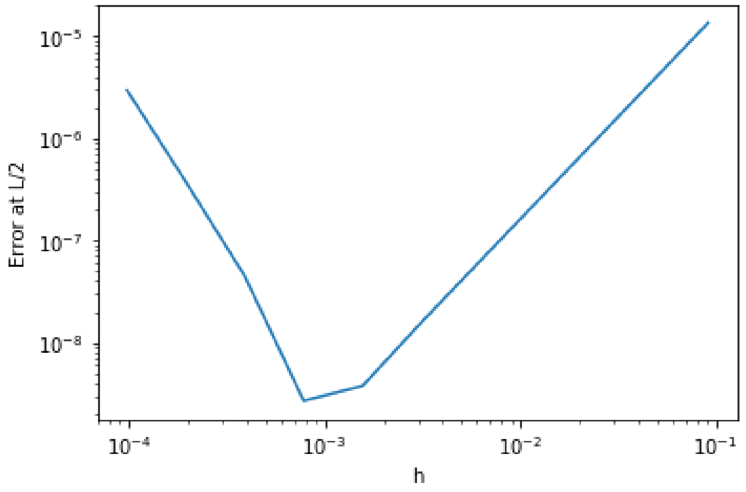
**Lausn:**

```
def y_x(x):
    y = (f_const/(24*E*I))*x**2*(L-x)**2-((p*g*L**2)/(np.pi**4*E*I))*(L**2*np.sin((np.pi/L)*x)
    return y

def rev_lst(lst):
    lst.reverse()
    return lst
def struct_2(n):
    l_1 = [16, -9, 8/3, -1/4]
    l_2 = [-4, 6, -4, 1]
    const = [1, -4, 6, -4, 1]
    A = np.zeros((n,n))
    A[0, 0:4] = l_1
    A[1, 0:4] = l_2
    b_1 = rev_lst(l_1)
    b_2 = rev_lst(l_2)
    A[n-2, -4:] = b_2
    A[n-1, -4:] = b_1
    col_index = 0
    for i in range(2, n-2):
        A[i, col_index:col_index+5] = const
        col_index +=1
    return A
mid_errors = []
cond_nums_2 = []
h_list_2 = []

for k in range(1,12):
    n = 10*(2**k)+1
    h = L/(n+1)# ath að bil orðin n+1 þegar tveir punktar fastir
    x = np.linspace(h, L-h, n)# báðir endapunktar fastir
    h_list_2.append(h)
    y_corr = y_x(x)
    y_corr_mid = y_corr[int((n)/2)] #floor-um fyrir index
    f_iter = ((h**4)/(E*I))*(f_const-g*p*np.sin((np.pi*x)/L))
    A = struct_2(n)
    A_sparse = csc_matrix(A)
    A_cond = sp_cond(A_sparse)
    cond_nums_2.append(A_cond)
    y = spsolve(A_sparse, f_iter)
    y_mid = y[int((n)/2)]
    mid_error = np.abs(y_mid-y_corr_mid)
    mid_errors.append(mid_error)
```

Teiknum svipað graf og í lið 5 og útbúum sömu töflu fyrir umbreytt fylki.

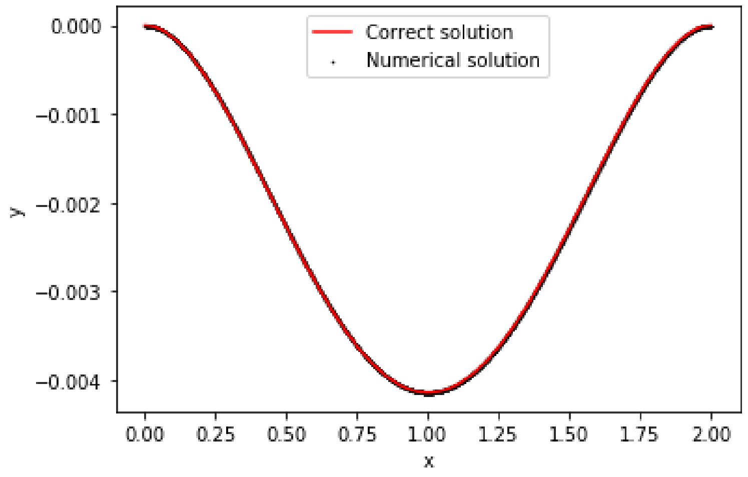


| n | Error at L/2 | cond(A) |
|---:|---:|---:|
| 21 | 1.34083e-05 | 17030.3 |
| 41 | 3.67525e-06 | 226219 |
| 81 | 9.63842e-07 | 3.28691e+06 |
| 161 | 2.46922e-07 | 5.00717e+07 |
| 321 | 6.24974e-08 | 7.81548e+08 |
| 641 | 1.57216e-08 | 1.23502e+10 |
| 1281 | 3.7557e-09 | 1.96374e+11 |
| 2561 | 2.6849e-09 | 3.13219e+12 |
| 5121 | 4.4451e-08 | 5.00364e+13 |
| 10241 | 3.72275e-07 | 8.00037e+14 |
| 20481 | 2.9447e-06 | 1.27854e+16 |

Sjáum að þetta graf hefur sömu eiginleika og grafið í lið 5, þ.e. frávikið er í hlutfalli við $h^2$ upp að ákeðnu gildi á $n$, eftir það er fylkið orðið svo óstöðugt að frávikið eykst hratt. Skoðum nú graf nálgunarlausnarinnar borið saman við graf réttu lausnarinnar fyrir besta gildi á $n$.

```
n_opt_2 = 2561 # from previous table
h = L/(n_opt_2+1)# ath að bil orðin n+1 þegar tveir punktar fastir
x = np.linspace(h, L-h, n_opt_2)# báðir endapunktar lausir
y_corr = y_x(x)
f_iter = ((h**4)/(E*I))*(f_const-g*p*np.sin((np.pi*x)/L))
A = struct_2(n_opt_2)
A_sparse = csc_matrix(A)
y = spsolve(A_sparse, f_iter)

plt.plot(x, y_corr, color = 'red', label='Correct solution')
plt.scatter(x, y, color = 'black', label='Numerical solution', s = 0.3)
plt.xlabel('x')
plt.ylabel('y')
plt.legend()
plt.show()
```



## 8.

Ideas for further exploration: If the width of the diving board is doubled, how does the displacement of the diver change? Does it change more or less than if the thickness is doubled? (Both beams have the same mass.) How does the maximum displacement change if the cross-section is circular or annular with the same area as the rectangle? (The area moment of inertia for a circular cross-section of radius $r$ is $I = \pi r^4/4$, and for an annular cross-section with inner radius $r_1$ and outer radius $r_2$ is $I = \pi \left(r_2^4 - r_1^4\right)/4$.) Find out the area moment of inertia for I-beams, for example. The Young's modulus for different materials are also tabulated and available. For example, the density of steel is about 7850 kg/m3 and its Young's modulus is about $2 \times 10^{11}$ Pascals.

The Euler–Bernoulli beam is a relatively simple, classical model. More recent models, such as the Timoshenko beam, take into account more exotic bending, where the beam cross-section may not be perpendicular to the beam's main axis.

**Lausn:**

Skoðum þversnið fyrir tvöfalda breidd, tvöfalda þykkt og hringlaga þversnið.

```python
w = 0.3
w_2 = 2*w # 1. double width
d_2 = 2*d # 2. double thick

def deflect(w,d):
    h = L/n_opt
    x = np.linspace(L/n_opt, L, n_opt)
    f_const = -480*w*d*g
    f_iter = np.zeros(n_opt)
    I = (w*d**3)/12
    for i in range(0, n_opt):
        if x[i]<1.8:
            f_iter[i]=((h**4)/(E*I))*(f_const)
        else:
            f_iter[i]=((h**4)/(E*I))*(f_const-g*(70/0.2))
    A = struct(n_opt)
    A_sp = csc_matrix(A)
    y = spsolve(A_sp, f_iter)
    deflect = y[-1]
    return deflect
# 3. circular cross section, same area as the original beam
r = np.sqrt((w*d)/np.pi)
def deflect_circle(w, d):
    h = L/n_opt
    x = np.linspace(L/n_opt, L, n_opt)
    f_const = -480*w*d*g
    r = np.sqrt((w*d)/np.pi)
    f_iter = np.zeros(n_opt)
    I = (np.pi*r**4)/4
    for i in range(0, n_opt):
        if x[i]<1.8:
            f_iter[i]=((h**4)/(E*I))*(f_const)
        else:
            f_iter[i]=((h**4)/(E*I))*(f_const-g*(70/0.2))
    A = struct(n_opt)
    A_sp = csc_matrix(A)
    y = spsolve(A_sp, f_iter)
    deflect = y[-1]
    return deflect
deflect_1 = deflect(w_2, d)
deflect_2 = deflect(w, d_2)
deflect_3 = deflect_circle(w, d)
print('Deflection when width doubled:', deflect_1,
     ', deflect chance from original beam:', np.abs(deflect_1-deflect_0))
print('Deflection when thickness doubled:', deflect_2,
      ', deflect chance from original beam:', np.abs(deflect_2-deflect_0))
print('Deflection when cross section is circular:', deflect_3,
      ', deflect chance from original beam:', np.abs(deflect_3-deflect_0))
```

```
> Deflection when width doubled: -0.10649853516332321 , deflect chance from original beam:
0.09684277300497733
> Deflection when thickness doubled: -0.026624633790830802 , deflect chance from original
beam: 0.17671667437746974
> Deflection when cross section is circular: -0.021293851997095645 , deflect chance from orig-
inal beam: 0.18204745617120488
```

Úttakið sýnir að mesta breyting á færslu í endapunkti er fyrir hringlaga þversnið, þá má sjá að breytingin er mun meiri þegar þykkt er tvöfölduð heldur en þegar breiddin er tvöfölduð.

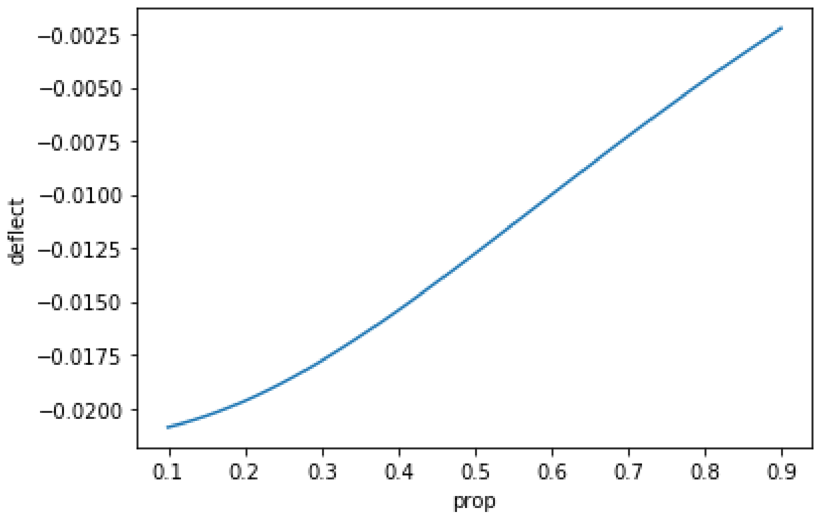
Skoðum nú tilfellið þegar þverskurður bitans er holur hringur. Skoðum nokkur tilfelli þegar þverskurðarflatarmál er það sama og í fyrri dæmum. Notum að flatarmál slíks bita er $A = \pi\left(r_2^2 - r_1^2\right)$, þar sem $r_2 > r_1$. Því má velja $r_1$ sem hlutfall af $r_2$ og reikna færsluna fyrir nokkra mismunandi bita sem allir hafa sama þverskurðarflatarmál. Eftirfarandi graf sýnir hverning færslan í endapunkti breytist með mismunandi gildum á $r_1$ og $r_2$.

```
def deflect_circle_ann(w, d, prop):
    h = L/n_opt
    x = np.linspace(L/n_opt, L, n_opt)
    f_const = -480*w*d*g
    area = w*d
    r_2 = np.sqrt(area/((1-prop**2)*np.pi))
    r_1 = prop*r_2
    f_iter = np.zeros(n_opt)
    I = (np.pi*(r_2**4-r_1**4))/4
    for i in range(0, n_opt):
        if x[i]<1.8:
            f_iter[i]=((h**4)/(E*I))*(f_const)
        else:
            f_iter[i]=((h**4)/(E*I))*(f_const-g*(70/0.2))
    A = struct(n_opt)
    A_sp = csc_matrix(A)
    y = spsolve(A_sp, f_iter)
    deflect = y[-1]
    return deflect

#skoðum nokkrar keyrslur fyrir mismunandi hlutfal, plottum svo deflect á móti
# prop(hlutfall r_1 af heildarradíus)
y_deflects = []
props = np.linspace(0.1, 0.9, 50)

for prop in props:
    deflect = deflect_circle_ann(w, d, prop)
    y_deflects.append(deflect)

plt.plot(props, y_deflects)
plt.xlabel('prop')
plt.ylabel('deflect')
plt.show()
```



Af grafinu sést að þegar innri radíus nálgast ytri radíus þá lækkar neikvæð færsla. Athuga skal að ef lítill munur er á innri og ytri radíus þá mun efnið í hringlaga kraganum líklega gefa undan, því er óraunhæft að notast við slík hlutföll. Næst er færsla fyrir I bita skoðuð. Hönnunarforsenda I bitans er sú að þverskurðaflatarmál hans er það sama og fyrir rétthyrnda þversniðið með 30 cm breidd og 3 cm þykkt. Notum okkur að hverfitregða I bita er $\frac{ah^3}{12}+\frac{b}{12}\left(H^3-h^3\right)$, veljum $(H-h)=2a$, $b=6a$ og $h=1.5b$. Reiknum síðan a út frá jöfnunni fyrir hverfitregðu þverskurðaflatarmál I bita, þ.e. $A=(H-h)b+ha$ að $a=\sqrt{\frac{A}{21}}$.

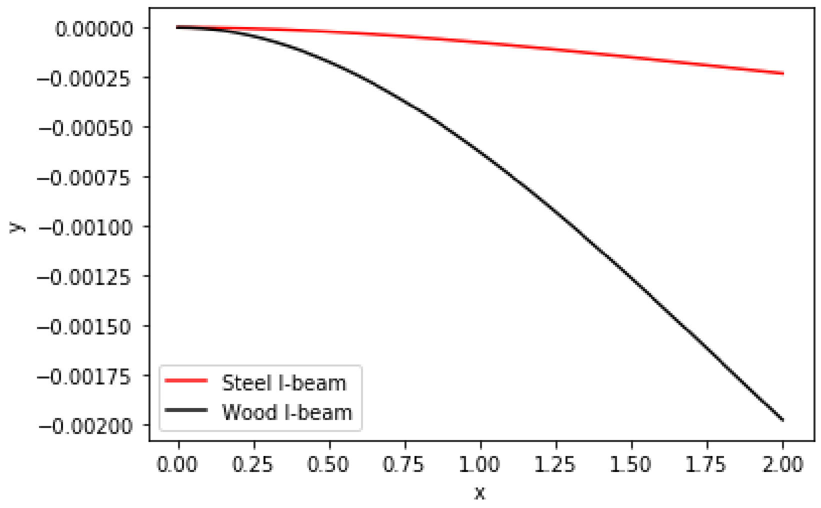
```
A_1 = w*d
a_1 = np.sqrt(A_1/21)
b = 6*a_1
h_1 = 1.5*b
H = 2*a_1+h_1
I = (a_1*h_1**3)/12+(b/12)*(H**3-h_1**3)
```

Reiknum færslu og teiknum lausn fyrir I bita eins og hann er skilgreindur að ofan þegar þyngd er lögð á öftustu 20 cm hans, bæði fyrir stál- og viðarþversnið.

```
#laus biti, notum n = 1280
from scipy.sparse.linalg import spsolve
from scipy.sparse import csc_matrix
n = 1280
A = struct(n)
A_sp = csc_matrix(A)
h = L/n
x = np.linspace(L/n, L, n)
f_iter = np.zeros(n)
f_const = -480*w*d
E = 1.3e10
for i in range(0, n):
    if x[i]<1.8:
        f_iter[i]=((h**4)/(E*I))*(f_const)
    else:
        f_iter[i]=((h**4)/(E*I))*(f_const-g*(70/0.2))

y = spsolve(A_sp, f_iter)
#Stál
E_steel = 2e11
f_const_steel = -7850*w*d*g
f_iter_steel = np.zeros(n)
for i in range(0, n):
    if x[i]<1.8:
        f_iter_steel[i]=((h**4)/(E_steel*I))*(f_const_steel)
    else:
        f_iter_steel[i]=((h**4)/(E_steel*I))*(f_const_steel-g*(70/0.2))
y_steel = spsolve(A_sp, f_iter_steel)

plt.plot(x, y_steel,label='Steel I-beam',color='red')
plt.plot(x, y, label='Wood I-beam', color ='black')
plt.xlabel('x')
plt.ylabel('y')
plt.legend()
plt.show()
```



Af myndinni sést augljóslega að I bitinn er töluvert sterkari en öll hin þversniðin sem við höfum skoðað. Fyrir sama efniskostnað eykst styrkurinn margfalt, við getum því sagt að I bitinn sé ákjósanlegt þversnið fyrir bita. Sjáum einnig að stálbitinn er mun sterkari en viðarbitinn, vegna þess að krafturinn á bitann er háður Young's stuðli efnisins sem um ræðir.

$2^{43}$