I. Problem:

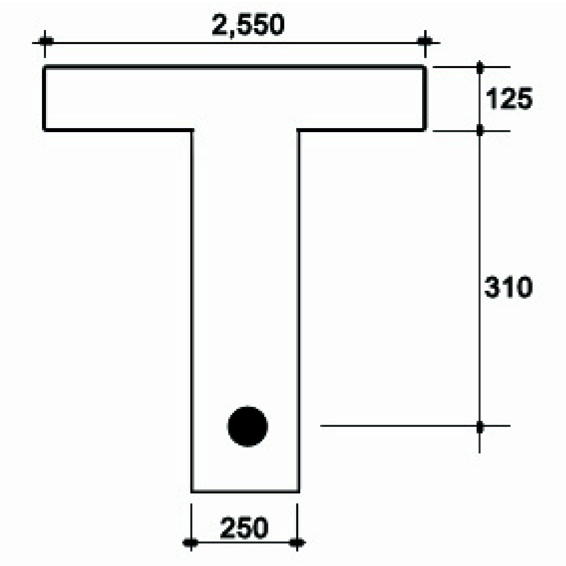
Draw the Moment-Capacity and Tension Steel (Mn - As) relationship curve of a reinforced concrete t-beam of different cases with parameters as follows:

General Cases

Cases	fc'	fy
Case 1	20	300
Case 2	30	300
Case 3	20	400
Case 4	30	400

Beam Properties

Figure 1: Given section



Property	Value	Unit
f'c	[20, 40, 20, 40]	MPa
fy	[300, 300, 400, 400]	MPa
Effective depth (d)	435	mm
Web width (b_w)	250	mm
Thickness of flange (t_f)	125	mm
ϵ_{cu}	0.003	
E_s	200,000	MPa
f'c reference	28	MPa
Student number (last 3-digits)	338	
Factor based of on reversed student number α	0.833	
$bf = bw + \alpha \cdot tf$	2550	mm

$\beta1$ is solved for each value of $f'c$.

```
fcPrime = [20, 40, 20, 40]
fy = [300, 300, 400, 400]
β1 = []
# Calculate for corresponding β1
for fcx in fcPrime:
    if fcx <= fc_base:
        β1.append(0.85)
    else:
        β1.append(round(0.85 - (0.05 / 7)*(fcx - fc_base), 3))
```

$\beta1$ now is `[0.85, 0.764, 0.85, 0.764]`

II. Solutions / Methodology

A. Assumptions

- In this problem, tensile strength of concrete is neglected, so that at $As = 0$, $Mn = 0$.
- Stress block used is the Whitney stress block distribution for simplicity of calculation.

B. Solution

The solution to the problem is to solve each cases in one go using programming language **python** by looping through a number of four (4) cases.

1. Create list of arrays to hold values for moments and steel areas for each case:

```
# With initial values of zeros (0's) for Mn and As for each case
M = ([0], [0], [0], [0])
As = ([0], [0], [0], [0])
```

2. Now for looping on 4 cases:

2.1. Calculation of balanced steel area, needed for the curve and analysis limit.

```
    for i in range(4):
        c_bal = 600 * d / (600 + fy[i])          # Location of neutral axis
        a_bal = β1[i] * c_bal                    # Equivalent height of rectangular
                                                 #  compression block
        z_bal = a_bal - tf if a_bal > tf else 0  # Web component of compression zone
                                                 #  if a > tf
        As_bal = (0.85 * fcPrime[i] * bf * tf + 0.85 * fcPrime[i] * bw * z_bal)
                    / fy[i]
        As_limit = 2 * As_bal                    # Limit to be analyzed
        As_trial = 100                           # Start calculating here for As
```

2.2. Create a loop that tries for each value of As, then calculates the corresponding moment capacity, Mn until As_limit is reached.

```
    while (As_trial <= As_limit):
        a = 10                                  # Trial for height of stress block
        c = a / β1[i]                           # Neutral axis height (kd)
        As_calc = 0                             # Variable to hold calculated As
                                                # To be compared with As_trial
```

2.3. Find the height of stress block by trial and error

```
        while (As_calc < As_trial):
            c = a / β1[i]                       # Neutral axis height (kd)
            fs = 600 * (d - c) / c              # Calculated fs
            if (fs >= fy[i]):                   # Use fy if steel yields
                fs = fy[i]
            Ac = area_of_tbeam(bf, tf, bw, a)   # Calling a function to calculate
                                                #  area of a given t-beam
            As_calc = 0.85 * fcPrime[i] * Ac / fs # Calculate As by equilibrium
            a += 0.02                           # Increment a with 0.02mm
```

2.4. Nominal moment is now then calculated by taking a moment at centroid of stress block

```
        Mn = As_calc * fs * (d - centroid_of_tbeam(bf, tf, bw, a))

        # Add the calculated moment and steel area to the array
        M[i].append(Mn/1000**2)
        As[i].append(As_calc)
```

2.5. To finalize a single loop, trial for steel area is incremented by $100mm^2$ each time

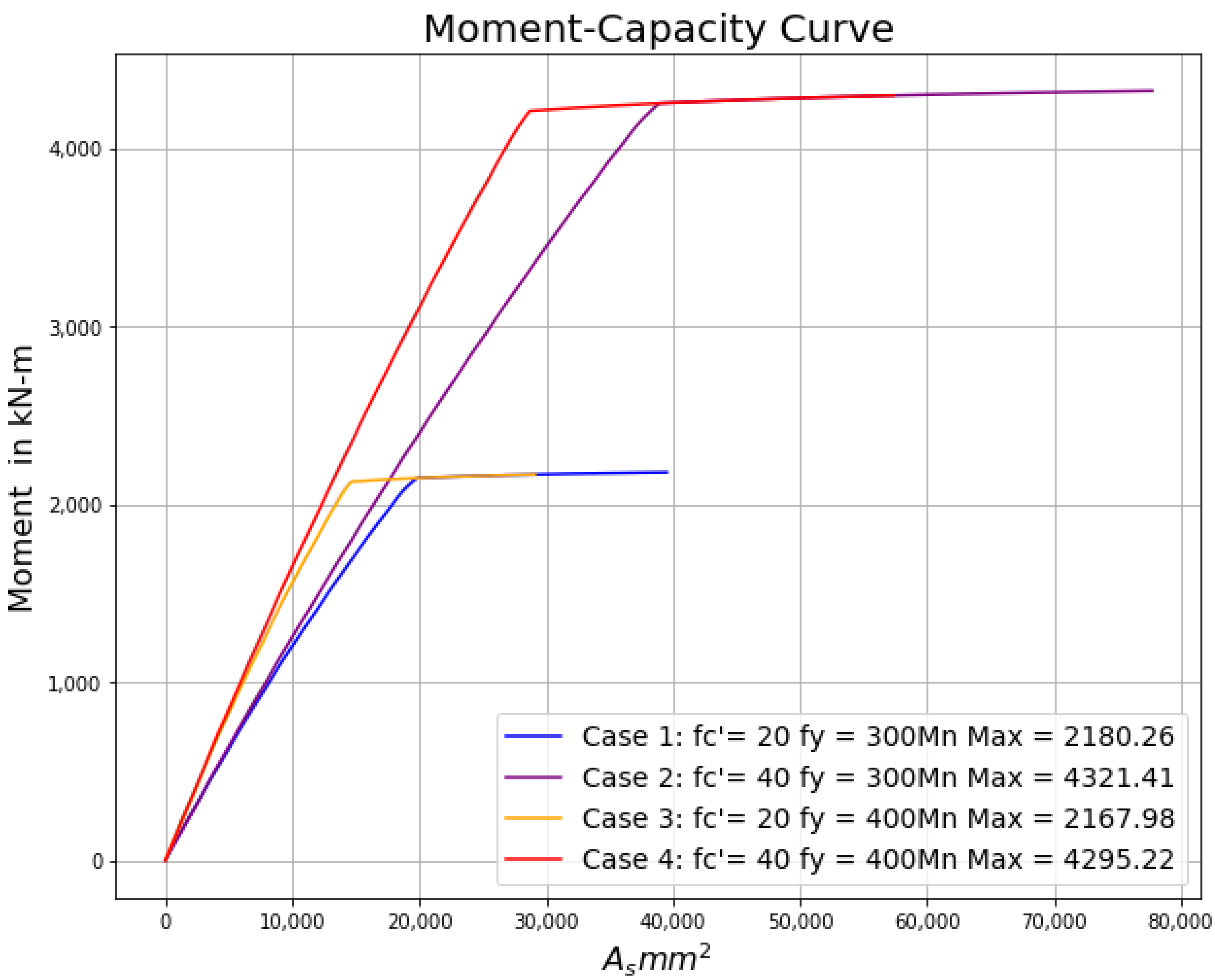
```
    As_trial += 100
```

This covers the whole process of the calculation. To add some more observations, some data have been collected and inserted to the actual code, (not shown in the main process above) to understand the result more deeply like; maximum nominal moment, strain at the compression fiber and actual steel stress is also collected to find if steel yields in each loop.

III. Results / Charts

The resulting graph for all cases is shown below:

Figure 2: Moment Capacity vs. Steel Reinforcement relationship curve



Also, a snippet of the resulting table of calculated As and Mn is shown below showing the transition between the steel yielding and not yielding.

Figure 3: Snippet of case 4 results

```
As =  28200 M =  4162.61 c =  231.75 a =  177.08 fs =  400 fs (actual) =  526.1945103353721 ϵc =  0.0022805255023188578
As =  28300 M =  4172.86 c =  237.93 a =  181.8 fs =  400 fs (actual) =  496.95235999548265 ϵc =  0.0024147183846976966
As =  28400 M =  4182.88 c =  244.08 a =  186.5 fs =  400 fs (actual) =  469.30501930489487 ϵc =  0.002556972439326059
As =  28500 M =  4192.72 c =  250.24 a =  191.2 fs =  400 fs (actual) =  443.0170519027547 ϵc =  0.002708609348258101
As =  28600 M =  4202.4 c =  256.41 a =  195.92 fs =  400 fs (actual) =  417.88667687581926 ϵc =  0.00287159190853217
As =  28700 M =  4209.63 c =  261.1 a =  199.5 fs =  399.62 fs (actual) =  399.6190094243616 ϵc =  0.003
As =  28800 M =  4210.15 c =  261.44 a =  199.76 fs =  398.32 fs (actual) =  398.31781315696185 ϵc =  0.0029999999999999996
As =  28900 M =  4210.67 c =  261.78 a =  200.02 fs =  397.02 fs (actual) =  397.0199999998574 ϵc =  0.0029999999999999996
As =  29000 M =  4211.18 c =  262.12 a =  200.28 fs =  395.73 fs (actual) =  395.7255567760479 ϵc =  0.003
As =  29100 M =  4211.7 c =  262.46 a =  200.54 fs =  394.43 fs (actual) =  394.4344703768765 ϵc =  0.0030000000000000005
As =  29200 M =  4212.22 c =  262.8 a =  200.8 fs =  393.15 fs (actual) =  393.1467277615858 ϵc =  0.0029999999999999996
As =  29300 M =  4212.74 c =  263.14 a =  201.06 fs =  391.86 fs (actual) =  391.86231595687957 ϵc =  0.0030000000000000005
As =  29400 M =  4213.25 c =  263.48 a =  201.32 fs =  390.58 fs (actual) =  390.5812220564878 ϵc =  0.003
As =  29500 M =  4213.73 c =  263.8 a =  201.56 fs =  389.4 fs (actual) =  389.4016076211714 ϵc =  0.0029999999999999996
As =  29600 M =  4214.25 c =  264.14 a =  201.82 fs =  388.13 fs (actual) =  388.12685827537564 ϵc =  0.0029999999999999996
As =  29700 M =  4214.76 c =  264.48 a =  202.08 fs =  386.86 fs (actual) =  386.855389488126 ϵc =  0.003
As =  29800 M =  4215.24 c =  264.79 a =  202.32 fs =  385.68 fs (actual) =  385.68462679174814 ϵc =  0.003
As =  29900 M =  4215.75 c =  265.13 a =  202.58 fs =  384.42 fs (actual) =  384.4194312794753 ϵc =  0.003
As =  30000 M =  4216.22 c =  265.45 a =  202.82 fs =  383.25 fs (actual) =  383.2544378696767 ϵc =  0.003
As =  30100 M =  4216.74 c =  265.79 a =  203.08 fs =  382.0 fs (actual) =  381.99546931926693 ϵc =  0.0029999999999999996
As =  30200 M =  4217.21 c =  266.1 a =  203.32 fs =  380.84 fs (actual) =  380.8362026560269 ϵc =  0.0029999999999999996
As =  30300 M =  4217.68 c =  266.41 a =  203.56 fs =  379.68 fs (actual) =  379.67966984361885 ϵc =  0.003
As =  30400 M =  4218.19 c =  266.75 a =  203.82 fs =  378.43 fs (actual) =  378.42983316962756 ϵc =  0.003
As =  30500 M =  4218.66 c =  267.07 a =  204.06 fs =  377.28 fs (actual) =  377.27896490869443 ϵc =  0.003
As =  30600 M =  4219.14 c =  267.38 a =  204.3 fs =  376.13 fs (actual) =  376.1308008614153 ϵc =  0.002999999999999999
As =  30700 M =  4219.61 c =  267.7 a =  204.54 fs =  374.99 fs (actual) =  374.9853315077735 ϵc =  0.0029999999999999996
As =  30800 M =  4220.08 c =  268.01 a =  204.78 fs =  373.84 fs (actual) =  373.842547372386 ϵc =  0.0029999999999999996
As =  30900 M =  4220.55 c =  268.32 a =  205.02 fs =  372.7 fs (actual) =  372.70243902424227 ϵc =  0.003
As =  31000 M =  4221.01 c =  268.64 a =  205.26 fs =  371.56 fs (actual) =  371.5649970764451 ϵc =  0.0030000000000000005
As =  31100 M =  4221.48 c =  268.95 a =  205.5 fs =  370.43 fs (actual) =  370.43021218595237 ϵc =  0.003
```

Note that the snippet above is part of the **Case 4** of this problem.

IV. Comments/Observations

Following are comments and findings in this problem set.

- Case 1 and 3 shows nearly same amount of Moment Capacity despite having different values of fy. It can be noted though that the one with greater fy (Case 3), reaches its max moment with less steel area (As). Similar is true with Case 2 and 4.
- For each case, it can be noted that at a specific point, each curve changed its slope abruptly, from steep to almost flat. This point is the transition between under and over reinforced or at balanced steel reinforcement. Numerical data for this point is also presented in the ***figure 2*** above. Beyond this point, is where the concrete yields at strain of 0.003 and steel no longer reaches its yield strength.
- It was also observed that when steel no longer yields, moment capacity doesn't increase much with respect to steel even if we increase its area significantly. This is observed in the part of each curve with small slope.

V. Appendix

References

- Gillesania, DI T., *Simplified Reinforced Concrete Design*, Diego Innocencio Tapang Guillesania, 2013
- American Concrete Institute, *Building Code Requirements for Structural Concrete (ACI 318-95) and Commentary (ACI 318R-95)*, 1995
- Nilson, A. H., Darwin, D., Dolan, C. W., *Design of Concrete Structures 14th ed.*, McGraw-Hill, 2010, Retrieved from http://www.engineeringbookspdf.com

Source Code

The programming language used in this problem set is **Python3** with the help of **Jupyter Notebook** for presenting the result. The full source code used is shown below. This source code is also available at github (https://github.com/alexiusacademia/masteral-advanced-concrete-design/blob/master/Notebooks/Problem%20Set%202.ipynb)

```python
import matplotlib.pyplot as plt
import matplotlib.ticker as tkr
import math

def centroid_of_tbeam(bf, tf, bw, y):
    # Calculate centroid of t-beam from top of flange
    # y - height of beam
    kd_prime = 0
    if (y <= tf):
        kd_prime = y / 2
    else:
        a_total = bf * tf + bw * (y - tf)
        ay = bf * tf * tf / 2
        ay += bw * (y - tf) * ((y - tf) / 2 + tf)
        kd_prime = ay / a_total
    return kd_prime

def area_of_tbeam(bf, tf, bw, y):
    # Calculate the area of t-beam
    area = 0
    if (y <= tf):
        area = bf * y
    else:
        area = bf * tf + bw * (y - tf)
    return area

student_number = [3, 3, 8]
d = 435
bw = 250
tf = 125
εcu = 0.003
Es = 200000
fc_base = 28         # Basis for calculating β1
student_number_reversed = ['.']

# Reverse the student number then append decimal at the start
for i in range(len(student_number)):
    student_number_reversed.append(str(student_number[len(student_number) - 1 - i]))

# Converted student number
student_number_reversed = float(''.join(student_number_reversed))

# Factor for calculating flange width
```

```python
α = 10 + 10 * student_number_reversed

# Calculate for b then round to the nearest 25mm
bf = bw + α * tf
bf = int(round(bf / 100 * 4) / 4 * 100)
print('bf = ', bf)
# Given arrays
fcPrime = [20, 40, 20, 40]
fy = [300, 300, 400, 400]
β1 = []

# Calculate for corresponding β1
for fcx in fcPrime:
    if fcx <= fc_base:
        β1.append(0.85)
    else:
        β1.append(round(0.85 - (0.05 / 7)*(fcx - fc_base), 3))

# Results array
M = ([0], [0], [0], [0])
As = ([0], [0], [0], [0])
MnMax = []

# -------------------------------------
# Start of problem main calculation
# -------------------------------------
for i in range(4):     # 4 cases
    print('= = = = = = = = = = = = =')
    print('Case # ', i+1)
    print('= = = = = = = = = = = = =')

    # Calculate balanced value for 'c'
    c_bal = 600 * d / (600 + fy[i])

    # Balanced equivalent compression block height
    a_bal = β1[i] * c_bal

    # Web component of the compression, z
    z_bal = a_bal - tf if a_bal > tf else 0

    # Balanced equation
    # Asb.fy = 0.85 f'c.bf.tf + 0.85f'c.bw.z
    As_bal = (0.85 * fcPrime[i] * bf * tf + 0.85 * fcPrime[i] * bw * z_bal) / fy[i]

    print('cb = ', c_bal, 'ab = ', a_bal, 'zb = ', z_bal, 'Asb = ', round(As_bal, 3))

    As_limit = 2 * As_bal

    As_trial = 100
    Mmax = 0.0
    while (As_trial <= As_limit):
```

```
        a = 10
        c = a / β1[i]
        As_calc = 0
        fs = 0.0
        fs_actual = 0.0
        steel_yields = False
        while (As_calc < As_trial):
            c = a / β1[i]
            fs = 600 * (d - c) / c
            fs_actual = fs
            if (fs >= fy[i]):
                fs = fy[i]
                steel_yields = True
            Ac = area_of_tbeam(bf, tf, bw, a)

            As_calc = 0.85 * fcPrime[i] * Ac / fs
            # Try for a
            a += 0.02

        # Calculate for the strain in concrete
        εc = (fs/Es) / (d-c) * c

        # Calculate moment
        Mn = As_calc * fs * (d - centroid_of_tbeam(bf, tf, bw, a))
        Mmax = Mn
        M[i].append(Mn/1000**2)
        As[i].append(As_calc)

        print('As = ', round(As_trial, 2),'M = ', round(Mn / 1000**2, 2), 'c = ', \
              round(c, 2), 'a = ', round(a, 2), 'fs = ', round(fs, 2), 'fs (actual) = ', fs_actual,
             'εc = ', εc)
        # Increment steel area each loop
        As_trial += 100

    MnMax.append(Mmax)

# Plot the curves
plt.figure(figsize=(10,8))
plt.title("Moment-Capacity Curve", fontsize=20)
plt.xlabel(r'$A_s mm^2$', fontsize=16)
plt.ylabel('Moment  in kN-m', fontsize=16)
plt.grid()

# Plot the converted values
case1, = plt.plot(As[0], M[0], label='Case 1: fc\'= '+ str(fcPrime[0]) + ' fy = ' +\
                  str(fy[0]) + 'Mn Max = ' + str(round(MnMax[0]/1000**2,2)), color='blue')
case2, = plt.plot(As[1], M[1], label='Case 2: fc\'= '+ str(fcPrime[1]) + ' fy = ' +\
```

```
                  str(fy[1]) + 'Mn Max = ' + str(round(MnMax[1]/1000**2,2)), color='purple')
case3, = plt.plot(As[2], M[2], label='Case 3: fc\'= '+ str(fcPrime[2]) + ' fy = ' +\
                  str(fy[2]) + 'Mn Max = ' + str(round(MnMax[2]/1000**2,2)), color='orange')
case4, = plt.plot(As[3], M[3], label='Case 4: fc\'= '+ str(fcPrime[3]) + ' fy = ' +\
                  str(fy[3]) + 'Mn Max = ' + str(round(MnMax[3]/1000**2,2)), color='red')

def func(x, pos):  # formatter function takes tick label and tick position
    s = '%d' % x
    groups = []
    while s and s[-1].isdigit():
        groups.append(s[-3:])
        s = s[:-3]
    return s + ','.join(reversed(groups))

y_formatter = tkr.FuncFormatter(func)
x_formatter = tkr.FuncFormatter(func)

ax = plt.subplot(111)
ax.yaxis.set_major_formatter(y_formatter)
ax.xaxis.set_major_formatter(x_formatter)

plt.legend(handles=[case1, case2, case3, case4], loc='best', fontsize=14)
plt.show()
```