I. Problem:

Moment-Curvature relationship curve of a reinforced concrete beam of different cases with parameters as follows:

General Cases

Cases	As	As'
Case 1	A_{sb}	0
Case 2	$0.5A_{sb}$	0
Case 3	A_{sb}	$0.5A_{sb}$

Beam Properties

Property	Value	Unit
f'c	21	MPa
fy	275	MPa
fr = 0.7 $\sqrt{f'c}$	3.208	MPa
Es	200,000	MPa
Ec = (4,700$\sqrt{f'c}$)	21538.10	MPa
β_1	0.85	
η = Es / Ec	9.28	
b (beam width)	300	mm
h (beam height)	450	mm
d (effective depth)	400	mm
d' (compression steel location)	50	mm

II. Solutions / Methodology

As a general solution to the problem, analysis as doubly reinforced beam is applied to address all the cases (singly or doubly reinforced). The following are the steps used:

1. Compute for the balanced steel at tension.

   $Asb = 4,539.92mm^2$

2. Steel area is assigned to both tension and compression side as indicated in the general cases.
3. For the 3 stages of the behavior of the beam:

   ### Stage 1 : Cracking point of concrete in tension

   1. Neutral axis location (kd) from the compression fiber of concrete is calculated by transforming area of steel to area of concrete using the modular ratio η:

      a. Uncrack section (Case 1 : Doubly Reinforced)

Particulars	Calculated Values
$As(transformed) = (\eta - 1)As$	37,617.24 mm^2
$As'(transformed) = (\eta - 1)As'$	18,808.62 mm^2
By taking moment of areas of concrete and steel to topmost fiber:	
kd	242.19 mm

   2. Calculation of $M\ cr$ and $\phi\ cr$

Particulars	Calculated Values
$I\ cr$	$3.949x10^9 mm^4$
$Mcr = fr\dfrac{Icr}{h - kd}$	$60.97x10^6 kN - m$
$\phi cr = \frac{fr}{Ec*(h-kd)}$	$7.167e - 07rad/mm$

   3. Calculate the curvature ϕc right after cracking

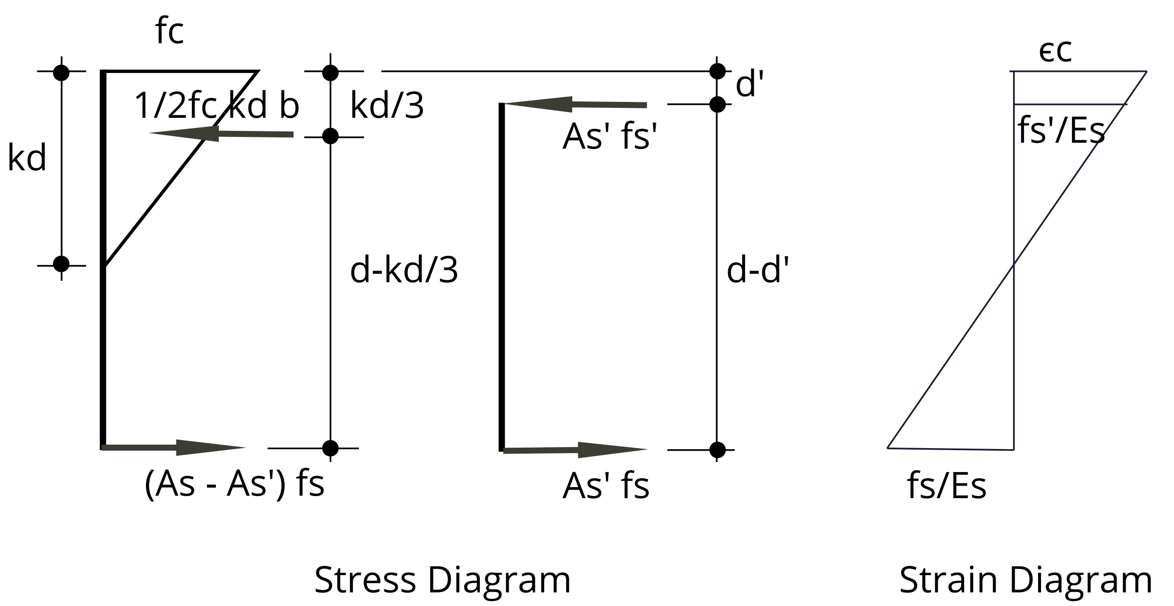
      The neutral axis will shift after the crack, so taking moment of area for transformed steel in tension (ηAs) and compression ($(\eta - 1)As$) and concrete at the compressive area into the neutral axis:

$$b \cdot kd \cdot \frac{kd}{2} + (\eta - 1)As' \cdot (kd - d') = \eta As \cdot (d - kd) \quad (1)$$

Particulars	Calculated Values
kd	196.76 mm
$Ic\ (aftercrack)$	$2.908x10^9 mm^4$
$\phi c = \dfrac{Mcr}{Ec * Ic}$	$9.733e-07rad/mm$

Stage 2 : Concrete compression yield at ($fc = 0.5f'c$)

At this stage, compression block is still assumed linear and so can be represented by a triangular shape as shown.



By equilibrium, $C = T$

$$\frac{1}{2} fc \cdot kd \cdot b + (As - As')fs = As' \cdot fs \tag{2}$$

Solving kd using quadratic formula, $kd = 194.03mm$

By deriving from the **Strain Diagram** above, we get:

$$fs = Es \cdot \epsilon c \cdot \frac{d - kd}{kd} \tag{3}$$

$$fs' = Es \cdot \epsilon c \cdot \frac{kd - d'}{kd} \tag{4}$$

After this, fs and fs' are compared to fy, if any of them is greater than fy, steel yields and, so use $fs = fy$ or $fs' = fy$ correspondingly is solving for Moments.

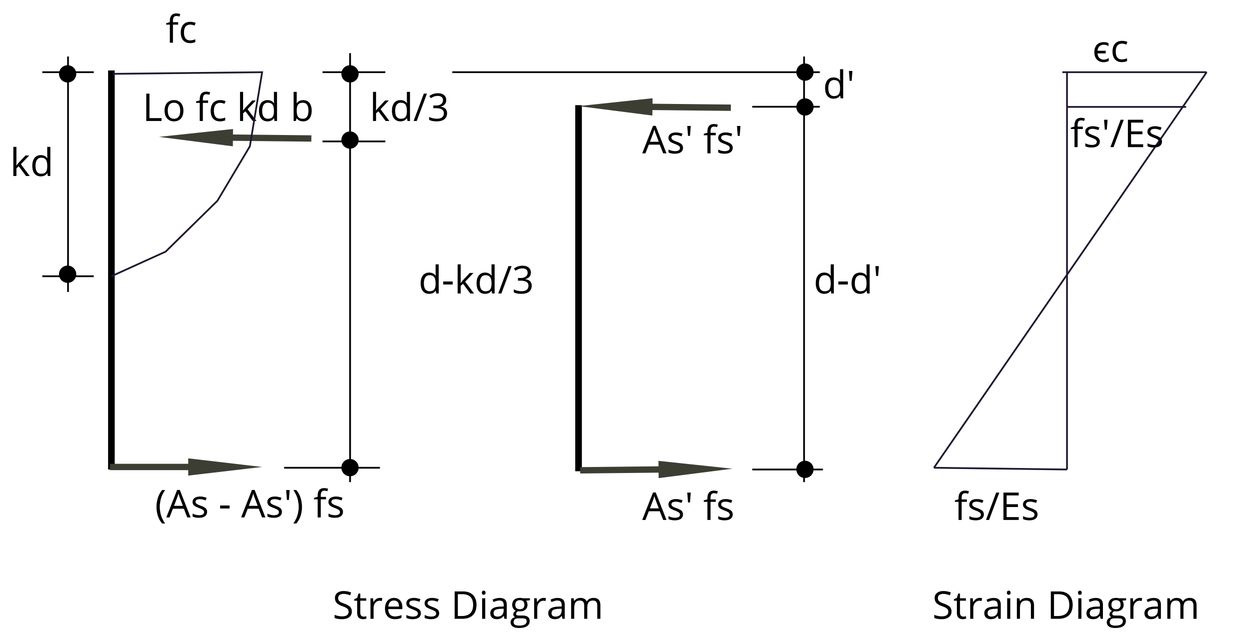
Moment can now be solved by taking moment to tension steel:

$$Mc = \frac{1}{2} \cdot fc \cdot kd \cdot b \cdot (d - \frac{kd}{3}) + As' \cdot fs'(d - d') \tag{5}$$

Particulars	Calculated Values
$fc = 0.5f'c$	$10.50MPa$
fs	$103.50MPa$
fs'	$72.37MPa$
Mc	$159.98x10^6 kN - m$
ϕc	$2.512e - 07rad/mm$

Stage 3 : Inelastic Stage

At this stage, compression block is no longer triangular, concrete modulus of elasticity is also no longer constant.



Solving for the moment and curvature is divided into two (2) more stages:

1. $0 < \epsilon c < \epsilon o$
2. $\epsilon o < \epsilon c < 0.003$

   wherein we iterate in the value of ϵc

Calculation inside these stages are almost the same except for the calculations of factors such as $k2$, Lo and fc:

1. First, fs and fs' are assumed to yield, so to solve for kd using equilibrium:

$$C = T \tag{6}$$

$$Lo \cdot fc \cdot kd \cdot b = (As - As') \cdot fy \tag{7}$$

$$kd = (As - As') \cdot \frac{fy}{Lo \cdot fc \cdot b} \tag{8}$$

2. Now, steel stresses fs and fs' are computed using the calculated kd with equations (3) and (4) respectively then compares them to fy:

   a. If $fs > fy$:

   This means steel yields at tension. We then test:

   a.1. if $fs' > fy$: Since the assumption that steel in compression and tension yields, we accept the calculated value of kd then proceeds with $fs = fy$ and $fs' = fy$.

   a.2. if $fs' < fy$:

   Instead of both As and As' multiplying to fy in equation (7), we multiply As' by fs'

   in equation (4), thus

$$Lo \cdot fc \cdot kd \cdot b = Asfy - As'fs' \tag{9}$$

$$Lo \cdot fc \cdot kd \cdot b = Asfy - As' \cdot Es \cdot \epsilon c \cdot \frac{kd - d'}{kd} \tag{10}$$

   kd can now be recalculated using quadratic formula, then fs' based on the new calculated kd

   b. if $fs < fy$:

   Now, assume the steel at compression yields, $fs' = fy$, we now then get re-write equation (9):

$$Lo \cdot fc \cdot kd \cdot b = As \cdot fs - As'fy \tag{11}$$

$$Lo \cdot fc \cdot kd \cdot b = As \cdot Es \cdot \epsilon c \cdot \frac{d - kd}{kd} - As'fy \tag{12}$$

   We now recalculate kd using the quadratic equation above.

   After calculating kd, we might want to check fs' again if compression steel yields to check if assumption in equation (13) is correct. If not, we just replace fy in equation (12) with equation (4) then recalculate kd.

   fs and fs' can now be recalculated based on the new kd.

3. After getting the stresses in steel, we can now solve for moment by taking moment at the tension steel

$$Mc = Lo \cdot fc \cdot kd \cdot b \cdot (d - k2 \cdot kd) + As' \cdot fs' \cdot (d - d') \tag{13}$$

4. For the curvature, ϕc

$$\$\phi c = \epsilon c / kd \tag{14}$$

III. Results / Charts

Following is the result of the run of the script for the problem in all three (3) cases:

Case 1

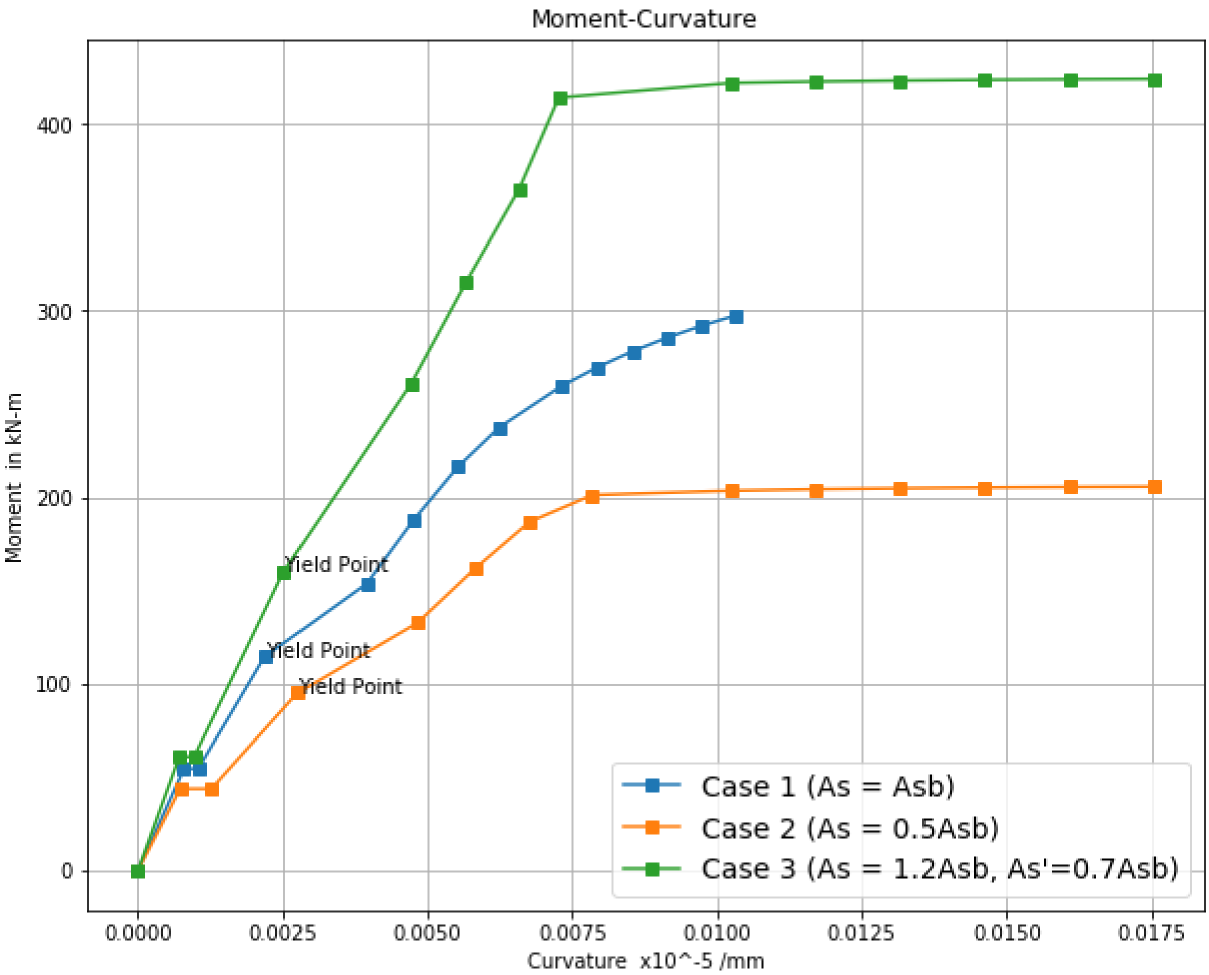
Moment ($kN \cdot m$)	ϕc **(rad/mm)**	ϵc	**Tension Steel Yield**	**Compression Steel Yield**
54.57	7.9703e-07			
54.57	1.0428e-06			
114.39	2.1859e-06	0.00048		
For 0 < ϵc < ϵo				
153.69	3.95e-06	0.00103	False	False
187.86	4.76e-06	0.00123	False	False
216.42	5.52e-06	0.00143	False	False
237.47	6.23e-06	0.00163	False	False
For ϵo < ϵc < ϵcu				
259.5	7.3e-06	0.00196	False	True
269.74	7.93e-06	0.00216	False	True
278.35	8.54e-06	0.00236	False	True
285.68	9.14e-06	0.00256	False	True
292.0	9.73e-06	0.00276	False	True
297.51	1.032e-05	0.00296	False	True

Case 2

Moment ($kN \cdot m$)	ϕc **(rad/mm)**	ϵc	**Tension Steel Yield**	**Compression Steel Yield**
43.86	7.315e-07			
43.86	1.270e-06			
95.07	2.754e-06	0.00048		
For 0 < ϵc < ϵo				
132.65	4.82e-06	0.00103	False	False
161.97	5.83e-06	0.00123	False	False
186.87	6.76e-06	0.00143	False	False
201.24	7.85e-06	0.00163	True	False
For ϵo < ϵc < ϵcu				
203.58	1.024e-05	0.00196	True	True
204.35	1.17e-05	0.00216	True	True
204.89	1.316e-05	0.00236	True	True
205.27	1.462e-05	0.00256	True	True
205.55	1.608e-05	0.00276	True	True
205.76	1.754e-05	0.00296	True	True

Case 3

Moment ($kN \cdot m$)	ϕc **(rad/mm)**	ϵc	**Tension Steel Yield**	**Compression Steel Yield**
60.97	7.167e-07			
60.97	9.733e-07			
159.98	2.512e-06	0.00048		
For 0 < ϵc < ϵo				
260.98	4.72e-06	0.00103	False	False
315.58	5.67e-06	0.00123	False	False
365.39	6.58e-06	0.00143	False	False
414.18	7.27e-06	0.00163	True	False
For ϵo < ϵc < ϵcu				
422.07	1.024e-05	0.00196	True	True
422.84	1.17e-05	0.00216	True	True
423.37	1.316e-05	0.00236	True	True
423.75	1.462e-05	0.00256	True	True
424.03	1.608e-05	0.00276	True	True
424.25	1.754e-05	0.00296	True	True

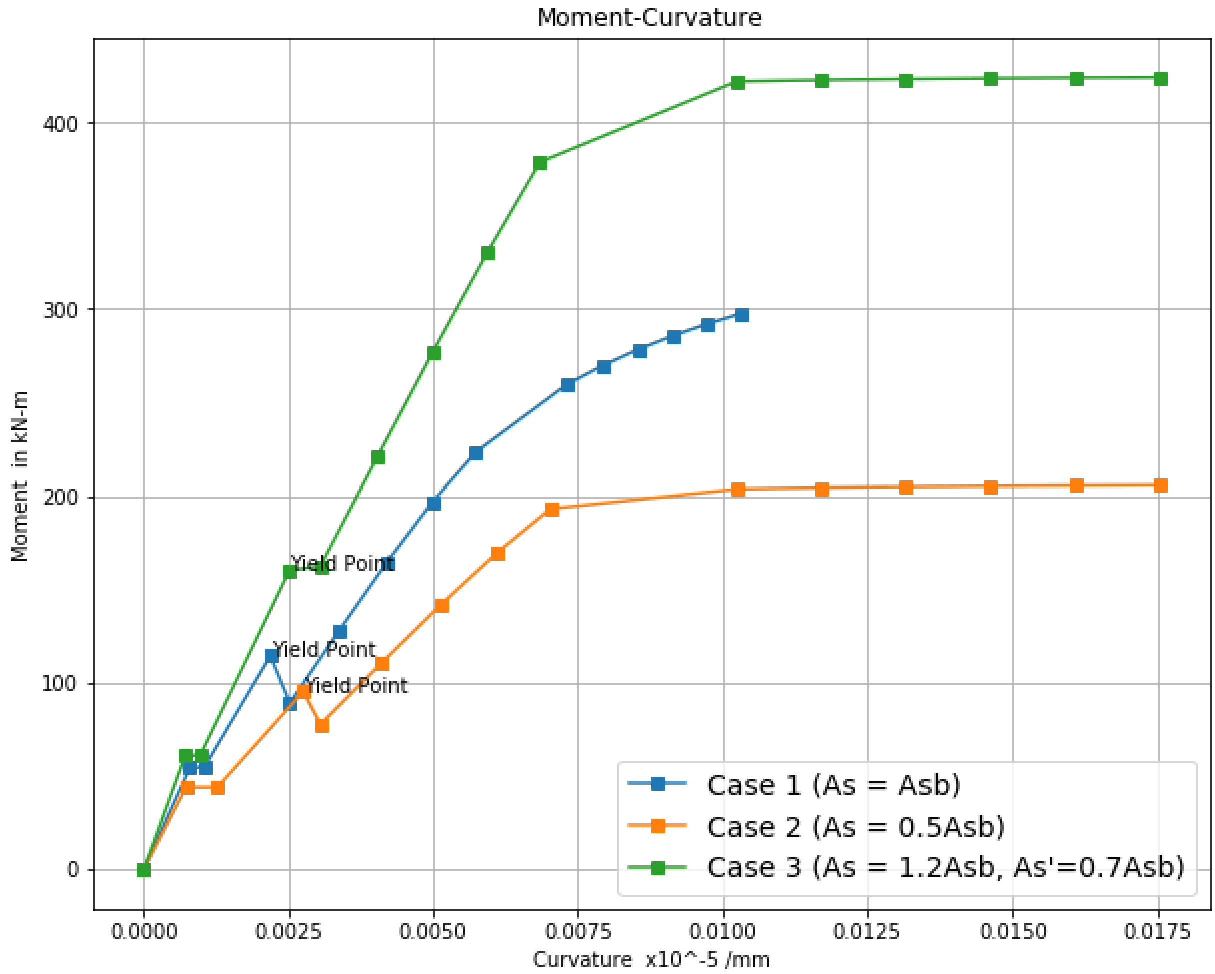


IV. Comments

Following are comments and findings in this problem set.

- The first that I find here is in the chart above, the curve for **Case 1**. It can be seen that it has the smallest curvature. Compared to the curve of **Case 2** which has a smallest amount of tensile reinforcement, shows a gradual change in Moment/Load with a high degree of visibility in change in curvature. The same goes to **Case 3**. This indicates, in my opinion, that they shows ductile behavior. The beams shows a large change in curvature which can be relate to the beams deflection (the larger the angle of curvature, the larger the deflection) while the beam at case 1 shows a brittle behavior. The beam reached its allowable strain of 0.003 in concrete without much change in curvature relative to load.
- Following the 1st comment, if we look at the table in the Results section at Case 1, we can see that the tensile reinforcements did not yield until the beam failed. This could be the reason why we avoid to have a balanced design or even an over reinforced design for that matter.
- After the elastic stage, I also noticed the change in slope in the range of $\epsilon c < \epsilon o$. The slope goes steeper and steeper until ϵc approaches ϵo and then it abruptly goes almost flat. I'm not sure if this though if this is what's called the *strain hardening* before the crushing of concrete at failure.

- Lastly is the comparison of calculated moment at $fc = 0.5f'c$ from that computed at yield point using triangular stress block (*end of elastic stage*) and that of computing it using PCA stress block (*considering inelastic stage*). In the figure at the result above, I started the ϵc iteration way far ahead of calculated strain ($\epsilon_{0.5f'c}$). The moment calculated using inelastic approach is less than of that computed using elastic approach at $fc = 0.5f'c$ as shown below (for Case 1 and Case 2).



V. Appendix

References

- Gillesania, DI T., *Simplified Reinforced Concrete Design*, Diego Innocencio Tapang Guillesania, 2013
- Ćurić, I., Radić, J., Franetović, M., *DETERMINATION OF THE BENDING MOMENT – CURVATURE RELATIONSHIP FOR REINFORCED CONCRETE HOLLOW SECTION BRIDGE COLUMNS*, n.d.
- 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 data. 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%201.ipynb)

```
# Imports
import math
import matplotlib.pyplot as plt

# Define parameters
b = 300                            # Beam width
h = 450                            # Beam height
clearance = 50                     # Clearance from tension steel to bottom of concrete
d = h - clearance                  # d - Effective depth
d_prime = 50                       # d' - Distance from compression steel to concrete compression fiber
fcprime = 21                       # f'c - Concrete compressive strength
fy = 275                           # fy - Steel tensile strength
fr = 0.7 * math.sqrt(fcprime)      # Modulus of fructure
Es = 200000                        # Modulus of elasticity of steel
Ec = 4700 * math.sqrt(fcprime)     # Modulus of elasticity of concrete
β1 = 0.85                          # Beta
η = Es / Ec                        # Modular ratio

# As balance
ρb = (0.85 * fcprime * β1 * 600) / (fy * (600 + fy)) # Balance concret-steel ratio
Asb = ρb * b * d                   # As balance

# Cases
As = [Asb, 0.5*Asb, Asb]       # Tension reinforcements
AsPrime = [0.0, 0.0, 0.5*Asb]      # Compression reinforcements

# Data holders
M = ([], [], [])                   # Array of moments for the 3 cases
φ = ([], [], [])                   # Array of curvature for the 3 cases
I = ([], [], [])                   # Array of all computed moment of inertias
kd = ([], [], [])                  # Array of values of neutral axis to compression fiber
fsm = ([], [], [])                 # Array of strains in concrete
yield_pts = []
```

```
# =========================================
# Utilities
# =========================================
def solveLo(case_no, λ):
    if case_no ==1:
        return 0.85 / 3 * λ * (3 - λ)
    else:
        return 0.85 * (3*λ - 1) / (3 * λ)
```

```
# Insert initial values for moment and curvature
for i in range(3):
    M[i].append(0.0)
    φ[i].append(0.0)

for i in range(3):
    # =========================================== #
    # Calculation before cracking                 #
    # =========================================== #
    # Calculate for kd of each case
    At = b * h                                          # Concrete alone
    At += (η-1) * As[i]                                 # Concrete plus transformed
tension steel
    At += (η-1) * AsPrime[i]                            # Plus transformed compression
steel
    Ma = (b * h) * (h / 2)                              # Moment of area of concrete to
compression fiber
    Ma += (η-1) * As[i] * d                             # Moment of tension reinf. to
compression fiber
    Ma += (η-1) * AsPrime[i] * d_prime                  # Moment of compression reinf.
to compression fiber
    kdCalculated = Ma / At
    kd[i].append(kdCalculated)                          # Insert to list of kd

    # Calculate for moment of inertia of each case
    Ic = (b * kdCalculated**3 / 12) + (b * kdCalculated * (kdCalculated / 2)**2)
    Ic += (b * (h - kdCalculated)**3 / 12) + (b * (h - kdCalculated) * ((h -
kdCalculated) / 2)**2)
    Ic += (η-1) * As[i] * (d - kdCalculated)**2
    Ic += (η-1) * AsPrime[i] * (kdCalculated - d_prime)**2
    I[i].append(Ic)                                     # Insert to list of I

    # Calculate the cracking moment
    Mcr = fr* Ic / (h - kdCalculated)                   # Cracking moment
    M[i].append(Mcr)                                    # Insert to list of M

    # Calculate the curvature
    φc = fr / (Ec * (h - kdCalculated))                 # Curvature right before
cracking
    φ[i].append(φc)                                     # Insert to list of φ

    # =========================================== #
    # Calculation after cracking                  #
    # =========================================== #
    # Finding the neutral axis using equilibrium of moment of areas
    # b(kd)(kd/2) + (n-1)As'(kd-d') = nAs(d-kd)
    # -- solve the quadratic equation
    qa = b
    qb = 2 * ((η-1) * AsPrime[i] + η * As[i])
    qc = -2 * ((η-1) * AsPrime[i] * d_prime + η * As[i] * d)
    qd = (qb**2) - (4 * qa * qc)                        # Discriminant
    kdCalculated = (-1 * qb + math.sqrt(qd)) / (2 * qa) # Neutral axis after cracking
    kd[i].append(kdCalculated)
```

```
    # Calculate moment of inertia
    Ic = (b * kdCalculated**3 / 12) + (b * kdCalculated * (kdCalculated / 2)**2)
    Ic += (η) * As[i] * (d - kdCalculated)**2
    Ic += (η-1) * AsPrime[i] * (kdCalculated - d_prime)**2
    I[i].append(Ic)

    # Calculate the curvature
    φc = M[i][1] / (Ec * Ic)                              # Curvature right after
cracking

    M[i].append(Mcr)
    φ[i].append(φc)

    # =========================================== #
    # Calculation at yield point                  #
    # =========================================== #
    fc = 0.5 * fcprime
    ϵc = fc / Ec

    qa = 0.5 * fc * b
    qb = (Es * ϵc) * (AsPrime[i] + As[i])
    qc = -(Es * ϵc) * (AsPrime[i] * d_prime + As[i] * d)
    qd = (qb**2) - (4 * qa * qc)            # Discriminant
    kdCalculated = (-1 * qb + math.sqrt(qd)) / (2 * qa)

    fs = (Es * ϵc) * (d - kdCalculated) / kdCalculated
    fsPrime = Es * ϵc / kdCalculated * (kdCalculated - d_prime)
    if fs > fy:
        fs = fy

    if fsPrime > fy:
        fsPrime = fys

    Mc = 0.5 * fc * b * kdCalculated * (d - kdCalculated / 3) +\
                AsPrime[i] * fsPrime * (d - d_prime)
    φc = ϵc / kdCalculated

    M[i].append(Mc)
    φ[i].append(φc)

    yield_pts.append((φc*1000, Mc / 1000**2))

    # =========================================== #
    # Calculation at inelastic behaviour          #
    # =========================================== #
    # Calculate for ϵo
    ϵo = 2 * 0.85 * fcprime / Ec        # This is overridden below

    # Iterator increment
    iterator_increment = 0.0002

    # For 0 < ϵc < ϵo
```

```
    εc = 0.5 * εo                       # To override above εo

    # For case 0 < εc < εo
    while (εc + iterator_increment) <= εo:
        εc = εc + iterator_increment
        λo = εc / εo
        k2 = 1 / 4 * (4 - λo) / (3 - λo)
        Lo = solveLo(1, λo)
        fc = 0.85 * fcprime * (2 * λo - λo**2)
        kdCalculated = (As[i] - AsPrime[i]) * fy / (Lo * fc * b)
        fs = (Es * εc) * (d - kdCalculated) / kdCalculated
        fsPrime = Es * εc / kdCalculated * (kdCalculated - d_prime)

        if fs >= fy:                                # Tension steel yields
            # Solve for the stress in compression steel
            if fsPrime < fy:
                # Compression steel does not yields
                qa = Lo * fc * b
                qb = (Es * εc) * AsPrime[i] - As[i] * fy
                qc = -(Es * εc) * AsPrime[i] * d_prime
                qd = (qb**2) - (4 * qa * qc)            # Discriminant
                kdCalculated = (-1 * qb + math.sqrt(qd)) / (2 * qa)
                fs = (Es * εc) * (d - kdCalculated) / kdCalculated
                fsPrime = Es * εc / kdCalculated * (kdCalculated - d_prime)
            else:
                # fs and fs' > fy
                kdCalculated = (As[i] - AsPrime[i]) * fy / (Lo * fc * b)
                fs = fy
                fsPrime = fy
        else:
            qa = Lo * fc * b
            qb = AsPrime[i] * fy + As[i] * Es * εc
            qc = -As[i] * Es * εc * d
            qd = (qb**2) - (4 * qa * qc)            # Discriminant
            kdCalculated = (-1 * qb + math.sqrt(qd)) / (2 * qa)
            fs = (Es * εc) * (d - kdCalculated) / kdCalculated
            fsPrime = Es * εc / kdCalculated * (kdCalculated - d_prime)

            if fsPrime < fy:
                # Compression syeel did not yield
                # Compression steel does not yields
                qa = Lo * fc * b
                qb = (Es * εc) * (AsPrime[i] + As[i])
                qc = -(Es * εc) * (As[i] * d + AsPrime[i] * d_prime)
                qd = (qb**2) - (4 * qa * qc)            # Discriminant
                kdCalculated = (-1 * qb + math.sqrt(qd)) / (2 * qa)
                fs = (Es * εc) * (d - kdCalculated) / kdCalculated
                fsPrime = Es * εc / kdCalculated * (kdCalculated - d_prime)

        Mc = Lo * fc * b * kdCalculated * (d - k2 * kdCalculated) +\
                AsPrime[i] * fsPrime * (d - d_prime)
        φc = εc / kdCalculated
```

```
        M[i].append(Mc)
        φ[i].append(φc)

    # For case ϵo < ϵc < ϵcu
    ϵc = ϵo + 0.0001
    while (ϵc + iterator_increment) <= 0.003:
        ϵc = ϵc + iterator_increment
        ζc = ϵo / ϵc
        λo = 1 / ζc
        Lo = solveLo(2, λo)
        k2 = (6 * λo**2 - 4 * λo + 1) / (4 * λo * (3 * λo - 1))
        fc = 0.85 * fcprime
        kdCalculated = (As[i] - AsPrime[i]) * fy / (Lo * fc * b)
        fs = (Es * ϵc) * (d - kdCalculated) / kdCalculated
        fsPrime = Es * ϵc / kdCalculated * (kdCalculated - d_prime)

        if fs >= fy:                                # Tension steel yields
            # Solve for the stress in compression steel
            if fsPrime < fy:
                # Compression steel does not yields
                qa = Lo * fc * b
                qb = (Es * ϵc) * AsPrime[i] - As[i] * fy
                qc = -(Es * ϵc) * AsPrime[i] * d_prime
                qd = (qb**2) - (4 * qa * qc)            # Discriminant
                kdCalculated = (-1 * qb + math.sqrt(qd)) / (2 * qa)
                fs = (Es * ϵc) * (d - kdCalculated) / kdCalculated
                fsPrime = Es * ϵc / kdCalculated * (kdCalculated - d_prime)
            else:
                # fs and fs' > fy
                kdCalculated = (As[i] - AsPrime[i]) * fy / (Lo * fc * b)
                fs = fy
                fsPrime = fy
        else:
            qa = Lo * fc * b
            qb = AsPrime[i] * fy + As[i] * Es * ϵc
            qc = -As[i] * Es * ϵc * d
            qd = (qb**2) - (4 * qa * qc)            # Discriminant
            kdCalculated = (-1 * qb + math.sqrt(qd)) / (2 * qa)
            fs = (Es * ϵc) * (d - kdCalculated) / kdCalculated
            fsPrime = Es * ϵc / kdCalculated * (kdCalculated - d_prime)

            if fsPrime < fy:
                # Compression syeel did not yield
                # Compression steel does not yields
                qa = Lo * fc * b
                qb = (Es * ϵc) * (AsPrime[i] + As[i])
                qc = -(Es * ϵc) * (As[i] * d + AsPrime[i] * d_prime)
                qd = (qb**2) - (4 * qa * qc)            # Discriminant
                kdCalculated = (-1 * qb + math.sqrt(qd)) / (2 * qa)
                fs = (Es * ϵc) * (d - kdCalculated) / kdCalculated
                fsPrime = Es * ϵc / kdCalculated * (kdCalculated - d_prime)

        Mc = Lo * fc * b * kdCalculated * (d - k2 * kdCalculated) +\
```

```python
                AsPrime[i] * fsPrime * (d - d_prime)
        φc = ϵc / kdCalculated

        φ[i].append(φc)
        M[i].append(Mc)
```

```python
# Convert the values of data to smaller figures before plotting
φ_converted = ([], [], [])
M_converted = ([], [], [])
for i in range(3):
    for curvature in φ[i]:
        φ_converted[i].append(curvature * 1000)
    for moment in M[i]:
        M_converted[i].append(moment / 1000**2)

# Plot the curves
plt.figure(figsize=(10,8))
plt.title("Moment-Curvature")
plt.xlabel('Curvature  x10^-5 /mm')
plt.ylabel('Moment  in kN-m')
plt.grid()

for yp in yield_pts:
    plt.text(yp[0], yp[1], 'Yield Point')

# Plot the converted values
case1, = plt.plot(φ_converted[0], M_converted[0], marker='s', label='Case 1 (As = 
Asb)')
case2, = plt.plot(φ_converted[1], M_converted[1], marker='s', label='Case 2 (As = 
0.5Asb)')
case3, = plt.plot(φ_converted[2], M_converted[2], marker='s', label='Case 3 (As = 
1.2Asb, As\'=0.7Asb)')
plt.legend(handles=[case1, case2, case3], loc='best', fontsize=14)
plt.show()
```