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

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:

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

```
# Imports
from sympy import *
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):
    print('=======================================')
    print('Case No.', i+1)
    print('=======================================')
    # =========================================== #
    # 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
    print('Mcr = ', round(Mcr / 1000**2, 2), 'φc = ', φc, 'kd = ', kdCalculated)

    # =========================================== #
    # 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)
```

```python
    print('Mcr = ', round(Mcr / 1000**2, 2), 'φc = ', φc, 'kd = ', kdCalculated, "After
cracking...")

    # ============================================ #
    # 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)
    print('Myield = ', round(Mc / 1000**2, 2), 'φc = ', φc, 'ϵc = ', ϵc, 'kd = ',
kdCalculated)

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

    # ============================================ #
    # Calculation at inelastic behaviour           #
    # ============================================ #
    # Calculate for ϵo
    ϵo = 2 * 0.85 * fcprime / Ec

    # Iterator increment
    iterator_increment = 0.0002

    # For 0 < ϵc < ϵo
    ϵc = 0.5 * ϵo                                # My setting for starting
strain iteration
    print('For 0 < ϵc < 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)
                print('Tension steel yields but compression steel did not.')
            else:
                # fs and fs' > fy
                kdCalculated = (As[i] - AsPrime[i]) * fy / (Lo * fc * b)
                fs = fy
                fsPrime = fy
                print('Both tension and compression steel yields.')
        else:
            print('Tension steel did not yield.')
            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)

        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)

        print('Mc = ', round(Mc / 1000**2, 2), 'φc = ', round(φc, 8), 'εc = ',
round(εc, 5), 'kd = ', round(kdCalculated, 0), 'fc = ', fc)

    print("For εo < εc < εcu")
    # 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)
                print('Tension steel yields but compression steel did not.')
            else:
                # fs and fs' > fy
                kdCalculated = (As[i] - AsPrime[i]) * fy / (Lo * fc * b)
                fs = fy
                fsPrime = fy
                print('Both tension and compression steel yields.')
        else:
            print('Tension steel did not yield.')
            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)

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

        print('Mc = ', round(Mc / 1000**2, 2), 'φc = ', round(φc, 8), 'εc = ',
round(εc, 5), 'kd = ', round(kdCalculated, 0), 'fc = ', fc)
        φ[i].append(φc)
        M[i].append(Mc)
```

```
=======================================
Case No. 1
=======================================
Mcr =  54.57 φc =  7.970318780711861e-07 kd =  263.13649515099587
Mcr =  54.57 φc =  1.0428634969669626e-06 kd =  223.02262969808856 After cracking...
Myield =  114.39 φc =  2.1859129421746513e-06 εc =  0.0004875080526548766 kd =
223.0226296980886
For 0 < εc < εo
Tension steel did not yield.
Mc =  153.69 φc =  3.95e-06 εc =  0.00103 kd =  261.0 fc =  15.281428223688067
Tension steel did not yield.
Mc =  187.86 φc =  4.76e-06 εc =  0.00123 kd =  258.0 fc =  16.655591741493783
```

```
Tension steel did not yield.
Mc =  216.42 φc =  5.52e-06 ϵc =  0.00143 kd =  259.0 fc =  17.509990553417143
Tension steel did not yield.
Mc =  237.47 φc =  6.23e-06 ϵc =  0.00163 kd =  261.0 fc =  17.844624659458155
For ϵo < ϵc < ϵcu
Tension steel did not yield.
Mc =  259.5 φc =  7.3e-06 ϵc =  0.00196 kd =  268.0 fc =  17.849999999999998
Tension steel did not yield.
Mc =  269.74 φc =  7.93e-06 ϵc =  0.00216 kd =  272.0 fc =  17.849999999999998
Tension steel did not yield.
Mc =  278.35 φc =  8.54e-06 ϵc =  0.00236 kd =  276.0 fc =  17.849999999999998
Tension steel did not yield.
Mc =  285.68 φc =  9.14e-06 ϵc =  0.00256 kd =  280.0 fc =  17.849999999999998
Tension steel did not yield.
Mc =  292.0 φc =  9.73e-06 ϵc =  0.00276 kd =  283.0 fc =  17.849999999999998
Tension steel did not yield.
Mc =  297.51 φc =  1.032e-05 ϵc =  0.00296 kd =  287.0 fc =  17.849999999999998
=======================================
Case No. 2
=======================================
Mcr =  43.86 φc =  7.315136942596765e-07 kd =  246.40002452244482
Mcr =  43.86 φc =  1.27052110091158e-06 kd =  177.01582310560858 After cracking...
Myield =  95.07 φc =  2.7540365832947417e-06 ϵc =  0.0004875080526548766 kd =
177.01582310560858
For 0 < ϵc < ϵo
Tension steel did not yield.
Mc =  132.65 φc =  4.82e-06 ϵc =  0.00103 kd =  213.0 fc =  15.281428223688067
Tension steel did not yield.
Mc =  161.97 φc =  5.83e-06 ϵc =  0.00123 kd =  211.0 fc =  16.655591741493783
Tension steel did not yield.
Mc =  186.87 φc =  6.76e-06 ϵc =  0.00143 kd =  211.0 fc =  17.509990553417143
Tension steel yields but compression steel did not.
Mc =  201.24 φc =  7.85e-06 ϵc =  0.00163 kd =  208.0 fc =  17.844624659458155
For ϵo < ϵc < ϵcu
Both tension and compression steel yields.
Mc =  203.58 φc =  1.024e-05 ϵc =  0.00196 kd =  191.0 fc =  17.849999999999998
Both tension and compression steel yields.
Mc =  204.35 φc =  1.17e-05 ϵc =  0.00216 kd =  184.0 fc =  17.849999999999998
Both tension and compression steel yields.
Mc =  204.89 φc =  1.316e-05 ϵc =  0.00236 kd =  179.0 fc =  17.849999999999998
Both tension and compression steel yields.
Mc =  205.27 φc =  1.462e-05 ϵc =  0.00256 kd =  175.0 fc =  17.849999999999998
Both tension and compression steel yields.
Mc =  205.55 φc =  1.608e-05 ϵc =  0.00276 kd =  172.0 fc =  17.849999999999998
Both tension and compression steel yields.
Mc =  205.76 φc =  1.754e-05 ϵc =  0.00296 kd =  169.0 fc =  17.849999999999998
=======================================
Case No. 3
=======================================
Mcr =  60.97 φc =  7.167101269032306e-07 kd =  242.19468984442693
Mcr =  60.97 φc =  9.733433018711496e-07 kd =  196.764121236301 After cracking...
Myield =  159.98 φc =  2.5125377830969175e-06 ϵc =  0.0004875080526548766 kd =
194.0301379484058
```

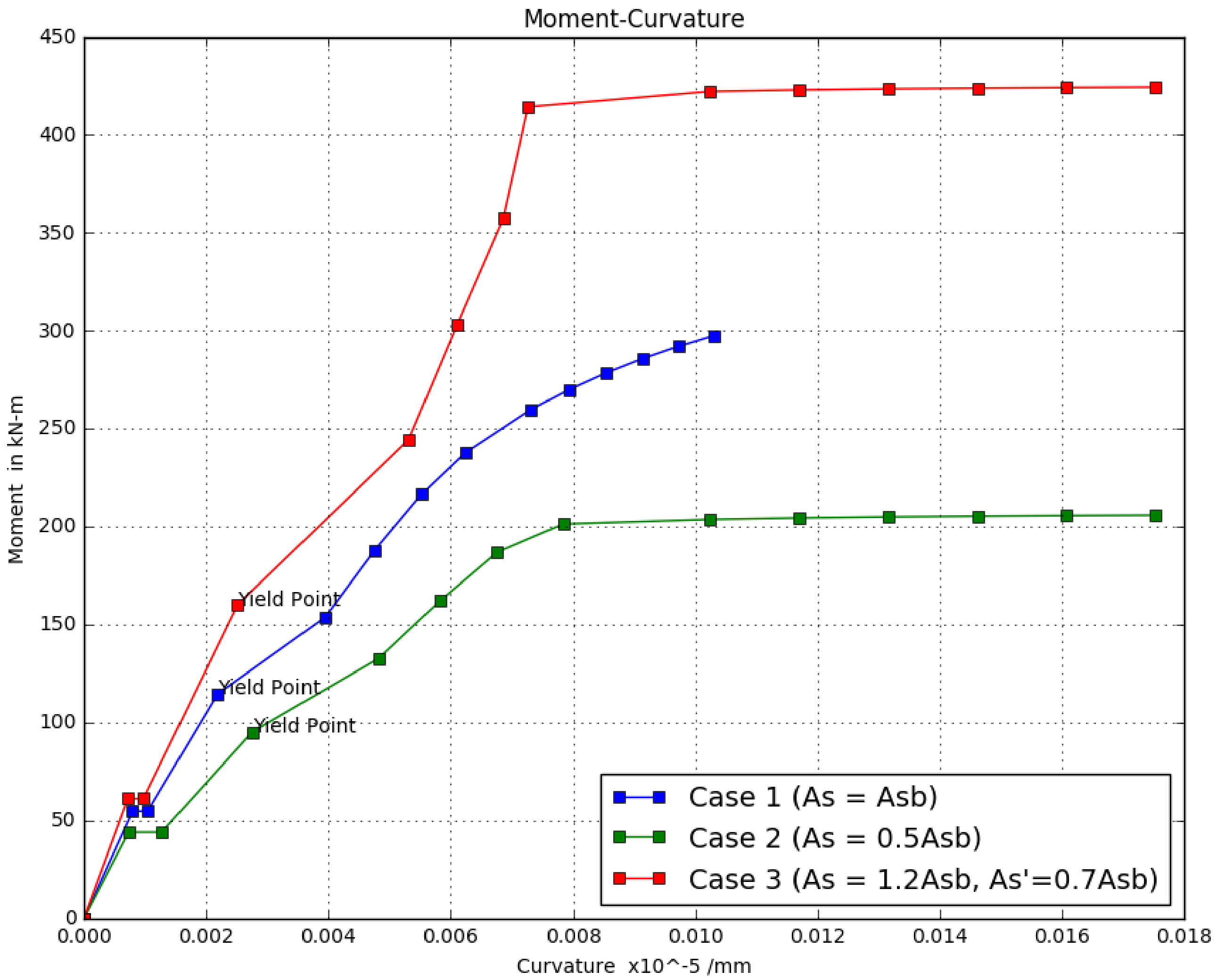
```
For 0 < ϵc < ϵo
Tension steel did not yield.
Mc =  244.28 φc =  5.31e-06 ϵc =  0.00103 kd =  194.0 fc =  15.281428223688067
Tension steel did not yield.
Mc =  302.97 φc =  6.1e-06 ϵc =  0.00123 kd =  201.0 fc =  16.655591741493783
Tension steel did not yield.
Mc =  357.3 φc =  6.86e-06 ϵc =  0.00143 kd =  208.0 fc =  17.509990553417143
Tension steel yields but compression steel did not.
Mc =  414.18 φc =  7.27e-06 ϵc =  0.00163 kd =  224.0 fc =  17.844624659458155
For ϵo < ϵc < ϵcu
Both tension and compression steel yields.
Mc =  422.07 φc =  1.024e-05 ϵc =  0.00196 kd =  191.0 fc =  17.849999999999998
Both tension and compression steel yields.
Mc =  422.84 φc =  1.17e-05 ϵc =  0.00216 kd =  184.0 fc =  17.849999999999998
Both tension and compression steel yields.
Mc =  423.37 φc =  1.316e-05 ϵc =  0.00236 kd =  179.0 fc =  17.849999999999998
Both tension and compression steel yields.
Mc =  423.75 φc =  1.462e-05 ϵc =  0.00256 kd =  175.0 fc =  17.849999999999998
Both tension and compression steel yields.
Mc =  424.03 φc =  1.608e-05 ϵc =  0.00276 kd =  172.0 fc =  17.849999999999998
Both tension and compression steel yields.
Mc =  424.25 φc =  1.754e-05 ϵc =  0.00296 kd =  169.0 fc =  17.849999999999998
```

```
# Convert the values of data to smaller figures
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()
```


Moment-Curvature
Moment in kN-m
Curvature x10^-5 /mm
0
50
100
150
200
250
300
350
400
450
0.000
0.002
0.004
0.006
0.008
0.010
0.012
0.014
0.016
0.018
Yield Point
Yield Point
Yield Point
Case 1 (As = Asb)
Case 2 (As = 0.5Asb)
Case 3 (As = 1.2Asb, As'=0.7Asb)