Dimensionamento da Longarina

Equipe

Arthur Brito Gomes

Francisco Gustavo

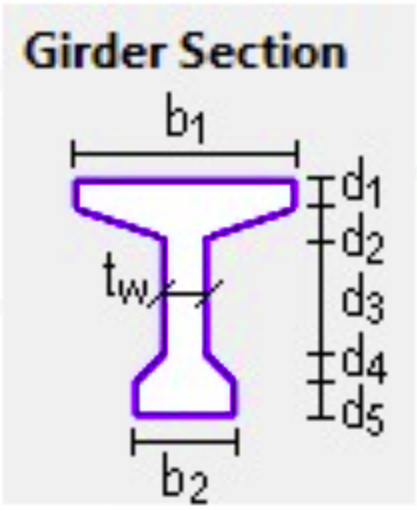
Gustavo Gomes

José Enrico

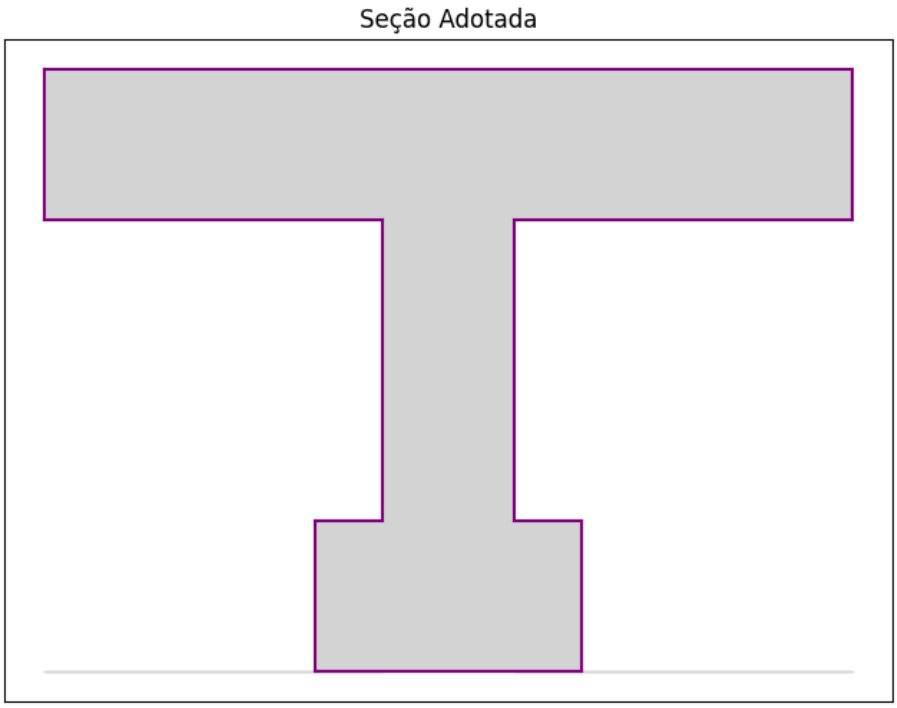
Imports

In [ ]:
```
import math
from funcoes import (
    to_red as re,
    to_yellow as ye,
    arredonda_pra_cima,
    arredonda_pra_baixo,
    show_section,
    integrate_linear_function
)
from ABNT_NBR_7188 import P, p, CNF, CIA, CIV
```

Dados



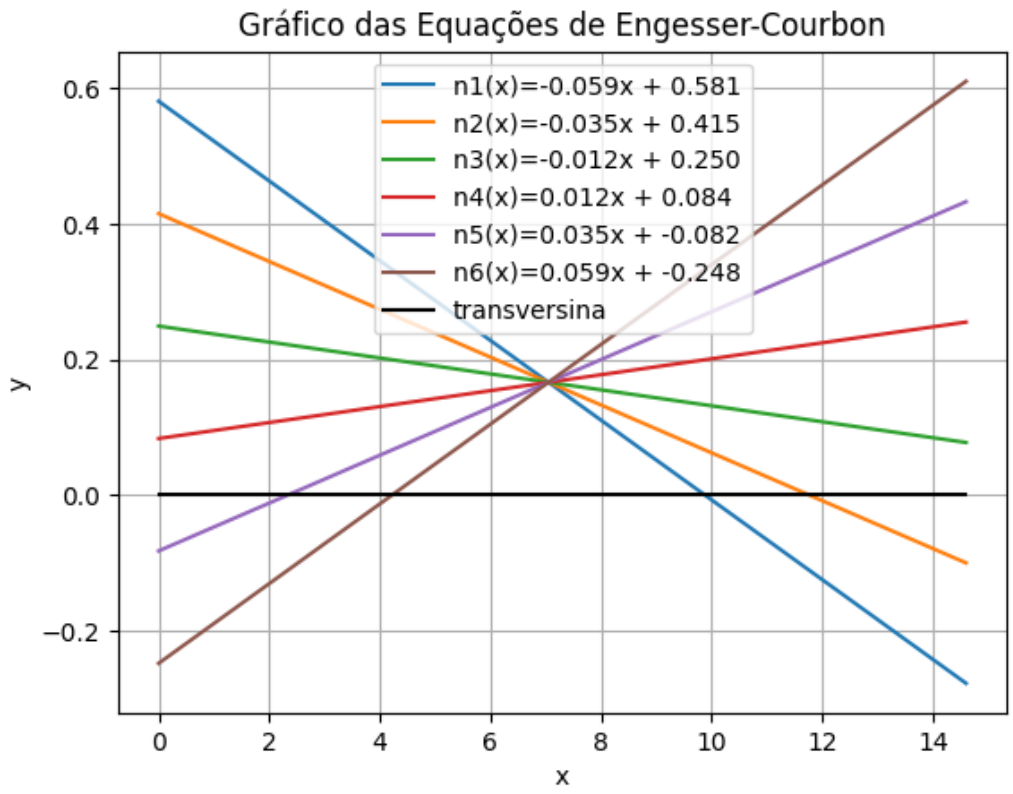
In [ ]:
```python
# Dimensões da viga (Seção T) OBS: Verificar secao.png
b1 = 2.43
b2 = 0.8
tw = 0.4

d1 = 0.2
d2 = 0
d3 = 0.4
d4 = 0
d5 = 0.2

d_linha = 0.04

d = 0.7375 # Valor modificado para deixar com menor valor de diferença entre d'-d_r
# d = (d1 + d2 + d3 + d4 + d5) - d_linha


# Concreto
bitola_agregado = (3 / 4) * 2.54e-2
fck = 30e6
fcd = fck / 1.4

# Aço
num_ramos = 2
fy = 500e6
fyd = fy / 1.15
```



Trem Tipo

In [ ]:
```python
n = 6  # número de longarinas
E = 2.43  # distância entre eixos
equacoes = []

for i in range(1, n + 1):
    aux = (6 / n) * ((2 * i - n - 1) / ((n**2 - 1) * E))
    a = aux
    b = -aux * 7.05 + 1 / n
    equacoes.append((a, b))

print(f'Equações de Engesser-Courbon:')
for i, equacao in enumerate(equacoes):
    a, b = equacao
    print(f'n{i+1}(x)={a:.3f}x + {b:.3f}')

# print(f'Pontos de análise:')
coordenada_x_do_comeco_trem_tipo = 2.5
pontos = [ponto + coordenada_x_do_comeco_trem_tipo for ponto in [0, 0.5, 2.5, 3]]
pontos_de_analise = []
for i, equacao in enumerate(equacoes):
    analise = []
    for ponto in pontos:
        a, b = equacao
        ni = a * ponto + b
        analise.append(ni)
        # print(f'n{i+1}({ponto})={ni:.3f}')
    pontos_de_analise.append(analise)
    # print()

comprimento_sem_trem_tipo = 3.4
print(f'Trem tipo:')
for i, (a, y1, y2, b) in enumerate(pontos_de_analise):
    a = abs(a)
    y1 = abs(y1)
    y2 = abs(y2)
    b = abs(b)
    A1 = integrate_linear_function(equacoes[i][0], equacoes[i][1], pontos[0], ponto
    A2 = integrate_linear_function(equacoes[i][0], equacoes[i][1], pontos[len(ponto
    Q1 = 75 * (y1 + y2)
    q1 = 5 * (A1 + A2)
    q2 = 5 * A2
    print(f'Viga {i+1}: Q1={Q1:.2f} q1={q1:.2f} q2={q2:.2f}')
```

```
Equações de Engesser-Courbon:
n1(x)=-0.059x + 0.581
n2(x)=-0.035x + 0.415
n3(x)=-0.012x + 0.250
n4(x)=0.012x + 0.084
n5(x)=0.035x + -0.082
n6(x)=0.059x + -0.248
Trem tipo:
Viga 1: Q1=51.90 q1=7.87 q2=2.68
Viga 2: Q1=41.14 q1=6.86 q2=2.74
Viga 3: Q1=30.38 q1=5.84 q2=2.80
Viga 4: Q1=19.62 q1=4.83 q2=2.86
Viga 5: Q1=8.86 q1=3.81 q2=2.92
Viga 6: Q1=8.82 q1=3.64 q2=2.98
```



Peso Próprio

In [ ]:
```
'''
  Entrada de dados:
'''

# Trem tipo intermediário
distancia_1o_vao = b1
distancia_2o_vao = b1
# Trem tipo canto
distancia_entre_longarinas = b1
```

```
# Pesos específicos Materiais
peso_especifico_concreto = 25  # kN/m³
peso_especifico_pavimento = 24  # kN/m³

# AREAS DO PROJETO CAD
# Longarina
area_secao_longarina = (
    b1 * d1 +
    (tw + b1) * d2 / 2 +
    tw * d3 + (tw + b2) * d4 / 2 +
    d5 * b2
)

# Guarda_roda
area_guarda_roda = 0.218  # m²

# Guarda_corpo
q_guarda_corpo = 0.157  # kN/m

# Transversinas
area_septo_pilar = 0.7 * 0.35  # altura x base
comprimento_septo_pilar = b1  # m

# Recapeamento
q_recapeamento = 2  # kN/m²

# Comprimentos adotados
espessura_de_asfalto = 0.04  # m
altura_passeio = 0.04  # m

# CARREGAMENTOS FTOOL
# Distribuído
ppV1 = area_secao_longarina * peso_especifico_concreto
ppV1 += q_guarda_corpo
ppV1 += 2 * 0.04 * peso_especifico_pavimento
ppV1 += 2 * q_recapeamento
ppV1 += area_guarda_roda * peso_especifico_concreto

ppV2 = area_secao_longarina * peso_especifico_concreto
ppV2 += 2.36 * 0.04 * peso_especifico_pavimento
ppV2 += 2.36 * q_recapeamento

ppV3 = area_secao_longarina * peso_especifico_concreto
ppV3 += b1 * 0.04 * peso_especifico_pavimento
ppV3 += b1 * q_recapeamento

ppV4 = area_secao_longarina * peso_especifico_concreto
ppV4 += area_guarda_roda * peso_especifico_concreto
ppV4 += (b1 - 0.4) * 0.04 * peso_especifico_pavimento
ppV4 += (b1 - 0.4) * q_recapeamento

ppV5 = area_secao_longarina * peso_especifico_concreto
ppV5 += q_guarda_corpo
ppV5 += (b1 - 0.1) * 0.04 * peso_especifico_pavimento
ppV5 += (b1 - 0.1) * q_recapeamento
```

```
ppV6 = area_secao_longarina * peso_especifico_concreto
ppV6 += q_guarda_corpo
ppV6 += (b1 - 0.1) * 0.04 * peso_especifico_pavimento
ppV6 += (b1 - 0.1) * q_recapeamento

# Concentrado
concentrada = area_septo_pilar * comprimento_septo_pilar * peso_especifico_concreto

# Fator de impacto
fator_de_impacto = CNF(2) * CIA('concreto') * CIV(20)
```

In [ ]:
```
print(f'CARGAS DISTRIBUÍDAS:')
print(f'    V1: {ppV1:.2f}')
print(f'    V2: {ppV2:.2f}')
print(f'    V3: {ppV3:.2f}')
print(f'    V4: {ppV4:.2f}')
print(f'    V5: {ppV5:.2f}')
print(f'    V6: {ppV6:.2f}')

print(f'CARGAS CONCENTRADAS:')
print(f'    Transversina no pilar: {concentrada:.2f}')

print(f'FATOR DE IMPACTO={fator_de_impacto:.2f}')
```

```
CARGAS DISTRIBUÍDAS:
    V1: 31.68
    V2: 27.14
    V3: 27.34
    V4: 31.61
    V5: 27.20
    V6: 27.20
CARGAS CONCENTRADAS:
    Transversina no pilar: 14.88
FATOR DE IMPACTO=1.63
```

Dimensionamento Longarina

Funções

In [ ]:
```
def calcular_epslon(Md: float, b1: float, d: float, fcd: float) -> tuple[float, flo
    '''
    Equação que retorna epslon (x/d).
    '''
    a = 0.4
    b = -1
    c = Md / (0.68 * b1 * (d**2) * fcd)

    delta = b**2 - 4 * a * c

    if delta < 0:
        raise Exception('Dimensões da viga não válidas.')
```

```
    raiz1 = (-b + math.sqrt(delta)) / (2 * a)
    raiz2 = (-b - math.sqrt(delta)) / (2 * a)
    return raiz1, raiz2
```

In [ ]:
```
def verifica_dominio(epslon: float) -> str:
    '''
    Verifica o domínio com base em um epslon (x/d).
    '''
    if epslon < 0:
        return '1'
    elif 0 <= epslon <= 0.259:
        return '2'
    elif 0.259 < epslon <= 0.450:
        return '3a'
    elif 0.450 < epslon <= 0.628:
        return '3b'
    elif 0.628 < epslon <= 1:
        return '4'
    elif 1 < epslon:
        return '5'
    else:
        raise Exception('Intervalo do domínio não definido.')
```

Carregamentos

In [ ]:
```
# Comprimento ponte
L = 25

# Carregamentos
PP = max(ppV1, ppV2, ppV3, ppV4, ppV5, ppV6)
print(f'Peso próprio = {PP:.2f} kN/m')

# Momento Fletor
Mg = (PP * L**2 / 8) * 1e3
Mq = 2142.2e3
Md = 1.35 * Mg + 1.5 * Mq
print(f'Md={Md:.2e}')

# Esforço cortante
Vsg = (PP * L) * 1e3 / 2
Vsq = 357.6e3
Vsd = 1.35 * Vsg + 1.5 * Vsq
print(f'Vsd={Vsd:.2e}')
```

```
Peso próprio = 31.68 kN/m
Md=6.55e+06
Vsd=1.07e+06
```

Verificação Domínio

In [ ]:
```
raiz1, raiz2 = calcular_epslon(Md, b1, d, fcd)
epslon = min(raiz1, raiz2)
x = epslon * d
```

```
y = 0.8 * x

if y > d1 and not (b1 == b2 == tw):
    Md1 = 0.85 * fcd * (b1 - tw) * d1 * (d - 0.5 * d1)
    Md2 = Md - Md1
    Md = Md2
    raiz1, raiz2 = calcular_epslon(Md2, b1, d, fcd)
    epslon = min(raiz1, raiz2)
    x = epslon * d
    y = 0.8 * x

if epslon > 0.45:
    print(y(f'Necessita armadura dupla. O calculo não considera isso.'))

dominio = verifica_dominio(epslon)
print(f'x/d={epslon} x={x} y={y} Domínio {dominio}')
```

```
x/d=0.09949568860818117 x=0.07337807034853362 y=0.0587024562788269 Domínio 2
```

Cálculo da área de aço

In [ ]:
```
diametro_bitola = 25  # mm

# Area de aço
As_calculado = Md / (fyd * (d - 0.4 * x))
As = As_calculado
taxa_armadura = 0.208 / 100
As_min = taxa_armadura * tw * (d + d_linha)

if As_min > As_calculado:
    As = As_min

area_bitola = math.pi * (diametro_bitola / 1e3)**2 / 4
num_bitolas = arredonda_pra_cima(As / area_bitola)
print(f'Area de aço calculado={As_calculado}; Area de aço min={As_min} -> {num_bito
```

```
Area de aço calculado=0.0059758890840216254; Area de aço min=0.0006468800000000001 -
> 13 Ø 25mm
```

Cálculo da armadura de pele

In [ ]:
```
diametro_bitola_pele = 10  # mm

# Armadura de pele
if d1 + d2 + d3 + d4 + d5 >= 0.6:
    As_pele = (0.1 / 100) * tw * (d1 + d2 + d3 + d4 + d5)
    area_bitola = math.pi * (diametro_bitola_pele / 1e3)**2 / 4
    num_bitolas_pele = arredonda_pra_cima(As_pele / area_bitola)

    print(f'Area de aço pele={As_pele} -> {num_bitolas_pele} Ø {diametro_bitola_pel
else:
    print(f'Não é necessário armadura de pele.')
```

```
Area de aço pele=0.00032 -> 5 Ø 10mm
```

Verificação esforço cortante

In [ ]:
```
diametro_estribo = 10  # mm

# Verificação do esforço cortante
alfa_v2 = 1 - (fck / 1e6) / 250
Vrd2 = 0.27 * alfa_v2 * fcd * tw * d
if Vsd <= Vrd2:
    # Não ocorre ruptura das diagonais de compressão. (Vsd<Vrd2)
    # Resistência a compressão do concreto
    fctd = 0.7 * (0.3 * math.pow(fck / 1e6, 2 / 3)) / 1.4
    Vc = 0.6 * fctd * 1e6 * tw * d
    Vsw = Vsd - Vc
    fywd: int = None
    if fyd <= 435e6:
        fywd = fyd
    else:
        fywd = 435e6
    alfa = math.radians(90)
    Asw = Vsw / (0.9 * d * fywd * (math.sin(alfa) + math.cos(alfa)))
    fctm = 0.3 * math.pow(fck / 1e6, 2 / 3)
    fywk = fy
    Asw_min = 0.2 * (fctm * 1e6 / fywk) * tw * math.sin(alfa)

    if Asw < Asw_min:
        Asw = Asw_min

    area_estribo = num_ramos * math.pi * (diametro_estribo / 1e3)**2 / 4
    num_estribos = arredonda_pra_cima(Asw / area_estribo)
    # Espaçamento de estribos - item 18.3.3.2 da NBR 6118 (2014)
    espacamento_max_estribos: float = None
    if Vsd <= 0.67 * Vrd2:
        if 0.6 * d >= 0.3:
            espacamento_max_estribos = 0.3
        else:
            espacamento_max_estribos = 0.6 * d
    elif Vsd > 0.67 * Vrd2:
        if 0.3 * d >= 0.3:
            espacamento_max_estribos = 0.2
        else:
            espacamento_max_estribos = 0.3 * d
    else:
        raise Exception(re('Ocorreu um erro no espaçamento de estribos.'))

    print(f'Area de aço estribos={Asw}/m; Area de aço min={Asw_min}/m -> {num_estri
    print(f'Consumo esforço cortante = {Vsd/Vrd2 * 100:.2f}%')
    # print(f'Espaçamento máximo entre estribos={espacamento_max_estribos}')
else:
    raise Exception(re(f'Ocorre ruptura das diagonais de compressão. Vsd/Vrd2={Vsd/
```

```
Area de aço estribos=0.0028227607831119725/m; Area de aço min=0.0004634349046107023/
m -> 18 Ø 10mm
Consumo esforço cortante = 71.30%
```

Verificação dos espaçamentos

Considera os valores minimos de espaçamento permitidos pela NBR 6118

In [ ]:
```
espacamento_min_horizontal = max(1.2 * bitola_agregado, 0.02, diametro_bitola * 1e-
# print(f'Espacamento min horizontal={espacamento_min_horizontal}')

espacamento_min_vertical = max(0.5 * bitola_agregado, 0.02, diametro_bitola * 1e-3)
# print(f'Espacamento min vertical={espacamento_min_vertical}')

num_max_de_bitolas_por_camada = arredonda_pra_baixo(
    ((b2 - d_linha * 2 - num_ramos * diametro_estribo * 2e-3) + espacamento_min_hor
    (diametro_bitola * 1e-3 + espacamento_min_horizontal)
)

if num_bitolas <= num_max_de_bitolas_por_camada:
    print(f'1 camada com {num_bitolas} Ø de {diametro_bitola}mm')
    d_real = (d1 + d2 + d3 + d4 + d5 - d_linha - diametro_estribo * 1e-3 - diametro
    print(f'Diferença d_real e d utilizado = {math.fabs(d_real - d) * 100:.2f}%')
else:
    print(f'Precisa de mais de uma camada. 1 camada suporta apenas {num_max_de_bito
    num_de_camadas = arredonda_pra_cima(num_bitolas / num_max_de_bitolas_por_camada
    print(f'Numero de camadas: {num_de_camadas}')
    num_max_de_camadas = arredonda_pra_baixo(
        ((d5 - diametro_estribo * 1e-3 - d_linha) + espacamento_min_vertical) /
        (diametro_bitola * 1e-3 + espacamento_min_vertical)
    )
    if num_de_camadas > num_max_de_camadas:
        raise Exception(f'Não existe seção suficiente para a quantidade de bitolas.
    d_real = d1 + d2 + d3 + d4
    d_real += (
        d5 - d_linha - diametro_estribo * 1e-3 -
        (num_de_camadas * diametro_bitola * 1e-3 + (num_de_camadas - 1) * espacamen
    )  # Folga
    d_real += (num_de_camadas * (diametro_bitola * 1e-3) + (num_de_camadas - 1) * e
    if d_real != d:
        print(f'd={d} d_real={d_real} {((d - d_real) / (d1 + d2 + d3 + d4 + d5)*100

if (d - d_real) / (d1 + d2 + d3 + d4 + d5) > 0.1:
    raise Exception(f'(d-d_real)/h > 10%')
```

```
1 camada com 13 Ø de 25mm
Diferença d_real e d utilizado = 0.00%
```