fat_free_calculation_v01-01

February 14, 2025

1 Fat Free Simulation

1.1 Constantes (input)

[93]:
```
import math

# Número pi
pi_number = math.pi

# Modulo de Elasticidade
young_modulus_steel = 207 * (10**9)  # N/m²

# Coeficiente de expansão térmica
temperature_coefficient = 0.000017

# Coeficiente de Poisson
poisson_coefficient_steel = 0.3

# Constante Gravitacional
gravity = 9.80665 # m/s²
```

1.2 Densidades (input)

[94]:
```
# Densidade
specific_mass_water = 1027 # Kg/m³

specific_mass_steel = 7850 # Kg/m³
specific_mass_concrete = 1800 # Kg/m³
specific_mass_coating = 940 # kg/m³
specific_mass_content = 0 # Kg/m³
```

1.3 Dimensões do duto (input)

[95]:
```
# Diametro
diameter = 0.4 # metros

#Espessura do aço
```

```python
steel_thickness = 0.02 # metros

#Espessura do concreto
concrete_thickness = 0 # metros

# Espessura do revestimento
coating_thickness = 0 # metros

# Diâmetro Interno
diameter_intern = diameter - 2*steel_thickness
area_intern = pi_number * ((diameter_intern/2) ** 2)

# Diametro Externo
diameter_outer = diameter + 2*concrete_thickness + 2*coating_thickness
area_outer = pi_number * ((diameter_outer/2) ** 2)

print('Diâmetro Externo(m) :' + str(diameter_outer))
```

```
Diâmetro Externo(m) :0.4
```

1.4 Carregamentos Funcionais (input)

[96]:
```python
# Tensão Residual Efetiva
heff = 90000 # N

# Pressão interna no duto
pression_intern = 0 # bar
p = pression_intern * (10 ** 5) # 1 bar = 10^5 Pa

# Mudança de temperatura em relação à temperatura ambiente durante a instalação
delta_t = 0
```

1.5 Dados do Vão Livre (input)

[97]:
```python
# Comprimento da tubulação (em metros)
length = 40 # metros

# Profundidade da água
#Water_depth = 110 # metros

# Distância entre o duto e o leito marinho (gap)
e_gap = 0.40 # metros

print('razão L/Ds :' + str(length/diameter))
```

```
razão L/Ds :100.0
```

1.6 Dados de Revestimento (input)

[98]:
```python
# Fator de rigidez do concreto
kc = 0.25

# Resistência normalizada de compressão
fcn = 42 # Mpa
```

1.7 Massa Efetiva (valores auxiliares)

[99]:
```python
# Cálculo da massa da estrutura
diameter_outer_steel = diameter
diameter_intern_steel = diameter_intern
area_steel_pipe = pi_number * (((diameter_outer_steel/2)**2) - ((diameter_intern_steel/2)**2))
mass_steel_pipe =  specific_mass_steel * area_steel_pipe  # mass of the pipe

diameter_outer_concrete = diameter_outer - 2*coating_thickness
diameter_intern_concrete = diameter_outer_steel
area_concrete_pipe = pi_number * (((diameter_outer_concrete/2)**2) - ((diameter_intern_concrete/2)**2))
mass_concrete_pipe =  specific_mass_concrete* area_concrete_pipe  # mass of the pipe

diameter_outer_coating = diameter_outer
diameter_intern_coating = diameter_outer_concrete
area_coating_pipe = pi_number * (((diameter_outer_coating/2)**2) - ((diameter_intern_coating/2)**2))
mass_coating_pipe =  specific_mass_coating* area_coating_pipe  # mass of the pipe

mass_structure_pipe = mass_steel_pipe + mass_concrete_pipe + mass_coating_pipe

area_water = pi_number * ((diameter_outer/2) ** 2)
mass_water_displaced = specific_mass_water * area_water  #  mass of the displaced water

area_content = pi_number * ((diameter_intern/2) ** 2)
mass_content = specific_mass_content * area_content # mass of fluid


# Cálculo do coeficiente
if e_gap/diameter_outer < 0.8:
  coefficient_mass_added = 0.68 + (1.6/(1+5*(e_gap/diameter_outer)))

else:
```

```
  coefficient_mass_added = 1

# Calculo da massa adicionada
mass_added = coefficient_mass_added * mass_water_displaced

mass_effetive =  mass_structure_pipe + mass_added + mass_content

print('massa efetiva (kg/m): ' + str(mass_effetive))
```

```
massa efetiva (kg/m): 316.4840439226358
```

1.8 Rigidez (valores auxiliares)

[100]:
```
# Rigidez à flexão do aço (EI_steel), dada em N.m²
moment_inertia_steel = (pi_number/4) * (((diameter/2) ** 4) - (((diameter/2) -␣
↪steel_thickness)**4))
ei_steel = moment_inertia_steel * young_modulus_steel


print('rigidez do aço (N.m²) : ' + str(ei_steel))
```

```
rigidez do aço (N.m²) : 89456599.48355705
```

1.9 Contribuição adicional para a rigidez (valores auxiliares)

[101]:
```
# Módulo de Young do Concreto
young_modulus_concrete = 10000 * (fcn ** 0.3)
moment_inertia_concrete = (pi_number/4) * (((diameter_outer_concrete/2) ** 4) -␣
↪((diameter_intern_concrete/2)**4))
ei_concrete =  moment_inertia_concrete * young_modulus_concrete

# Contribuição da rigidez à flexão do concreto e revestimento expressa como␣
↪porcentagem de EI aço
csf = kc * ((ei_concrete/ei_steel) ** 0.75)

print('Momento de inércia do concreto(N.m²) :' + str(ei_concrete))
print('Contribuição da rigidez à flexão do concreto e revestimento expressa␣
↪como porcentagem de EI aço :' + str(csf))
```

```
Momento de inércia do concreto(N.m²) :0.0
Contribuição da rigidez à flexão do concreto e revestimento expressa como
porcentagem de EI aço :0.0
```

1.10 Força Axial Efetiva (valores auxiliares)

[102]:
```
import math

# Tensão axial aplicada durante a instalação da tubulação (N)
effective_lay_tension = heff

# Diferença de pressão interna (assumida como zero neste caso)
internal_pressure_diff = p

# Diferença de temperatura entre a instalação e a operação (assumida como zero␣
↪neste caso)
temperature_diff = delta_t

# **Cálculo da força axial efetiva**
effective_axial_force = (effective_lay_tension -  # Tensão axial de instalação
                        internal_pressure_diff * area_intern *  # Termo da␣
↪pressão interna
                        (1 - 2 * poisson_coefficient_steel) -  # Ajuste devido␣
↪ao efeito de Poisson
                        area_outer * temperature_coefficient *␣
↪temperature_diff)  # Termo da variação térmica

print('Força Axial Efetiva (N): ' + str(effective_axial_force))
```

```
Força Axial Efetiva (N): 90000.0
```

[103]:
```
# tentar encontrar o valor de lambda

area_pipe_outer = pi_number * ((diameter_outer/2) ** 2)
lambda_max = effective_axial_force / area_pipe_outer

print(lambda_max/1000000)
```

```
0.716197243913529
```

1.11 Parâmetros do solo (user_defined)

---

[104]:
```
# Effetive Lenght
ms = mass_structure_pipe + mass_content
m = mass_water_displaced

ds_per_d = ms/m

user_defined = False
```

```
if user_defined == True:
  kvs = 10 ** 5
  kv = 10 ** 5
  kl = 10 ** 5


else:

  poisson_coefficient_soil = 0.45

  cv = 600000 # boundary condition coefficient (vertical dynamic stiffness)
  cl = 500000 # boundary condition coefficient (lateral dynamic stiffness)

  # Definição dos parâmetros de rigidez do solo para argila muito mole (Clay␣
↪Very Soft)
  kvs = 75000 # N/m/m - Rigidez estática vertical do solo por unidade de␣
↪comprimento
  kv = (cv/(1-poisson_coefficient_soil))*((2/3)*(ds_per_d) + 1/3)*math.
↪sqrt(diameter_outer) # N/m/m - Rigidez dinâmica vertical do solo por unidade␣
↪de comprimento
  kl = (cl*(1+poisson_coefficient_soil))*((2/3)*(ds_per_d) + 1/3)*math.
↪sqrt(diameter_outer) # N/m/m - Rigidez dinâmica lateral do solo por unidade␣
↪de comprimento

print('Rigidez dinâmica vertical do solo: ' + str(kv))
print('Rigidez dinâmica lateral do solo: ' + str(kl))
```

```
Rigidez dinâmica vertical do solo: 897989.4426679794
Rigidez dinâmica lateral do solo: 596788.817106428
```

1.12 Carga Crítica de Flambagem (dados de resposta)

[105]:
```
import math
def effetive_length(k: int, length: int, csf: int, stiffness: float):

  if k == kv:
    print('Rigidez dinâmica vertical do solo')

  elif k == kl:
    print('Rigidez dinâmica lateral do solo')

  else:
    print('Rigidez estática vertical do solo')

  # Cálculo do logaritmo do valor adimensional que influencia o comprimento␣
↪efetivo
  value_log = k * (length ** 4) / ((1 + csf) * stiffness)
```

```python
    print('value_log: ' + str (value_log))

    beta = math.log10(value_log)  # Cálculo do logaritmo de base 10 do valor␣
 ↪obtido
    print('beta: ' + str(beta))  # Exibe o valor de beta

    # Cálculo da razão entre o comprimento efetivo e o comprimento total da␣
 ↪tubulação
    ratio_length_eff = 4.73 / ((-0.066 * (beta ** 2)) + (1.02 * beta) + 0.63)

    # Cálculo do comprimento efetivo da tubulação
    length_eff = ratio_length_eff * length

    print('Comprimento Efetivo: ' + str(length_eff))  # Exibe o comprimento␣
 ↪efetivo da tubulação

    # Coeficiente de condição de contorno
    c2 = 4

    # Cálculo da carga crítica de flambagem (Pcr)
    pcr = (1 + csf) * c2 * (pi_number ** 2) * stiffness / (length_eff ** 2)

    print('Carga Crítica de Flambagem: ' + str(pcr))  # Exibe o valor da carga␣
 ↪crítica de flambagem

    # Razão entre uma carga específica (90000 N) e a carga crítica de flambagem
    print('Razão entre Força Axial Efetiva e Carga Crítica de Flambagem: ' +␣
 ↪str(effective_axial_force / pcr))  # Exibe a relação entre a carga axial␣
 ↪efetiva e a carga crítica de flambagem
    print('')

    return length_eff, pcr

length_eff_vertical_dynamic, pcr_vertical_dynamic = effetive_length(k=kv,␣
 ↪length=length, csf=csf, stiffness=ei_steel)
length_eff_lateral_dynamic, pcr_lateral_dynamic = effetive_length(k=kl,␣
 ↪length=length, csf=csf, stiffness=ei_steel)
length_eff_vertical_static, pcr_vertical_static = effetive_length(k=kvs,␣
 ↪length=length, csf=csf, stiffness=ei_steel)

pcr = min([pcr_vertical_dynamic, pcr_lateral_dynamic, pcr_vertical_static])
```

```
Rigidez dinâmica vertical do solo
value_log: 25697.969590858167
beta: 4.409898810867445
Comprimento Efetivo: 49.21212364625961
Carga Crítica de Flambagem: 1458236.3135555368
```

```
Razão entre Força Axial Efetiva e Carga Crítica de Flambagem:
0.06171839170604522

Rigidez dinâmica lateral do solo
value_log: 17078.442290591156
beta: 4.232448256549039
Comprimento Efetivo: 50.25501474871205
Carga Crítica de Flambagem: 1398341.7112712332
Razão entre Força Axial Efetiva e Carga Crítica de Flambagem:
0.06436195049790866

Rigidez estática vertical do solo
value_log: 2146.2921808836627
beta: 3.3316888434025222
Comprimento Efetivo: 57.40791693322816
Carga Crítica de Flambagem: 1071589.7350489332
Razão entre Força Axial Efetiva e Carga Crítica de Flambagem:
0.08398736667246093
```

1.13 Deflexão Estática (dados de resposta)

[106]:
```python
# Coeficiente c6 de condição de contorno
c6 = 1/384

# Peso do tubo por unidade de comprimento
pipe_structure_weight = (mass_structure_pipe + mass_content) * gravity

# Cálculo do Empuxo
empuxo = mass_water_displaced * gravity

# Peso Submerso por Unidade de Comprimento
q = pipe_structure_weight - empuxo

# Solicitacao Axial Efetiva
seff = effective_axial_force

# Cálculo da Deflexão Estática
deflection = c6 * ((q * (length_eff_vertical_static**4))/(ei_steel * (1 +
 ↪csf))) * (1 / (1 + seff/pcr_vertical_static))
deflection_vertical_dynamic = c6 * ((q * (length_eff_vertical_dynamic**4))/
 ↪(ei_steel * (1 + csf))) * (1 / (1 + seff/pcr_vertical_dynamic))

print('Peso Submerso(N/L): ' + str(q))
print('Deflexão: ' + str(deflection))
print('Razão entre deflexão e diâmetro: '+ str(deflection/diameter_outer))
```

```
Peso Submerso(N/L): 572.4219224997441
Deflexão: 0.16696889845604465
Razão entre deflexão e diâmetro: 0.4174222461401116
```

1.14 Amplitude máxima de tensão (dados de resposta)

[107]:
```
length_eff_dic = {'vertical' : length_eff_vertical_dynamic, 'lateral' :␣
 ↪length_eff_lateral_dynamic}

print(length_eff_dic['vertical'])
```

```
49.21212364625961
```

[108]:
```
# Coeficiente C4 de condição de contorno

length_eff_dic = {'vertical' : length_eff_vertical_dynamic, 'lateral' :␣
 ↪length_eff_lateral_dynamic}
length_eff_list = [length_eff_vertical_dynamic, length_eff_lateral_dynamic]

amplitude_dic = {'vertical' : [], 'lateral': []}

for key, length_eff in length_eff_dic.items():

  c4_midspan = 8.6
  c4_shouder = 14.1 * ((length/length_eff)**2)
  c4 = [c4_midspan, c4_shouder]

  if key == 'vertical':
    name_length_eff = 'Rigidez dinâmica vertical do solo'

  elif key == 'lateral':
    name_length_eff = 'Rigidez dinâmica lateral do solo'

  for c in c4:

    if c == c4_midspan:
      name_c = 'midspan'

    else:
      name_c = 'shouder'

    # Distância à linha neutra
    middle_line = (diameter_outer - 1*steel_thickness)/2

    print(name_length_eff + ', ' + name_c +', ' + 'linha média')

    # Cálculo seguindo a fórmulda do manual da fatfree
```

```
    amplitude_max = 2 * c * (1 + csf) * diameter * young_modulus_steel *␣
 ↪middle_line / (length_eff ** 2)

    amplitude_dic[key].append(amplitude_max/1000000)


    print('Amplitude (MPa): ' + str(amplitude_max/1000000))
    print('')

print('Amplitude máx. in-line (Mpa) :' + str(max(amplitude_dic['lateral'])))
print('Amplitude máx. cross-flow (Mpa) :' + str(max(amplitude_dic['vertical'])))
```

```
Rigidez dinâmica vertical do solo, midspan, linha média
Amplitude (MPa): 111.72958139062249

Rigidez dinâmica vertical do solo, shouder, linha média
Amplitude (MPa): 121.02207716398449

Rigidez dinâmica lateral do solo, midspan, linha média
Amplitude (MPa): 107.14047688226859

Rigidez dinâmica lateral do solo, shouder, linha média
Amplitude (MPa): 111.28468724478375

Amplitude máx. in-line (Mpa) :111.28468724478375
Amplitude máx. cross-flow (Mpa) :121.02207716398449
```

1.15 Frequência Natural Fundamental (dados de resposta)

[109]:
```
# Boundary condition coefficients
c1_inline, c1_crossflow = 3.56, 3.56
c3_inline, c3_crossflow = 0, 0.4

# Massa efetiva
me = mass_effetive

# Cálculo das Frequências
f_inline = c1_inline * math.sqrt(1 + csf) * math.sqrt(ei_steel/(me *␣
 ↪(length_eff_lateral_dynamic ** 4)) * (1 + (seff/pcr_lateral_dynamic) +␣
 ↪c3_inline * (deflection/diameter_outer) ** 2))
f_crossflow = c1_crossflow * math.sqrt(1 + csf) * math.sqrt(ei_steel/(me *␣
 ↪(length_eff_vertical_dynamic ** 4)) * (1 + (seff/pcr_vertical_dynamic) +␣
 ↪c3_crossflow * (deflection/diameter_outer) ** 2))

print('Frequência na direção in-line (Hz) :'+ str(f_inline))
print('Frequência na direção cross-flow (Hz) :'+ str(f_crossflow))
```

```
Frequência na direção in-line (Hz) :0.773154239826187
Frequência na direção cross-flow (Hz) :0.831279402352052
```

2 Resumo dos resultados

[110]:

```python
print('----------------MAIN----------------')
print('STRUCTURAL MODELLING')
print('coating data')
print('kc : ' + str(kc))
print('fcn : ' + str(fcn))
#print('k[m] : ' + str())
print('')

print('Functional Loads')
print('Heff: ' + str(effective_lay_tension))
print('p[bar] : ' + str(pression_intern))
print('delta_T[ºC] : ' + str(delta_t))
print('')

print('Pipe Dimensions')
print('Ds: ' + str(diameter))
print('t_steel: ' + str(steel_thickness))
print('t_concrete: ' + str(concrete_thickness))
print('t_coating: ' + str(coating_thickness))
print('')

print('Constants')
print('v: ' + str(poisson_coefficient_steel))
print('alfa_exp_term: ' + str(temperature_coefficient))
print('E(N/m): ' + str(young_modulus_steel))
print('')

print('Densities: ')
print('d_steel: ' + str(specific_mass_steel))
print('d_concrete: ' + str(specific_mass_concrete))
print('d_coating: ' + str(specific_mass_coating))
print('d_cont: ' + str(specific_mass_content))
print('')

print('FREE SPAN SCENÁRIO')
#print('h[m] : ' + str(Water_depth))
print('L[m] : ' + str(length))
print('e[m] : ' + str(e_gap))
print('D[m] : ' + str(diameter_outer))
print('L/Ds : ' + str(length/diameter))
print('')
```

```python
print('RESPONSE DATA')
print('f1 (in-line) : ' + str(f_inline))
print('f1 (cr-flow) : ' + str(f_crossflow))
print('A1 (in-line) : ' + str(max(amplitude_dic['lateral'])))
print('A1 (cr-flow) : ' + str(max(amplitude_dic['vertical'])))
print('delta/D : ' + str(deflection/diameter_outer))
print('Seff/Pe : ' + str(effective_axial_force / pcr))
print('')

print('SOIL PROPERTIES')
print('Kv : ' + str(kv))
print('Kl : ' + str(kl))
print('Kv,s : ' + str(kvs))
print('')

print('------------------ABA------------------')
print('STRUCTURAL MODELLING INTERMEDIATE RESULTS')
print('Transfer Values')
print('EI_steel : ' + str(ei_steel))
print('me : ' + str(me))
print('q : ' + str(q))
print('Seff : ' + str(effective_axial_force))
print('Ca : ' + str(coefficient_mass_added))
print('CSF : ' + str(csf))
print('ds/d : ' + str(ds_per_d))
print('')

print('Areas [m²]')
print('Ai : ' + str(area_intern))
print('A_steel : ' + str(area_steel_pipe))
print('A_concrete : ' + str(area_concrete_pipe))
print('A_coating : ' + str(area_coating_pipe))
print('Ae : ' + str(area_outer))
```

```
----------------MAIN----------------
STRUCTURAL MODELLING
coating data
kc : 0.25
fcn : 42

Functional Loads
Heff: 90000
p[bar] : 0
delta_T[ºC] : 0

Pipe Dimensions
Ds: 0.4
```

```
t_steel: 0.02
t_concrete: 0
t_coating: 0

Constants
v: 0.3
alfa_exp_term: 1.7e-05
E(N/m): 207000000000

Densities:
d_steel: 7850
d_concrete: 1800
d_coating: 940
d_cont: 0

FREE SPAN SCENÁRIO
L[m] : 40
e[m] : 0.4
D[m] : 0.4
L/Ds : 100.0

RESPONSE DATA
f1 (in-line) : 0.773154239826187
f1 (cr-flow) : 0.831279402352052
A1 (in-line) : 111.28468724478375
A1 (cr-flow) : 121.02207716398449
delta/D : 0.4174222461401116
Seff/Pe : 0.08398736667246093

SOIL PROPERTIES
Kv : 897989.4426679794
Kl : 596788.817106428
Kv,s : 75000

------------------ABA------------------
STRUCTURAL MODELLING INTERMEDIATE RESULTS
Transfer Values
EI_steel : 89456599.48355705
me : 316.4840439226358
q : 572.4219224997441
Seff : 90000.0
Ca : 1
CSF : 0.0
ds/d : 1.4522882181110035

Areas [m²]
Ai : 0.10178760197630932
A_steel : 0.023876104167282437
```

```
A_concrete : 0.0
A_coating : 0.0
Ae : 0.12566370614359174
```

3 Exporting to PDF

[111]:
```
!sudo apt-get install texlive-xetex texlive-fonts-recommended␣
↪texlive-plain-generic
```

```
Reading package lists… Done
Building dependency tree… Done
Reading state information… Done
texlive-fonts-recommended is already the newest version (2021.20220204-1).
texlive-plain-generic is already the newest version (2021.20220204-1).
texlive-xetex is already the newest version (2021.20220204-1).
0 upgraded, 0 newly installed, 0 to remove and 20 not upgraded.
```

[112]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call
drive.mount("/content/drive", force_remount=True).
```

[113]:
```
!apt-get install pandoc
```

```
Reading package lists… Done
Building dependency tree… Done
Reading state information… Done
pandoc is already the newest version (2.9.2.1-3ubuntu2).
0 upgraded, 0 newly installed, 0 to remove and 20 not upgraded.
```

[114]:
```
!jupyter nbconvert --to pdf '/content/drive/MyDrive/fat_free_calculation_v01-02.
↪ipynb'
```

```
[NbConvertApp] Converting notebook
/content/drive/MyDrive/fat_free_calculation_v01-02.ipynb to pdf
[NbConvertApp] Writing 77914 bytes to notebook.tex
[NbConvertApp] Building PDF
[NbConvertApp] Running xelatex 3 times: ['xelatex', 'notebook.tex', '-quiet']
[NbConvertApp] Running bibtex 1 time: ['bibtex', 'notebook']
[NbConvertApp] WARNING | bibtex had problems, most likely because there were no
citations
[NbConvertApp] PDF successfully created
[NbConvertApp] Writing 75889 bytes to
/content/drive/MyDrive/fat_free_calculation_v01-02.pdf
```