

Super Códigos do Gabagol: O Mago do Concreto Estrutural I

Os códigos apresentados devem ser utilizados exclusivamente para a resolução de problemas de nFNC, nFOC e verificação de estabilidade de pilares, referentes ao conteúdo do segundo bimestre e exame de EDI-38. Cada arquivo contém uma função necessária para a execução das rotinas de cálculo, permitindo obter as respostas para os problemas propostos.

Embora os códigos disponibilizados tratem especificamente do caso nFOC, é possível resolver todos os exercícios do bimestre com eles, desde que se atente às diferenças entre as convenções de sinais adotadas para nFNC e nFOC.

As explicações abaixo não se aprofundam na teoria de Concreto Estrutural I, apenas indicam como as funções se relacionam para a resolução de cada um dos tipos de problema.

1 Tipos de Problemas

No estudo de nFNC e nFOC, temos os seguintes problemas para resolver:

1. **Cálculo dos esforços resultantes** na seção de concreto, dadas as deformações impostas.
2. **Cálculo das deformações e verificação** da seção transversal, dados os esforços de projeto aplicados. O código para resolução desses problemas utiliza o código de **Cálculo dos esforços resultantes**.
3. **Determinação da área mínima de aço** necessária para resistir aos esforços solicitantes, considerando o arranjo geométrico da seção (com todas as variáveis conhecidas, exceto o diâmetro das barras). O código para resolução desses problemas utiliza o código de **Cálculo das deformações e verificação**.

No estudo do Estado Limite Último de Serviço (ELUi), temos ainda outros tipos de problema:

4. **Verificação de Estabilidade de Pilares.** Quer-se determinar a flecha máxima a que o pilar estará sujeito e se, com essas deformações, todas as suas seções transversais estarão de acordo com o os critérios do ELU. Utiliza o código de **Cálculo de deformações e verificação**.
5. **Trajetória de equilíbrio**
6. **Pilar Padrão**
7. **Compressão Uniforme**

2 Descrição dos Arquivos

Códigos usados na resolução de problemas de nFOC:

- **Ac, Sx, Sy, Ixx, Iyy, Ixy:** Calculam propriedades geométricas da seção de concreto, necessárias no cálculo da jacobiana do concreto, e na função de translação em relação ao CG da seção transversal.
- **translacao_cg:** função que recebe as coordenadas dos vértices da seção transversal de concreto e das barras de aço e retorna as coordenadas desses mesmos pontos em relação do Centro de Gravidade da seção de concreto. Necessário implementar antes de desenhar as seções ou iniciar os algoritmos de resolução dos problemas.
- **parametros:** função que determina os parâmetros de cálculo do concreto e do aço (tais como σ_{cd} , ε_{c2} , ε_{cu} , f_{yd} etc.), a partir das classes de materiais adotadas e dos coeficientes de ponderação γ_c e γ_s . Necessária nos códigos de esforços e verificação. (ATENÇÃO: ao resolver problemas de provas antigas, lembrar de desconsiderar o η_c).

- **sigmasi, sigmac, dsigmasi, dsigmac**: calculam as tensões em função da deformação e as derivadas da tensão em função da deformação para o aço e para o concreto. Necessárias em diversos passos dos algoritmos.
- **potencial_I, potencial_J, potencial_K, tk**: funções auxiliares para construção das funções f_i .
- **funcoes_f**: funções f_i utilizadas para o cálculo dos esforços resistentes no concreto e para o cálculo da jacobiana do concreto.
- **esforços**: retorna os esforços totais na seção dadas suas propriedades geométricas, propriedades dos materiais e deformações. Retorna também os esforços apenas no concreto, pois eles são necessários no cálculo da jacobiana do concreto.
- **jacobiana_concreto, jacobiana_aco, inversor_matriz**: funções necessárias para executar o algoritmo de Newton-Raphson, durante os problemas de verificação. Elas atualizam a solução estimada a cada iteração.
- **verificacao_nFOC**: função principal para resolução de problemas de verificação. Dadas as propriedades geométricas da seção, classes dos materiais e os esforços, retorna a deformação encontrada, se a seção passou pelo algoritmo de verificação com sucesso, e se a configuração resultante respeita os limites do ELU.
- **verificaELU_nFOC**: dadas as propriedades dos materiais, propriedades geométricas da seção e deformações, a função retorna true caso os limites do ELU sejam respeitados e false caso não sejam. Usada no algoritmo de verificação.
- **dimensionamento_nFOC**: função principal para resolução de problemas de dimensionamento. Dadas as propriedades geométricas da seção, os materiais e os esforços, retorna a mínima área teórica de aço necessária para equilibrar a seção, o mínimo diâmetro comercial que deve ser usado, e as deformações na seção quando as barras de aço assumem esse mínimo diâmetro comercial.
- **d_aco**: função auxiliar que calcula o mínimo diâmetro comercial disponível que atende às condições do problema de dimensionamento.
- **diagrama_e0k e diagrama_nm**: alguns problemas de nFNC pedem o desenho do diagrama de ϵ_0 por κ e do diagrama de esforços resistentes. Esses códigos não são funções, mas tem os algoritmos necessários para gerar os diagramas, bastando trocar os dados iniciais. (ATENÇÃO: o diagrama de esforços resistentes, como está montado, apresenta convenção de sinais de nFOC, não de nFNC).

Além desses, os problemas de ELUi utilizam os seguintes códigos:

- **verifica_pilar**: função utilizada no contexto de ELUi. Determina a máxima flecha de um pilar sujeito a momentos N_d e M_d e se todas as suas seções transversais atendem aos critérios do Estado Limite Último. Utiliza o algoritmo de verificação nFOC.

Para rodar os códigos, basta criar outros arquivos na pasta que os contém, declarar os dados iniciais e chamar as funções. Os códigos para resolução da P2 de 2025 estão na pasta como exemplos.

Em geral, as provas pedem que seja elaborado um desenho da seção, com a linha neutra e barras de aço. Nos códigos utilizados para resolver diretamente as questões, há os comandos necessários para gerar esses gráficos.

3 Detalhamento dos códigos diagrama_e0k e diagrama_nm

Utilizaremos o código anexado q1b.p2_antoniogarcia.m para entender o que deve ser alterado e o que deve ser mantido no código para rodar os diagramas.

```
1 %% P2 | EDI-38
2 %% Ant nio Gouv a Garcia
3 %% Quest o 1b
4
5
6
7 close all;
8 clear;
9 clc;
10
11 %% PROPRIEDADES DOS MATERIAIS (ALTERAR)
12 classe_concreto = 80;
13 classe_aco = 50;
14 gamac = 1.4;
15 gamas = 1.15;
16 tol_k = 1e-3;
17 tol_de = 1e-3;
18
19
20 %% Parmetros dos materiais (NAO ALTERAR, A FUNCAO parametros CALCULA
    AUTOMATICAMENTE)
21 [sigmacd, epsilonc2, epsiloncu, n, fyd, epsilonyd] = parametros(classe_concreto,
    classe_aco, gamac, gamas);
22
23 %% Geometria (ALTERAR, LEMBRANDO QUE O ULTIMO ELEMENTO DOS VETORES DE COORDENADAS
    DOS VERTICES DEVE REPETIR O PRIMEIRO ELEMENTO)
24 x_vertices = [0; 0.25; 0.25; 0; 0];
25 y_vertices = [0; 0; 0.7; 0.7; 0];
26
27 n_arestas = length(x_vertices)-1;
28
29 diametro_aco = [0.025; 0.025; 0.025; 0.025; 0.025; 0.025];
30 x_aco = [0.1; 0.1; 0.1; 0.15; 0.15; 0.15]; % Por se tratar de um problema de nFNC,
    n o temos os dados de x_aco. Mas, desde que as barras estejam dispostas
    simetricamente em rela o ao eixo Y, n o h altera o no resultado final.
31 y_aco = [0.05; 0.13; 0.65; 0.05; 0.13; 0.65];
32
33 n_barras = length(x_aco);
34
35 %% NAO FAZER ALTERACOES A PARTIR DAQUI
36 [x_vertices,y_vertices, x_aco, y_aco] = translacao_cg(x_vertices, y_vertices,
    x_aco, y_aco);
37
38 ytopo = max(y_vertices);
39 ybase = min(y_vertices);
40 ysmin = min(y_aco);
41 ysmx = max(y_aco);
42 h = ytopo-ybase;
43
44 A = [0.0 epsilonc2];
45 B = [epsiloncu/h epsiloncu*ytopo/h];
46 C = [(epsiloncu+10.0)/(ytopo-ysmin) epsiloncu-ytopo*(epsiloncu+10.0)/(ytopo-ysmin)];
47 D = [0 -10.0];
48 E = [(epsiloncu+10.0)/(ybase-ysmx) epsiloncu-ybase*(epsiloncu+10.0)/(ybase-ysmx)];
49 F = [-epsiloncu/h -epsiloncu*ybase/h];
50
51 pontos = 1000;
52 incremento= 1/(pontos -1);
53 i = 1;
54 v=zeros(1000);
55 u=zeros(1000);
56 for s = 0:incremento:1
57     if s <= 1/6
58         c1(1) = A(1);
59         c1(2) = A(2);
```

```

60     ca(1) = 6.0*(B(1)-A(1));
61     ca(2) = 6.0*(B(2)-A(2));
62     k = ca(1)*s + c1(1);
63     epsilon0 = ca(2)*s + c1(2);
64     [v(i), u(i),~,~,~,~] = esforcos(epsilon0, -k, 0, n_barras, n_arestas,
        diametro_aco, x_vertices, y_vertices, x_aco, y_aco, fyd, epsilonyd, n,
        sigmacd, epsilonc2, tol_k, tol_de);
65 else
66     if s<= 2/6
67         c1(1) = 2.0*B(1)-C(1);
68         c1(2) = 2.0*B(2)-C(2);
69         ca(1) = 6.0*(C(1)-B(1));
70         ca(2) = 6.0*(C(2)-B(2));
71         k = ca(1)*s + c1(1);
72         epsilon0 = ca(2)*s + c1(2);
73         [v(i), u(i),~,~,~,~] = esforcos(epsilon0, -k, 0, n_barras, n_arestas,
            diametro_aco, x_vertices, y_vertices, x_aco, y_aco, fyd, epsilonyd, n,
            sigmacd, epsilonc2, tol_k, tol_de);
74     else
75         if s <= 3/6
76             c1(1) = 3.0*C(1)-2.0*D(1);
77             c1(2) = 3.0*C(2)-2.0*D(2);
78             ca(1) = 6.0*(D(1)-C(1));
79             ca(2) = 6.0*(D(2)-C(2));
80             k = ca(1)*s + c1(1);
81             epsilon0 = ca(2)*s + c1(2);
82             [v(i), u(i),~,~,~,~] = esforcos(epsilon0, -k, 0, n_barras,
                n_arestas, diametro_aco, x_vertices, y_vertices, x_aco, y_aco,
                fyd, epsilonyd, n, sigmacd, epsilonc2, tol_k, tol_de);
83         else
84             if s <= 4/6
85                 c1(1) = 4.0*D(1)-3.0*E(1);
86                 c1(2) = 4.0*D(2)-3.0*E(2);
87                 ca(1) = 6.0*(E(1)-D(1));
88                 ca(2) = 6.0*(E(2)-D(2));
89                 k = ca(1)*s + c1(1);
90                 epsilon0 = ca(2)*s + c1(2);
91                 [v(i), u(i),~,~,~,~] = esforcos(epsilon0, -k, 0, n_barras,
                    n_arestas, diametro_aco, x_vertices, y_vertices, x_aco,
                    y_aco, fyd, epsilonyd, n, sigmacd, epsilonc2, tol_k,
                    tol_de);
92             else
93                 if s <= 5/6
94                     c1(1) = 5.0*E(1)-4.0*F(1);
95                     c1(2) = 5.0*E(2)-4.0*F(2);
96                     ca(1) = 6.0*(F(1)-E(1));
97                     ca(2) = 6.0*(F(2)-E(2));
98                     k = ca(1)*s + c1(1);
99                     epsilon0 = ca(2)*s + c1(2);
100                     [v(i), u(i),~,~,~,~] = esforcos(epsilon0, -k, 0, n_barras,
                        n_arestas, diametro_aco, x_vertices, y_vertices, x_aco,
                        y_aco, fyd, epsilonyd, n, sigmacd, epsilonc2, tol_k,
                        tol_de);
101                 else
102                     if s <= 6/6
103                         c1(1) = 6.0*F(1)-5.0*A(1);
104                         c1(2) = 6.0*F(2)-5.0*A(2);
105                         ca(1) = 6.0*(A(1)-F(1));
106                         ca(2) = 6.0*(A(2)-F(2));
107                         k = ca(1)*s + c1(1);
108                         epsilon0 = ca(2)*s + c1(2);
109                         [v(i), u(i),~,~,~,~] = esforcos(epsilon0, -k, 0,
                            n_barras, n_arestas, diametro_aco, x_vertices,
                            y_vertices, x_aco, y_aco, fyd, epsilonyd, n,
                            sigmacd, epsilonc2, tol_k, tol_de);
110                     end
111                 end
112             end
113         end
114     end

```

```

115     end
116     i = i+1;
117 end
118 plot(v, u);
119 hold on;
120 grid on;
121 title('Diagrama de Esforços Resultantes');
122 xlabel('N (Força Normal)');
123 ylabel('M (Momento Fletor)');

```

4 Entrada e saída das funções Verificação, e Dimensionamento e Verificação de Pilares

Função dimensionamento

```

1 [At, d, e0, kx, ky] = dimensionamento_nFOC(classe_concreto, classe_aco, x_vertices,
    y_vertices, x_aco, y_aco, gamac, gamas, Nd, Mxd, Myd);

```

Saídas

- At - área teórica mínima de aço necessária para equilibrar a seção na configuração apresentada
- d - diâmetro comercial mínimo necessário para equilibrar a seção
- e0, kx, ky - deformações na seção para a configuração que utiliza o diâmetro comercial mínimo d

A própria função dimensionamento tem alguns fprintf's que revelam os resultados desejados quando ela é chamada.

Entradas

- classe_concreto - classe do concreto (se o concreto é C80, classe_concreto é 80)
- classe_aco - classe do aço (se o aço é CA50, classe_aco é 50)
- x_vertices e y_vertices - são os vetores que armazenam as coordenadas x e y dos vértices da seção, em metros. Os valores devem ser inseridos na ordem em que os vértices aparecem no contorno do polígono, garantindo que $x_vertices(i)$ e $y_vertices(i)$ representem o mesmo ponto. O último ponto deve repetir o primeiro para fechar corretamente a seção. Exemplo:

```

1 x_vertices = [0; 0.25; 0.25; 0; 0];
2 y_vertices = [0; 0; 0.7; 0.7; 0];

```

- x_aco e y_aco - coordenadas x e y das barras de aço, em metros. Devem ser feitos de forma que $x_vertices(i)$ e $y_vertices(i)$ correspondam ao mesmo vértice. Não é necessário obedecer a nenhuma ordem específica. Não se deve repetir nenhuma barra ao se declarar as suas coordenadas. Exemplo:

```

1 x_aco = [0.1; 0.1; 0.1; 0.15; 0.15; 0.15];
2 y_aco = [0.05; 0.13; 0.65; 0.05; 0.13; 0.65];

```

Em problemas de nFNC em que as coordenadas x das barras não são fornecidas, basta declarar x_aco de forma que as barras estejam simetricamente dispostas. (Foi o caso do exemplo acima!)

- gamac e gamas - coeficientes de ponderação de resistência do concreto a aço. Em geral, 1.4 e 1.15, respectivamente.
- Nd, Mxd e Myd - esforços a que a seção está sujeita. Escrever Nd em MN, Mxd e Myd em MN.m, respeitando *sempre* a convenção de nFOC para o sinal de Mxd.

Função verificação

```
1 [elu, verificacao, e0, kx, ky] = verificacao_nFOC(classe_concreto, classe_aco,  
    x_vertices, y_vertices, x_aco, y_aco, diametro_aco, gamac, gamas, Nd, Mxd, Myd)
```

Saídas

- elu - booleana que indica se o ELU foi respeitado na verificação
- verificacao - booleana que indica se o processo de iteração por Newton-Raphson foi bem sucedido
- e0, kx, ky - deformações na seção para a última iteração do algoritmo

Entradas

A única entrada diferente nesse caso é diametro_aco.

- diametro_aco - vetor que armazena o diâmetro de cada barra de aço (deve respeitar a ordem em que elas foram declaradas nos vetores x_aco e y_aco. Exemplo:

```
1 x_aco = [0.1; 0.1; 0.1; 0.15; 0.15; 0.15];  
2 y_aco = [0.05; 0.13; 0.65; 0.05; 0.13; 0.65];  
3 diametro_aco = [0.025; 0.025; 0.025; 0.025; 0.025; 0.025];
```

Função verificação de pilares

```
1 [verificacao, f, count] = verifica_pilar(classe_concreto, classe_aco, gamac, gamas,  
    Nd, Md, diametro_aco, x, y, xs, ys, m, z, prec)
```

Saídas

- verificacao - booleana que indica se o pilar é seguro
- f - máxima flecha do pilar
- count - número de iterações necessários para chegar ao resultado

Entradas

- Md - o momento deve ser colocado segundo a convenção de sinais de nFNC! Em geral, os problemas de ELUi apresentam o momento já com essa convenção de sinal.
- m - número de divisões ao longo do comprimento do pilar para verificação das seções transversais
- z - comprimento, em metros, do pilar
- prec - precisão de cálculo para o algoritmo (em geral, escolhe-se $\text{prec} = 1\text{e-}10$.)

Há um exemplo de aplicação do código e de entradas no script testepilares.m, que resolve o seguinte problema da apostila:

4.3.4 Exemplo:

Verifique a estabilidade do pilar engastado-livre ilustrado na Figura 4.11, submetido aos esforços N_d e M_d na sua extremidade livre

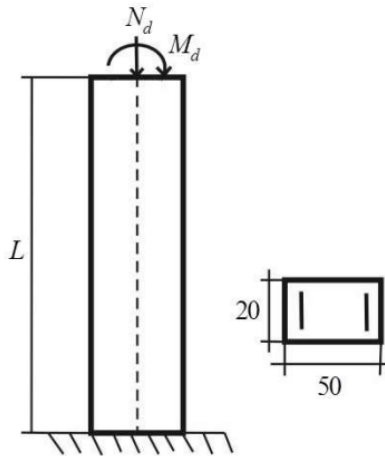


Figura 4.11 Pilar engastado-livre.

Dados:

Seção transversal: Retangular $b = 20 \text{ cm} \times h = 50 \text{ cm}$

Comprimento efetivo $l_e = 2L$

Razão conhecida $\frac{l_e}{h} = 20$

Arranjo:

$n_c = 2$

$d' = 2,5 \text{ cm}$

$A_s = 11,17 \text{ cm}^2 \text{ } (\omega = 0,4)$

Materiais:

C20

CA-50

Esforços $N_d = 364,3 \text{ kN}$ ($\mu = 0,3$) e $M_d = 6.070,0 \text{ kN.cm}$ ($\nu = 0,1$)

em que $e = M_d/N_d = 16,67 \text{ cm}$ é a excentricidade de N_d . As respostas de flecha f após a convergência para $m = 5, 10$ ou 100 divisões do pilar foram:

iterações	1	7	14
m	f	f	f
5	2,4558 cm	3,1518 cm	3,1519 cm
10	2,4536 cm	3,1471 cm	3,1473 cm
100	2,4526 cm	3,1456 cm	3,1457 cm

5 Plotagem das seções transversais

As avaliações costumam cobrar o desenho da seção transversal de concreto e de sua linha neutra após a verificação ou dimensionamento. Segue exemplo de plotagem para um problema de dimensionamento.

```
1 %% DECLARACAO DOS PARAMETROS
2
3 x_vertices = [0; 0.25; 0.25; 0; 0];
4 y_vertices = [0; 0; 0.7; 0.7; 0];
5 x_aco = [0.05; 0.05; 0.05; 0.20; 0.20; 0.20]; % Por se tratar de um problema de
   nFNC, nao temos os dados de x_aco. Mas, desde que as barras estejam dispostas
   simetricamente em relacao ao eixo Y, nao ha alteracao no resultado final.
6 y_aco = [0.05; 0.13; 0.65; 0.05; 0.13; 0.65];
7 classe_concreto = 80;
8 classe_aco = 50;
9 gamac = 1.4;
10 gamas = 1.15;
11 Nd = 7.2;
12 Mxd = 0;
13 Myd = 0;
14
15 %% CHAMADA DA FUNCAO DE DIMENSIONAMENTO E ATRIBUICAO DOS RESULTADOS
16
17 [At, d, e0, kx, ky] = dimensionamento_nFOC(classe_concreto, classe_aco, x_vertices,
   y_vertices, x_aco, y_aco, gamac, gamas, Nd, Mxd, Myd);
18 resultado = [At, d, e0, kx, ky];
19 disp(resultado);
20
21 %% REESCREVE OS DADOS DE ENTRADA EM RELACAO AO CG DA SECAO TRANSVERSAL (FUNDAMENTAL
   PARA QUE A PLOTAGEM SEJA FEITA ADEQUADAMENTE)
22 [x_vertices,y_vertices, x_aco, y_aco] = translacao_cg(x_vertices, y_vertices,
   x_aco, y_aco);
23
24 %% EFETUA A PLOTAGEM DO GRAFICO CASO O DIMENSIONAMENTO RETORNE UM VALOR VALIDO DE
   DIAMETRO COMERCIAL
25 if (~isnan(d))
26     plot(x_vertices,y_vertices,'-r');
27     xlabel('x');
28     ylabel('y');
29     title('Secao Transversal e LN');
30     hold on;
31     grid on;
32     plot(x_aco, y_aco, '.b');
33     v = 0.05; % fator para descentralizar o plot
34
35     g = @(x) (ky*x + e0) / kx; % Sabendo que kx = 0
36
37     ylim([min(y_vertices)-v, max(y_vertices)+v]);
38
39     fplot(g, [min(x_vertices)-v, max(x_vertices)+v], '--k', 'LineWidth', 1);
40 end
```

Para problemas de verificação, um algoritmo similar pode ser utilizado, mudando o trigger da plotagem para `if(elu == true)`, por exemplo.