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 ELUi, 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 exclusivamente do caso da nFOC, é possível resolver todos os exercícios do 2º bimestre com eles, desde que se atente às diferenças entre as convenções de sinais adotadas para nFNC e nFOC.

Nos exercícios de ELUi, a convenção de sinais adotada é, via de regra, a de nFNC. Atenção para isso na elaboração das funções e scripts.

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.** Determina a estabilidade de um pilar pelo algoritmo de diferenças finitas. Determina também a flecha máxima a que o pilar estará sujeito e se, com essa flecha, todas as suas seções transversais estarão de acordo com o os critérios do ELU. Utiliza o código de **Cálculo de deformações e verificação**. Os exercícios pedem também para plotar a flecha em função da altura no pilar.

5. **Trajetória de equilíbrio.** Plotagem da trajetória de equilíbrio de uma seção e determinação da carga normal crítica para uma excentricidade dada.

6. **Compressão Uniforme.** O algoritmo trabalha com pilares em compressão uniforme. (Momento nulo e excentricidade nula). Quer descobrir se o pilar falha primeiro por perda de equilíbrio (Normal crítica menor que a normal calculada para o caso $\epsilon = \epsilon_{c2}$) ou por falha de capacidade resistiva (ELU).

7. **Pilar Padrão.** Outro método para determinar a flecha máxima a que o pilar estará sujeito, o método do Pilar Padrão. Os exercícios também costumam pedir para plotar as curvas de momento resistente x curvatura e momento externo x curvatura.

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.
- **`esforcos`**: 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 Nd e Md e se todas as suas seções transversais atendem aos critérios do Estado Limite Último. Utiliza o algoritmo de verificação nFOC.
- **`plota_teq`**: função que calcula a máxima força normal que pode ser aplicada em um pilar dada uma excentricidade. Também plota a curva da trajetória de equilíbrio.
- **`compressao_uniforme.m`**: função utilizada para a situação de compressão uniforme: momento e excentricidade nulos. Tem saídas de texto que mostram qual a normal crítica para instabilidade e qual a normal máxima para a capacidade resistiva da seção (ELU).
- **`pilar_padrao.m`**: basta colocar as entradas adequadas e o código tem saídas de texto que dizem a distribuição de deformações crítica e a flecha.

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.

```
%% P2 | EDI-38
%% Ant nio Gouv a Garcia
%% Quest o 1b



close all;
clear;
clc;

%% PROPRIEDADES DOS MATERIAIS (ALTERAR)
classe_concreto = 80;
classe_aco = 50;
gamac = 1.4;
gamas = 1.15;
tol_k = 1e-3;
tol_de = 1e-3;


%% Par metros dos materiais (NAO ALTERAR, A FUNCAO parametros CALCULA
    AUTOMATICAMENTE)
[sigmacd, epsilonc2, epsiloncu, n, fyd, epsilonyd] = parametros(classe_concreto,
    classe_aco, gamac, gamas);

%% Geometria (ALTERAR, LEMBRANDO QUE O ULTIMO ELEMENTO DOS VETORES DE COORDENADAS
    DOS VERTICES DEVE REPETIR O PRIMEIRO ELEMENTO)
x_vertices = [0; 0.25; 0.25; 0; 0];
y_vertices = [0; 0; 0.7; 0.7; 0];

n_arestas = length(x_vertices)-1;

diametro_aco = [0.025; 0.025; 0.025; 0.025; 0.025; 0.025];
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.
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 ysmax = 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-ysmax) epsiloncu-ybase*(epsiloncu+10.0)/(ybase-ysmax)];
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);
```

```
                else
                    if s <= 5/6
                        c1(1) = 5.0*E(1)-4.0*F(1);
                        c1(2) = 5.0*E(2)-4.0*F(2);
                        ca(1) = 6.0*(F(1)-E(1));
                        ca(2) = 6.0*(F(2)-E(2));
                        k = ca(1)*s + c1(1);
                        epsilon0 = ca(2)*s + c1(2);
                        [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);
                    else
                        if s <= 6/6
                            c1(1) = 6.0*F(1)-5.0*A(1);
                            c1(2) = 6.0*F(2)-5.0*A(2);
                            ca(1) = 6.0*(A(1)-F(1));
                            ca(2) = 6.0*(A(2)-F(2));
                            k = ca(1)*s + c1(1);
                            epsilon0 = ca(2)*s + c1(2);
                            [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);
                        end
                    end
                end
            end
        end
    end
    i = i+1;
end
plot(v, u);
hold on;
grid on;
title('Diagrama de Esfor os Resultantes');
xlabel('N (For a Normal)');
ylabel('M (Momento Fletor)');
```

4 Entrada e saída das funções Verificação nFOC e Dimensionamento nFOC

Função dimensionamento

```
[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:

```
    x_vertices = [0; 0.25; 0.25; 0; 0];
    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:

```
    x_aco = [0.1; 0.1; 0.1; 0.15; 0.15; 0.15];
    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

```
[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:

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

5 Entrada e saída da função Verificação de Pilares, Pilar Padrão e Compressão Uniforme

Função verificação de pilares

```
[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 prec = 1e-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

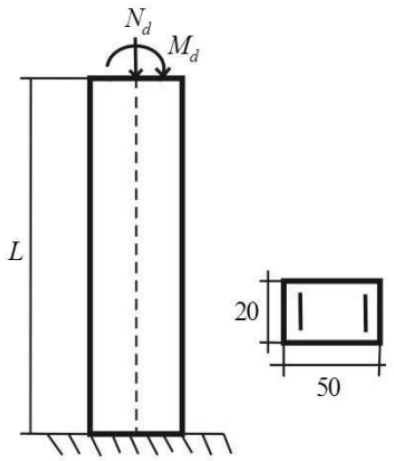

Figura 4.11 Pilar engastado-livre.

Dados:
- Seção transversal: Retangular $b = 20$ cm $\times$ $h = 50$ cm
- Comprimento efetivo $l_e = 2L$
- Razão conhecida $\frac{l_e}{h} = 20$
- Arranjo:
  - $n_c = 2$
  - $d' = 2,5$ cm
  - $As = 11,17$ cm^2 $(\omega = 0,4)$
- Materiais:
  - C20
  - CA-50
- Esforços $N_d = 364,3$ kN $(\mu = 0,3)$ e $M_d = 6.070,0$ kN.cm $(\nu = 0,1)$

em que $e = M_d/N_d = 16,67$ 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

Função pilar padrão

```
pilar_padrao(classe_concreto, classe_aco, gamac, gamas, diametro_aco, x, y, xs, ys,
    Nd, e, z)
```

Entradas

A única entrada diferente é a excentricidade e. Pode-se calculá-la por $e = \text{Md}/\text{Nd}$, antes de colocar as entradas da função no script.

Função compressão uniforme

```
compressao_uniforme(classe_concreto, classe_aco, gamac, gamas, diametro_aco, x, y,
    xs, ys, z)
```

Entradas

Todas as entradas já foram previamente comentadas e detalhadas.

Função compressão uniforme

```
plota_teq(classe_concreto, classe_aco, gamac, gamas, e, diametro_aco, x, y, xs, ys,
    m, z, prec)
```

Entradas

Todas as entradas já foram previamente comentadas e detalhadas.

6 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.

```
%% DECLARACAO DOS PARAMETROS

x_vertices = [0; 0.25; 0.25; 0; 0];
y_vertices = [0; 0; 0.7; 0.7; 0];
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.
y_aco = [0.05; 0.13; 0.65; 0.05; 0.13; 0.65];
classe_concreto = 80;
classe_aco = 50;
gamac = 1.4;
gamas = 1.15;
Nd = 7.2;
Mxd = 0;
Myd = 0;

%% CHAMADA DA FUNCAO DE DIMENSIONAMENTO E ATRIBUICAO DOS RESULTADOS

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

%% REESCREVE OS DADOS DE ENTRADA EM RELACAO AO CG DA SECAO TRANSVERSAL (FUNDAMENTAL PARA QUE A PLOTAGEM SEJA FEITA ADEQUADAMENTE)
[x_vertices,y_vertices, x_aco, y_aco] = translacao_cg(x_vertices, y_vertices, x_aco, y_aco);

%% EFETUA A PLOTAGEM DO GRAFICO CASO O DIMENSIONAMENTO RETORNE UM VALOR VALIDO DE DIAMETRO COMERCIAL
if (~isnan(d))
    plot(x_vertices,y_vertices,'-r');
    xlabel('x');
    ylabel('y');
    title('Secao Transversal e LN');
    hold on;
    grid on;
    plot(x_aco, y_aco, '.b');
    v = 0.05; % fator para descentralizar o plot

    g = @(x) (ky*x + e0) / kx; % Sabendo que kx = 0

    ylim([min(y_vertices)-v, max(y_vertices)+v]);

    fplot(g,[min(x_vertices)-v, max(x_vertices)+v],'--k','LineWidth',1);
end
```

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