

Universidade Federal Fluminense

NATALIE VON PARASKI

Análise Estática Não Linear de Pórticos Planos via  
Matlab

Volta Redonda

2012

NATALIE VON PARASKI

## Análise Estática Não Linear de Pórticos Planos via Matlab

Dissertação apresentada ao Programa de Pós-graduação em Modelagem Computacional em Ciência e Tecnologia da Universidade Federal Fluminense, como requisito parcial para obtenção do título de Mestre em Modelagem Computacional em Ciência e Tecnologia. Área de Concentração: Modelagem Computacional.

Orientador:

Alexandre da Silva Galvão

Coorientador:

Diomar Cesar Lobão

UNIVERSIDADE FEDERAL FLUMINENSE

Volta Redonda

2012

P221

Paraski, Natalie von.

Análise estática não linear de pórticos planos via  
MATLAB / Natalie von Paraski; Orientador: Alexandre da  
Silva Galvão; Coorientador: Diomar Cesar Lobão - Volta  
Redonda, 2012.  
152f.

Dissertação (Mestrado em Modelagem Computacional em Ciência  
e Tecnologia) – Universidade Federal Fluminense.

1. MATLAB. 2. Pórticos planos. 3. Estruturas esbeltas.  
4. Estabilidade. 5. Análise não linear. 6. Elementos finitos.  
7. Implementação computacional. I. Galvão, Alexandre da Silva  
(orientador); Lobão, Diomar Cesar (coorientador). II. Título.

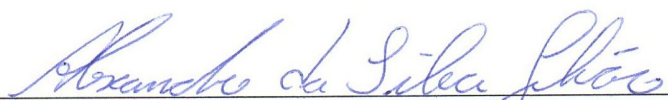
CDD 001.642

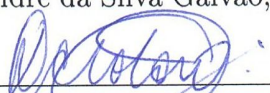
## Análise Estática Não Linear de Pórticos Planos via Matlab

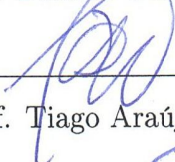
Natalie von Paraski


Dissertação apresentada ao Programa de Pós-graduação em Modelagem Computacional em Ciência e Tecnologia da Universidade Federal Fluminense, como requisito parcial para obtenção do título de Mestre em Modelagem Computacional em Ciência e Tecnologia. Área de Concentração: Modelagem Computacional.

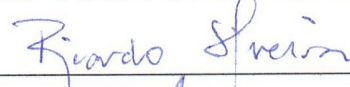
Aprovada por:

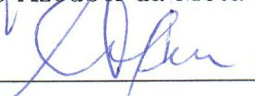
  
\_\_\_\_\_  
Prof. Alexandre da Silva Galvão, D.Sc. / MCCT-UFF (Presidente)

  
\_\_\_\_\_  
Prof. Diomar Cesar Lobão, Ph.D. / MCCT-UFF

  
\_\_\_\_\_  
Prof. Tiago Araújo Neves, D.Sc. / MCCT-UFF

  
\_\_\_\_\_  
Prof. Gustavo Benitez Alvarez, D.Sc. / MCCT-UFF

  
\_\_\_\_\_  
Prof. Ricardo Azoubel da Mota Silveira, D.Sc. / DECIV-UFOP

  
\_\_\_\_\_  
Prof. Anderson Pereira, D.Sc. / Tecgraf-PUC-Rio

Volta Redonda, 09 de Agosto de 2012.

*Dedicatória.*  
*Para Ion Vasile Vancea,*  
*que me devolveu à vida.*

## Agradecimentos

Para Deus, acima de tudo, pela saúde e pela ajuda que mais ninguém pode dar.

A minha família, em especial a minha mãe Aurora, meu pai Roberto e minha irmã Roberta por todo apoio e incentivo em todos os momentos.

Aos responsáveis por este programa de mestrado, por generosamente me aceitarem e me permitirem buscar o crescimento acadêmico.

A todos os meus amigos do mestrado que sempre estiveram ao meu lado durante os estudos e nas pouquíssimas horas de descanso no "Pombo".

Aos amigos de fora do mestrado, que me socorriam e faziam de tudo para eu descansar um pouco mais.

Aos professores que me ensinaram muito, mas principalmente a lutar e a sempre me superar.

Ao Prof. Tiago Neves não só pela gentileza de doar este *Template*, mas principalmente por me guiar tão gentilmente durante a fase final deste trabalho, que é tão importante.

Ao prof. Lobão por representar a minha esperança em permanecer no mestrado, desde o começo.

Aos professores André e Simone por abrirem a minha mente para novos e valiosos conhecimentos e me mostrarem que eu sempre posso melhorar.

Ao prof. Galvão por, pacientemente, responder (inúmeras vezes) às mesmas dúvidas, por sempre dizer a coisa certa pra me acalmar ou pra me acelerar nos estudos. Mas, principalmente, por me olhar nos olhos nas horas mais difíceis e me dizer: “eu sei que você consegue” e ser capaz de me fazer acreditar.

A todos os demais que contribuíram para a realização deste trabalho.

## Resumo

Encontram-se na literatura diversos trabalhos dedicados ao desenvolvimento de formulações de modelos de estruturas reticuladas planas que levem em consideração os diferentes comportamentos não lineares inerentes a essas estruturas. Paralelamente, há inúmeras pesquisas de metodologias de solução desses problemas. Este trabalho inicia um projeto de reestruturação, para uma linguagem de codificação mais simples, compacta e com excelentes recursos de visualização gráfica, de alguns procedimentos estruturados em linguagem Fortran no sistema computacional CS-ASA, que vem sendo desenvolvido há mais de 15 anos por um grupo de pesquisadores da Engenharia Estrutural. No presente trabalho é desenvolvido um código em linguagem Matlab com os procedimentos necessários para realizar análises estáticas de modelos estruturais compostos por elementos finitos de viga-coluna plana com não linearidade geométrica.

## Abstract

In the literature, there are several works devoted to the development of models of planar frame structures that take into account the different non-linear behavior inherent to these structures. In addition, the research of the methodology to solving the problems resulting from these properties is abundant. The aim of the present work is to start a restructuring project for a coding language which is more simple, compact and with excellent features of the graphic display of some procedures structured in Fortran and used to develop the computer system CS-ASA, project which has been developed for over 15 years by a group of researchers in Structural Engineering. In the present thesis, we present a code in Matlab with the procedures required to perform the static analysis of the structural models composed of finite elements of the beam-column which has geometric nonlinearity.



## Palavras-chave

1. Matlab
2. Pórticos Planos
3. Estruturas Esbeltas
4. Estabilidade
5. Análise Não Linear
6. Elementos Finitos
7. Implementação Computacional

## Glossário

CS-ASA	:	<i>Computational System for Advanced Structural Analysis</i>
DLDES	:	Delta L Desejado - Comprimento de Arco Desejado
GSP	:	<i>Generalized Stiffness Parameter</i>
Matlab	:	<i>Matrix Laboratory</i>
RLA	:	Referencial Lagrangiano Atualizado
RLT	:	Referencial Lagrangiano Total

# Sumário

<b>Lista de Figuras</b>	<b>xii</b>
<b>Lista de Tabelas</b>	<b>xvi</b>
<b>Lista de Símbolos</b>	<b>xvii</b>
<b>1 Introdução</b>	<b>20</b>
1.1 Considerações Gerais . . . . .	20
1.2 Objetivo e Descrição do Trabalho . . . . .	21
1.3 Revisão Bibliográfica . . . . .	23
<b>2 Problema Não Linear: Modelagem e Solução</b>	<b>26</b>
2.1 Introdução . . . . .	26
2.2 Formulação do Elemento Finito . . . . .	27
2.2.1 Referenciais Lagrangianos . . . . .	28
2.2.2 Definição do Elemento Finito . . . . .	29
2.2.3 Equações Básicas . . . . .	29
2.2.4 Discretização do Sistema Estrutural . . . . .	31
2.3 Metodologia de Solução Não Linear . . . . .	35
2.3.1 Solução Incremental Predita . . . . .	36
2.3.1.1 Matriz de Rigidez . . . . .	37
2.3.1.2 Vetor de Forças Aplicadas . . . . .	37
2.3.1.3 Vetor de Deslocamentos . . . . .	38

2.3.2	Ciclo de Iterações . . . . .	38
2.3.2.1	Vetor de Forças Internas . . . . .	40
2.3.2.2	Iteração à Carga Constante . . . . .	41
2.3.2.3	Iteração baseada no Comprimento de Arco Cilíndrico . . . . .	42
2.3.3	Estratégias de Incremento de Carga . . . . .	45
2.3.3.1	Incremento do Comprimento de Arco . . . . .	45
2.3.3.2	Sinal do Incremento Inicial do Parâmetro de Carga . . . . .	47
<b>3</b>	<b>Implementação Computacional</b>	<b>48</b>
3.1	Introdução . . . . .	48
3.1.1	Características . . . . .	49
3.2	Visão Geral do Código Computacional . . . . .	51
3.3	Entendendo a Estrutura . . . . .	53
3.4	Entrada de Dados . . . . .	55
3.4.1	Arquivo de Entrada de Dados . . . . .	56
3.4.1.1	Função LerDadosArquivo . . . . .	58
3.4.1.2	Função CriaAnima . . . . .	60
3.4.1.3	Função LimitesIni . . . . .	62
3.4.2	Definição do Vetor de Restrições . . . . .	63
3.4.2.1	Função CriaVetGrauLib . . . . .	64
3.4.3	Matriz de Graus de Liberdade . . . . .	65
3.4.3.1	Função criaMtzGrauLib . . . . .	66
3.4.4	Vetor de Cargas de Referência . . . . .	67
3.4.4.1	Função CriaVetForce . . . . .	68
3.5	Ciclo Incremental - Solução Predita . . . . .	69
3.5.1	Matriz de Rigidez do Elemento - $\mathbf{K}^e$ . . . . .	70
3.5.1.1	Função criaMtzKL . . . . .	71

3.5.1.2	Função criaMtzKt . . . . .	72
3.5.1.3	Matriz de Rotação do Elemento - $R^e$ . . . . .	73
3.5.1.4	Função criaMtzRot1 . . . . .	73
3.5.1.5	Matriz de Rigidez do Elemento no Sistema Global - $\mathbf{K}_{gl}^e$ . . . . .	74
3.5.2	Matriz de Rigidez da Estrutura - $\mathbf{K}$ . . . . .	75
3.5.2.1	Dimensão da Matriz de Rigidez da Estrutura - $\mathbf{K}$ . . . . .	75
3.5.2.2	Inserção de $K_{gl}^e$ em $K$ . . . . .	76
3.5.2.3	Função CriaMtzK . . . . .	80
3.5.3	Cálculo do Incremento de Carga . . . . .	81
3.5.3.1	Função CalcIncForce . . . . .	82
3.5.4	Vetor de Cargas Aplicadas . . . . .	83
3.5.4.1	Função ReduzV2 . . . . .	83
3.5.5	Vetor de Deslocamentos . . . . .	85
3.5.5.1	Função MontaVetDes . . . . .	86
3.6	Ciclo Incremental-Iterativo . . . . .	87
3.6.1	Atualização de Coordenadas . . . . .	88
3.6.1.1	Função atualizaCoord . . . . .	88
3.6.2	Vetor de Forças Internas . . . . .	89
3.6.2.1	Função CriaVetForceInt . . . . .	89
3.6.2.2	Função criaVetDuNat . . . . .	92
3.6.3	Verificação da Convergência - Função compara . . . . .	92
3.6.4	Atualização Iterativa . . . . .	93
3.6.4.1	Função arc . . . . .	94
3.7	Ciclo Incremental - Atualizações para o próximo incremento . . . . .	95
3.7.1	Função AtualizaVar . . . . .	95
3.7.2	Função fatCorrDLDES . . . . .	96

Sumário	xi
3.8 Pós-Processamento . . . . .	97
3.8.1 Função AnimaEst . . . . .	98
3.8.2 Função ArmazenaDados . . . . .	100
3.8.3 Função grafico2 . . . . .	100
<b>4 Validação e Resultados</b>	<b>102</b>
4.1 Exemplos Clássicos . . . . .	103
4.1.1 Viga Engastada Livre . . . . .	103
4.1.2 Coluna Engastada Livre . . . . .	106
4.2 Exemplos Fortemente Não Lineares . . . . .	107
4.2.1 Pórticos em L . . . . .	108
4.2.1.1 Pórtico de Lee . . . . .	108
4.2.1.2 Roorda Frame . . . . .	112
4.2.2 Arco Circular Birrotulado . . . . .	113
4.2.2.1 Carga Centrada . . . . .	113
4.2.2.2 Carga Excêntrica . . . . .	114
4.3 Outros exemplos . . . . .	116
4.3.1 Galpão . . . . .	116
4.3.2 Galpão Duplo . . . . .	118
4.4 Comparações . . . . .	120
<b>5 Conclusões e Sugestões de Trabalhos Futuros</b>	<b>123</b>
5.1 Introdução . . . . .	123
5.2 Conclusões . . . . .	124
5.3 Sugestões de Trabalhos Futuros . . . . .	125
<b>Referências</b>	<b>126</b>

## Lista de Figuras

2.1	Referencial Lagrangiano Total (Alves, 1995). . . . .	28
2.2	Referencial Lagrangiano Atualizado (Alves, 1995). . . . .	29
2.3	Elemento Finito de Viga Coluna. . . . .	29
2.4	Estratégia de Comprimento de Arco Cilíndrico (Crisfield, 1991) . . . . .	44
3.1	Análise Não Linear: Fluxograma. . . . .	51
3.2	Código Principal. . . . .	51
3.3	Exemplo de L-Frame. . . . .	53
3.4	Discretização do pórtico L-Frame em 20 elementos finitos. . . . .	54
3.5	Visualização do Elemento Finito. . . . .	54
3.6	Modelo do Arquivo de Entrada de Dados. . . . .	56
3.7	Arquivo de entrada de dados do L-Frame. . . . .	58
3.8	Função LerDadosArquivo. . . . .	59
3.9	Função CriaAnima. . . . .	61
3.10	Função LimitesIni. . . . .	62
3.11	Caso Geral de um Vetor de Restrições e Exemplo aplicado ao L-Frame. . .	64
3.12	Função CriaVetGrauLib. . . . .	64
3.13	Função CriaMtzGrauLib. . . . .	66
3.14	Caso Geral de um Vetor de Forças de Referência e Exemplo aplicado ao L-Frame. . . . .	68
3.15	Função CriaVetForce. . . . .	68
3.16	Função criaMtzKL. . . . .	71
3.17	Função criaMtzKt. . . . .	72

3.18	Função criaMtzRot1. . . . .	74
3.19	Elemento referenciado no Sistema Local e Global. . . . .	74
3.20	Definição da dimensão da Matriz de um elemento qualquer do Sistema Local para o Sistema Global. . . . .	76
3.21	Código de inserção de $K_{gl}^e$ em $K$ . . . . .	78
3.22	Exemplo de dimensão da Matriz nos Sistemas Local e Global. . . . .	79
3.23	Exemplo - Localização das células do Elemento 1 do L-Frame da Matriz do Sistema Local para a Matriz do Sistema Global. . . . .	79
3.24	Função CriaMtzK. . . . .	80
3.25	Função CalcIncForce. . . . .	82
3.26	Cálculo do Vetor de Forças Aplicadas. . . . .	83
3.27	Função ReduzV2. . . . .	84
3.28	Inclusão das Condições de Contorno no Vetor de Deslocamentos. . . . .	86
3.29	Função MontaVetDes. . . . .	86
3.30	Ciclo Incremental-Iterativo. . . . .	88
3.31	Função AtualizaCoord. . . . .	88
3.32	Função CriaVetForceInt. . . . .	90
3.33	Função criaVetDuNat. . . . .	92
3.34	Função compara. . . . .	93
3.35	Função arc. . . . .	94
3.36	Função AtualizaVar. . . . .	96
3.37	Função fatCorrDLDES. . . . .	97
3.38	Função AnimaEst. . . . .	98
3.39	Função ArmazenaDados. . . . .	100
3.40	Função grafico2. . . . .	101
4.1	Viga Engastada-Livre. . . . .	103
4.2	trajetória de equilíbrio (10 elem). . . . .	104



4.3	trajetória de equilíbrio com 30 incrementos. . . . .	105
4.4	Coluna Engastada-Livre. . . . .	106
4.5	Matlab: Carregamento utilizando elemento de excentricidade. . . . .	106
4.6	Trajetoária de Equilíbrio com 11 elementos. . . . .	107
4.7	Pórtico de Lee. . . . .	108
4.8	Trajetoária de equilíbrio do deslocamento axial e transversal do pórtico de Lee. . . . .	108
4.9	Trajetoária de Equilíbrio da rotação $\theta$ . . . . .	109
4.10	Pontos Limites de Carregamento e Deslocamento. . . . .	109
4.11	Deformação da Estrutura nos Pontos Limites. . . . .	110
4.12	Pórtico de Lee: Tolerância x Incrementos e Iterações Acumuladas. . . . .	111
4.13	Pórtico de Roorda. . . . .	112
4.14	Roorda Frame: trajetórias de equilíbrio. . . . .	112
4.15	Arco Circular Birrotulado. . . . .	113
4.16	Carga centrada: $P \times w$ . . . . .	113
4.17	Carga Excêntrica: $P \times w$ . . . . .	114
4.18	Carga Excêntrica: $P \times u$ . . . . .	114
4.19	Carga Excêntrica: $P \times \theta$ . . . . .	115
4.20	Estrutura tipo Galpão. . . . .	116
4.21	Galpão: Trajetória de Equilíbrio axial. . . . .	117
4.22	Galpão: Trajetória de Equilíbrio Transversal. . . . .	117
4.23	Galpão: Trajetória de Equilíbrio da Rotação. . . . .	117
4.24	Trajetoárias de Equilíbrio: $w_2xP$ . . . . .	118
4.25	Galpão Duplo. . . . .	118
4.26	Trajetoárias de equilíbrio do galpão Duplo. . . . .	119
4.27	Aproximação das Trajetórias de equilíbrio. . . . .	119

---

4.28	Trajетórias de equilíbrio do deslocamento axial e da Rotação. . . . .	119
4.29	Trajетórias de equilíbrio para o nó 11. . . . .	119
4.30	Trajетórias de equilíbrio transversal para o nó 16. . . . .	120
4.31	Pórtico de Lee: Deslocamento Axial x Incrementos. . . . .	121
4.32	Pórtico de Lee: Deslocamento Transversal x Incrementos. . . . .	121

## Lista de Tabelas

3.1	Tabela-base de Entrada de Dados. . . . .	56
3.2	Tabela de geração do Vetor de Restrições. . . . .	63
3.3	Tabela de Graus de Liberdade. . . . .	65
3.4	Exemplo de Tabela de Graus de Liberdade do L-Frame. . . . .	66
3.5	Tabela de geração do Vetor de Forças de Referência. . . . .	67
4.1	Dados pontuais: Timoshenko e Gere (1982). . . . .	104
4.2	Tempo de processamento por Incrementos e Id. . . . .	105
4.3	Dados pontuais: Southwel (1941). . . . .	107
4.4	Deslocamento Axial: Schweizerhof e Wriggers(1986). . . . .	109
4.5	Deslocamento Transversal: Schweizerhof e Wriggers(1986). . . . .	110
4.6	Pontos Limites - Galvão(2000). . . . .	110
4.7	Carregamento Centrado: Yang e Kuo (1994). . . . .	114
4.8	Carregamento Excêntrico: Yang e Kuo(1994). . . . .	115
4.9	Galpão - Carregamentos. . . . .	116
4.10	Galpão Duplo - Carregamentos. . . . .	118

## Lista de Símbolos

$\lambda$	: Parâmetro de carga responsável pelo escalonamento de $F_r$ .
$\xi$	: Tolerância ao resíduo requerida no processo de convergência.
$\delta\lambda$	: Correção do parâmetro de carga ao longo do ciclo iterativo.
$\Delta\lambda$	: Incremento do parâmetro de carga.
$\Delta\lambda^{(k-1)}$ e $\Delta\lambda^k$	: Incremento do parâmetro de carga avaliado na iteração anterior (k-1) e na iteração corrente k.
$\Delta\lambda^0$	: Incremento inicial do parâmetro de carga.
$\delta\lambda^{(k-1)}$ e $\delta\lambda^k$	: Correção do parâmetro de carga, avaliado na iteração anterior (k-1) e na iteração corrente k.
$F_i$	: Vetor de forças internas.
$F_r$	: Vetor de forças de referência.
$g$	: Vetor de forças residuais.
$H$	: Deslocamento generalizado.
$I_d$	: Número de iterações desejadas para cada incremento.
${}^tI$	: Número de iterações que foram necessárias para fazer convergir o passo de carga anterior.
$K$	: Matriz de rigidez representativa do sistema estrutural.
$\Delta l$	: Comprimento de arco da trajetória de equilíbrio.
$t$	: Última configuração de equilíbrio processada.
$t + \Delta t$	: Configuração de equilíbrio procurada no passo de carga corrente.
$u$	: Vetor de deslocamentos nodais.
$\Delta u$	: Vetor de deslocamentos nodais incrementais.
$\Delta u^{(k-1)}$ e $\Delta u^k$	: Vetor de deslocamentos nodais incrementais avaliado na iteração anterior (k-1) e na iteração corrente k.
$\Delta u^0$	: Incremento inicial dos deslocamentos nodais.
$\delta u$	: Vetor de deslocamentos residuais.
$\delta u_g$	: Parcela de $\delta u$ referente às forças residuais $g$ .
$\delta u_r$	: Parcela de $\delta u$ referente às forças de referência $F_r$ .
$\Pi$	: Energia potencial total.
$\Psi$	: Rotação de corpo rígido (total ou incremental).
$\Delta\theta$	: Rotação incremental de um ponto qualquer do elemento.

$\Delta\Pi$	: Incremento da energia potencial total.
$\Delta\sigma$	: Incremento de tensão axial.
$\Delta\varepsilon_{xx}$	: Incremento de deformação axial.
$\Delta e_{xx}$	: Parcela linear de $\Delta\varepsilon_{xx}$ .
$\Delta\eta_{xx}$	: Parcela de $\Delta\varepsilon_{xx}$ que contém os termos quadráticos.
${}^t\varepsilon$	: Deformação axial na configuração de equilíbrio $t$ .
$\delta, \phi_1$ e $\phi_2$	: Deslocamentos naturais (totais ou incrementais).
$A$	: Área da seção transversal do elemento.
$\Delta d$	: Vetor de deslocamentos incrementais.
$E$	: Módulo de elasticidade do material que compõe o elemento.
$EI$	: Rigidez à flexão da viga.
${}^{t+\Delta t}F_i$	: Vetor de forças internas calculadas na iteração corrente.
$\Delta^t F_i$	: Incremento do vetor de forças internas.
${}^t F_i$	: Vetor de forças internas calculadas na configuração de equilíbrio $t$ .
$H$	: Matriz que contém as funções de interpolação que relacionam os deslocamentos incrementais $\Delta d$ com os deslocamentos nodais incrementais $\Delta u$ .
$K^e$	: Matriz de rigidez elementar no sistema local de coordenadas.
$K_{gl}^e$	: Matriz de rigidez elementar no Sistema global de coordenadas.
$K$	: Matriz de rigidez global do sistema estrutural.
$K_\tau^e$	: Matriz de rigidez geométrica.
$K_L^e$	: Matriz de rigidez linear.
$L$	: Comprimento inicial do elemento.
${}^t L$	: Comprimento do elemento na configuração de equilíbrio $t$ .
${}^{t+\Delta t} L$	: Comprimento do elemento na configuração corrente $t + \Delta t$ .
${}^t M$	: Momento fletor na configuração de equilíbrio $t$ .
$M_1$ e $M_2$	: Momentos nodais na configuração de equilíbrio $t$ .
${}^t P$	: Força axial na configuração de equilíbrio $t$ .
${}^t Q$	: Esforço cortante na configuração de equilíbrio $t$ .
$\Delta u_n$	: Vetor de deslocamentos naturais incrementais.
$\Delta U$	: Incremento da energia interna de deformação.
$\Delta V$	: Incremento da energia potencial das forças externas.
$\Delta u$ e $\Delta v$	: Deslocamentos incrementais de um ponto qualquer do elemento, na direção dos eixos $x$ e $y$ respectivamente.
$U$	: Energia interna de deformação.
$U_\tau$	: Parcela de $\Delta U$ que corresponde à influência das deformações iniciais.

- $V$  : Energia potencial das forças externas.
- $\Delta^t \tau_{xy}$  : Incremento de tensão cisalhante.
- $\Delta \varepsilon_{xy}$  : Incremento de deformação cisalhante.
- $\Delta e_{xy}$  : Parcela linear de  $\Delta \varepsilon_{xy}$ .
- $\Delta \eta_{xy}$  : Parcela de  $\Delta \varepsilon_{xy}$  que contém os termos quadráticos.
- $R^e$  : Matriz de rotação do elemento.
- ${}^t R_a$  : Matriz de rotação entre o sistema global de referências e o sistema local atualizado na última iteração processada, t.
- ${}^t R^e$  : Matriz de rotação do elemento calculada na última configuração de equilíbrio computada, t.

# Capítulo 1

## Introdução

### 1.1 Considerações Gerais

Vários pesquisadores têm se empenhado e direcionado suas pesquisas para o desenvolvimento de metodologias práticas e eficientes para uma análise não linear de sistemas estruturais. Estas pesquisas deram origem a códigos estruturados em Fortran que foram compilados e unidos em um grande sistema computacional que, apesar de ser uma ferramenta poderosa na análise não linear de estabilidade de estruturas esbeltas, é um código de difícil compreensão, utilização e extensão para usuários e novos desenvolvedores. Em contrapartida, os constantes avanços tecnológicos fazem com que se torne cada vez mais necessário o desenvolvimento de soluções computacionais mais adequadas e eficientes quanto ao custo computacional, complexidade de código, e interação com o usuário.

Para que o desenvolvimento dessa ferramenta computacional se torne viável, o conhecimento das metodologias de solução, bem como dos detalhes da implementação computacional de todas as etapas do processo de análise estrutural, se torna de fundamental importância para a solução dos problemas envolvidos neste tipo de análise, pois somente através da compreensão do código será possível aos engenheiros e profissionais da área realizar as manipulações mais relevantes.

A análise da estabilidade de sistemas estruturais esbeltos através do Método dos Elementos Finitos (MEF) envolve invariavelmente a solução de um sistema de equações algébricas não lineares. Como relatado no artigo de Riks (1979) e também destacado em Silveira (1995), as técnicas baseadas em relações de rigidez tangente e os esquemas que combinam procedimentos incrementais e iterativos, são atualmente considerados os mais eficientes e, portanto, serão utilizadas no presente trabalho.

Muitos pesquisadores têm desenvolvido formulações geometricamente não lineares para elementos finitos (Alves, 1995; Crisfield, 1991; Yang e Kuo, 1994; Pacoste e Eriksson, 1997; Torkamani et al., 1997; Neuenhofen e Fillippou, 1998), que têm sido adequadamente empregadas na modelagem de vários sistemas estruturais esbeltos. Essas formulações permitem a determinação da matriz de rigidez e do vetor de forças internas de forma direta e podem ser acopladas com relativa facilidade às várias estratégias de solução não linear. Recentemente, muitos pesquisadores têm se esforçado para aprimorar as técnicas utilizadas nas formulações teóricas (Lavall et al., 2011; Silva e Silva, 2011) assim como as técnicas computacionais utilizadas em metodologias numéricas sofisticadas (Brito et al., 2011; Coelho et al., 2011; Leon et al., 2011).

Essas considerações motivaram a adoção deste tema para a presente dissertação de mestrado.

## 1.2 Objetivo e Descrição do Trabalho

Este trabalho tem o objetivo de iniciar o projeto de reestruturação, para a linguagem Matlab, do sistema computacional de análise avançada de estruturas CS-ASA (Silva, 2009), que vem sendo desenvolvido há mais de 15 anos por um grupo que envolve pesquisadores da Engenharia Estrutural. Dessa forma, será desenvolvido um código em linguagem Matlab que possibilitará a realização de análises de estruturas aperticadas planas com não linearidade geométrica. O objetivo deste trabalho consiste na criação de um código que seja uma base computacional didática voltada para a análise não linear estática de pórticos planos, além de servir para futuras pesquisas envolvendo análise não linear de estruturas e, posteriormente, para implementação de análises mais avançadas.

O presente trabalho é parte integrante das seguintes sublinhas de pesquisa da grande linha de investigação do programa de pós-graduação (*strictu sensu*) em Modelagem Computacional em Ciência e Tecnologia (MCCT/EEIMVR/UFF):

- **Métodos Matemáticos e Computacionais Aplicados à Engenharia e Ciência:**

*Métodos Numéricos para Equações em Derivadas Parciais:*

Tem como objetivo a aplicação de métodos numéricos, como o método dos elementos finitos (MEF), na determinação de respostas de sistemas de engenharia;

*Análise de Sistemas Estruturais:*



Objetiva o estudo do equilíbrio e estabilidade de elementos estruturais submetidos a carregamentos diversos.

Em uma visão mais específica, o principal objetivo deste trabalho é o estudo e implementação computacional de formulações geometricamente não lineares, para elementos finitos reticulados planos, em ambiente Matlab.

Na Seção 1.3, é feita uma revisão bibliográfica onde atenção especial é dada aos trabalhos que tratam diretamente de formulações geometricamente não lineares e implementações de soluções computacionais desenvolvidas em ambiente de programação Matlab.

No Capítulo ?? é realizada uma explanação geral sobre o sistema computacional desenvolvido, apresentando de maneira resumida suas características assim como um breve histórico do código computacional utilizado na comparação de resultados para a validação da análise.

O Capítulo 2 aborda o desenvolvimento teórico das formulações de elementos finitos não lineares. Ainda neste Capítulo é apresentada a metodologia de solução não linear adotada, assim como as estratégias iterativas e de incremento de carga adotadas neste trabalho.

O Capítulo 3 apresenta, de forma detalhada, a implementação computacional desta análise, realizando uma conexão solidamente estruturada entre as teorias estudadas e suas aplicações no âmbito computacional.

No Capítulo 4 são analisados exemplos de problemas estruturais encontrados na literatura, utilizados na validação da análise realizada pelo código desenvolvido. São utilizados neste capítulo alguns exemplos clássicos de solução analítica e alguns exemplos fortemente não lineares que possuem soluções numéricas obtidas por diversos pesquisadores. Neste mesmo capítulo serão realizadas também algumas comparações de desempenho computacional entre o código desenvolvido em ambiente Matlab e o CS-ASA (Silva, 2009), encontrado na literatura.

Finalmente, o Capítulo 5 traz as conclusões sobre as análises realizadas e também sobre o código desenvolvido. São fornecidas também algumas sugestões para o desenvolvimento de trabalhos futuros, tendo em vista a continuidade desta pesquisa.

## 1.3 Revisão Bibliográfica

Nas últimas décadas as técnicas de análise de estruturas geometricamente não lineares têm tido grande interesse por parte dos pesquisadores. Uma atenção particular tem sido direcionada ao desenvolvimento de formulações de elementos finitos reticulados planos que, além de possibilitar uma análise rápida e eficaz de muitos sistemas estruturais reais, possibilita o emprego direto de estratégias de solução não lineares que, posteriormente, podem ser adaptadas para outros tipos de elementos.

Formulações em referenciais Lagrangianos totais e atualizados (RLT e RLA) têm sido apresentadas por vários pesquisadores, dos quais pode-se citar: Wen et al. (1983); Chajes et al. (1987); Goto et al. (1987); Wong e Loi (1990); Alves (1995) e Torkamani et al. (1997). Yang e Kuo (1994) sugeriram uma forma incremental de se calcular o vetor de forças internas com duas abordagens diferentes para os deslocamentos nodais: *deslocamentos naturais incrementais e rigidez externa*. Pacoste e Eriksson (1995 e 1997) introduziram formulações em RLT baseadas em relações deformação-deslocamento denominadas *relações melhoradas*, com a não linearidade expressa por funções trigonométricas.

Essas formulações foram integradas à metodologia de solução numérica implementada por Silveira (1995) e expandida por Galvão (2000), que realizou um estudo comparativo de várias formulações lagrangianas totais, atualizadas e corrotacionais, onde podem ser observados seus aspectos positivos, assim como aspectos negativos, no contexto da análise não linear de estruturas. Este estudo foi complementado por Pinheiro (2003), que estudou uma estratégia de solução não linear para pórticos planos com ligações semirrígidas. Instabilidades estáticas e dinâmicas foram estudadas por Galvão (2004). Machado (2005) implementou formulações não lineares considerando o efeito da inelasticidade do aço em pórticos planos com ligações rígidas para, em seguida, Rocha (2006) e Santos (2007) considerarem em um único elemento finito de pórtico plano os efeitos não lineares, possibilitando a análise inelástica de segunda ordem em estruturas metálicas com ligações semi rígidas. Todos esses códigos foram unificados por Silva (2009) em um só código computacional, o CS-ASA.

Nas últimas décadas, os avanços tecnológicos e as exigências do mercado de engenharia, que introduziram maior complexidade e eficiência aos cálculos estruturais, levaram os pesquisadores a procurar metodologias de solução que ao mesmo tempo produzissem resultados precisos e fossem de rápido processamento. Juntamente com as pesquisas relativas ao desenvolvimento de formulações não lineares, muitos trabalhos têm sido produzidos

com a finalidade de se determinar a melhor estratégia de solução não linear. Os métodos que têm mostrado maior eficiência são os que combinam procedimentos incrementais e iterativos. Como trabalhos pioneiros podem ser citados os desenvolvidos por: Argyris (1964), com a aplicação de um método incremental para solução não linear; Mallet e Margal (1968), que utilizaram iterações do tipo Newton para contornarem os possíveis erros nas aproximações incrementais; Zienkiewicz (1971), que apresentou uma modificação no método de Newton-Raphson, fazendo com que a matriz de rigidez só fosse atualizada a cada passo de carga.

Diversos trabalhos têm sido publicados apresentando diferentes estratégias de controle automático do processo incremental, bem como diferentes estratégias de iteração. Utilizando um parâmetro de rigidez corrente como indicador do grau de não linearidade do sistema, Bergan et al. (1978) e Bergan (1980) suprimiram as iterações de equilíbrio nas zonas críticas da trajetória, até os pontos limites serem atravessados; os trabalhos de Bergan et al. (1978) e Heijer e Rheinbold (1981) forneceram diferentes estratégias de incremento de carga.

Batoz e Dhatt (1979) apresentaram uma técnica na qual o ciclo iterativo é realizado não à carga constante, mas a deslocamento constante, o que permite se obter os pontos limites de carga mas não os de deslocamento; Riks (1979) apresentou um método, baseado no parâmetro comprimento de arco  $\Delta l$ , capaz de calcular pontos limites de carga e de deslocamento com a introdução de um parâmetro que controla o progresso dos cálculos ao longo do caminho de equilíbrio; Meek e Tan (1984) apresentaram um resumo das principais técnicas para se ultrapassar os pontos limites, das quais a técnica do comprimento de arco foi reconhecida como uma das mais eficientes. Contribuíram com essa técnica: Riks (1972 e 1979), Ramm (1981), Schweizerhof e Wriggers (1986), e Crisfield (1981, 1991 e 1997). Yang e Kuo (1994) introduziram estratégias de incremento de carga e iteração baseadas em relações de restrição para as quais é definido um parâmetro generalizado, Krenk (1993 e 1995) elaborou uma nova estratégia de iteração, introduzindo duas condições de ortogonalidade: a primeira entre o vetor de cargas residuais e o incremento de deslocamento e a outra entre o incremento de forças internas e o vetor de deslocamentos iterativos.

Crisfield (1997) introduziu procedimentos numéricos que permitem avaliar com precisão os pontos críticos existentes, e obter as trajetórias de equilíbrio secundárias.

Silveira et al. (1999) apresentaram uma metodologia geral de solução de sistemas de equações algébricas não lineares que, num contexto computacional pode ser resumida em

duas partes: (i) a partir de uma dada configuração de equilíbrio da estrutura é calculada uma solução incremental inicial; (ii) em seguida, essa solução é corrigida com iterações do tipo Newton até ser atingida a nova configuração de equilíbrio. Nesses dois trabalhos diversas estratégias de iteração e incremento de carga foram testadas. Utilizando a mesma metodologia, Rocha (2000) realizou um estudo comparativo de diversas estratégias de iteração e incremento de carga através da análise de vários exemplos numéricos de sistemas estruturais.

Esforços vêm sendo aplicados por vários pesquisadores no sentido de se desenvolver e aperfeiçoar códigos de análise de estruturas, tanto no sentido de aprimoramento da tecnologia como na padronização das implementações das análises não lineares, assim como na otimização de topologias. Em adição ao popular código *99 line* (Sigmund 2001) e do seu sucessor (Andreassen et al. 2011), Allaire e Pantz (2006) apresentou um código de otimização estrutural baseado FreeFem++. Liu et al. (2005) introduziu um método *Coupled Level Set* usando o pacote FEMLAB. Challis (2010) apresentou um código *Discrete Level Set* em Matlab muito no espírito do código *99 line*. Mais recentemente, Suresh (2010) desenvolveu um código de 199 linhas para o traçado de *Pareto-optimal* com o auxílio de derivativas topológicas. Mais especificamente, no ramo da análise estrutural, Aranha Jr. et al. (2003) apresenta um código computacional para a formulação de um elemento finito de barra para análise estática e dinâmica geometricamente não linear de pórticos planos e estruturas formadas por cabos. Queiros (2007) apresenta uma interface desenvolvida em Matlab para a realização de análises de galpões pré-moldados em concreto considerando a influência da rigidez nas ligações viga-pilar. Carvalho (2010) desenvolveu o algoritmo PEFNL-2D visando o estudo de análise de vigas com elementos finitos através de uma formulação corrotacional. Leon et al. (2011), desenvolveu uma biblioteca em linguagem C++ utilizando-se dos conceitos de orientação a objeto, com o objetivo de simplificar e unificar os esquemas de solução das equações não lineares. Maciel (2012), criou um novo módulo a ser utilizado no CS-ASA para a realização de estudos sobre o equilíbrio e a estabilidade de elementos estruturais com restrições bilaterais de contato impostas por bases elásticas. Visando facilitar a entrada dos dados referentes às informações de estruturas a serem analisadas pelo CS-ASA, Prado (2012) desenvolveu um sistema gráfico interativo de pré-processamento voltado para a análise avançada de estruturas. Dentro desse contexto, este trabalho pretende criar um código que se utilize de uma linguagem de programação mais simplificada, aproximando a mesma dos estudos de análise de estruturas de dados, facilitando a compreensão do problema e a observação das aplicações das técnicas pelos usuários e novos desenvolvedores.

## Capítulo 2

# Problema Não Linear: Modelagem e Solução

### 2.1 Introdução

A análise linear das estruturas, apesar de ser muito útil em grande parte dos problemas práticos da engenharia estrutural, possui a desvantagem de não simular o comportamento realista da estrutura sob condições não usuais de carregamento, ou mesmo de não obter com precisão os carregamentos limites de estruturas cujo comportamento é não linear mesmo para carregamentos inferiores às cargas críticas (Galvão, 2000).

Em contrapartida, o fato de a natureza do problema não ser simples resulta em um aumento considerável do custo computacional da implementação da solução. Conforme a resposta do modelo estrutural pode-se adotar variadas técnicas de análise, cada qual com suas respectivas estratégias de refinamento. Deve-se adotar a estratégia mais apropriada a cada tipo de estrutura. Neste trabalho serão adotadas estratégias eficazes na aplicação da maioria dos casos estruturais.

No que concerne ao comportamento não linear de uma estrutura, duas principais fontes de não linearidade podem ser destacadas: a física e a geométrica. A não linearidade física ocorre quando o material não apresenta uma relação tensão-deformação linear. Esse efeito não linear não será abordado neste trabalho.

Uma Estrutura, porém, pode se comportar de modo não linear, ainda que seja composta por materiais que obedeçam à lei de Hooke. Quando há grandes deslocamentos, mas pequenas deformações, como em geral ocorre com as estruturas esbeltas, pode ser observada a não linearidade geométrica. Quando há grandes deflexões laterais de um membro da estrutura podem haver, como consequência, momentos fletores adicionais, em

virtude de um esforço normal. Esses efeitos, também conhecidos como de segunda ordem, devem ser considerados no âmbito global e local (elemento finito). Eles compõem uma importante fonte de não linearidade no problema estrutural, exigindo formulações numéricas específicas para a solução do problema.

Este capítulo se propõe a apresentar uma metodologia voltada para a análise estática não linear de pórticos planos. A Seção 2.2 apresenta a formulação do elemento finito decorrente do processo de discretização de um sistema estrutural em elementos finitos de pórtico plano. Cada um desses elementos será responsável pela simulação da não linearidade geométrica, onde grandes deslocamentos e pequenas deformações serão considerados.

A Seção 2.3 refere-se à apresentação da metodologia utilizada na solução do problema não linear, incluindo as estratégias de incremento de carga e de iteração que serão empregados nas análises das estruturas abordadas neste trabalho.

## 2.2 Formulação do Elemento Finito

No contexto da análise estrutural, o método mais utilizado para a discretização de um problema contínuo e posterior obtenção de soluções numéricas é o Método dos Elementos Finitos, devido à sua eficiência e aplicabilidade (Galvão, 2000). Como dito anteriormente, essa técnica visa discretizar, ou seja, dividir o meio contínuo em subdomínios, chamados elementos, interligados por pontos nodais, onde os graus de liberdade são definidos.

A precisão deste método se encontra diretamente ligada às condições de convergência e ao refinamento da malha, que em condições limites tenderiam à obtenção da solução exata do problema. Em outras palavras, quanto maior o número de pontos na malha, maior o número de elementos, ou seja, mais refinada estará a malha e, portanto, mais próxima da solução analítica do problema será a resposta obtida. No entanto, deve-se ressaltar que a utilização de uma quantidade de elementos a gerar uma solução satisfatória dentro da precisão desejada e do tempo esperado refletem automaticamente em uma considerável redução no custo computacional, sem prejuízos referentes à confiabilidade dos resultados, pois após certo grau de precisão, as alterações na resposta obtida assumem valores muito pouco significativos (Galvão, 2000).

### 2.2.1 Referenciais Lagrangianos

As formulações Euleriana e Lagrangiana são propostas para a descrição do movimento de corpos sólidos. Na formulação Euleriana as coordenadas espaciais (referentes ao corpo deformado) são definidas como as coordenadas de referência do sistema, enquanto que, na formulação Lagrangiana, adotada neste trabalho, os deslocamentos decorrentes de um carregamento dado são medidos em relação à configuração inicial do sistema. A maioria das formulações de elementos finitos para análise de segunda ordem de estruturas encontradas na literatura é baseada em referenciais Lagrangianos.

A formulação Lagrangiana pode ser dividida em dois referenciais: o Referencial Lagrangiano Total (RLT) e o Referencial Lagrangiano Atualizado (RLA).

No RLT os deslocamentos são medidos em relação à configuração inicial indeformada, como mostra o esquema da Figura 2.1.

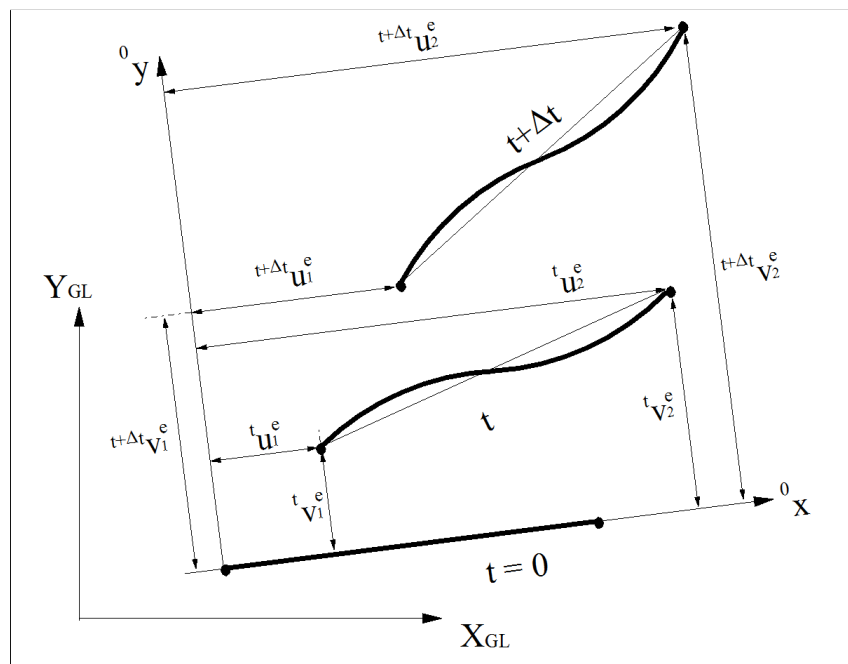


Figura 2.1: Referencial Lagrangiano Total (Alves, 1995).

Pode-se observar, na Figura 2.1, que as variáveis  $X_{GL}$  e  $Y_{GL}$  representam os eixos axiais e transversais globais, onde toda a estrutura é definida. Os eixos  $^0x$  e  $^0y$  referem-se a configuração local inicial ( $t = 0$ ) do elemento, que servirá de referência para todas as configurações de equilíbrio subsequentes ( $t, t + \Delta t$ , conseqüentemente). Pode-se observar, também, que, para cada nova configuração da estrutura, são obtidos novos deslocamentos axiais e transversais nos nós iniciais ( $\Delta u_1^e$  e  $\Delta v_1^e$ ) e finais ( $\Delta u_2^e$  e  $\Delta v_2^e$ ) do elemento.

Alves (1995) mostrou que, devido aos eventuais deslocamentos de corpo rígido ocorridos durante o processo incremental, cujas influências não são perfeitamente consideradas, bem como devido à utilização de funções de interpolação simplificadas, a tendência é que os resultados obtidos em RLT se afastem do comportamento real à medida que a configuração deformada se distancia da configuração original.

No RLA (abordagem adotada neste trabalho) os deslocamentos são medidos em relação à última configuração de equilíbrio obtida no processo incremental, ou seja, em relação a um referencial que é atualizado a cada incremento de carga, conforme ilustrado na Figura 2.2. Nesse caso as rotações de corpo rígido são divididas em partes menores e podem ser melhor aproximadas pelas funções de interpolação. (Alves, 1995)

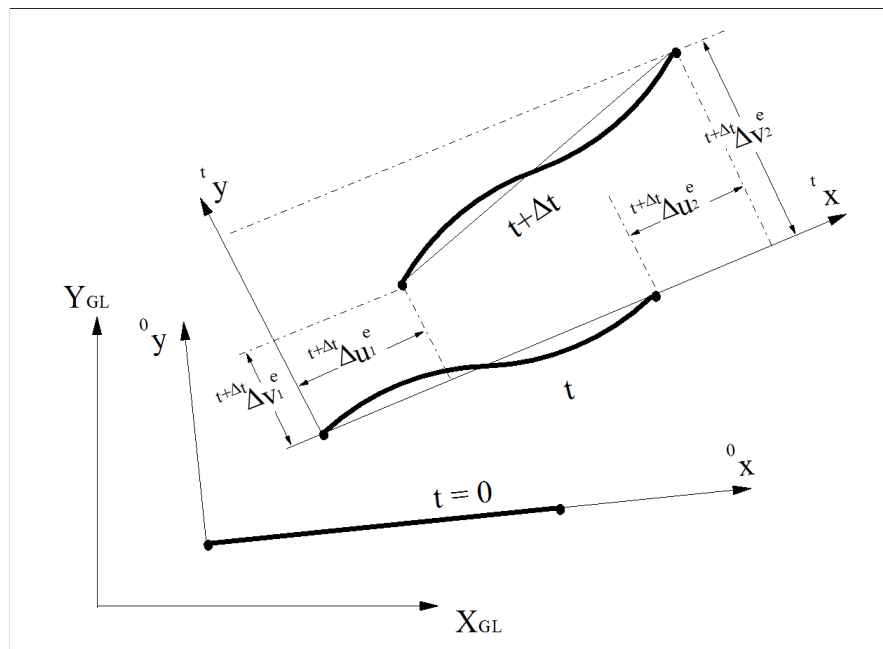


Figura 2.2: Referencial Lagrangiano Atualizado (Alves, 1995).

Pode-se observar, na Figura 2.2, que as variáveis  $X_{GL}$  e  $Y_{GL}$  também representam os eixos axiais e transversais globais, onde a estrutura é definida. Os eixos  $^0x$  e  $^0y$  referem-se a configuração local inicial ( $t = 0$ ) do elemento, que servirá de referência para a configuração seguinte ( $t$ ) e esta servirá de referência para a próxima configuração ( $t + \Delta t$ ). Pode-se observar, também, que, para cada nova configuração da estrutura, são obtidos novos deslocamentos axiais e transversais nos nós iniciais ( $\Delta u_1^e$  e  $\Delta v_1^e$ ) e finais ( $\Delta u_2^e$  e  $\Delta v_2^e$ ) do elemento, sendo os mesmos referenciados pela configuração de equilíbrio anterior.



### 2.2.2 Definição do Elemento Finito

O elemento finito a ser utilizado no presente trabalho é o elemento de pórtico plano com seis graus de liberdade, como mostra a Figura 2.3 . Definido o modelo da estrutura, a mesma deverá ser discretizada em elementos finitos, ou seja, dividida em múltiplas partes menores que serão chamadas de elementos, que possuem nós em suas extremidades definindo o seu início e o seu final.

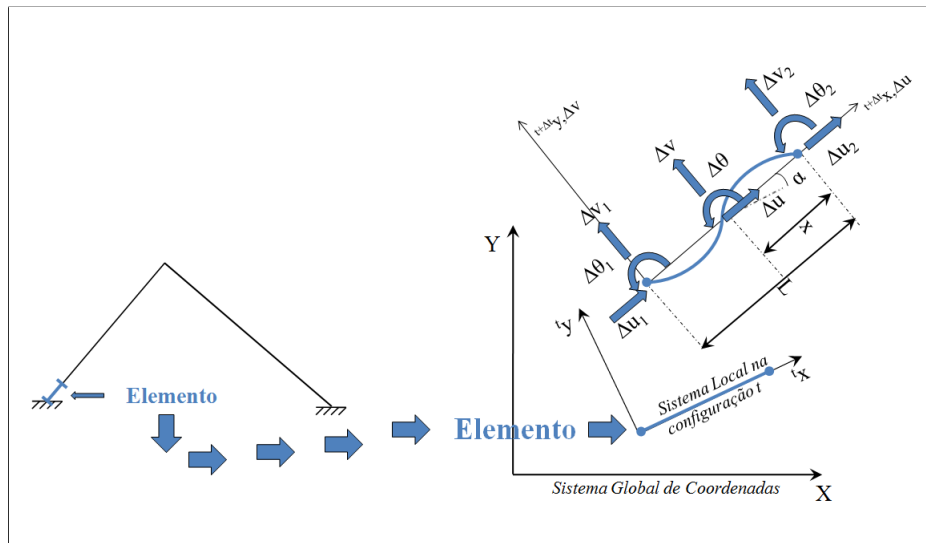


Figura 2.3: Elemento Finito de Viga Coluna.

### 2.2.3 Equações Básicas

A Lei de Hooke, representada pela equação (2.1), diz que, no caso utópico de um sistema perfeitamente elástico (cuja deformação desapareça com a retirada das forças que a originaram) e linear, as forças deformantes,  $\Delta F$ , são proporcionais às deformações elásticas,  $\Delta U$ , produzidas, de acordo com um parâmetro de rigidez constante  $K$ .

$$\Delta F = K \Delta U \tag{2.1}$$

Na prática, entretanto, verifica-se que o parâmetro de rigidez varia com os deslocamentos. Um modelo mais realista deve, portanto, levar em conta as relações não lineares entre as forças aplicadas e os deslocamentos estruturais. O equacionamento a seguir descreve a formulação de um elemento finito de pórtico plano (Galvão, 2000).

O Princípio da Energia Total Estacionária estabelece que entre todas as configurações

admissíveis de um sistema conservativo, aquelas que satisfazem as condições de equilíbrio tornam a energia potencial estacionária (Cook et al., 1989). Podemos então, definir que a energia potencial total do sistema,  $\Pi$ , consiste da energia interna de deformação elástica  $\Delta U$  e do potencial das cargas externas  $\Delta V$  ou seja,

$$\Pi = \Delta U + \Delta V \quad (2.2)$$

A energia armazenada na estrutura  $\Delta U$  para mover-se de uma configuração de equilíbrio  $t$  para uma secundária  $t + \Delta t$ , no Referencial Lagrangiano Atualizado, pode ser escrita na seguinte forma,

$$\Delta U = \iint_{Vol} ({}^t\tau_{xx}\Delta\varepsilon_{xx} + 2{}^t\tau_{xy}\Delta\varepsilon_{xy}) dAdx + \iint_{Vol} \left( \frac{E}{2}\Delta e_{xx}^2 \right) dAdx \quad (2.3)$$

onde  $\tau$  representa as componentes do tensor de Cauchy e  $\Delta\varepsilon$  representa as deformações axiais e transversais do tensor de Green-Lagrange.

A parcela  $\Delta\varepsilon_{xx}$ , referente às deformações axiais é representada por,

$$\Delta\varepsilon_{xx} = \Delta e_{xx} + \Delta\eta_{xx} \quad (2.4)$$

onde  $\Delta e_{xx}$  representa a parcela linear, descrita em 2.5

$$\Delta e_{xx} = \frac{d\Delta u}{dx} - y \frac{d^2\Delta v}{dx^2} \quad (2.5)$$

e  $\Delta\eta_{xx}$  representa a parcela não linear da equação 2.6,

$$\Delta\eta_{xx} = \frac{1}{2} \left[ \left( \frac{d\Delta u}{dx} \right)^2 - 2y \frac{d\Delta u}{dx} \frac{d^2\Delta v}{dx^2} + y^2 \left( \frac{d^2\Delta v}{dx^2} \right) + \left( \frac{d\Delta v}{dx} \right) \right] \quad (2.6)$$

A parcela  $\Delta\varepsilon_{xy}$ , referente às deformações transversais por ser escrita na forma,

$$\Delta\varepsilon_{xy} = \Delta e_{xy} + \Delta\eta_{xy} \quad (2.7)$$

onde  $\Delta e_{xy}$  representa a parcela linear, que é nula

$$\Delta e_{xy} = 0 \quad (2.8)$$

e  $\Delta \eta_{xy}$ , representado pela parcela não linear, resumindo as deformações transversais à forma representada pela Equação 2.9.

$$\Delta \eta_{xy} = \frac{1}{2} \left[ -\frac{d\Delta u}{dx} \frac{d\Delta v}{dx} + y \left( \frac{d\Delta v}{dx} \frac{d^2\Delta v}{dx^2} \right) \right] \quad (2.9)$$

A energia potencial das forças externas na configuração deformada,  $\Delta V$  é definida como,

$$\Delta V = - \int_s \Delta F_i \Delta u_i ds = - \left[ \int_s {}^t F_i \Delta U_i ds + \int_s F_i \Delta U_i ds \right] \quad (2.10)$$

sendo  $s$  a região onde as forças  $F_i$  são aplicadas e  $\Delta u_i$  representam as componentes dos deslocamentos incrementais.

#### 2.2.4 Discretização do Sistema Estrutural

Com base no elemento de viga coluna definido na Figura 2.3, deve-se definir uma relação entre o deslocamento de um ponto qualquer do elemento e os deslocamentos nodais incrementais. O equacionamento desta Seção (Galvão, 2000) trata das relações que levam às Matrizes de Rigidez utilizadas na implementação computacional da solução não linear.

Para que haja continuidade de deslocamentos e rotação nos bordos dos elementos adjacentes, é suficiente considerar, para aproximar o deslocamento axial incremental  $\Delta u$ , uma função linear, enquanto para a componente transversal  $\Delta v$ , admitindo-se  $\Delta \theta = d\Delta v/dx$ , deve ser usada uma função do terceiro grau (Yang e Kuo, 1994). Podemos, portanto, definir  $\Delta u$  e  $\Delta v$ , como,

$$\Delta u = a_0 + a_1 x \quad (2.11)$$

$$\Delta v = b_0 + b_1 x + b_2 x^2 + b_3 x^3 \quad (2.12)$$

onde  $a_0, a_1, b_0, b_1, \dots$  e  $b_n$  são constantes a serem determinadas através das condições de contorno do elemento: em  $x = 0$ ,  $\Delta u = \Delta u_1$ ,  $\Delta v = \Delta v_1$  e  $\Delta \theta_1 = d\Delta v_1/dx$ ; e em  $x = L$ ,  $\Delta u = \Delta u_2$ ,  $\Delta v = \Delta v_2$  e  $\Delta \theta_2 = d\Delta v_2/dx$ . Dessas condições chegam-se às expressões para  $\Delta u$  e  $\Delta v$  em termos dos valores nodais,

$$\Delta u = H_1 \Delta u_1 + H_2 \Delta u_2 \quad (2.13)$$

$$\Delta v = H_3 \Delta v_1 + H_4 \Delta \theta_1 + H_5 \Delta v_2 + H_6 \Delta \theta_2 \quad (2.14)$$

onde  $H_1, H_2, \dots$  e  $H_6$  são as funções de interpolação,

$$\begin{aligned} H_1 &= 1 - \frac{x}{L} \\ H_2 &= \frac{x}{L} \\ H_3 &= 1 - \frac{3x^2}{L^2} + \frac{2x^3}{L^3} \\ H_4 &= x - \frac{2x^2}{L} + \frac{x^3}{L^2} \\ H_5 &= \frac{3x^2}{L^2} - \frac{2x^3}{L^3} \\ H_6 &= -\frac{x^2}{L} + \frac{x^3}{L^2} \end{aligned}$$

Matricialmente, tem-se que os deslocamentos  $\Delta u$  e  $\Delta v$ , e a rotação  $\Delta \theta$  de um dado ponto do elemento, a uma distância  $x$  do nó 1, são dadas pela relação:

$$\Delta d = H \Delta u \quad (2.15)$$

onde a equação anterior assume a seguinte forma,

$$\begin{bmatrix} \Delta u \\ \Delta v \\ \Delta \theta \end{bmatrix} = \begin{bmatrix} H_1 & 0 & 0 & H_2 & 0 & 0 \\ 0 & H_3 & H_4 & 0 & H_5 & H_6 \\ 0 & H_{3,x} & H_{4,x} & 0 & H_{5,x} & H_{6,x} \end{bmatrix} \begin{bmatrix} \Delta u_1 \\ \Delta v_1 \\ \Delta \theta_1 \\ \Delta u_2 \\ \Delta v_2 \\ \Delta \theta_2 \end{bmatrix}$$

Obtidos os resultados, transforma-se os valores de  $\Delta u$  para o sistema de coordenadas global (referencial comum), através da equação,

$$\Delta u_{gl} = R^T \Delta u \tag{2.16}$$

onde  $R$  é a matriz de rotação do elemento, representada pela expressão 3.29.

O indicador variacional (energia potencial total do sistema) pode ser expresso em função dos deslocamentos e forças nodais através da equação,

$$\Delta \Pi = \Delta u^{eT} \left[ \frac{1}{2} (K_L^e + K_\tau^e) \right] \Delta u^e + \Delta u^{eT} {}^t F_i^e - \Delta u^{eT} {}^{(t+\Delta t)} \lambda F_r^e \tag{2.17}$$

Obtemos, a partir da equação anterior, a sua primeira variação,

$$\delta^1 \Delta \Pi = (K_L^e + K_\tau^e) \Delta u^e + {}^t F_i^e - {}^{(t+\Delta t)} \lambda F_r^e \tag{2.18}$$

Sabendo que a condição de equilíbrio para o sistema pede que a primeira variação seja igual a zero:

$$\delta^1 \Delta \Pi = 0 \tag{2.19}$$

Chega-se à equação na forma a seguir,

$$(K_L^e + K_\tau^e) \Delta u^e + {}^t F_i^e = ({}^{t+\Delta t}) \lambda F_r^e \quad (2.20)$$

onde a matriz de rigidez  $K^e$  é representada pelas parcelas linear  $K_L^e$  e não linear  $K_\tau^e$ , referente às tensões,

$$K^e = K_L^e + K_\tau^e \quad (2.21)$$

As matrizes  $K_L^e$  e  $K_\tau^e$  são obtidas diretamente da energia interna de deformação do sistema. Sendo  $K_L^e$  representada por,

$$K_{L(i,j)}^e = \frac{\partial^2 U_L}{\partial \Delta u_i \partial \Delta u_j} \quad (2.22)$$

que é representada pela matriz  $K_L^e$  em (2.23):

$$\mathbf{K}_L^e = \begin{bmatrix} EA/L & 0 & 0 & -EA/L & 0 & 0 \\ 0 & 12EI/L^3 & 6EI/L^2 & 0 & -12EI/L^3 & 6EI/L^2 \\ 0 & 6EI/L^2 & 4EI/L & 0 & -6EI/L^2 & 2EI/L \\ -EA/L & 0 & 0 & EA/L & 0 & 0 \\ 0 & -12EI/L^3 & -6EI/L^2 & 0 & 12EI/L^3 & -6EI/L^2 \\ 0 & -6EI/L^2 & 2EI/L & 0 & -6EI/L^2 & 4EI/L \end{bmatrix} \quad (2.23)$$

e  $K_\tau^e$  representado por,

$$K_{\tau(i,j)}^e = \frac{\partial^2 U_\tau}{\partial \Delta u_i \partial \Delta u_j} \quad (2.24)$$

que é representada pela matriz simétrica  $K_\tau^e$  na forma (2.25):

$$\mathbf{K}_\tau^e = \begin{bmatrix} \frac{P}{L} & 0 & \frac{-M_1}{L} & -\frac{P}{L} & 0 & -\frac{M_2}{L} \\ 0 & \frac{6P}{5L} + \frac{12PI}{AL^3} & \frac{P}{10} + \frac{6PI}{AL^2} & 0 & -\left(\frac{6P}{5L} + \frac{12PI}{AL^3}\right) & \frac{P}{10} + \frac{6PI}{AL^2} \\ \frac{-M_1}{L} & \frac{P}{10} + \frac{6PI}{AL^2} & \frac{2PL}{15} + \frac{4PI}{AL} & -\frac{-M_1}{L} & -\left(\frac{P}{10} + \frac{6PI}{AL^2}\right) & -\frac{PL}{30} + \frac{2PI}{AL} \\ -\frac{P}{L} & 0 & -\frac{-M_1}{L} & \frac{P}{L} & 0 & \frac{M_2}{L} \\ 0 & -\left(\frac{6P}{5L} + \frac{12PI}{AL^3}\right) & -\left(\frac{P}{10} + \frac{6PI}{AL^2}\right) & 0 & \frac{6P}{5L} + \frac{12PI}{AL^3} & -\left(\frac{P}{10} + \frac{6PI}{AL^2}\right) \\ -\frac{M_2}{L} & \frac{P}{10} + \frac{6PI}{AL^2} & -\frac{PL}{30} + \frac{2PI}{AL} & \frac{M_2}{L} & -\left(\frac{P}{10} + \frac{6PI}{AL^2}\right) & \frac{2PL}{15} + \frac{4PI}{AL} \end{bmatrix} \quad (2.25)$$

Após a definição das matrizes de rigidez referentes a cada elemento do pórtico, as mesmas devem ser rotacionadas para o sistema global de referência da estrutura. A informação contida em cada uma dessas matrizes será inserida em uma matriz de rigidez global, referente à toda a estrutura. Esta matriz conterá as informações de todas as matrizes de rigidez de cada elemento. Esse assunto será abordado mais detalhadamente nos Capítulos posteriores.

## 2.3 Metodologia de Solução Não Linear

A estratégia de solução de um sistema estrutural que se comporta não linearmente, através do método dos elementos finitos, envolve necessariamente a solução de um sistema de equações algébricas não lineares. Para isso, serão utilizados métodos de solução combinando procedimentos incrementais e iterativos.

Após a leitura do arquivo de entrada, onde os dados referentes à situação inicial da estrutura, carregamento e análise serão lidos, inicia-se o processo de solução não linear do problema.

Em uma análise incremental não linear que incorpore procedimentos iterativos em cada passo incremental, duas diferentes fases podem ser identificadas. A primeira delas, denominada fase predita, envolve a solução dos deslocamentos incrementais, através das equações de equilíbrio da estrutura, a partir de um determinado acréscimo de carregamento. A segunda fase, denominada corretiva, tem por objetivo a correção das forças internas incrementais obtidas dos acréscimos de deslocamentos pela utilização de um processo iterativo. Tais forças internas são então comparadas com o carregamento externo, obtendo-se daí a quantificação do desequilíbrio  $g$  existente entre forças internas e externas

(Silva, 2010). Ambas as fases serão abordadas nas próximas Seções.

$$g = F_{ext} - F_i \quad (2.26)$$

Estabelecido um critério de convergência  $\xi$ , o processo iterativo será interrompido quando a diferença entre a força interna e externa da estrutura atingir esse valor. Isso significa que a mesma se encontra em estado de equilíbrio.

$$\xi \geq \frac{\|g\|}{\|F_{ext}\|} \quad (2.27)$$

Após a confirmação da convergência, ou seja, verificado que a estrutura encontra-se em equilíbrio, atualizam-se as variáveis e repete-se todo o processo até o último incremento.

Num contexto computacional, para um dado passo de carga, esse processo pode ser resumido em duas etapas. Inicialmente, a partir da última configuração de equilíbrio da estrutura, é selecionado um incremento de carga,  $\Delta\lambda^0$ , definido aqui como incremento inicial do parâmetro de carga, procurando satisfazer alguma equação de restrição imposta ao problema. Após a seleção de  $\Delta\lambda^0$ , determina-se o incremento inicial dos deslocamentos nodais  $\Delta U^0$ . As aproximações  $\Delta\lambda^0$  e  $\Delta U^0$  caracterizam o que é comumente chamado de solução incremental predita. Na segunda etapa de solução, mediante uma determinada estratégia de iteração, tem-se como objetivo corrigir a solução incremental, inicialmente proposta na etapa anterior, para restaurar o equilíbrio da estrutura. Se as iterações realizadas envolvem não só os deslocamentos nodais,  $U$ , mas também o parâmetro de carga,  $\lambda$ , então uma equação adicional de restrição é requerida. A forma dessa equação de restrição é o que distingue as várias estratégias de iteração (Silveira, 1995; Galvão, 2000).

### 2.3.1 Solução Incremental Predita

A solução incremental predita é composta de três etapas bem definidas: criação da matriz de rigidez, definição do vetor de forças aplicadas e cálculo do vetor de deslocamentos predito. Essas três etapas serão abordadas a seguir.



### 2.3.1.1 Matriz de Rigidez

A primeira etapa da solução incremental predita é a montagem da matriz de rigidez da estrutura,  $K$ . Essa matriz é obtida através da realocação de todas as matrizes de rigidez dos elementos,  $K_{gl}^e$ . A montagem das matrizes  $K_{gl}^e$  pode ser feita utilizando-se, em um primeiro momento, os dados de entrada obtidos pela leitura do arquivo de entrada com as informações da estrutura ou, após o primeiro ciclo incremental-iterativo, através dos dados atualizados referentes à última configuração de equilíbrio da estrutura. A matriz de Rigidez do elemento pode ser expressa pela equação:

$$K_{gl}^e = R^{eT}(K_L + K_\tau)R^e \quad (2.28)$$

onde  $R^e$  é a matriz de rotação entre o sistema global de coordenadas e o sistema local, representada a seguir:

$$R^e = \begin{bmatrix} \cos\theta & \sin\theta & 0 & 0 & 0 & 0 \\ -\sin\theta & \cos\theta & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & \cos\theta & \sin\theta & 0 \\ 0 & 0 & 0 & -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \quad (2.29)$$

Sendo  $\cos\theta$  o cosseno do ângulo entre o sistema global e local de coordenadas e  $\sin\theta$  o seno do ângulo entre o sistema global e local de coordenadas.

### 2.3.1.2 Vetor de Forças Aplicadas

A segunda etapa da solução incremental predita consiste na definição do vetor de cargas aplicadas,  $F_{ext}$ , através da equação 2.30,

$$F_{ext} = \Delta\lambda F_r \quad (2.30)$$

onde  $F_r$  é o vetor de cargas de referência, criado a partir do arquivo de entrada de dados, e  $\Delta\lambda$  é o parâmetro de carga definido para o passo corrente do ciclo incremental.

### 2.3.1.3 Vetor de Deslocamentos

Montada a Matriz de Rigidez da Estrutura,  $K$ , é realizado o cálculo do vetor de deslocamentos incremental  $\Delta U$ , resolvendo-se o sistema linear:

$$K\Delta U = F_{ext} \quad (2.31)$$

O vetor  $F_{ext}$ , expresso na equação 2.31, representa o vetor de carregamento externo. Vale observar que, para o cálculo ser efetuado, as informações referentes às condições de contorno do problema devem ser removidas de  $F_{ext}$ , o que implica na geração do vetor  $\Delta U$  sem as condições de contorno. Essa inclusão deverá ser realizada após a obtenção do resultado da equação 2.31. Essas condições de contorno representam os valores das restrições aos movimentos da estrutura, assunto que será abordado nos próximos Capítulos.

O vetor de deslocamentos  $\Delta U$ , calculado na solução predita, obedece às regras da análise estrutural linear, o que resulta, na grande maioria dos casos, em uma solução aproximada do comportamento real da estrutura.

Esta solução necessitará de um tratamento corretivo, onde refinamentos seguindo princípios iterativos serão realizadas até que o resultado do deslocamento se aproxime o suficiente do resultado real da estrutura. Esse processo será rapidamente abordado na Seção 2.3.2.

## 2.3.2 Ciclo de Iterações

Obtido o deslocamento  $\Delta U$  referente às cargas aplicadas  $F_{ext}$  (solução predita), o primeiro passo do ciclo iterativo é a atualização das coordenadas para o cálculo do vetor de forças internas  $F_i$ . Essa atualização é necessária pelo fato de o vetor de cargas aplicadas  $F_{ext}$  implicar no deslocamento (representado pelo vetor de deslocamentos  $\Delta U$ ) a ser utilizado no cálculo do vetor de forças internas  $F_i$ .

Os valores de  $F_i$  serão comparados aos do vetor de cargas aplicadas ou forças externas,  $F_{ext}$ . Essa comparação baseia-se no princípio Variacional, que diz que para a condição de equilíbrio ser alcançada, a primeira variação do funcional de energia  $\Delta\Pi$  deve ser nula,

ou seja, igual a zero. A representação da primeira variação do funcional de energia é mostrada a seguir:

$$\delta^1 \Delta \Pi = [(K_L^e + K_\tau^e)] \Delta u^e + {}^t F_i^e - {}^{t+\Delta t} \lambda F_r^e \quad (2.32)$$

Como,

$$\delta^1 \Delta \Pi = 0 \quad (2.33)$$

o último termo da expressão pode ser enviado para o outro lado da igualdade, tendo seu sinal trocado. A equação, então, toma a forma da equação 2.20. Lembrando que  $[(K_L^e + K_\tau^e)] = K^e$ , a equação pode ser reescrita na forma,

$$K^e \Delta u^e + {}^t F_i^e = {}^{t+\Delta t} \lambda F_r^e \quad (2.34)$$

Sabendo que a expressão  $K^e \Delta u^e$  resulta no vetor de incremento de força interna  $\Delta^t F_i^e$ , a equação se torna,

$$\Delta^t F_i^e + {}^t F_i^e = {}^{t+\Delta t} \lambda F_r^e \quad (2.35)$$

onde o vetor de forças internas pode ser reescrito na forma,

$$\Delta^t F_i^e + {}^t F_i^e = {}^{t+\Delta t} F_i^e \quad (2.36)$$

Pode ser observado que, para a condição de equilíbrio ser alcançada, as forças internas  ${}^{t+\Delta t} F_i^e$  devem ser iguais às forças externas  ${}^{t+\Delta t} \lambda F_r^e$ , ou seja, às cargas aplicadas.

$${}^{t+\Delta t} F_i^e = {}^{t+\Delta t} \lambda F_r^e \quad (2.37)$$

### 2.3.2.1 Vetor de Forças Internas

Com o objetivo de contornar o problema gerado pelo aparecimento de tensões residuais nos deslocamentos de corpo rígido, o vetor de forças internas é calculado através dos deslocamentos naturais incrementais  $\Delta u_n$  e da matriz de rigidez  $K$ . Sabendo que para cada elemento finito temos a seguinte relação para obtenção do vetor de forças internas incremental  $\Delta^t F_i^e$ ,

$$\Delta^t F_i^e = K^e \Delta u_n^e \quad (2.38)$$

sendo o vetor de deslocamentos naturais incrementais  $\Delta u_n^e$  para cada elemento definido por,

$$\Delta u_n^e = \begin{bmatrix} 0 \\ 0 \\ \phi_1 \\ \delta \\ 0 \\ \phi_2 \end{bmatrix} \quad (2.39)$$

onde,

$$\phi_1 = \Delta \theta_1 - \Psi \quad (2.40)$$

$$\delta = {}^{t+\Delta t} L - {}^t L \quad (2.41)$$

$$\phi_2 = \Delta \theta_2 - \Psi \quad (2.42)$$

e, sabendo que,

$$v = \Delta v_2 - \Delta v_1 \quad (2.43)$$

$$u = \Delta u_2 - \Delta u_1 \quad (2.44)$$

$$\Psi = \tan^{-1} \left( \frac{v}{{}^tL + u} \right) \quad (2.45)$$

Vale lembrar que os valores das coordenadas atualizadas se encontram no sistema global de referência, sendo necessária a sua rotação para o sistema local de cada elemento, referente ao passo de carga de seu respectivo ciclo incremental, antes do cálculo de  $\Delta u_n^e$ .

Após a definição de  $\Delta u_n^e$ , calcula-se, através da expressão (2.38), o vetor de forças internas incremental  $\Delta^t F_i^e$  de cada elemento.

Definido  $\Delta^t F_i^e$ , este deverá ser somado ao vetor de carregamentos acumulados até o passo anterior,  ${}^t F_i^e$ , que deverá ser rotacionado da última configuração processada no ciclo iterativo para o sistema de referência global da estrutura.

Definido o vetor de forças internas de cada elemento,  $\Delta^t F_i^e$ , e realizadas as respectivas inserções no vetor de forças internas global,  $F_i$ , a diferença  $g$  calculada entre a força interna  $F_i$  e a força externa  $F_{ext}$  será comparada ao valor definido para a tolerância  $\xi$  aceita para o critério de convergência. Enquanto o valor da diferença entre força externa e força interna for maior que  $\xi$ , os valores de deslocamento e carga aplicada serão recalculados com base no vetor  $g$  e o processo iterativo se repetirá, sendo calculado um novo vetor de forças internas  $F_i$  para a realização de nova comparação.

A determinação do parâmetro de carga iterativo,  $\Delta \lambda^{n+1}$  é função de uma dada estratégia de iteração, ou equação de restrição imposta ao problema, que tem a função de otimizar a convergência do processo iterativo (Galvão, 2000). A seguir são apresentadas duas estratégias bastante eficientes que serão utilizadas nos Capítulos seguintes.

### 2.3.2.2 Iteração à Carga Constante

O processo de iteração à carga constante, como o próprio nome diz, consiste em não se alterar o valor do carregamento externo (utilizado na solução predita do ciclo incremental) no decorrer de todo o ciclo iterativo, sendo atualizados apenas os valores referentes ao vetor de deslocamentos,  $\Delta U$ . Isso implica na expressão simplificada para o cálculo do carregamento externo,

$$\delta\lambda^n = 0 \quad (2.46)$$

onde cada  $\delta\lambda^n$  representa o incremento de carga em cada passo do ciclo iterativo. Como não há incremento a ser adicionado ao carregamento predito, a expressão se anula, restando apenas a atualização dos deslocamentos durante o ciclo,

$$\Delta U_{n+1} = \Delta U_n + \delta u_g^n \quad (2.47)$$

onde  $\delta u_g^n$  representa a correção do vetor de deslocamentos incrementais  $\Delta U_n$ , e é obtido da solução do sistema linear:

$$K\delta u_g^n = g \quad (2.48)$$

Essa estratégia, de simples implementação, é útil somente para análises de estruturas até pontos-limite, pois, como o incremento de carga possui apenas um sentido (positivo ou negativo) não é possível retornar à trajetória de equilíbrio, tornando-se ineficaz a análise após pontos limites da estrutura, onde o sentido do incremento de carga deveria ser modificado.

### 2.3.2.3 Iteração baseada no Comprimento de Arco Cilíndrico

A estratégia de comprimento de arco cilíndrico consiste na introdução de uma equação de restrição em forma de arco cilíndrico a partir do ponto encontrado na solução predita.

Dessas observações veio a proposta de Crisfield (1981) de que, a cada iteração, a seguinte relação fosse satisfeita:

$$(\Delta u^k)^T \Delta u^k = \Delta l^2 \quad (2.49)$$

onde  $\Delta u^k$  é o incremento de deslocamento atualizado durante o ciclo iterativo. Com base nessa informação, pode-se escrever a equação quadrática a seguir,

$$A(\delta\lambda^k)^2 + B\delta\lambda^k + C = 0 \quad (2.50)$$

onde, agora, os coeficientes A, B e C têm a seguinte forma:

$$A = (\delta u_r^k)^T \delta u_r^k \quad (2.51)$$

$$B = 2(\delta u_r^k)^T (\Delta u^{(k-1)} + \delta u_g^k) \quad (2.52)$$

$$C = (\Delta u^{(k-1)} + \delta u_g^k)^T (\Delta u^{(k-1)} + \delta u_g^k) - \Delta l^2 \quad (2.53)$$

Com a resolução de (2.50), chega-se às raízes da equação,  $\delta\lambda_1$  e  $\delta\lambda_2$ . Deve-se, então, escolher entre as soluções

$$\Delta u_1^k = \Delta u^{(k-1)} + \delta u_g^k + \delta\lambda_1^k \delta u_r^k \quad (2.54)$$

e

$$\Delta u_2^k = \Delta u^{(k-1)} + \delta u_g^k + \delta\lambda_2^k \delta u_r^k \quad (2.55)$$

O critério para escolha das expressões acima é a maior proximidade com a solução incremental da iteração anterior,  $\Delta u^{(k-1)}$ . Essa escolha deve prevenir um possível retorno, o que faria a solução regredir ao longo do caminho já calculado. Um procedimento bastante simples a ser seguido, reside na definição do menor ângulo entre  $\Delta u^k$  e  $\Delta u^{(k-1)}$ , o que equivale a se encontrar o máximo cosseno do ângulo:

$$\cos\theta_{1,2} = \frac{\Delta u^{(k-1)T} \Delta u^k}{\Delta l^2} = \frac{\Delta u^{(k-1)T} (\Delta u^{(k-1)} + \delta u_g^k)}{\Delta l^2} + \delta\lambda_{1,2} \frac{\Delta u^{(k-1)T} \delta u_r^k}{\Delta l^2} \quad (2.56)$$

Como a Equação (2.50) é quadrática, ela poderá ter raízes imaginárias se o termo  $B^2 - 4AC$  for menor que zero. Essa situação pode existir quando o incremento inicial do parâmetro de carga for muito grande, ou se a estrutura exibir múltiplos caminhos de equilíbrio em torno de um ponto (Meek e Tan, 1984).

Quando este tipo de situação ocorre, o critério de solução a ser adotado consiste na interrupção do ciclo iterativo e manutenção das últimas configurações encontradas durante o mesmo. Após a saída do ciclo iterativo, o novo comprimento de arco desejado é calculado com base no produto entre o fator de correção definido através da relação *limite/norma* (cujo resultado não poderá exceder os valores 0.1 e 0.5) e o comprimento de arco definido na fase predita do ciclo incremental corrente. Esse valor será utilizado no cálculo do próximo incremento do comprimento de arco, definido na fase predita do próximo passo do ciclo incremental dando continuidade à análise da estrutura. O procedimento acima descrito pode não ser suficiente para a continuidade do processo de análise. Nesse caso, duas situações podem ser observadas: A estrutura pode sofrer uma deformação incoerente com a análise ou simplesmente não sofrer mais alterações. Em ambos os casos, adota-se o procedimento de interrupção do ciclo incremental e finaliza-se a análise da estrutura.

A Figura 2.4 ilustra os dois primeiros incrementos de carga e deslocamento, partindo da solução predita, durante um passo do ciclo iterativo da estratégia de comprimento de arco cilíndrico.

Quando o critério de convergência for atingido, ou seja, quando a diferença entre a força externa e a força interna for tão pequena que seu valor será igual ou menor que o definido pelo critério de convergência, o ciclo iterativo se encerrará, indicando que a estrutura se encontra em equilíbrio, e um novo incremento de carga será definido, iniciando-se, assim um novo ciclo incremental na análise da estrutura.

A definição do incremento de carga deve ser definida com base em algumas estratégias de incremento de carga, visando a obtenção de uma análise não linear bem sucedida. No caso da utilização da estratégia iterativa de comprimento de arco cilíndrico, a utilização de uma estratégia incremental de comprimento de arco se torna necessária. Há ainda a questão da análise do sinal do incremento do carregamento, que deve ser definido corretamente para a continuidade da análise durante e após os carregamentos críticos, onde



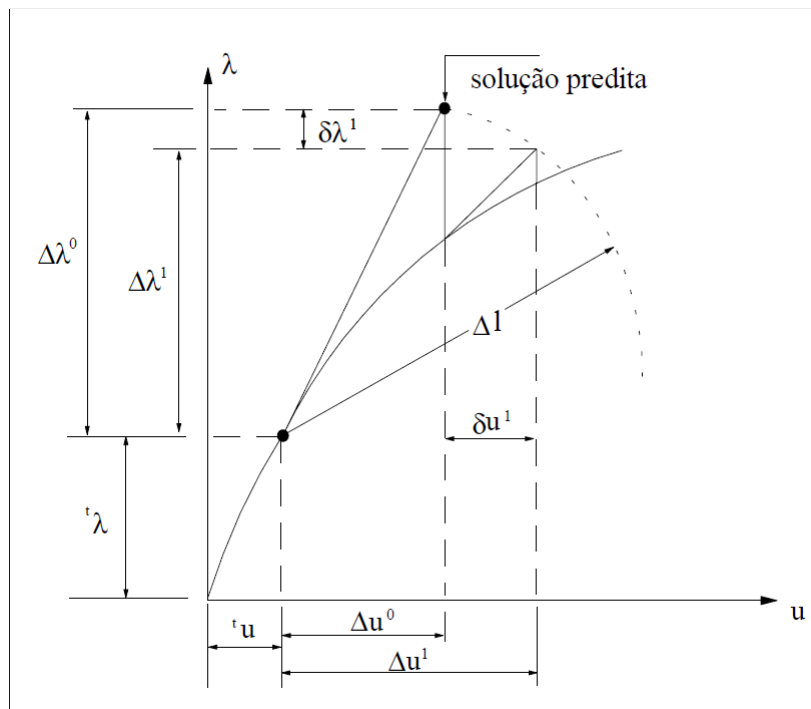


Figura 2.4: Estratégia de Comprimento de Arco Cilíndrico (Crisfield, 1991)

a estrutura possui seu comportamento alterado. Essas estratégias serão abordadas nas subseções a seguir.

### 2.3.3 Estratégias de Incremento de Carga

A definição do incremento do parâmetro de carga deve refletir o grau de não linearidade da análise realizada. Uma estratégia eficiente de incremento automático de carga deve satisfazer basicamente os seguintes requerimentos:

- (i) produzir grandes incrementos quando a resposta da estrutura for aproximadamente linear;
- (ii) gerar pequenos incrementos quando a resposta da estrutura for fortemente não linear;
- (iii) ser capaz de escolher o sinal correto para o incremento, introduzindo medidas capazes de detectar quando os pontos limites são ultrapassados.

A seguir serão abordadas as estratégias incrementais de parâmetro de carga utilizadas neste trabalho.

### 2.3.3.1 Incremento do Comprimento de Arco

O cálculo do incremento do comprimento de arco,  $\Delta l$ , é realizado durante a fase predita do ciclo incremental, através da condição de restrição:

$${}^t\Delta l^2 = {}^t\Delta u^{0T} {}^t\Delta u^0 \quad (2.57)$$

onde  ${}^t\Delta u$  é definido como o vetor de deslocamentos de referência, resultante do produto do vetor de forças de referência pela matriz de rigidez do incremento corrente. A equação, portanto, pode ser reescrita na forma,

$${}^t\Delta l = \sqrt{{}^t\Delta u^{0T} {}^t\Delta u^0} \quad (2.58)$$

Definido o incremento do comprimento de arco,  $\Delta l$ , o incremento do carregamento  $\Delta \lambda$  é calculado através da expressão:

$${}^t\Delta \lambda = \pm \frac{{}^{t-1}\Delta l_{des}}{{}^t\Delta l} \quad (2.59)$$

onde o sinal correto da expressão pode ser escolhido baseando-se no sinal do parâmetro GSP, que será apresentado na Seção 2.3.3.2.

O escalar  $\Delta l_{des}$  representa o comprimento de arco desejado, proposto por Crisfield(1991) para ser adotado como parâmetro de controle a ser utilizado no próximo passo de carga. Obtido após o término do ciclo iterativo, ao final do ciclo incremental do passo anterior, ele pode ser representado pela equação a seguir:

$${}^t\Delta l_{des} = {}^t\Delta l \sqrt{\frac{I_d}{tI}} \quad (2.60)$$

onde  $t$  é o passo incremental corrente,  $I_d$  é o número de iterações desejado (definido pelo usuário),  $tI$  é o número de iterações necessários para o alcance da convergência e  $\Delta l$  é o comprimento de arco definido na solução predita.

Somente após o cálculo do incremento do carregamento,  $\Delta \lambda$ , o valor de  $\Delta l_{des}$  é assumido por  $\Delta l$ , o qual será o novo comprimento de arco cilíndrico, servindo de base de cálculo durante o ciclo iterativo e utilizado após o término do mesmo no cálculo do novo incremento de arco desejado.

Deve-se observar que, durante o primeiro passo do ciclo incremental, o valor de  $\Delta l_{des}$  é inexistente, devendo, então, ser definido através da expressão:

$${}^0\Delta l_{des} = {}^0\Delta\lambda {}^0\Delta l \quad (2.61)$$

onde  ${}^0\Delta\lambda$  é o incremento de carga inicial definido pelo usuário e  ${}^0\Delta l$  é representado pela equação 2.58 calculada pela primeira vez. Em seguida, o primeiro  ${}^0\Delta\lambda$  a ser de fato utilizado na montagem do vetor de forças, para posterior cálculo dos deslocamentos, será calculado com case na equação 2.59.

### 2.3.3.2 Sinal do Incremento Inicial do Parâmetro de Carga

A análise do sinal a ser adotado no incremento do carregamento (equação 2.59) pode ser definida observando-se o sinal obtido através do cálculo do parâmetro GSP (*Generalized Stiffness Parameter*), realizado na fase predita do ciclo incremental corrente, antes do cálculo de  $\Delta\lambda$ . Yang e Kuo (1994) consideram o parâmetro GSP do sistema em sua forma:

$$GSP = \frac{{}^1\delta u_r^T {}^1\delta u_r}{{}^{t-1}\delta u_r^T {}^t\delta u_r} \quad (2.62)$$

onde  ${}^1\delta u_r^T$  é o vetor de deslocamentos de referência do primeiro passo, obtido através do produto entre a matriz de rigidez da estrutura em sua configuração inicial e o vetor de forças de referência. O vetor  ${}^{t-1}\delta u_r$  refere-se aos deslocamentos de referência do passo anterior enquanto  ${}^t\delta u_r$  é o vetor de deslocamentos de referência do passo corrente. Deve-se observar que esses vetores são obtidos durante a fase predita de seus respectivos ciclos incrementais.

De acordo com Yang e Kuo (1994), o sinal do parâmetro de rigidez corrente depende apenas dos vetores  ${}^{t-1}\delta u_r$  (passo de carga anterior) e  ${}^t\delta u_r$  (passo de carga corrente), lembrando que ambos são definidos durante a fase predita. O parâmetro de rigidez GSP torna-se negativo para os passos de carga localizados nas regiões próximas aos pontos limites. Para os demais, esse parâmetro permanecerá sempre positivo. Isso implica na alteração do sinal do incremento de carga  $\Delta\lambda$  toda vez que o sinal do parâmetro GSP se tornar negativo.

Vale lembrar que o sinal GSP ao início da análise é definido como positivo.

## Capítulo 3

# Implementação Computacional

### 3.1 Introdução

Um sistema computacional bem definido envolve não somente equipamentos adequados à solução do problema (*hardware*) como também a utilização de códigos escritos de forma clara e objetiva, com a estruturação mais simples possível. Tendo como base estas observações, este capítulo apresenta a ferramenta computacional para análise estrutural desenvolvida neste trabalho com suas características mais relevantes.

Para o desenvolvimento do sistema a ser abordado como tema desta dissertação foram utilizados como base os procedimentos desenvolvidos no CS-ASA (Silva, 2009).

Optou-se pelo Matlab pelos seguintes motivos:

- Interface de utilização da linguagem: a linguagem Matlab é própria para o desenvolvimento de códigos voltados para a resolução de cálculos numéricos, álgebra matricial e análises numéricas, tendo como elemento básico da informação matrizes. Além disso, como as soluções dos problemas são expressas quase exatamente como elas são formuladas matematicamente (ao contrário da programação tradicional - ex. Fortran, C, etc.), torna-se muito mais rápido o desenvolvimento do código, o qual usualmente será menor e mais simples, facilitando a sua compreensão por parte do usuário-desenvolvedor, e
- Avanços tecnológicos: com os avanços no mercado computacional, se torna necessário o uso de ferramentas que acompanhem esse processo evolutivo. Códigos de mais fácil compreensão, portáteis e que permitem codificações mais simples para aplicativos modernos, mais robustos e confiáveis se tornam fundamentais nessa etapa da

análise de estruturas. Soluções computacionais (paralelização, por exemplo) de alta performance são desejáveis devido ao alto custo computacional inerente a algumas análises. O software escolhido como ambiente de desenvolvimento deste trabalho permite a execução de todas as etapas da análise estrutural, incluindo a geração de resultados gráficos e animações.

Deve-se observar também, que este trabalho permite uma verificação maior e mais detalhada de todas as etapas da análise estrutural, combinada com a sua aplicação em um ambiente de desenvolvimento mais amigável ao desenvolvedor, que não necessitará possuir conhecimentos avançados em programação.

### 3.1.1 Características

O código computacional desenvolvido neste estudo foi criado em ambiente Matlab, numa linguagem de programação mais simples e direta, de mais fácil compreensão por profissionais da área de ciências exatas que não sejam tão familiarizados com a programação tradicional. Como o ambiente Matlab permite tratamento de dados, pós-processamento, gerações de gráficos e animações das estruturas, esses puderam ser incluídos sem grandes complicações no código desenvolvido.

Pelo fato dessa linguagem haver sido desenvolvida em conformidade com a linguagem matemática, possuindo suas bases no conceito tradicional de matrizes, os códigos computacionais se encontram mais claros e em concordância com as metodologias de solução da análise estrutural estática não linear de pórticos planos.

Para a validação dos dados foram utilizados exemplos clássicos da literatura científica. O código CS-ASA (desenvolvido em linguagem Fortran) foi utilizado para comparação dos resultados numéricos e geração dos resultados gráficos.

O CS-ASA teve sua base computacional desenvolvida inicialmente por Silveira (1995) para investigar a instabilidade elástica de colunas, arcos e anéis com restrições unilaterais de contato. Posteriormente, sob orientação do mesmo, outros trabalhos foram realizados usando essa base.

No primeiro deles, Galvão (2000) desenvolveu um código onde foram implementadas e testadas diversas formulações geometricamente não lineares para elementos de pórticos planos. Nesse mesmo ano, Rocha (2000) estudou estratégias de solução não linear para o traçado completo da trajetória de equilíbrio.

Pinheiro (2003) estudou uma estratégia de solução não linear para pórticos planos com ligações semi rígidas. Galvão (2004) realizou um detalhado estudo sobre instabilidade estática e dinâmica de estruturas esbeltas.

Formulações não lineares considerando o efeito da inelasticidade do aço em pórticos planos com ligações rígidas foram implementadas por Machado (2005). Finalmente, Rocha (2006) e Santos (2007) consideraram em um único elemento finito de pórtico plano os efeitos não lineares, possibilitando a análise inelástica de segunda ordem em estruturas metálicas com ligações semi rígidas.

Cabe ressaltar que em todos esses trabalhos foram desenvolvidos códigos computacionais a partir da mesma base computacional, em linguagem Fortran. Adicionalmente, todos esses códigos foram unificados por Silva (2009) em um só código, o CS-ASA.

A intenção deste trabalho é iniciar um projeto de reestruturação desses procedimentos em uma base computacional didática e de fácil compreensão e utilização para futuras pesquisas envolvendo análise não linear de estruturas e novas implementações.

No presente trabalho, é adotada a formulação de elemento finito de pórtico plano de Yang e Kuo para uma análise estrutural considerando os efeitos da não linearidade geométrica.

Em geral, o processo de simulação numérica é dividido em três etapas, pré-processamento, análise e pós-processamento, sendo usualmente tratadas de maneira independente. Tradicionalmente, o pré processamento, que é a etapa inicial da análise computacional, consiste na leitura de um ou mais arquivos texto em formatos específicos. Os dados contidos nesses arquivos serão, em seguida, processados para obtenção dos dados referentes às respostas que serão tratados no próprio código, gerando gráficos referentes ao pós-processamento. No caso do presente trabalho, durante o processamento pode-se também adicionar rotinas computacionais que tratem os dados referentes às respostas obtidas durante o mesmo, gerando assim simulações animadas do problema. Essas etapas serão abordadas detalhadamente nas próximas Seções deste Capítulo.

### 3.2 Visão Geral do Código Computacional

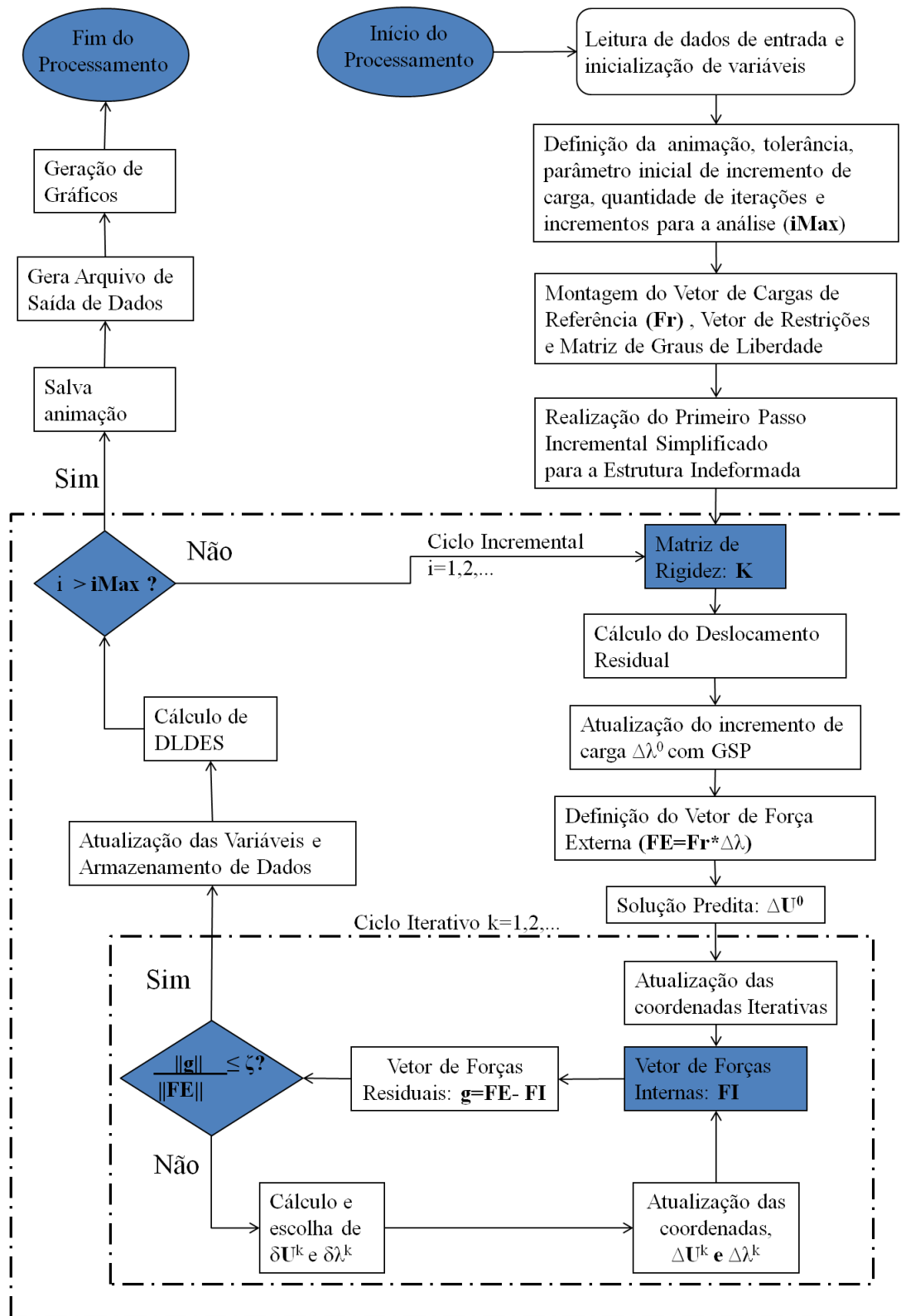


Figura 3.1: Análise Não Linear: Fluxograma.

Pode-se observar que este Capítulo tem como objetivo explicar detalhadamente toda a fase de implementação computacional da análise não linear estática de pórticos planos de forma que o leitor seja capaz de compreender a aplicação desta teoria através dos mecanismos computacionais de programação. Ao final deste Capítulo, o leitor usuário-desenvolvedor será capaz de reproduzir, manipular e complementar, quando necessário, esta análise em linguagem Matlab ou em outra de domínio do mesmo.

A Figura 3.1 exibe o fluxograma da análise não linear, retratando todo o processo de implementação computacional da teoria aplicada neste trabalho.

O código apresentado a seguir foi desenvolvido com a utilização de funções criadas em arquivos próprios, sendo os mesmos mantidos no mesmo diretório em que se localiza o código principal. A divisão das tarefas em funções é realizada para que o código seja simplificado, observando-se que um problema complexo quando dividido em problemas menores e, conseqüente, menos complexos, se torna de mais fácil resolução.

Todo o corpo do código principal pode ser observado na Figura 3.2.

Pode-se observar que as linhas 1 a 5 da Figura 3.2 representam a fase de pré-processamento da análise, constituída por entrada de dados e definições de limites e inicializações de variáveis. Na linha 6 é chamado o comando do Matlab *tic*, utilizado para marcar o início do tempo de processamento da análise.

As linhas 7 a 9 compreendem as definições dos vetores e matrizes que não sofrerão qualquer alteração durante a análise.

As linhas 10 a 15 compreendem o primeiro passo incremental da análise que, por ser diferenciado em alguns aspectos, constituindo uma versão simplificada de um passo do ciclo incremental, optou-se por mantê-lo fora da estrutura de repetição que compreende o ciclo incremental.

A estrutura de repetição compreendida entre as linhas 16 e 46 definem o ciclo incremental-iterativo, sendo dividido em solução predita (linhas 17 a 24), solução iterativa (linhas 25 a 39) e atualização de variáveis (linhas 40 a 45).

As linhas 47 a 52 referem-se à etapa de pós processamento, onde os arquivos são concluídos e salvos e os gráficos são gerados. Pode-se observar, na linha 49 a finalização da contagem do tempo pelo comando do Matlab *toc*, sendo a sua contagem armazenada na variável *tempo*, utilizada para verificação do tempo de processamento. Deve ser dada atenção, também, à linha 51, onde os dados da análise são gravados em arquivo.



```

1 -   clc; clear all; close all; format long e;
2 -   [Nnos, carr, ncarreg, Nelem, conect, Coord, E, A, I, Napoio, r, filme]=LerDadosArquivo();
3 -   [Xi, Xj, Yi, Yj, Anima, filme, nome]=CriaAnima(Nelem, Coord, conect, r, Napoio, filme);
4 -   [limite, deltaForceInc, incMax, Id, Xacum, Yacum, THETAcum, ret, PMM, ACoord, MATDATA]=...
5 -       LimitesIni(Coord, Nelem);
6 -   tic
7 -   VgrauLib=CriaVetGrauLib(Nnos, r, Napoio);
8 -   [MgrauLib1, cont]=criaMtzGrauLib(Nnos, VgrauLib);
9 -   ForceRef=CriaVetForce(Nnos, carr, ncarreg);
10 -  [FRef]=ReduzV2(VgrauLib, ForceRef);
11 -  [K]=CriaMtzK(Nelem, conect, Coord, E, A, I, PMM, MgrauLib1, cont);
12 -  URef10=K\FRef;
13 -  [dl, signal, deltaForceInc, deltaForce, Urefant]=...
14 -      CalcIncForce(URef10, URef10, URef10, 0, 1, 0, deltaForceInc);
15 -  [dldes]=fatCorrDLDES(ret, limite, 1, Id, 0, dl);
16 -  for inc=1:incMax
17 -      [K]=CriaMtzK(Nelem, conect, Coord, E, A, I, PMM, MgrauLib1, cont);
18 -      URef1=K\FRef;
19 -      [dl, signal, deltaForceInc, deltaForce, Urefant]=CalcIncForce...
20 -          (URef1, URef10, Urefant, dldes, signal, deltaForce, deltaForceInc);
21 -      ForceInc=ForceRef*deltaForceInc;
22 -      [FRed]=ReduzV2(VgrauLib, ForceInc);
23 -      u=K\FRed;
24 -      U=MontaVetDes(VgrauLib, u);
25 -      ret=0;
26 -      itera=0;
27 -  while ret==0
28 -      itera=itera+1;
29 -      [Coord, MU]=atualizaCoord(U, Nnos, ACoord);
30 -      [FI, PMM1]=CriaVetForceInt...
31 -          (Nelem, conect, Coord, E, A, I, ACoord, PMM, MgrauLib1, cont, MU);
32 -      [ret, g, norma]=compara(FRef, FI, deltaForce, limite);
33 -      deltaG=K\g;
34 -      if ret==0
35 -          [ret, desl, carga]=arc(URef1, U, deltaG, dl*dl, VgrauLib, ret);
36 -          U=MontaVetDes(VgrauLib, desl);
37 -          deltaForce=deltaForce+carga;
38 -      end
39 -  end
40 -  [PMM, ACoord, Xacum, Yacum, THETAcum]=...
41 -      AtualizaVar(PMM, PMM1, Coord, Xacum, Yacum, THETAcum, U, carr);
42 -  [dldes]=fatCorrDLDES(ret, limite, norma, Id, itera, dl);
43 -  [Anima]=AnimaEst(Nelem, Coord, conect, Anima, r, Napoio, Xi, Xj, Yi, Yj);
44 -  [MATDATA]=ArmazenaDados(inc, itera, deltaForceInc, ...
45 -      deltaForce, Xacum, Yacum, THETAcum, U, carr, Coord, MATDATA);
46 -  end
47 -  Anima=close(Anima);
48 -  hold off;
49 -  tempo=toc
50 -  ArqSaida=strcat('DADOS', filme(103:length(filme)), '.txt');
51 -  save(ArqSaida, 'MATDATA', '-ASCII');
52 -  grafico2(MATDATA, nome);

```

Figura 3.2: Código Principal.

Vale observar que a contagem de tempo realizada nesta análise é afetada pelos vários processos ativos no sistema. De forma mais detalhada o uso da função profile do

Matlab também pode ser usada e daí é possível ter ideia do tempo de processamento e compartilhamento do sistema.

As funções representadas na Figura 3.2, assim como maiores detalhes, serão apresentadas e explicadas em maiores detalhes no decorrer deste Capítulo. Vale lembrar alguns detalhes sobre a ferramenta utilizada (Matlab):

- Case Sensitive: o Matlab diferencia letras maiúsculas de minúsculas;
- Uso do ponto e vírgula: o ponto e vírgula, no Matlab é utilizado não só para definir o final da linha de comando, como para impedir que os valores das variáveis da linha sejam exibidos na janela de comando(tela de comando).
- Nomenclatura de função: O nome válido de uma função criada em um arquivo distinto é o nome dado ao arquivo, não importando se o nome da função é outro;
- Multiplicidade de argumento e retorno: As funções permitem a passagem de inúmeras variáveis como parâmetros, assim como permite o retorno de mais de uma variável.
- Ambiente Matricial: o Matlab considera suas variáveis como sendo matrizes, ainda que unidimensionais.
- Operações Matriciais: por conta do item anterior, as operações matriciais podem ser realizadas automaticamente, observando-se os mesmos princípios da matemática tradicional.
- Vetorização: São utilizados alguns princípios de vetorização em algumas funções, com o objetivo inicial da não utilização excessiva de estruturas de repetição, o que otimiza o código e com objetivo posterior de possível aplicação em paralelização.

Para ilustrar o processo de implementação computacional, será utilizada como exemplo uma estrutura clássica de elemento finito, o L-Frame. Esta estrutura foi escolhida por ser de simples compreensão e aplicação, mas possuir pontos limites ao longo da trajetória de equilíbrio, os quais permitem a observação de todo o processo de análise não linear.

Antes de começar, vale observar as linhas iniciais do código principal, onde podem ser percebidos comandos de inicialização, que são utilizados para a limpeza e preparação do ambiente computacional. Estes comandos são descritos no tópico de ajuda do Matlab, bastando apenas digitar na janela de comando a palavra *help* seguida do nome do comando.

Esta verificação pode ser realizada também para qualquer função previamente definida pelo Matlab. Funções criadas pelo próprio usuário também podem lançar mão deste recurso. Optou-se, neste trabalho pela não utilização do mesmo.

### 3.3 Entendendo a Estrutura

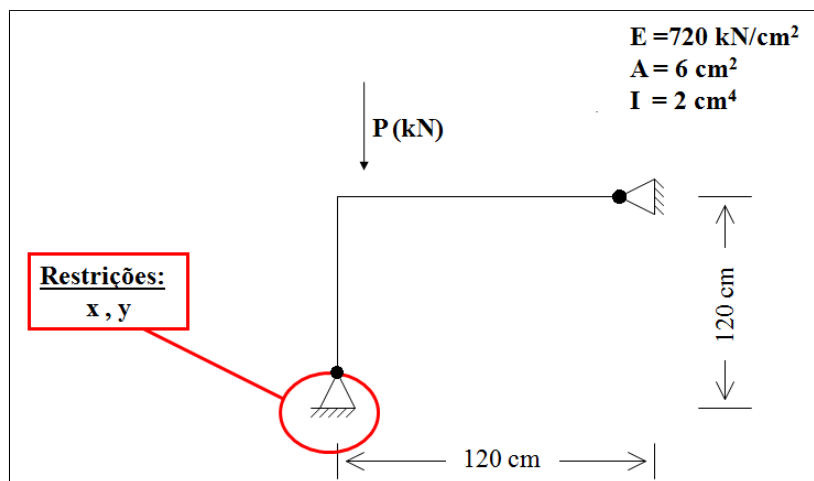


Figura 3.3: Exemplo de L-Frame.

O modelo da estrutura deve conter informações sobre dimensão e apoios da estrutura, as forças aplicadas e propriedades físicas e geométricas da mesma.

Como o tipo de estrutura adotado é o pórtico plano, as dimensões são definidas com base no sistema de coordenadas cartesiano bidimensional (eixo de coordenadas axial  $x$  e transversal  $y$ ).

Os apoios da estrutura admitidos são os apoios usuais, que podem possuir restrição de deslocamento axial, transversal ou à rotação, sendo admissíveis as possíveis combinações entre os mesmos.

Foram utilizadas neste trabalho as aplicações de força pontual axial e transversal, assim como carga de momentos localizados. Vale lembrar que a combinação das mesmas é admissível.

As propriedades físicas e geométricas da estrutura utilizadas neste trabalho são 3: módulo de elasticidade, área da seção e momento de inércia. Essas propriedades podem ser definidas com diferentes valores em partes da estrutura distintas, sendo que todas elas devem ser explicitadas no modelo da estrutura.

Definido o modelo da estrutura, a mesma deverá ser discretizada em elementos finitos, ou seja, dividida em múltiplas partes menores que serão chamadas de elementos, que possuem nós em suas extremidades que definem o seu início e o seu final.

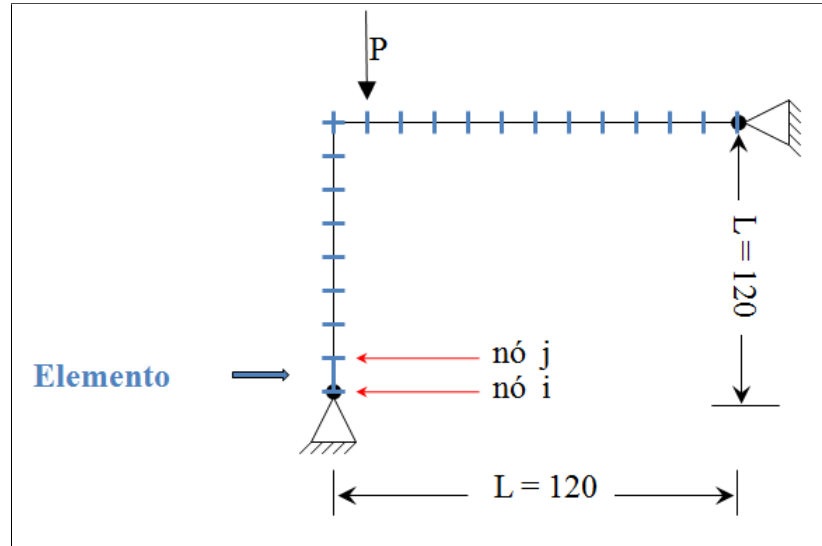


Figura 3.4: Discretização do pórtico L-Frame em 20 elementos finitos.

Para ilustrar, é apresentado na Figura 3.3 um pórtico em L (L-Frame) discretizado em 20 elementos (Figura 3.4), cada um possuindo 2 nós - um  $n_i$  (início) e um  $n_j$  (final) - totalizando uma estrutura com 21 nós. Deve-se observar que um mesmo nó em geral pertence a mais de um elemento, sendo eles os responsáveis pela conectividade da estrutura.

Pode-se observar, na Figura 3.5, que cada elemento possui 6 graus de liberdade, sendo 3 para o nó início e 3 para o nó final. E cada elemento possui o seu próprio sistema local de coordenadas que, usualmente, serão distintos do sistema geral de coordenadas, onde toda a estrutura é definida.

Deve-se observar que um elemento pode possuir características físicas e geométricas em geral distintas dos demais elementos da estrutura, as quais devem estar contidas no modelo estrutural para que a análise seja bem sucedida.

Antes de finalizar esta etapa, vale lembrar que, apesar de os elementos possuírem sistema de eixos de coordenadas próprios, todos os dados no modelo estrutural devem ser definidos a partir do sistema global de coordenadas, pois este vale para toda a estrutura e é com base nele que o arquivo entrada de dados será criado.

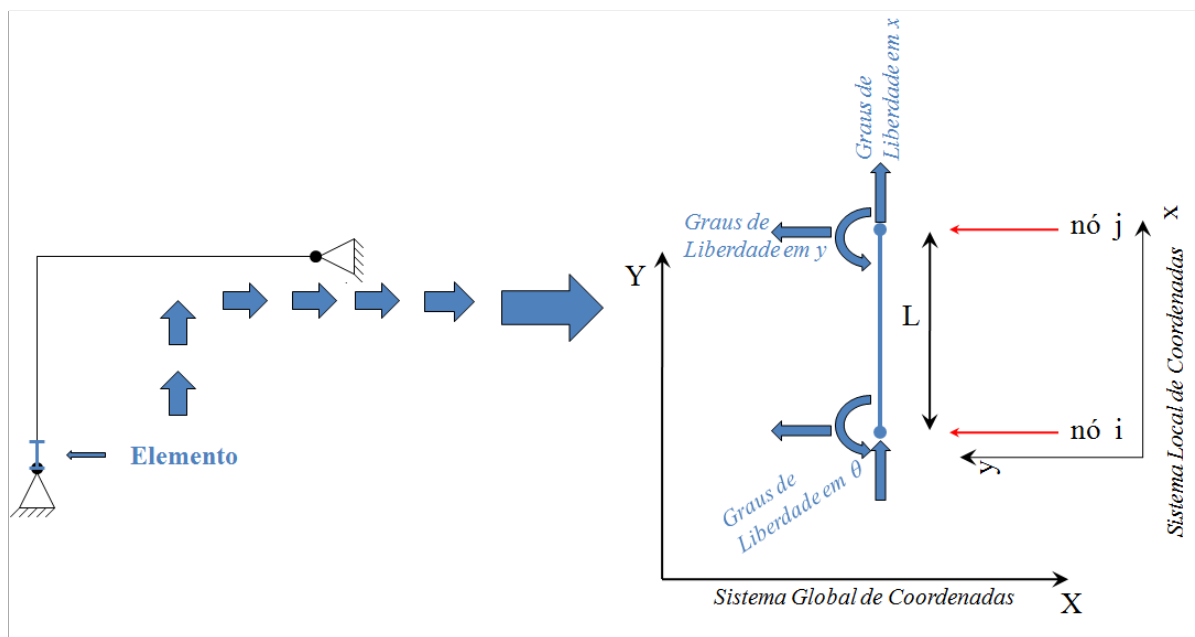


Figura 3.5: Visualização do Elemento Finito.

### 3.4 Entrada de Dados

A definição do arquivo de entrada de dados, relatada na Seção 3.4.1, é de extrema importância, pois é a partir desse arquivo que o código lerá os dados do modelo estrutural definido para a efetuação da análise da estrutura.

Optou-se, neste trabalho, pela definição da entrada de dados através da combinação entre a leitura de arquivo de dados, contendo informações referentes à estrutura a ser analisada, e a digitação, por parte do usuário, dos dados referentes às especificações da realização da análise propriamente dita, como, por exemplo, o parâmetro inicial de carga, a quantidade de incrementos (pontos de equilíbrio) necessários para a realização da análise, entre outros que serão abordados detalhadamente no decorrer deste Capítulo.

Deve-se observar, no arquivo de entrada de dados, que as unidades de medida não são escritas no arquivo de entrada de dados, sendo inseridos apenas os seus respectivos valores numéricos. No código definido neste trabalho, o arquivo de entrada de dados é lido diretamente pela função `LerDadosArquivo`, que separa os grupos de dados nas suas respectivas variáveis, conforme especificações que serão vistas mais adiante nesta Seção.

### 3.4.1 Arquivo de Entrada de Dados

Um arquivo de entrada de dados deve possuir suas localizações bem definidas, de forma a não permitir que dados sejam inseridos em locais errôneos e gerando, conseqüentemente, uma análise equivocada da estrutura. Para tal, foram utilizadas linhas de comentário no decorrer do arquivo que não devem ser removidas, pois elas servem de guia para a inserção dos dados do modelo estrutural no arquivo.

O arquivo de dados utilizado foi definido com base na Tabela 3.1:

Tabela 3.1: Tabela-base de Entrada de Dados.

Qtd. Elementos	Qtd.Nós	Qtd.Apoios	Qtd.Nós carregados		
Id. elemento	Id. Nó início	Id. Nó fim	Módulo de Elasticidade	Área da seção	Momento de Inércia
Id. Nó	Coordenada X	Coordenada Y			
Id. Nó apoiado	Restrição em X	Restrição em Y	Restrição no momento		
Id. Nó carregado	Carga em X	Carga em Y	Momento aplicado		

A Tabela 3.1 define as identificações dos dados que deverão ser inseridos no arquivo texto que servirá de entrada de dados para o código. Com base nela, o arquivo texto vazio poderia ser visualizado como na Figura 3.6:

Onde a primeira linha da Figura 3.6 define:

**Nelem:** A quantidade total de elementos finitos da estrutura

**Nnos:** A quantidade total de nós da estrutura

**Napoio:** A quantidade total de apoios da estrutura

**Ncarreg:** A quantidade total de nós que possuem carregamento na estrutura

**A segunda linha define:**

**Elem:** Identificação numérica do elemento

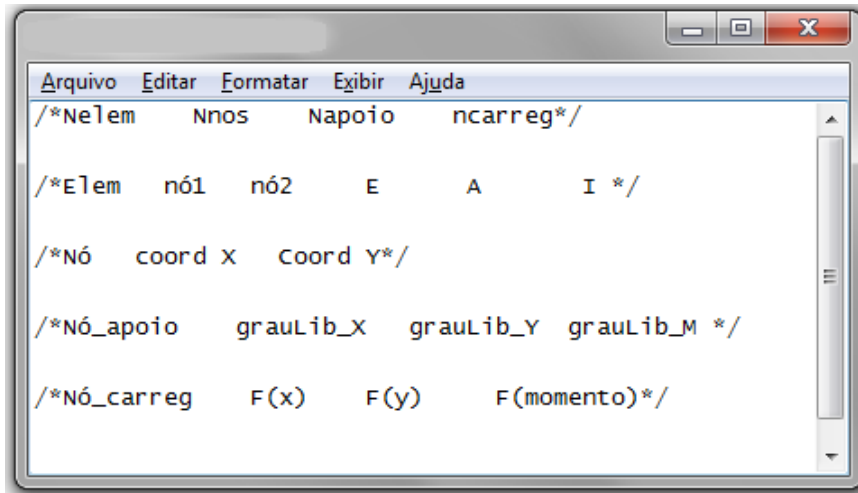


Figura 3.6: Modelo do Arquivo de Entrada de Dados.

**Nó1:** Identificação numérica do nó início do elemento

**Nó2:** Identificação numérica do nó final do elemento

**E:** Módulo de elasticidade do elemento

**A:** Área da seção do elemento

**I:** Momento de inércia do elemento

**A terceira linha define:**

**Nó:** Identificação do nó (em ordem)

**CoordX:** Posição do nó no eixo de coordenada  $x$  do sistema global

**CoordY:** Posição do nó no eixo de coordenada  $y$  do sistema global

**A quarta linha define:**

**NóApoio:** Identificação numérica do nó que possui apoio

**GrauLibX:** Se há restrição de deslocamento ou grau de liberdade em  $x$

**GrauLibY:** Se há restrição de deslocamento ou grau de liberdade em  $y$

**GrauLibM:** Se há restrição ou grau de liberdade na rotação

**A quinta e última linha define:**

**NóCarreg:** Identificação do nó que possui carregamento

**F(x):** Valor da força aplicada no sentido axial do sistema global

**F(y):** Valor da força aplicada no sentido transversal do sistema global

**F(momento):** Valor do momento aplicado

O arquivo preenchido com os dados do modelo estrutural do L-Frame poderia ser visualizado na forma da Figura 3.7.

Deve-se observar na Figura 3.7 que os dados referentes tanto aos elementos quanto aos nós devem ser inseridos ordenadamente, visando maior clareza na geração do arquivo de entrada de dados.

No que concerne aos apoios, foram definidos os valores 0 (zero) para restrição e 1 (um) para ausência de restrição tanto aos deslocamentos, quanto às rotações. Quanto às definições de forças, foram adotados os sinais negativos para as definições dos sentidos invertidos já definidos pelos padrões de engenharia (esquerdo, vertical e horário).

O que permite a entrada de comentários no arquivo de dados é o grupo de caracteres `/**/` sendo necessário, apenas, inserir o texto entre os asteriscos (\*) e sempre iniciados e terminados na mesma linha.

#### 3.4.1.1 Função LerDadosArquivo

Esta função é a responsável pela leitura dos dados de entrada e separação dos mesmos em variáveis que referem-se aos itens relacionados acima.

No código principal, ela é representada por apenas uma linha, na qual a função LerDadosArquivo é chamada sem argumentos de entrada (parâmetros) e as variáveis definidas a seguir recebem o retorno da função.

```
[Nnos,carr,ncarreg,Nelem,conect, Coord,E,A,I,Napioio,r,filme]=LerDadosArquivo();
```

Observando a linha de comando acima, podemos observar que a função retorna para o código principal 12 variáveis referentes à estrutura de pórtico plano. Cada uma delas é descrita a seguir.

- Nnos: Quantidade total de nós da estrutura - Matriz unidimensional (1 x 1)



```

Arquivo  Editar  Formatar  Exibir  Ajuda
/*Nelem Nnos Napoio ncarreg*/
20 21 2 1

/*Elem nó1 nó2 E A I */
1 1 2 720 6 2
2 2 3 720 6 2
3 3 4 720 6 2
4 4 5 720 6 2
5 5 6 720 6 2
6 6 7 720 6 2
7 7 8 720 6 2
8 8 9 720 6 2
9 9 10 720 6 2
10 10 11 720 6 2
11 11 12 720 6 2
12 12 13 720 6 2
13 13 14 720 6 2
14 14 15 720 6 2
15 15 16 720 6 2
16 16 17 720 6 2
17 17 18 720 6 2
18 18 19 720 6 2
19 19 20 720 6 2
20 20 21 720 6 2

/*Nó coord X Coord Y*/
1 0 0
2 0 12
3 0 24
4 0 36
5 0 48
6 0 60
7 0 72
8 0 84
9 0 96
10 0 108
11 0 120
12 12 120
13 24 120
14 36 120
15 48 120
16 60 120
17 72 120
18 84 120
19 96 120
20 108 120
21 120 120

/*Nó_apoio grauLib_x grauLib_Y grauLib_M */
1 0 0 1
21 0 0 1

/*Nó_carreg F(x) F(y) F(momento)*/
13 0 -1 0

```

Figura 3.7: Arquivo de entrada de dados do L-Frame.

- ncarreg: Quantidade total de nós da estrutura que possuem carregamento pontual - Matriz unidimensional (1 x 1)
- Nelem: Quantidade total de elementos da estrutura - Matriz unidimensional (1 x 1)
- Napoio: Quantidade total de apoios da estrutura - Matriz unidimensional (1 x 1)

```

1  function [Nnos,carr,ncarreg,Nelem,conect,Coord,E,A,I,Napoio,r,nomearq]=...
2      LerDadosArquivo()
3  -   nomearq = input('Nome do arquivo de dados: ','s');
4  -   arq = fopen(nomearq,'r');
5  -   [data]=textread(nomearq,'%f','commentstyle','c');
6  -   fclose(arq);
7  -   Nelem = data(1);
8  -   Nnos = data(2);
9  -   Napoio = data(3);
10 -   ncarreg = data(4);
11 -   j=5;
12 -   for i=1:Nelem
13 -       conect(i,1) = data(j+1);
14 -       conect(i,2) = data(j+2);
15 -       E(i)=data(j+3);
16 -       A(i)=data(j+4);
17 -       I(i)=data(j+5);
18 -       j = j+6;
19 -   end
20 -   for i=1:Nnos
21 -       Coord(i,1)= data(j+1);
22 -       Coord(i,2)= data(j+2);
23 -       j = j+3;
24 -   end
25 -   for i=1:Napoio
26 -       r(i,1)=data(j);
27 -       r(i,2)=data(j+1);
28 -       r(i,3)=data(j+2);
29 -       r(i,4)=data(j+3);
30 -       j=j+4;
31 -   end
32 -   for i=1:ncarreg
33 -       carr(i,1)=data(j);
34 -       carr(i,2)=data(j+1);
35 -       carr(i,3)=data(j+2);
36 -       carr(i,4)=data(j+3);
37 -       j = j+4;
38 -   end
39 -   end

```

Figura 3.8: Função LerDadosArquivo.

- conect: Matriz bidimensional que armazena o nó início e nó fim de cada elemento (Nelem x 2). A quantidade de linhas dessa matriz é a quantidade de elementos da estrutura e a quantidade de colunas representa a quantidade de nós do elemento. No caso do pórtico plano, apenas 2.
- Coord: Matriz bidimensional que armazena as coordenadas axiais e transversais dos nós da estrutura (Nnos x 2).
- E: Vetor (Array) que armazena o módulo de elasticidade de cada elemento da estru-

tura (1 x Nelem). Seu tamanho refere-se à quantidade de elementos da estrutura.

- A: Vetor (Array) que armazena a área da seção de cada elemento da estrutura (1 x Nelem). Seu tamanho refere-se à quantidade de elementos da estrutura.
- I: Vetor (Array) que armazena o momento de inércia de cada elemento da estrutura (1 x Nelem). Seu tamanho refere-se à quantidade de elementos da estrutura.
- r: Matriz bidimensional que armazena os graus de liberdade e restrições dos apoios da estrutura (Napoio x 4). A quantidade de linhas é a mesma quantidade de apoios da estrutura. A primeira coluna refere-se ao nó que serve de apoio. as 3 colunas seguintes armazenam, respectivamente, informações referentes às restrições axiais (coluna 2), transversais (coluna 3) e à rotação (coluna 4).
- carr: Matriz bidimensional que armazena as cargas aplicadas a cada nó da estrutura (ncarreg x 4). A quantidade de linhas é a mesma quantidade de nós carregados da estrutura. A primeira coluna refere-se ao nó que será carregado. as 3 colunas seguintes armazenam, respectivamente, informações referentes às cargas axiais (coluna 2), transversais (coluna 3) e momento fletor (coluna 4).
- filme: armazena o nome do arquivo de dados de entrada para posterior geração de gráficos animados, a serem salvos automaticamente.

As variáveis E, A e I são especificadas como vetores pois a estrutura pode ser composta por materiais diferentes para cada elemento, possuindo especificações distintas. Vale observar que a ordem das variáveis no código principal deve ser a mesma definida no corpo da função.

Realizada a interpretação das variáveis retornadas pela função, vamos observar o comportamento da mesma através de seu código.

A função ilustrada na Figura 3.8, após definição da mesma (linhas 1 e 2), começa com a solicitação dos dados ao usuário, com posterior leitura do arquivo de dados e armazenamento dos dados do mesmo na variável local *data* e fechamento do arquivo (linhas 3 a 6). Após esse momento, os dados serão copiados das respectivas posições da variável *data*, que tomou a forma de um vetor coluna com cada dado do arquivo armazenado em uma linha de *data*. O critério de separação de informações utilizado foi o de espaços em branco ou quebras de linha.

Após esses esclarecimentos, pode-se notar que, com exceção das linhas 7 a 10, a função utiliza-se de uma variável de contagem *j* para percorrer as posições do vetor *data* e uma

variável de contagem  $i$  para percorrer os vetores e matrizes das variáveis a ser retornadas, copiando as informações referentes às posições desejadas de *data* para cada uma das variáveis descritas anteriormente.

Devem ser definidos ainda, alguns dados de entrada e realizadas inicializações de algumas variáveis, antes da análise ser, efetivamente, iniciada.

#### 3.4.1.2 Função *CriaAnima*

A função *CriaAnima* é utilizada para a realização da animação das deformações da estrutura. Sua chamada é realizada pelo código principal, antes da entrada no ciclo incremental, podendo ser representada pela linha de comando a seguir:

```
[Xi,Xj,Yi,Yj,Anima,filme,nome]=CriaAnima(Nelem,Coord,conect,r,Napioio, filme);
```

A função *CriaAnima* recebe como parâmetros de entrada as variáveis *Nelem*, *Coord*, *conect*, *r*, *Napioio* e *filme*, retornadas pela função *LerDadosArquivo*. Ela retorna para o código principal as variáveis  $X_i$ ,  $X_j$ ,  $Y_i$  e  $Y_j$ , responsáveis pela definição dos limites da grade da animação. A função retorna também as variáveis *Anima*, que armazenará os frames da animação capturados ao final de cada ciclo incremental, *filme* devidamente atualizada (para casos de necessidade de alteração de endereço de armazenamento) e *nome*, que possui mais alterações que serão utilizadas posteriormente, na geração de gráficos.

Observando o corpo da função na Figura 3.9, percebe-se, na linha 3, a inicialização da variável *margem* com o valor zero, para garantia de entrada na estrutura de repetição compreendida entre as linhas 4 e 27, que será utilizada para a criação da imagem da estrutura indeformada.

As linhas 5 a 8 são utilizadas para solicitar ao usuário que informe as margens da Figura onde a estrutura será exibida. As linhas 9 a 24 são utilizadas para a geração da estrutura indeformada, sendo explicadas em maiores detalhes no tópico 3.8.1.

Após a geração e exibição da estrutura, a linha 25 é utilizada para solicitar ao usuário que confirme se as margens são satisfatórias ou se deseja defini-las novamente. Confirmadas as definições da estrutura a ser animada, na linha 28 é removida a extensão do nome do arquivo. A seguir, seu conteúdo é copiado para a variável *filme1*, que será utilizada, posteriormente na geração dos gráficos da estrutura.

Na linha 30 é inserido um caminho padrão, que deverá ser editado ou excluído pelo

```

1  function [Xi,Xj,Yi,Yj,Anima,filme,filme1]=...
2      CriaAnima(Nelem,Coord,conect,r,Napoio, filme)
3  -   margem=0;
4  -   while (margem==0)
5      Xi=input('Entre com a margem inicial de x: ');
6      Xj=input('Entre com a margem final de x: ');
7      Yi=input('Entre com a margem inicial de y: ');
8      Yj=input('Entre com a margem final de y: ');
9  -   for i=1:Nelem
10     x1 = Coord(conect(i,1),1);
11     y1 = Coord(conect(i,1),2);
12     x2 = Coord(conect(i,2),1);
13     y2 = Coord(conect(i,2),2);
14     x=[x1 x2];
15     y=[y1 y2];
16     plot(x,y,'b','linewidth',2);
17     hold on;
18   end
19   for i=1:Napoio
20     apx(i) = Coord(r(i,1),1);
21     apy(i) = Coord(r(i,1),2);
22   end
23   plot(apx,apy,'ro','linewidth',2);
24   axis([Xi Xj Yi Yj]);
25   margem=input('Redefinir margens?(Sim = 0) ');
26   hold off;
27 end
28 filme=filme(1:length(filme)-4);
29 filme1=filme;
30 filme=strcat('C:\Exemplo de Local\',filme);
31 Anima = avifile(filme);
32 [Anima]=AnimaEst(Nelem, Coord,conect,Anima,r,Napoio,Xi,Xj,Yi,Yj);
33 end

```

Figura 3.9: Função CriaAnima.

usuário para que o arquivo seja salvo corretamente. A linha 31 é responsável pela criação do arquivo do tipo AVI onde a animação será inserida. Essas informações serão salvas na variável *Anima*, que será utilizada na função *AnimaEst* para receber as imagens das deformações da estrutura.

Finalmente, na linha 32 é chamada pela primeira vez a função *AnimaEst*, explicada em maiores detalhes no tópico 3.8.1, para armazenamento da imagem da estrutura ainda indeformada.

### 3.4.1.3 Função LimitesIni

A função *LimitesIni* é utilizada para as definições de entradas de dados que não foram definidas no arquivo e inicializações de variáveis que se tornem necessárias. Sua chamada

pode ser observada no código principal pela linha de comando:

```
[limite,deltaForceInc,incMax,Id,Xacum,Yacum,THETAcum,...
    ret,PMM,ACoord,MATDATA]=LimitesIni(Coord,Nelem);
```

Deve-se observar que as variáveis *Coord* e *Nelem* são passadas como parâmetros de entrada, pois serão utilizadas na definição e inicialização de outras variáveis, que serão criadas no corpo da função e devolvidas ao código principal.

```
1 function [limite,deltaForceInc,incMax,Id,Xacum,Yacum,THETAcum,...
2         ret,PMM,ACoord,MATDATA]=LimitesIni(Coord,Nelem)
3 - limite=input('Entre com o limite para convergencia: ');
4 - deltaForceInc=input('Entre com o incremento INICIAL de força: ');
5 - incMax=input('Entre com a qtde de Incrementos: ');
6 - Id=input('Entre com a qtde de Iterações desejadas: ');
7 - Xacum=0;
8 - Yacum=0;
9 - THETAcum=0;
10 - ret=0;
11 - PMM(1:6,1:Nelem)=0;
12 - MATDATA=[0 0 0 0 0 0 0 0 0 0 0 0];
13 - ACoord=Coord;
14 - end
```

Figura 3.10: Função LimitesIni.

Observando o corpo da função, na Figura 3.10, pode-se observar que o limite para a convergência da estrutura pode ser inserido pelo usuário e armazenado na variável *limite* (linha 3). A função *input*, pré-definida pelo Matlab, recebe, neste caso, um texto que aparecerá na janela de comando contendo a instrução para o usuário digitar o valor desejado. Neste caso, o mesmo será referente ao limite desejado para convergência. Esse valor será armazenado na variável *limite*.

Para as variáveis *deltaForceInc* (variável que define o valor do incremento de força - linha 4), *incMax* (quantidade máxima de passos incrementais - linha 5) e *Id* (quantidade de iterações desejadas - linha 6), o procedimento será, basicamente o mesmo.

As linhas 7 a 12 são utilizadas para inicializar as variáveis com valores nulos, enquanto a linha 13 inicializa a variável *ACoord* (utilizada para armazenar as coordenadas da iteração e do incremento anterior) com os dados das coordenadas iniciais da estrutura, armazenados em *Coord*.

Observando novamente as linhas 7 a 9, deve-se perceber que as variáveis *Xacum*, *Yacum* e *THETAcum* serão utilizadas no armazenamento dos deslocamentos acumulados

da estrutura, cuja atualização será abordada posteriormente. A variável *ret*, utilizada como retorno que define o tipo de saída dos laços de iteração e incremento, também é inicializada com valor igual a zero, visando a garantia de entrada no ciclo iterativo.

Após a leitura dos dados de entrada, deve-se definir o Vetor de Restrições, a Matriz de Graus de Liberdade e o Vetor de Cargas de Referência, pois essas três estruturas não são modificadas ou atualizadas durante o processo de análise da estrutura. Sendo assim, é razoável que elas sejam montadas antes da entrada no ciclo incremental.

### 3.4.2 Definição do Vetor de Restrições

O vetor de restrições segue, basicamente, as mesmas regras do vetor de forças. Ele define as restrições de deslocamento e rotação sofridas em cada nó dos elementos da estrutura. No caso de um pórtico plano, são definidas 3 delas: Restrição ao deslocamento axial ( $x$ ), restrição ao deslocamento transversal ( $y$ ), e restrição à rotação ( $\theta$ ). Onde há restrição, o valor a ser definido no preenchimento do vetor será zero, ao passo que onde não houver restrição, o valor inserido será 1.

A Tabela 3.2 ilustra a sequência dos valores a serem inseridos no vetor de restrições:

Tabela 3.2: Tabela de geração do Vetor de Restrições.

Identificação do nó	Tipo de Restrição
nó 1	Restrição em $x$
	Restrição em $y$
	Restrição em $\theta$
nó 2	Restrição em $x$
	Restrição em $y$
	Restrição em $\theta$
nó 3	Restrição em $x$
	Restrição em $y$
	Restrição em $\theta$
...	...
nó N	Restrição em $x$
	Restrição em $y$
	Restrição em $\theta$

O vetor de restrições, então, é criado através da inserção desses valores em ordem definida tanto de nós como das restrições aos deslocamentos e rotações (como mostrado na Tabela 3.2).

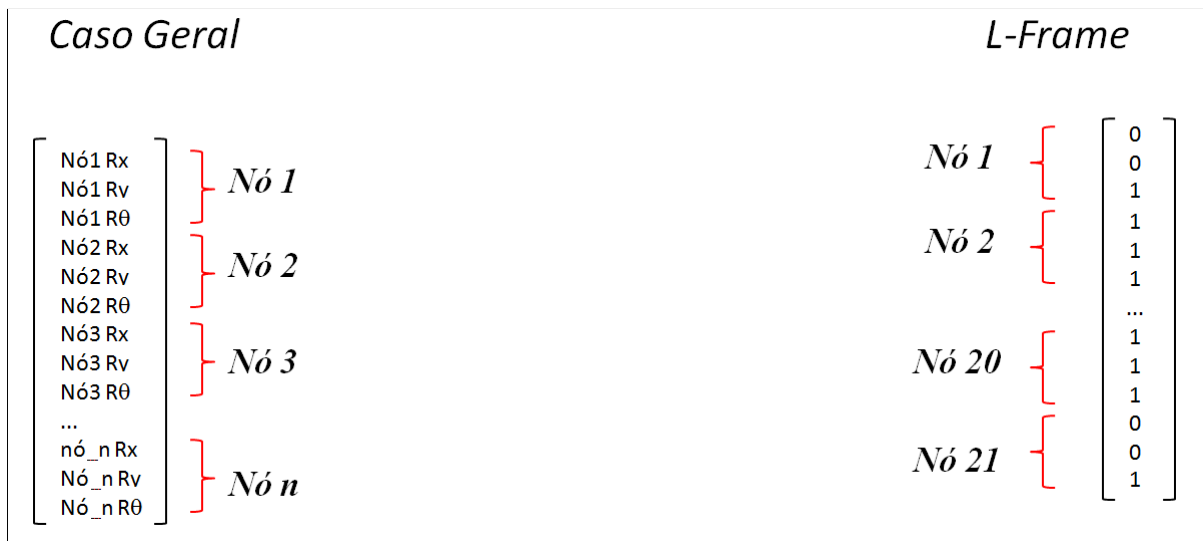


Figura 3.11: Caso Geral de um Vetor de Restrições e Exemplo aplicado ao L-Frame.

Pode-se observar na Figura 3.11 que, no caso do L-Frame, como foram definidos 2 engastes (restrição em  $x$ ,  $y$  e  $\theta$ ) como apoios no primeiro e último nós (nó 1 e nó 21), todas as outras posições do vetor serão preenchidas com 1 (valor referente à ausência de restrição). Este vetor sempre terá o mesmo tamanho do vetor de forças, sendo representadas as reticências do vetor (...) pelos mesmos motivos.

### 3.4.2.1 Função CriaVetGrauLib

A função *CriaVetGrauLib*, assim como *CriaVetForce*, serve para a criação e preenchimento do vetor, neste caso, de restrições com os valores referentes ao arquivo de entrada de dados. A chamada da função pelo código principal pode ser vista a seguir,

```
VgrauLib=CriaVetGrauLib(Nnos,r,Napoio);
```

onde a variável *VgrauLib* recebe o vetor de restrições retornado pela função. Seus argumentos são as variáveis *Nnos* (quantidade de nós da estrutura), *r* (matriz de restrições dos nós) e *Napoio* (quantidade total de apoios da estrutura).

Como pode ser observado na Figura 3.12, seu funcionamento é praticamente o mesmo da função *CriaVetForce* (Figura 3.15), com exceção da linha 5, onde o vetor é preenchido com valores unitários, e não zerado como o vetor de forças. Isso ocorre porque os laços de repetição *for* localizam para preenchimento com os dados do arquivo apenas as posições do vetor que se referem aos nós que possuem restrições.



```

1  function [GrauLib]=CriaVetGrauLib(Qtdnos,VetGrauLib,QtdApoio)
2  -   qtdGrauLib=length(VetGrauLib(1,:))-1;
3  -   cont=-qtdGrauLib+1;
4  -   d = Qtdnos*qtdGrauLib;
5  -   GrauLib(1:d) = 1;
6  -   GrauLib = GrauLib';
7  -   for i=1:QtdApoio
8  -   -   for j=1:qtdGrauLib
9  -   -   -   GrauLib(qtdGrauLib*VetGrauLib(i,1)+cont) = VetGrauLib(i,j+1);
10 -   -   -   cont=cont+1;
11 -   -   end
12 -   -   cont=-qtdGrauLib+1;
13 -   end
14 - end

```

Figura 3.12: Função CriaVetGrauLib.

No caso do vetor de restrições, ao contrário do vetor de forças de referência, as posições que dizem respeito aos nós que não possuem restrições devem ser preenchidas automaticamente com o valor 1, o que justifica a alteração da linha 5 da função. Isso se revelará de extrema utilidade no momento de criação da matriz de graus de liberdade.

Quanto ao restante da função, sua estrutura equivale à função *CriaVetForce*, descrita anteriormente (ressalvados os nomes das variáveis). Pode ser definida uma única função para a geração desses vetores. No entanto, por motivos didáticos, optou-se por manter as duas funções distintas.

### 3.4.3 Matriz de Graus de Liberdade

A matriz de graus de liberdade é criada utilizando-se as informações do vetor de restrições. Ela é de grande importância, pois seus dados servirão de base para a inserção das informações da matriz  $K^e$  na matriz  $K$ .

A montagem da matriz de graus de liberdade é feita inserindo-se na primeira coluna os nós dos elementos e nas colunas seguintes os valores adaptados das restrições aos deslocamentos axial, transversal e à rotação (respectivamente).

Uma modificação, no entanto, deve ser realizada: as posições referentes às restrições que possuírem valores diferentes de zero deverão ser inseridas através de um incremento sequencial unitário, iniciado a partir do primeiro valor 1 até o último valor diferente de zero do vetor de restrições, ordenadamente. Isso ocorre porque a matriz se comporta como uma espécie de "contador" dos graus de liberdade de todo o sistema, que definirá linha e coluna das células da matriz de rigidez do elemento  $K^e$  e dimensão da matriz de

Tabela 3.3: Tabela de Graus de Liberdade.

Nó	GL X	GL Y	GL $\theta$
1	1GLX	1GLY	1GL $\theta$
2	2GLX	2GLY	2GL $\theta$
3	3GLX	3GLY	3GL $\theta$
...	...	...	...
N	NGLX	NGLY	NGL $\theta$

rigidez da estrutura  $K$ .

Tabela 3.4: Exemplo de Tabela de Graus de Liberdade do L-Frame.

Nó	GL X	GL Y	GL $\theta$
1	0	0	1
2	2	3	4
3	5	6	7
...	...	...	...
20	56	57	58
21	0	0	59

Repare na Tabela 3.4 que a última posição diferente de zero da matriz possui o valor 59, que é exatamente a quantidade de graus de liberdade do sistema subtraído da quantidade de restrições. Este valor define a dimensão da matriz global do sistema ( $K$ ), onde serão inseridos os valores da matriz global do elemento ( $K^e$ ). Ambos serão descritos na Seção 3.4.3.1.

### 3.4.3.1 Função `criaMtzGrauLib`

A função `criaMtzGrauLib` é a responsável pela criação da Matriz de Graus de liberdade. Sua chamada pelo código principal pode ser observada na linha de comando,

```
[MgrauLib1,cont]=criaMtzGrauLib(Nnos,VgrauLib);
```

onde a função `criaMtzGrauLib` recebe como argumentos a variável  $Nnos$ , que armazena a quantidade total de nós da estrutura e o vetor  $VgrauLib$ , criado na Seção 3.4.2. A função retorna para o código principal a matriz  $MgrauLib$ , contendo as informações da matriz de graus de liberdade do sistema, e a variável `cont`, que guarda o valor da quanti-

dade total de graus de liberdade do sistema, o que equivale à última posição diferente de zero da matriz  $M_{\text{grauLib}}$ .

Observando o corpo da função na Figura 3.13, pode-se notar que ele se divide em duas etapas:

```

1  function[MgrauLib1,cont]=criaMtzGrauLib(Nnos,VgrauLib)
2  for i=1:Nnos
3      MgrauLib(i,:)= [i VgrauLib(i*3-2) VgrauLib(i*3-1) VgrauLib(i*3)];
4  end
5  cont=0;
6  for i=1:Nnos
7      for j=2:length(MgrauLib(1,:))
8          if MgrauLib(i,j)==1
9              MgrauLib1(i,j)=MgrauLib(i,j)+cont;
10             cont=cont+1;
11         end
12     end
13 end
14 MgrauLib1(:,1)=MgrauLib(:,1)
15 pause
16 end

```

Figura 3.13: Função CriaMtzGrauLib.

A primeira delas é definida no primeiro laço *for* (linhas 2 a 4), que apenas remonta o vetor de restrições em formato matricial, adicionando 1 coluna (a primeira) para a definição dos nós da estrutura.

A etapa seguinte, iniciada na linha 5, que define o momento inicial do contador ( $\text{cont}=0$ ), é seguida por dois laços *for*, sendo o primeiro responsável por percorrer as linhas da matriz e o segundo responsável pelas colunas.

As linhas 8 a 11 da função são marcadas por uma estrutura de decisão. É essa estrutura que permite à função diferenciar o conteúdo da matriz. Observe que, quando o valor encontrado na célula for igual a 1 (linha 8), é adicionado ao valor naquela posição o valor da variável *cont* (linha 9), que após isso, é incrementado de uma unidade (linha 10). Finalizados os laços, resta apenas copiar a primeira coluna da matriz definida na primeira etapa da função para a primeira posição da matriz final, que será retornada para o código principal, juntamente com a variável *cont*.

### 3.4.4 Vetor de Cargas de Referência

O vetor de cargas de referência, ou vetor de forças de referência, define a direção e o sentido das forças e serem aplicadas em cada nó dos elementos da estrutura. No caso de um pórtico plano, são definidas 3 direções: Força axial, aplicada na direção  $X$ , força transversal, aplicada na direção  $Y$ , e Momento Aplicado. Os sentidos são definidos como positivo e negativo. Onde não há força aplicada, o valor a ser definido no preenchimento do vetor será zero. Este vetor também é responsável pela manutenção da proporcionalidade entre os elementos da estrutura.

A Tabela 3.5 ilustra a sequência dos valores a serem inseridos no vetor de forças de referência:

Tabela 3.5: Tabela de geração do Vetor de Forças de Referência.

Identificação do nó	Força Aplicada
nó 1	Força em $x$
	Força em $y$
	Momento Aplicado
nó 2	Força em $x$
	Força em $y$
	Momento Aplicado
nó 3	Força em $x$
	Força em $y$
	Momento Aplicado
...	...
nó N	Força em $x$
	Força em $y$
	Momento Aplicado

O vetor de forças de referência, então, é criado através da inserção desses valores em ordem definida tanto de nós como das forças aplicadas (como mostrado na Tabela 3.5).

Pode-se observar na Figura 3.14 que, no exemplo do L-Frame, como só há forças aplicadas no nó 9, todos os outros espaços do vetor deverão ser preenchidos com zeros. Os espaços entre os valores do nó 2 e 9 e entre os nós 9 e 21 foram ilustrativamente preenchidos com reticências a fim de representar todos os outros nós que deverão ser preenchidos com zeros. Este vetor possuirá, na verdade, 63 posições.

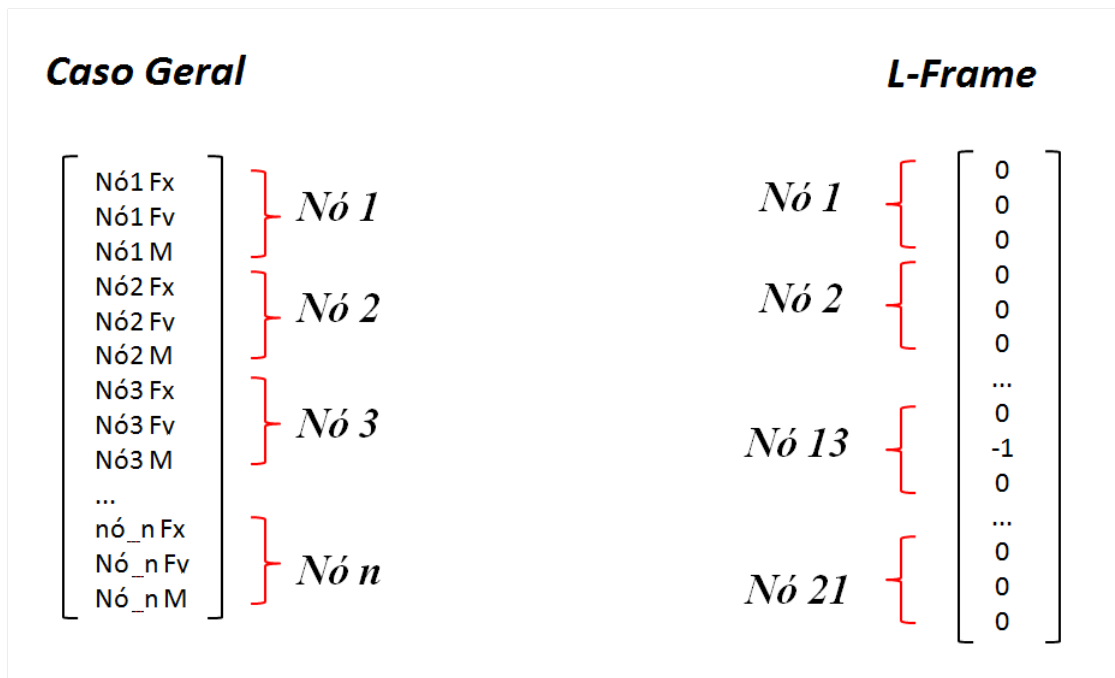


Figura 3.14: Caso Geral de um Vetor de Forças de Referência e Exemplo aplicado ao L-Frame.

### 3.4.4.1 Função CriaVetForce

A função *CriaVetForce* é a responsável pela montagem do Vetor de Forças de Referência. A sua chamada no código principal pode ser observada a seguir,

```
ForceRef=CriaVetForce(Nnos, carr, ncarreg);
```

onde o vetor *ForceRef* recebe o retorno da função. Pode ser observado que a função recebe como dados de entrada as variáveis *Nnos*, *carr* e *ncarreg*, respectivamente responsáveis pelas informações de quantidade de nós da estrutura, especificações de carga e quantidade total de nós carregados. Com apenas essas informações é possível a realização da montagem do vetor de Cargas de Referência.

Observando o corpo da função, ilustrado na Figura 3.15:

A função acima foi criada de forma generalizada, ou seja, com o objetivo de atender a qualquer estrutura, seja 2d ou 3d. Para isso, algumas mudanças no código são necessárias. A variável *qtdCarr* armazena a quantidade de eixos que a estrutura possui, em substituição ao valor 3, referente aos três eixos do pórtico plano: axial, transversal e momento. É definida uma variável *cont*, utilizada para auxiliar no posicionamento correto para a inserção dos valores referenciais de carregamento no vetor *force*, o qual será retornado

```

1  function [force]=CriaVetForce(Qtdnos,vetCarga,Qtdcarreg)
2  -   qtdCarr=length(vetCarga(1,:))-1;
3  -   cont=-qtdCarr+1;
4  -   d = Qtdnos*qtdCarr;
5  -   force(1:d) = 0;
6  -   force = force';
7  -   for i=1:Qtdcarreg
8  -   -   for j=1:qtdCarr
9  -   -   -   force(qtdCarr*vetCarga(i,1)+cont)= vetCarga(i,j+1);
10 -   -   -   cont=cont+1;
11 -   -   end
12 -   -   cont=-qtdCarr+1;
13 -   end
14 - end

```

Figura 3.15: Função CriaVetForce.

para o código principal. A variável  $d$  refere-se ao tamanho do vetor. As linhas 5 e 6 dizem respeito à criação do vetor, preenchimento do mesmo com zeros e transposição do mesmo.

A partir da linha 7, iniciam-se os laços para preenchimento do vetor. O primeiro laço serve para percorrer as linhas da matriz de carregamento, o que equivale a buscar cada nó que possui carregamento. O segundo laço, mais interno, é utilizado na busca das informações referentes aos eixos, que estão armazenadas coluna a coluna, conforme explicado na Seção 3.4.1. Observe que os laços não percorrerão todas as posições do vetor, e sim, todas as linhas da matriz *carr*. Daí a necessidade do preenchimento do mesmo com o valor zero.

Na linha 9 cada valor é, de fato, inserido no vetor de forças. Note que as informações dentro dos parênteses no lado esquerdo da igualdade resultarão em um valor numérico correspondente à cada posição do vetor. Observando agora o lado esquerdo da igualdade, a matriz *vetCarga* (nome referente à matriz *carr* do código principal) é percorrida ,para cada linha, todas as colunas sempre a partir da segunda posição, o que equivale às informações de cargas aplicadas em cada eixo de cada nó da estrutura. Percorridos os laços, o vetor *force* se encontra devidamente preenchido e é retornado ao código principal, sendo seu conteúdo recebido pelo vetor *ForceRef*. Vale lembrar que essa variável se refere ao vetor de cargas de referência,  $F_r$ .

Definidas nas subseções 3.4.2, 3.4.3 e 3.4.4, as variáveis que não sofrerão modificações, pode-se então ser iniciada a fase do ciclo incremental-iterativo, onde todas as análises da estrutura serão efetuadas.

## 3.5 Ciclo Incremental - Solução Preditada

Armazenados os dados do arquivo de entrada nas suas respectivas variáveis e definidos os vetores e matrizes impassíveis de modificações, deve-se agora tratar esses dados de maneira que a solução predita seja alcançada.

A solução predita é a primeira parte do ciclo incremental, que é definido pela entrada no laço *for* do código principal, cuja condição de saída do laço é definida pelo valor de incremento máximo dado pelo usuário. A linha de comando abaixo ilustra a entrada no laço responsável pelo ciclo incremental do código principal.

```
for inc=1:incMax
```

Repare que se deduz, pela linha de comando acima, que a entrada neste laço define o primeiro passo incremental do ciclo. No entanto, deve-se observar que, dadas as condições particulares pertinentes ao primeiro incremento da análise, neste código a primeira entrada no ciclo incremental refere-se, na realidade, ao segundo passo incremental da análise. A partir daí, enquanto a variável *inc* não alcançar o valor de *incMax*, o ciclo incremental se repetirá, com todas as atualizações pertinentes ao código, que serão abordadas posteriormente.

Para o cálculo da solução predita, torna-se necessária a montagem da Matriz de Rigidez da Estrutura. Realizada esta tarefa, deve-se calcular o incremento de carregamento para a montagem do Vetor de Cargas Aplicadas, também conhecido como Vetor de Forças Externas, onde serão inseridas as informações referentes às cargas aplicadas à estrutura. O Vetor de Restrições foi utilizado para a criação da Matriz de Graus de Liberdade e servirá para a inserção e remoção das condições de contorno do problema durante toda a análise da estrutura.

As Matrizes de Rigidez de cada elemento (que serão realizadas no sistema local de cada elemento) serão calculadas para, a seguir, serem rotacionadas para o sistema de referência global e montar a Matriz de Rigidez do Sistema, o que só será possível com o auxílio da Matriz de Graus de Liberdade.

Realizada esta tarefa, poderemos, finalmente, calcular o vetor de deslocamentos e inserir as condições de contorno no mesmo, obtendo assim, a solução predita do problema, ou seja, a primeira estimativa de deformação da estrutura.

### 3.5.1 Matriz de Rigidez do Elemento - $K^e$

A matriz de rigidez do elemento  $K^e$ , como visto no Capítulo anterior, será constituída, neste trabalho, pela soma das matrizes  $K_L^e$  e  $K_r^e$ , referentes às parcelas linear e não linear do elemento. Elas também são conhecidas como matriz de rigidez linear e matriz de tensões do elemento. O resultado da soma dessas duas matrizes será rotacionado para o sistema de referência da estrutura (conhecido como sistema de referencia global) definindo, finalmente, os valores de  $K_{gl}^e$ , que representa a Matriz de Rigidez do Elemento no Sistema Global.

A Matriz de Rigidez Linear guarda as informações referentes às propriedades físicas e geométricas do elemento no sistema local de coordenadas do mesmo. Na análise não linear, ela é definida através da formulação do elemento finito e possui dimensão 6 para pórticos planos.

Pode ser definida como uma função do módulo de elasticidade, momento de inércia, área da seção e comprimento do elemento, possuindo variações relativas apenas a esses valores. A equação (3.23) representa uma matriz de rigidez de um elemento qualquer, cuja definição foi abordada no Capítulo anterior.

#### 3.5.1.1 Função `criaMtzKL`

A função `criaMtzKL` é utilizada para a criação da parcela linear,  $K_L^e$ , da matriz de rigidez do elemento,  $K^e$ . Sua chamada, ao contrário das funções já analisadas, não é realizada pelo código principal. Essa chamada é realizada no corpo da função de criação da matriz de Rigidez da Estrutura  $K$ , cujo funcionamento será abordado mais à frente neste Capítulo. Tal chamada pode ser observada na linha de comando a seguir:

```
[RigLoc]=criaMtzKL(E,A,I,L,i);
```

Pode-se observar que a função recebe como argumentos de entrada os vetores  $E$ ,  $A$  e  $I$  (já conhecidos) e a variável  $L$ , referente ao comprimento do elemento, calculada com base nas coordenadas inicial e final de cada elemento. A variável  $i$  refere-se ao elemento corrente, servindo de índice de posicionamento dos vetores, como pode ser observado na Figura 3.16. O retorno da função será uma matriz, a ser recebido pela variável *RigLoc*, que armazenará os dados já calculados na função.

Observando o corpo da função, percebe-se que a mesma é dividida em duas partes. A



```

1  function [RigLoc]=criaMtzKL(E,A,I,L,i)
2
3  -     e1 = (E(i)*A(i))/L;
4  -     e2 = (12*E(i)*I(i))/(L^3);
5  -     e3 = (6*E(i)*I(i))/(L^2);
6  -     e4 = (4*E(i)*I(i))/L;
7  -     e5 = (2*E(i)*I(i))/(L);
8
9  -     RigLoc = [e1  0  0 -e1  0  0;
10 -             0  e2  e3  0 -e2  e3;
11 -             0  e3  e4  0 -e3  e5;
12 -            -e1  0  0  e1  0  0;
13 -             0 -e2 -e3  0  e2 -e3;
14 -             0  e3  e5  0 -e3  e4];
15 - end

```

Figura 3.16: Função criaMtzKL.

primeira (linha 3 à linha 7) realiza os cálculos referentes às parcelas da matriz definidas pelo elemento finito. A segunda parte (linhas 9 a 14) simplesmente monta as informações na forma da matriz RigLoc, que será retornada para a função CriaMtzK.

### 3.5.1.2 Função criaMtzKt

A função criaMtzKt é utilizada para a criação da matriz de tensões do elemento,  $K_{\tau}^e$ , que será somada à matriz  $K_L^e$  para a obtenção de  $K^e$ . Sua chamada, como no caso da função criaMtzKL, é realizada no corpo da função de criação da matriz de Rigidez da Estrutura,  $K$ , e pode ser observada pela linha de comando a seguir:

```
[TensLoc]=criaMtzKt(P,L,M1,M2,E(i),I(i),A(i));
```

Esta função, ao contrário da função criaMtzKL, não recebe como argumento os vetores  $A$  e  $I$  e sim, apenas as suas posições, definidas pelo índice  $i$ , que neste caso, não é passado como argumento, servindo somente para localizar a posição do vetor, cujo valor será passado como argumento da função. Optou-se por essa variação apenas com o objetivo de se ilustrar as diferentes maneiras de se implementar chamadas de funções em condições semelhantes. As variáveis  $P$ ,  $M1$  e  $M2$  são extraídas da matriz referente às tensões acumuladas na estrutura, armazenada na variável  $PMM$ , também no corpo da função CriaMtzK. Após tratamento dos dados, a função criaMtzKt retorna para a variável TensLoc a matriz de tensões do elemento, cujo cálculo pode ser observado na Figura 3.17, que mostra o corpo da função.

```

1  function [TensLoc]=criaMtzKt (P,L,M1,M2,I,A)
2  -   t1=P/L;
3  -   t2=0;
4  -   t3=-M1/L;
5  -   t4= 6/5*P/L+12*I*P/A/L^3;
6  -   t5=P/10+6*I*P/A/L^2;
7  -   t6=2/15*L*P+4*I*P/A/L;
8  -   t7=-L*P/30+2*I*P/A/L;
9  -   t8=-M2/L;
10
11 -       TensLoc = [t1    t2    t3    -t1    -t2    t8;
12 -                  t2    t4    t5    -t2    -t4    t5;
13 -                  t3    t5    t6    -t3    -t5    t7;
14 -                  -t1   -t2   -t3     t1     t2   -t8;
15 -                  -t2   -t4   -t5     t2     t4   -t5;
16 -                  t8    t5    t7    -t8    -t5    t6];
17 -   end

```

Figura 3.17: Função criaMtzKt.

Observando o corpo da função, percebe-se que ela também é dividida em duas partes. A primeira (linha 3 à linha 9) realiza os cálculos referentes às parcelas da matriz definidas pelo elemento finito. A segunda parte (linhas 9 a 16) simplesmente monta as informações na forma da matriz *TensLoc*, que será retornada para a função *CriaMtzK*.

Definidas as matrizes  $K_L^e$  e  $K_r^e$  e armazenadas em suas respectivas variáveis *RigLoc* e *TensLoc*, a matriz de rigidez do elemento no sistema local será calculada e armazenada em *Kelem*, o que pode ser observado na linha de comando a seguir:

```
Kelem=RigLoc+TensLoc;
```

### 3.5.1.3 Matriz de Rotação do Elemento - $R^e$

A matriz de rotação é utilizada para referenciar a matriz de rigidez do elemento em relação ao sistema global de coordenadas. Cada elemento da estrutura, por ser definido em seu próprio sistema de coordenadas, deve passar por este processo para que todos os elementos da estruturas possuam o mesmo referencial: o sistema global de coordenadas. Essa matriz, assim como a matriz rigidez do elemento, é quadrada e possui dimensão 6. No entanto, a matriz de rotação é função apenas do ângulo de inclinação da barra do elemento em seu sistema local de coordenadas em relação aos eixos do sistema global de coordenadas.

Definindo a representação do seno desse ângulo por  $s$  e o seu cosseno por  $c$ , a matriz de rotação pode ser escrita na forma da Equação (3.29), do Capítulo anterior.

#### 3.5.1.4 Função `criaMtzRot1`

A função `criaMtzRot1` é utilizada na criação das matrizes de rotação a serem utilizadas para a rotação dos elementos do sistema local para o global, e vice-versa. Ela é chamada em vários pontos do código. Uma das chamadas, neste caso, dentro da função `CriaMtzK`, pode ser observada na linha de comando a seguir:

```
[Ra]=criaMtzRot1(cosBeta,senBeta);
```

Neste caso, a variável  $Ra$ , que recebe o retorno da função, armazenará a matriz de rotação que será utilizada para rotacionar o elemento finito para o sistema de referência global. Seus argumentos de entrada são o cosseno e o seno do ângulo formado entre o sistema de Coordenadas local e o global, representados pelas variáveis  $cosBeta$  e  $senBeta$ , respectivamente. Esses valores são calculados nas linhas de comando:

```
cosBeta = (x2-x1)/L;
```

```
senBeta = (y2-y1)/L;
```

onde a parcela  $x2 - x1$  representa o deslocamento no eixo  $x$  e a parcela  $y2 - y1$  representa o deslocamento no eixo  $y$ . A variável  $L$ , como visto anteriormente, representa o comprimento do elemento calculado.

```

1  function [R]=criaMtzRot1(cosBeta,senBeta)
2
3  -      R = [cosBeta senBeta 0          0          0          0;
4          -senBeta cosBeta 0          0          0          0;
5          0          0 1          0          0          0;
6          0          0 0  cosBeta senBeta 0;
7          0          0 0 -senBeta cosBeta 0;
8          0          0 0          0          0          1];
9  -      end

```

Figura 3.18: Função `criaMtzRot1`.

Observando o corpo da função `criaMtzRot1`, representado pela Figura 3.18, podemos observar que ela apenas realiza o posicionamento dos valores na matriz, já que seus argu-

mentos foram passados já calculados. A função apenas realiza a montagem da matriz e a retorna para a função que realizou a chamada, neste caso, sendo a função *CriaMtzK*.

### 3.5.1.5 Matriz de Rigidez do Elemento no Sistema Global - $K_{gl}^e$

A Matriz do Elemento no sistema de referencia da estrutura  $K_{gl}^e$ , ou sistema global, é utilizada para referenciar a matriz de rigidez do elemento (originalmente referenciada pelo sistema de coordenadas local do elemento) em relação ao sistema global de coordenadas.

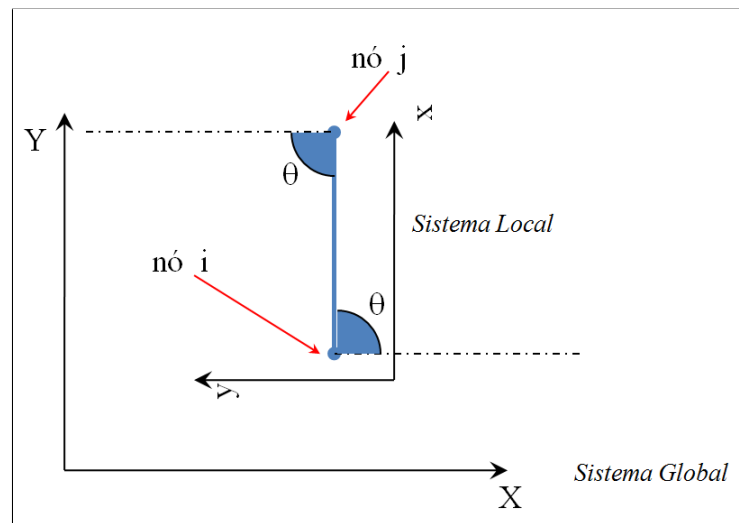


Figura 3.19: Elemento referenciado no Sistema Local e Global.

Repare que é utilizado apenas 1 ângulo na definição da matriz de rotação do elemento, sendo ele definido pela inclinação da barra em relação ao eixo *x* e ao eixo *y* do sistema global.

A matriz  $K_{gl}^e$  é definida através do produto matricial  $K_{gl}^e = (R^T)(K^e)(R)$ , sendo representada pela linha de comando a seguir:

```
Kelem = Ra'*Kelem*Ra;
```

Pode ser observado na linha de comando anterior que a matriz  $K_{gl}^e$ , armazenada na variável *Kelem* é apenas atualizada, pois seu cálculo, juntamente com a matriz de rotação armazenada na variável *Ra* é armazenado novamente na mesma variável, *Kelem*.

Após esta operação, os valores na variável *Kelem* estarão referenciados pelo sistema global de coordenadas e, finalmente todo o processo de criação da matriz de rigidez do

elemento no sistema global,  $K_{gl}^e$ , se conclui. Cada uma dessas matrizes  $K_{gl}^e$  deverá agora, ser inserida na Matriz de Rigidez do sistema de referência Global da estrutura ( $K$ ).

### 3.5.2 Matriz de Rigidez da Estrutura - $K$

#### 3.5.2.1 Dimensão da Matriz de Rigidez da Estrutura - $K$

A Matriz de Rigidez da Estrutura,  $K$  tem sua dimensão definida pelo produto da quantidade total de nós da estrutura pela quantidade total de graus de liberdade de um nó subtraídos da quantidade de restrições do sistema. Ou seja,

$$Dim_K = (QtdNs)(QtdGLN) - RestSist, \quad (3.1)$$

em se tratando de pórticos planos, temos sempre 3 graus de liberdade para cada nó. E no caso deste L-Frame, temos 21 nós na estrutura e duas restrições em cada um dos dois apoios, totalizando 4 restrições, ou seja,

$$59 = (21)(3) - 4, \quad (3.2)$$

Esse resultado possuirá sempre o mesmo valor da última célula diferente de zero da matriz de graus de liberdade, que no caso deste exemplo, obteve o valor 59. Essa simples conta pode ser utilizada para verificação da eficácia da análise, pois a não confirmação dos valores indicará erro no processo da mesma, e deverá ser verificada desde o início.

No código, essa operação é representada pela linha de comando a seguir, que foi inserida logo após o início da função `CriaMtzK`,

```
Rig=zeros(cont);
```

onde a variável `cont`, criada dentro da função `CriaMtzGrauLib` e retornada para o código principal, foi passada como argumento da função `CriaMtzK`, informando a dimensão da matriz de Rigidez do Sistema,  $K$ .

Na linha de comando observada anteriormente, a variável `cont` é passada como argumento da função `zeros` (pré-definida pelo Matlab) cujo objetivo é criar uma matriz quadrada de mesma dimensão do argumento passado para ela. Essa inicialização é necessária, pois onde não forem inseridos valores na matriz de rigidez  $K$ , deverá constar o valor 0 (zero). Pode-se observar, ainda, que a variável `Rig` será utilizada para o armazenamento dos valores da matriz de rigidez da estrutura,  $K$ .

### 3.5.2.2 Inserção de $K_{gl}^e$ em $K$

O primeiro passo para a inserção dos valores de  $K_{gl}^e$  em  $K$  reside em definir o local de destino dos elementos de  $K_{gl}^e$ . Para isso, são utilizados os valores da Matriz de Graus de Liberdade. Deve-se mudar a referência dimensional de  $K_{gl}^e$  de forma que a nova dimensão mostre em que posição de  $K$  o valor da célula será inserido.

Lembrando que cada elemento da estrutura possui um nó início e um nó final definido na matriz de graus de liberdade, e que esses possuem respectivos valores em  $x$ ,  $y$  e  $\theta$ , deve-se substituir os valores que dimensionam  $K_{gl}^e$  (1, 2, 3, 4, 5, 6) pelos valores contidos nas posições  $X_i, Y_i, \theta_i, X_j, Y_j$  e  $\theta_j$ , respectivamente.

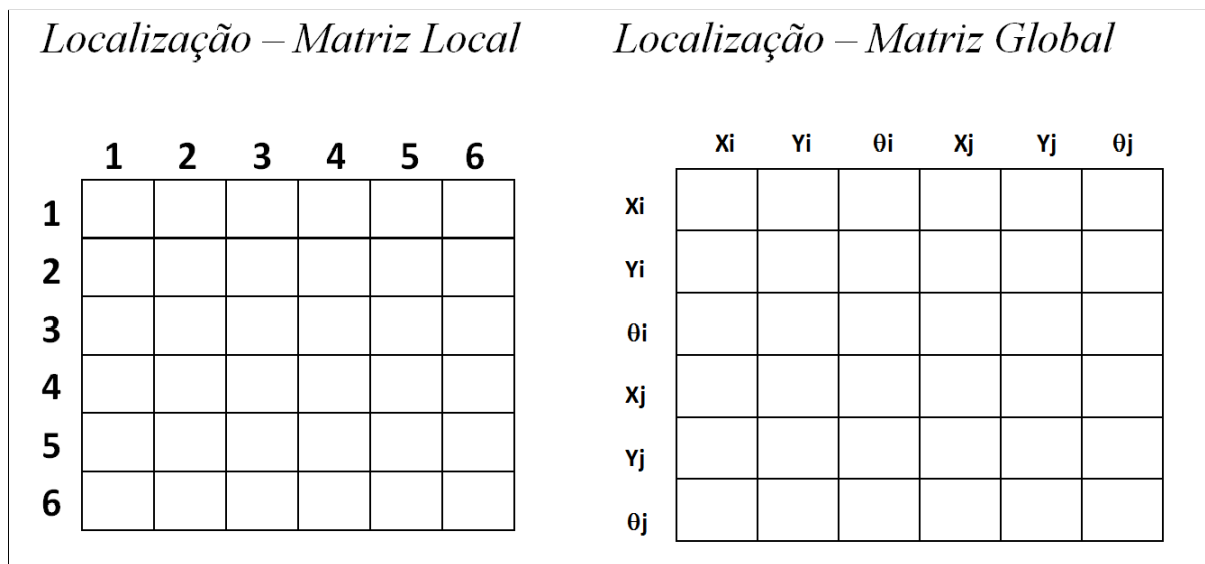


Figura 3.20: Definição da dimensão da Matriz de um elemento qualquer do Sistema Local para o Sistema Global.

Esses valores definirão quais valores de  $K_{gl}^e$  irão para  $K$  e em quais posições de  $K$  eles deverão ser inseridos.

No código, essa definição pode ser observada na linha de comando da função `CriaMtzK`, ilustrada a seguir:

```
vetGrau=[MgrauLib1(conect(i,1),2:length(MgrauLib1(i,:)))...
        MgrauLib1(conect(i,2),2:length(MgrauLib1(i,:)))];
```

A variável `vetGrau` recebe as informações referentes a um vetor de 6 posições, onde as novas referências de dimensão serão inseridas. Repare que a informação é retirada

da variável *MgrauLib1*, que armazena as informações da matriz de graus de liberdade. Essa informação é transferida para a variável *vetGrau* em duas partes: a primeira parte é definida pelo trecho de código,

```
MgrauLib1(conect(i,1),2:length(MgrauLib1(i,:)))
```

onde os índices da variável *MgrauLib* são passados entre parênteses e separados por dois pontos (:). Vale observar que este símbolo representa a notação implícita da estrutura de repetição *for*. A informação à esquerda de(:) determina o primeiro valor a ser inserido e a informação à direita de(:) define o último valor da sequência a ser capturado. O mesmo ocorre com a segunda parte da informação,

```
MgrauLib1(conect(i,2),2:length(MgrauLib1(i,:)))
```

que se refere às informações referentes ao nó 2 da estrutura.

Repare que a variável *conect* é utilizada na primeira parte para a definição do nó início e na segunda parte para a definição do nó fim do elemento finito.

Repare também que *MgrauLib1* define o início da captura pela coluna 2 (pois a linha é definida pela variável *conect*). Isso se deve ao fato de que as informações da primeira coluna dizem respeito a qual nó está sendo utilizado, sendo as informações dos graus de liberdade encontrados a partir da coluna seguinte.

A função *length* é utilizada para se definir a última coluna da matriz na respectiva linha *i* (lembrando que *i* define qual elemento está sendo utilizado).

Portanto, esse vetor possuirá, ao todo, 6 colunas, sendo as 3 primeiras referentes aos graus de liberdade do nó início do elemento e as 3 seguintes referentes aos graus de liberdade do nó fim.

A linha de comando descrita anteriormente foi criada com conceitos de vetorização, com o objetivo da redução da utilização de estruturas de repetição no decorrer do código.

Com base nesses novos valores de dimensão devem ser observadas as condições de contorno do sistema. Essas condições definem que os valores ligados às restrições do sistema não poderão ser alterados. Isso quer dizer que os valores das células onde houver zero nas posições definidas pela nova dimensão não serão inseridos em *K*, ou seja, serão ignorados.

Outro ponto a ser definido é o da ocupação de valores de elementos diferentes na

mesma célula de  $K$ . Quando isso ocorrer, os valores apenas serão adicionados e a célula possuirá, cumulativamente, os valores de mais de um elemento.

No código, as duas situações acima descritas podem ser observadas no trecho de código ilustrado na Figura 3.21.

```

for j=1:length(Kelem)
    for k=1:length(Kelem)
        if vetGrau(j)~=0 && vetGrau(k)~=0
            Rig(vetGrau(j),vetGrau(k))=Rig(vetGrau(j),vetGrau(k))+Kelem(j,k);
        end
    end
end
end

```

Figura 3.21: Código de inserção de  $K_{gl}^e$  em  $K$ .

Observando o código, podemos notar que o primeiro laço de repetição *for* é utilizado para percorrer as linhas da matriz  $K_{gl}^e$ , representada pela variável *Kelem*, enquanto a segunda estrutura de repetição serve para percorrer as colunas da mesma matriz.

Na parte mais interna dos laços pode ser observada uma estrutura de decisão, cuja condição é utilizada para verificar se o índice da linha ou a coluna da nova dimensão da matriz, representada pela variável do tipo array *vetGrau* é igual a zero. Caso qualquer um dos dois seja, a condição não será aceita e a linha de dentro da estrutura de decisão não será executada. Assim, a variável *Rig* não sofrerá alterações.

No entanto, caso ambos os índices de linha e coluna sejam diferentes de zero, a linha de comando dentro da estrutura de decisão será executada e o valor correspondente à  $K_{gl}^e$  será somado ao valor encontrado na posição definida pela nova dimensão (definida pelo vetor da variável *vetGrau*) em  $K$ .

Pode-se observar que esse processo de atualização da matriz de rigidez a cada nova inserção de elemento de  $K_{gl}^e$  se torna possível, entre outros motivos, pela inicialização de toda a matriz com os valores iguais a zero, o que foi realizado no início da função *CriaMtzK*.

Para melhor ilustrar o funcionamento do código observado, utilizaremos como exemplo o primeiro elemento do L-Frame. A dimensão de qualquer elemento de  $K_{gl}^e$  que, por padrão, é 1 2 3 4 5 6, assumiria os valores 0 0 1 2 3 4. Veja o esquema ilustrado na Figura 3.22.





### 3.5.2.3 Função CriaMtzK

Depois de definido todo o processo de criação da Matriz de Rigidez  $K$ , pode-se analisar a chamada da função CriaMtzK pelo código principal. Pode ser observado na linha de código a seguir,

```
[K]=CriaMtzK(Nnos,Nelem,conect,Coord,E,A,I,PMM,MgrauLib1,cont);
```

que a função retorna a matriz de rigidez  $K$  para a variável  $K$ . Para a criação de  $K$  foram necessários ser passadas como argumentos da função as variáveis  $Nnos$ ,  $Nelem$ ,  $conect$ ,  $Coord$ ,  $E$ ,  $A$  e  $I$ , provenientes da leitura do arquivo de dados de entrada pela função LerDadosArquivo. As variáveis  $MgrauLib1$  e  $cont$ , retornadas pela função criaMtzGrauLib, também foram passadas como argumentos. Já a variável  $PMM$ , utilizada para o armazenamento das tensões acumuladas, também é passada como argumento da função CriaMtzK para cálculo da matriz de tensões  $K_{\tau}^e$ , representada pela variável  $Kt$ . Essa variável é inicializada com zeros e suas atualizações são feitas no corpo da função CriaVetForceInt, que será abordado mais à frente.

Observando o corpo da função CriaMtzK, podemos perceber que, após a criação de  $Rig$  (linha 2), é iniciada uma estrutura de repetição que envolve todo o resto do código da função (linhas 3 a 30). Esse laço é utilizado para percorrer cada elemento finito da estrutura, ou seja, durante cada passo do laço os dados de cada elemento são acessados, tratados e sua matriz de rigidez inserida na matriz  $K$ , representada pela variável  $Rig$ .

Depois da entrada no laço, as coordenadas  $x$  e  $y$  dos nós início e fim do elemento são extraídas das variáveis passadas como argumento (linhas 4 a 7) e é calculado, com base nesses dados, o comprimento do elemento (linha 8).

Com base nas informações anteriores, são calculados o cosseno e o seno entre o sistema de referência local e global (linhas 9 e 10), para montagem da matriz de rotação (linha 11).

As linhas 13 a 15 mostram a extração dos dados da matriz  $PMM$  para cálculo da matriz de tensões  $K_{\tau}^e$ , representada pela variável  $Kt$  (linha 17).

Na linha 16 é montada a matriz  $K_L^e$ . Após a criação das parcelas linear e de tensões (linhas 16 e 17), é calculada a matriz do elemento finito no sistema local (linha 18) e rotacionada para o sistema global de coordenadas (linha 19).

Nas linhas 20 e 21 é criado o vetor de posições que definirá a nova dimensão para

```

1  function [Rig]=CriaMtzK(Nelem,conect,Coord,E,A,I,PMM,MgrauLib1,cont)
2  - Rig=zeros(cont);
3  - for i=1:Nelem
4  -     x1 = Coord(conect(i,1),1);
5  -     y1 = Coord(conect(i,1),2);
6  -     x2 = Coord(conect(i,2),1);
7  -     y2 = Coord(conect(i,2),2);
8  -     L = sqrt((x2-x1)^2 + (y2-y1)^2);
9  -     senBeta = (y2-y1)/L;
10 -     cosBeta = (x2-x1)/L;
11 -     [Ra]=criaMtzRot1(cosBeta,senBeta);
12 -     PMMe1=PMM(:,i);
13 -     M1=PMMe1(3);
14 -     M2=PMMe1(6);
15 -     P=PMMe1(4);
16 -     [RigLoc]=criaMtzKL(E,A,I,L,i);
17 -     [TensLoc]=criaMtzKt(P,L,M1,M2,I(i),A(i));
18 -     Kelem=RigLoc+TensLoc;
19 -     Kelem = Ra'*Kelem*Ra;
20 -     vetGrau=[MgrauLib1(conect(i,1),2:length(MgrauLib1(i,:))) ...
21 -             MgrauLib1(conect(i,2),2:length(MgrauLib1(i,:)))];
22 -     for j=1:length(Kelem)
23 -         for k=1:length(Kelem)
24 -             if vetGrau(j)~=0 && vetGrau(k)~=0
25 -                 Rig(vetGrau(j),vetGrau(k))=Rig(vetGrau(j),...
26 -                 vetGrau(k))+Kelem(j,k);
27 -             end
28 -         end
29 -     end
30 - end
31 - end

```

Figura 3.24: Função CriaMtzK.

inserção de  $K_{gl}^e$  nas posições corretas de  $K$ . As linhas 22 a 30 tratam, especificamente, da inserção desses dados e atualização da matriz de rigidez da estrutura (como explicado anteriormente).

Após a criação de  $K$ , a mesma, após inserção na variável  $Rig$ , é retornada pela função CriaMtzK para o código principal.

### 3.5.3 Cálculo do Incremento de Carga

O incremento do carregamento é sempre calculado com base na relação entre a matriz de rigidez do incremento corrente e o vetor de forças de referência. O vetor de deslocamentos tangente obtido através dessa relação, representada pela linha de comando a seguir,

```
URef1=K\FRef;
```

é passado como argumento para a função `CalcIncForce`, juntamente com outros dados, para a definição do próximo incremento de carregamento.

Deve-se atentar para o sinal de divisão invertido `\`, o qual faz com que a operação na linha de comando anteriormente apresentada se comporte como a solução da equação  $(K)(URef1) = FRef$ , vista anteriormente. Em sua forma geral, a utilização deste operador na forma  $X = A$  representa a solução da equação  $AX = B$  pelo método de eliminação de Gauss, onde  $A$  é uma matriz cheia  $N \times N$  e  $B$  é um vetor coluna. Essas informações são encontradas utilizando a ajuda do Matlab. Deve-se observar que, para uma otimização desse operador e, conseqüentemente, do código, é necessária a utilização da função `sparse` na definição da matriz  $A$ .

### 3.5.3.1 Função CalcIncForce

A chamada da função `CalcIncForce` ocorre no código principal, sendo representada pela linha de comando,

```
[dl,signal,deltaForceInc,deltaForce,URefant]=CalcIncForce...
(URef1,URef10,URefant,dldes,signal,deltaForce,deltaForceInc);
```

Pode-se observar que são passados como argumentos da função as variáveis `URef1`, `URef10` e `URefant`, representando, respectivamente, os vetores de deslocamento tangente do incremento corrente, da estrutura inicial e do incremento anterior. A variável `dldes` representa o comprimento de arco desejado, calculado ao final de cada ciclo incremental. A variável `signal` é responsável pela alteração do sinal do incremento do carregamento, enquanto as variáveis `deltaForce` e `deltaForceInc` representam o incremento de carga acumulado e o incremento de carga, respectivamente.

Observando o corpo da função exibido na Figura 3.25, pode-se perceber que as linhas 3 e 4 definem o valor do comprimento de arco `dl`, que será utilizado posteriormente no cálculo do incremento do carregamento. A linha 5 define o valor do parâmetro `GSP`, armazenado na variável `GSP`. Toda vez que esse valor for negativo, o sinal do incremento de carga, armazenado na variável `signal`, será alterado, como indicado na estrutura de decisão compreendida pelas linhas 9 a 11. De volta à estrutura de decisão referente às linhas 6 a 8, é recalculado o incremento de arco desejado `dldes` apenas se o mesmo

```

1  function [dl, signal, deltaForceInc, deltaForce, URefant]=CalcIncForce...
2      (URef1, URef10, URefant, dlDes, signal, deltaForce, deltaForceInc)
3  -   dl=URef1'*URef1;
4  -   dl=sqrt(dl);
5  -   GSP=(URef10'*URef10)/(URefant'*URef1);
6  -   if dlDes==0
7  -       dlDes=deltaForceInc*dl;
8  -   end
9  -   if GSP<0
10 -       signal=signal*-1;
11 -   end
12 -   deltaForceInc = signal*(dlDes/dl);
13 -   dl=dlDes;
14 -   deltaForce=deltaForce+deltaForceInc;
15 -   URefant=URef1;
16 -   end

```

Figura 3.25: Função CalcIncForce.

for igual a zero, o que pode ocorrer quando, durante o ciclo iterativo, são encontradas raízes complexas durante a implementação da estratégia iterativa de comprimento de arco. Na linha 12, o incremento do comprimento de carga *deltaIncForce* finalmente é calculado. Na linha 13 o valor do comprimento de arco *dl* é passado para a variável *dlDes*, armazenando assim o comprimento de arco desejado. Na linha 14 o incremento de carga acumulado *deltaForce* é atualizado enquanto na linha 15 a variável *URefant* recebe o vetor de deslocamentos tangente corrente. As variáveis *dl*, *signal*, *deltaForceInc*, *deltaForce* e *URefant* devidamente atualizadas são devolvidas ao código principal, o que pode ser observado na primeira linha da função.

### 3.5.4 Vetor de Cargas Aplicadas

Depois de montado o vetor de cargas de referência  $F_r$ , O cálculo do vetor de Cargas Aplicadas  $F_{ext}$  é realizado através do produto do Vetor de Cargas de Referência  $F_r$  pelo escalar  $\lambda$ , que armazena a informação do incremento acumulado de carga durante a análise da estrutura. Isso pode ser observado mais detalhadamente na Figura 3.26.

A sequência dos valores a serem inseridos no vetor de cargas aplicadas é a mesma ilustrada no vetor de cargas de referência, na Seção 3.4.4.

Como a criação do vetor já foi realizada na Seção 3.4.4, resta ao código apenas a realização da operação, observada na linha de comando a seguir,

```
ForceInc=ForceRef*deltaForceInc;
```

$$\begin{array}{l}
 \Delta\lambda * F_r = F_{ext} \\
 \left[ \begin{array}{l}
 \text{Nó1 Fx} \\
 \text{Nó1 Fy} \\
 \text{Nó1 M} \\
 \text{Nó2 Fx} \\
 \text{Nó2 Fy} \\
 \text{Nó2 M} \\
 \text{Nó3 Fx} \\
 \text{Nó3 Fy} \\
 \text{Nó3 M} \\
 \dots \\
 \text{nó n Fx} \\
 \text{Nó n Fy} \\
 \text{Nó n M}
 \end{array} \right] = F_{ext}
 \end{array}
 \qquad
 \begin{array}{l}
 \text{Exemplo:} \\
 \Delta\lambda * F_r = F_{ext} \\
 2.5 * \left[ \begin{array}{l}
 0 \\
 0 \\
 0 \\
 0 \\
 0 \\
 0 \\
 0 \\
 0 \\
 \dots \\
 0 \\
 -1 \\
 0 \\
 0 \\
 0 \\
 0 \\
 0
 \end{array} \right] = \left[ \begin{array}{l}
 \text{Nó1} \\
 \text{Nó2} \\
 \dots \\
 \text{Nó13} \\
 \dots \\
 \text{Nó21}
 \end{array} \right] \left[ \begin{array}{l}
 0 \\
 0 \\
 0 \\
 0 \\
 0 \\
 0 \\
 0 \\
 0 \\
 \dots \\
 0 \\
 -2.5 \\
 0 \\
 0 \\
 0 \\
 0 \\
 0
 \end{array} \right]
 \end{array}$$

Figura 3.26: Cálculo do Vetor de Forças Aplicadas.

onde pode-se notar que o vetor *ForceInc* recebe o resultado da operação entre o vetor de forças de referência *ForceRef* e a variável escalar de incremento de força, *deltaForce*, definida pelo usuário. É o vetor de Cargas Aplicadas que será utilizado, juntamente com a matriz de rigidez, no cálculo do deslocamento da estrutura.

### 3.5.4.1 Função ReduzV2

A função *ReduzV2* serve para reduzir um vetor conforme as condições de contorno aplicadas ao sistema. A sua chamada, neste caso, é feita pelo código principal para a redução do vetor de forças aplicadas, para que se torne possível o cálculo dos deslocamentos. É sabido que a remoção e inserção de valores em vetores, tendo como consequência mudanças em seu tamanho e posição de alguns componentes, podem ocasionar perda de eficiência no algoritmo. No entanto, por motivos didáticos, optou-se pela não utilização desta técnica.

A chamada da função pelo código principal pode ser observada na linha de comando a seguir:

```
[FRed]=ReduzV2(VgrauLib,ForceInc);
```

A função *ReduzV2* recebe como argumentos os vetores de restrições e de forças aplicadas, armazenados nas variáveis *VgrauLib* e *ForceInc*. O resultado retornado pela função é o vetor de forças aplicadas com dimensão reduzida na quantidade de restrições e sem os

valores que ocupavam essas posições.

```

1  - function [forceRed]=ReduzV2 (VetgrauLib,Vet1)
2  -     Vet=Vet1';
3  -     d=length(VetgrauLib);
4  -     d1=d;
5  -     n=0;
6  -     for i=1:d
7  -         if VetgrauLib(i)==0
8  -             if i==1
9  -                 forceRed=Vet (i+1:length(Vet));
10 -                 Vet=forceRed;
11 -                 n=n+1;
12 -                 d=d-1;
13 -             else
14 -                 i=i-n;
15 -                 forceRed=[Vet (1:i-1 ) Vet (i+1:length(Vet))];
16 -                 Vet=forceRed;
17 -                 i=i+n;
18 -                 n=n+1;
19 -                 d=d-1;
20 -             end
21 -         end
22 -     end
23 -     if n==0
24 -         forceRed=Vet1';
25 -     end
26 -     if n==d1
27 -         forceRed=0;
28 -     end
29 -     forceRed=forceRed';
30 - end

```

Figura 3.27: Função ReduzV2.

Observando o corpo da função, ilustrado na Figura 3.27, percebemos que a variável Vet1, que armazena o vetor de forças aplicadas, tem sua transposta copiada para uma variável auxiliar (linha 2) para fins de manipulação dos dados. A seguir, define-se a dimensão do vetor original (linha 3) e seu conteúdo também é copiado para uma variável auxiliar (linha 4). A variável  $n$ , criada para contabilizar quantas restrições existem no decorrer do laço, é inicializada com o valor zero (linha 5).

Da linha 6 à linha 22 é definida a estrutura de repetição que garantirá que cada posição do vetor de restrições seja percorrida. São inseridas nele duas estruturas de decisão.

A primeira, mais externa, é iniciada na linha 7 e finalizada na linha 21 e serve apenas para verificar se a posição corrente possui restrição ( $VetGrauLib==0$ ) ou não. Caso possua, a condição é aceita e entra-se na estrutura de decisão seguinte (linhas 8 a 20).

Esta estrutura é responsável pela verificação da existência de restrição na primeira posição do vetor. Caso sim, entra-se na primeira parte da estrutura (linhas 9 a 12) onde é copiado para a variável  $a$  ser reduzida o vetor de forças sem a primeira posição (linha 9) e copiado de volta para a variável auxiliar que está sendo manipulada (linha 10). Após isso, atualiza-se a variável  $n$ , na qual é incrementado o valor referente à uma restrição localizada (linha 11) e a variável  $d$ , cuja dimensão será decrementada de uma posição (linha 12).

Para todas as outras posições do vetor de restrições, a segunda parte da estrutura de decisão é utilizada (linhas 13 a 19). Pode-se observar, na linha 14, que a variável utilizada para percorrer o laço mais externo é decrementada da quantidade de restrições encontradas até o momento, para localização adequada da posição original alterada para os vetores que estão sendo manipulados e atualizados e, logo após a remoção do valor ela volta ao seu valor original, para continuar percorrendo o laço de onde parou (linha 17). Na linha 15, é realizada operação similar à da linha 9, com a diferença de, neste caso, serem passados para o vetor as posições até antes da posição a ser removida e as posteriores à removida. As outras operações tem o mesmo propósito das operações realizadas na primeira parte da estrutura de condição.

Após a saída do laço, são testadas mais duas condições:

A primeira (linhas 23 a 25) testa o vetor para o caso de ausência de restrições, devolvendo o vetor original para o código principal. A segunda (linhas 26 a 28) testa o vetor para o caso da existência apenas de restrições no vetor, o que acarreta em um retorno de um vetor de forças vazio.

A última linha transpõe o vetor de volta à sua posição original, sendo após essa operação, o vetor retornado ao código principal.

### 3.5.5 Vetor de Deslocamentos

O vetor de deslocamentos é obtido através da equação de equilíbrio entre forças internas e forças externas,

$$K\Delta U = F, \quad (3.3)$$

e seu cálculo no código principal pode ser observado na linha de comando a seguir:



$u = K \setminus F \text{red};$

O sinal de divisão invertido  $\setminus$  faz com que a operação na linha de comando anteriormente apresentada se comporte como a solução da equação  $K\Delta U = F$ , vista anteriormente.

Deve-se lembrar que o cálculo do deslocamento armazenado na variável  $u$  é realizado sem as condições de contorno incluídas, gerando somente os valores de deslocamentos não-nulos da estrutura do sistema. Conseqüentemente, se torna necessária a inclusão das mesmas em um vetor que contenha todas as informações de deslocamento da estrutura.

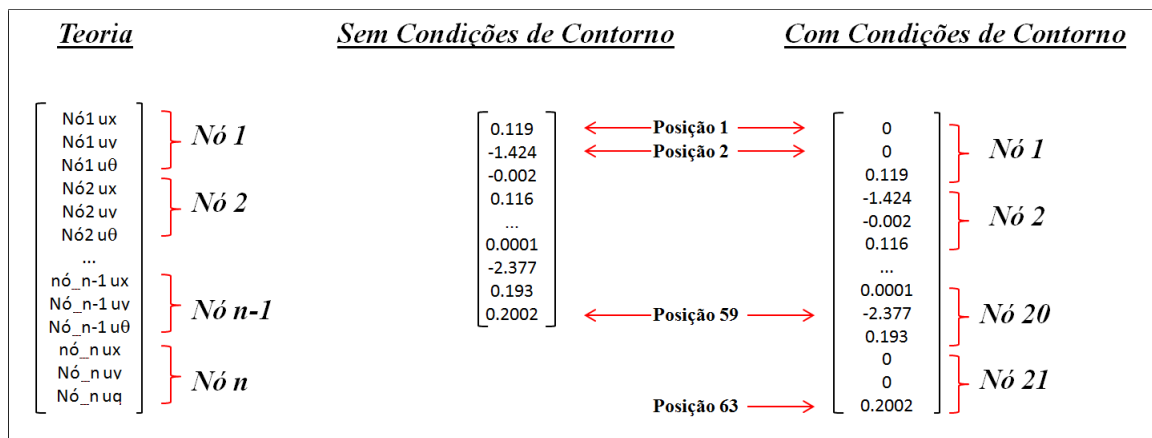


Figura 3.28: Inclusão das Condições de Contorno no Vetor de Deslocamentos.

Para tal, utiliza-se novamente o vetor de restrições para a montagem do vetor de deslocamentos com as condições de contorno incluídas. Basicamente, o processo de montagem envolve a inserção dos valores do vetor  $u$  nas posições do vetor de Restrições que são diferentes de zero, ordenadamente. Deve-se prestar atenção especial ao fato de que, quando as condições de contorno são removidas do sistema, o vetor de deslocamentos não pode ser referenciado pela relação de três posições para cada nó. Veja na Figura 3.28:

Pode ser observada na Figura 3.28 uma sobreposição dos valores dos dois primeiros vetores, gerando finalmente o vetor de deslocamentos  $U$ , que contempla todas as informações de deslocamento do sistema.

### 3.5.5.1 Função MontaVetDes

A função MontaVetDes é a responsável pela inserção das condições de contorno de volta ao vetor de deslocamentos. Sua chamada é realizada pelo código principal, e pode ser observada na linha de código a seguir:

```
U=MontaVetDes(VgrauLib,u);
```

Pode ser observado que a função `MontaVetDes` recebe como argumentos o vetor de restrições e o vetor de deslocamentos sem as condições de contorno incluídas e retorna para a variável  $U$  o vetor de deslocamentos com as condições de contorno incluídas em seus respectivos lugares.

```
1  function [VetU]=MontaVetDes (VetgrauLib,VetDes)
2  -     Vet=VetDes';
3  -     d=length(VetgrauLib);
4  -     n=0;
5  -     for i=1:d
6  -         if VetgrauLib(i)==0
7  -             VetU=[Vet(1:i-1) 0 Vet(i:length(Vet))];
8  -             Vet=VetU;
9  -             n=1;
10 -        end
11 -    end
12 -    if n==0
13 -        VetU=VetDes';
14 -    end
15 -    VetU=VetU';
16 - end
```

Figura 3.29: Função `MontaVetDes`.

Observando o corpo da função, que pode ser visto na Figura 3.29, é criada uma variável para armazenamento da transposta do vetor de deslocamentos (linha 1). A seguir, a função `length` passa para a variável  $d$  o tamanho do vetor de restrições (linha 3), e, por fim, a variável  $n$  é inicializada com o valor zero (linha 4).

A seguir, pode-se observar o laço que será utilizado para percorrer todas as posições do vetor de restrições (linhas 5 a 11). Dentro dele, há uma estrutura de decisão que verifica se na variável corrente há restrição ou não (linhas 6 a 10).

Caso haja restrição ( $VetgrauLib(i) == 0$ ), os comandos dentro da estrutura de decisão serão executados. Neles, pode-se observar que é passado para o vetor todas as posições anteriores à corrente, na posição seguinte é inserido o valor zero e após esse valor são inseridas todas as posições posteriores do vetor de deslocamentos (7). Após esse processo, o vetor é copiado para a variável auxiliar que continuará a ser incrementada com as condições de contorno (linha 8) e a variável de contagem de restrições é acrescida de uma unidade (linha 9).

Finalizado o laço, outra estrutura de decisão testa se foram incluídas restrições ao

vetor (linhas 12 a 14). Caso não tenham havido alterações ( $n == 0$ ), o vetor original é copiado para a variável a ser retornada para o código principal (linha 13). Após essa verificação, o vetor, com as condições de contorno incluídas, é transposto de volta para a sua posição original (linha 15) para ser retornado para o código principal.

Antes da entrada no ciclo iterativo, os valores das variáveis *ret* e *itera*, responsáveis pela definição de tipo de saída do ciclo iterativo e pela contagem de iterações do mesmo, são iguados a zero. As duas linhas de comando a seguir representam a situação anteriormente descrita, finalizando o processo de obtenção da solução predita.

```
ret=0;
```

```
itera=0;
```

### 3.6 Ciclo Incremental-Iterativo

Definida a solução predita, a análise não linear entra na fase iterativa, que consiste na verificação e correção (se necessário) dos deslocamentos obtidos na fase anterior. No código principal, ela pode ser definida pela estrutura de repetição definida na Figura 3.30:

```

27 - while ret==0
28 -     itera=itera+1;
29 -     [Coord,MU]=atualizaCoord(U,Nnos,ACoord);
30 -     [FI,PMM1]=CriaVetForceInt...
31 -         (Nelem,conect,Coord,E,A,I,ACoord,PMM,MgrauLib1,cont,MU);
32 -     [ret,g,norma]=compara(FRef,FI,deltaForce,limite);
33 -     deltaG=K\g;
34 -     if ret==0
35 -         [ret,desl,carga]=arc(URef1,U,deltaG,dl*dl,VgrauLib,ret);
36 -         U=MontaVetDes(VgrauLib,desl);
37 -         deltaForce=deltaForce+carga;
38 -     end
39 - end

```

Figura 3.30: Ciclo Incremental-Iterativo.

A linha de comando a seguir representa a contagem de passagens em um mesmo ciclo iterativo.

```
itera=itera+1;
```

### 3.6.1 Atualização de Coordenadas

Iniciada a contagem de iterações do ciclo iterativo, a primeira tarefa a ser realizada consiste na atualização das coordenadas da estrutura, com base, inicialmente, no deslocamento predito obtido, para que o vetor de forças internas seja criado adequadamente.

#### 3.6.1.1 Função atualizaCoord

A função `atualizaCoord` é utilizada para a atualização das coordenadas da estrutura analisada. Sua chamada é realizada no código principal e pode ser observada na linha de comando que segue.

```
[Coord,MU]=atualizaCoord(U,Nnos,ACoord);
```

Pode-se observar que a função recebe como parâmetros de entrada o vetor de deslocamentos predito  $U$ , a variável  $Nnos$ , que armazena a quantidade de nós da estrutura e a variável  $ACoord$ , que armazena as coordenadas do ciclo incremental anterior. As coordenadas atualizadas são retornadas pela função para a variável  $Coord$ , que armazena as coordenadas atualizadas durante o ciclo iterativo. A variável  $MU$  recebe os dados do vetor de deslocamentos  $U$  reorganizados em formato matricial, como será visto a seguir.

```

1  function [newCoord,MU]=atualizaCoord(U,Nnos,Coord)
2  -  for i=1:Nnos
3  -      MU(i,:)= [i U(i*3-2) U(i*3-1) U(i*3)];
4  -  end
5  -      newCoord=Coord+[MU(:,2) MU(:,3)];
6  -  end

```

Figura 3.31: Função `AtualizaCoord`.

Observando o corpo da função na Figura 3.31, pode-se notar uma estrutura de repetição (linhas 2 a 4) na qual a variável  $MU$  recebe os deslocamentos do vetor de deslocamentos  $U$  reorganizados na forma de uma matriz onde as colunas 1, 2, 3 e 4 referem-se, respectivamente, ao elemento da estrutura e aos deslocamentos axiais, transversais e rotação. A seguir, na linha 5, a variável  $newCoord$  recebe as coordenadas da estrutura atualizadas, resultantes da soma das variáveis  $Coord$  e das posições das colunas 2 e 3 da matriz armazenada em  $MU$ .

Repare que a variável  $Coord$ , contida na função `atualizaCoord`, na verdade recebeu os valores contidos na variável  $ACoord$ , passada como parâmetro da função no código

principal. Vale lembrar, também, que a variável *Coord*, utilizada no corpo da mesma função, existe somente dentro desta função, não correspondendo, portanto, à variável *Coord* utilizada no código principal.

Finalmente, os valores das coordenadas atualizados contidos na variável *newCoord* são retornados pela função para o código principal onde, como dito anteriormente, são armazenados na variável *Coord* existente no código principal. Vale lembrar que a variável *MU* também é retornada para o código principal.

### 3.6.2 Vetor de Forças Internas

Atualizadas as coordenadas, é criado o Vetor de Forças Internas com o objetivo de compará-lo com o vetor de forças Aplicadas. Quanto mais próximos seus resultados, mais próximo da solução exata se encontrará o resultado da análise.

#### 3.6.2.1 Função CriaVetForceInt

A função *CriaVetForceInt* é a responsável pela criação do vetor de forças internas da estrutura. Sua chamada é realizada pelo código principal e pode ser vista na linha de comando a seguir:

```
[FI,PMM1]=CriaVetForceInt(Nelem,connect,Coord,E,A,I,ACoord,PMM,MgrauLib1,cont,MU);
```

Pode-se observar, na linha de comando anterior, que a função recebe como argumentos as variáveis *MgrauLib1*, *E*, *A*, *I*, *Nelem* e *connect*, já conhecidas. As variáveis *Coord* e *ACoord* armazenam os dados das coordenadas da estrutura atualizadas a cada iteração e a cada incremento, respectivamente. A variável *PMM* é uma matriz que armazena os dados das tensões de cada elemento da estrutura a cada passo do ciclo incremental. A variável *cont* armazena a quantidade de graus de liberdade da estrutura e a variável *MU* armazena os deslocamentos atualizados durante o ciclo iterativo.

Pode-se notar que esta função retorna a variável *FI*, que armazena o vetor de forças internas da estrutura e a variável *PMM1*, cuja função é armazenar as tensões da estrutura atualizadas durante o ciclo iterativo.

Observando o corpo da função, representada na Figura 3.32, na linha 3 é criado o vetor de forças internas sem as condições de contorno, preenchido com zeros e armazenado na variável *FiGlobal*. A seguir, a estrutura de repetição que se inicia na linha 4 e termina

```

1  function [FI, PMM1]=CriaVetForceInt...
2      (Nelem, conect, Coord, E, A, I, ACoord, PMM, MgrauLib1, cont, MU)
3  -  FiGlobal(1:cont) = 0;
4  -  for i=1:Nelem
5  -      x1 = Coord(conect(i,1),1);
6  -      y1 = Coord(conect(i,1),2);
7  -      x2 = Coord(conect(i,2),1);
8  -      y2 = Coord(conect(i,2),2);
9  -      L = sqrt((x2-x1)^2 + (y2-y1)^2);
10 -     senBeta = (y2-y1)/L;
11 -     cosBeta = (x2-x1)/L;
12 -     [Ra]=criaMtzRot1(cosBeta, senBeta);
13 -     Ax1 = ACoord(conect(i,1),1);
14 -     Ay1 = ACoord(conect(i,1),2);
15 -     Ax2 = ACoord(conect(i,2),1);
16 -     Ay2 = ACoord(conect(i,2),2);
17 -     AL = sqrt((Ax2-Ax1)^2 + (Ay2-Ay1)^2);
18 -     AsenBeta = (Ay2-Ay1)/AL;
19 -     AcosBeta = (Ax2-Ax1)/AL;
20 -     [Rt]=criaMtzRot1(AcosBeta, AsenBeta);
21 -     Uloc=[MU(conect(i,1),2) MU(conect(i,1),3) MU(conect(i,1),4) ...
22           MU(conect(i,2),2) MU(conect(i,2),3) MU(conect(i,2),4)]';
23 -     Uloc=Rt*Uloc;
24 -     [vetDuNat]=criaVetDuNat(AL, L, Uloc);
25 -     [RigLoc]=criaMtzKL(E, A, I, AL, i);
26 -     PMMe1=PMM(:, i);
27 -     M1=PMMe1(3);
28 -     M2=PMMe1(6);
29 -     P=PMMe1(4);
30 -     [TensLoc]=criaMtzKt(P, AL, M1, M2, I(i), A(i));
31 -     Fi=(RigLoc + TensLoc)*vetDuNat;
32 -     PMM1(:, i)=Fi;
33 -     Fi=Fi+PMM(:, i);
34 -     FiG=Ra'*Fi;
35 -     FiG=FiG';
36 -     vetGrau=[MgrauLib1(conect(i,1), 2:length(MgrauLib1(i, :))) ...
37             MgrauLib1(conect(i,2), 2:length(MgrauLib1(i, :)))];
38 -     for j=1:length(FiG)
39 -         if vetGrau(j)~=0
40 -             FiGlobal(vetGrau(j))=FiGlobal(vetGrau(j))+FiG(j);
41 -         end
42 -     end
43 - end
44 -     FI=FiGlobal';
45 - end

```

Figura 3.32: Função CriaVetForceInt.

na linha 43 trata da criação do vetor de forças internas de cada elemento e montagem dos mesmos no vetor referente à estrutura. Essa estrutura de repetição é utilizada para percorrer todos os elementos da estrutura.

As linhas 5 a 8 são utilizadas para armazenar nas variáveis  $x_1$ ,  $y_1$ ,  $x_2$  e  $y_2$  as coordena-

das atualizadas na iteração anterior do nós iniciais e finais do elemento, respectivamente. Na linha 9 é calculado o comprimento do elemento. As linhas 10 e 11 são utilizadas para o cálculo do seno e do cosseno, respectivamente, do ângulo entre o sistema local atualizado na última iteração e o sistema global enquanto na linha 12 a variável  $Ra$  recebe o retorno da função  $criaMtzRot1$ , explicada anteriormente na Seção 3.5.1.4. Nesse caso, a função cria a matriz de rotação entre o sistema local atualizado na última iteração e o global.

As linhas 13 a 20 possuem as mesmas funções das linhas 5 a 9, sendo que as coordenadas utilizadas são da última iteração do ciclo incremental anterior. Sendo assim, a variável  $Rt$  armazena a matriz de rotação entre o sistema local atualizado no último incremento e o sistema global.

Nas linhas 21 e 22 é montado o vetor de deslocamentos do elemento e seus dados armazenados na variável  $Uloc$ . A seguir, seus dados são rotacionados do sistema global de coordenadas para o sistema local atualizado no último incremento (linha 23). Na linha 24 é chamada a função  $criaVetDuNat$ , responsável pelo cálculo do vetor de deslocamentos naturais, e o vetor resultante armazenado na variável  $vetDuNat$ . maiores detalhes dessa função serão abordados no tópico 3.6.2.2.

A seguir, na linha 25, é chamada a função  $criaMtzKL$  (explicada na Seção 3.5.1.1) para a realização do cálculo da parcela linear da matriz de rigidez. Na linha 26, a variável  $PMMel$  recebe os valores de  $PMM$  referentes ao elemento. Nas linhas 27 a 29 são armazenados nas variáveis  $M1$ ,  $M2$  e  $P$  os valores de tensão referentes ao momento aplicado aos nós 1 e 2 do elemento e tensão axial, respectivamente. Na linha 30, é calculada a matriz de tensões (explicada na Seção 3.5.1.2) e armazenada na variável  $TensLoc$ .

Na linha 31, é realizado o cálculo do vetor de incremento de forças internas no sistema local de coordenadas do elemento e seu resultado armazenado na variável  $Fi$ . A seguir, esses mesmos valores são inseridos na variável  $PMM1$  para fins de atualização durante o ciclo iterativo (linha 32).

Na linha 33 a variável  $Fi$  é atualizada através da realização da soma do incremento da força interna  $Fi$  com o acumulado até o incremento anterior  $PMM$ . Na linha 34,  $Fi$  é rotacionado de volta para o sistema global de coordenadas, através de  $Ra$ . Esse resultado é armazenado em  $FiG$ . Vale lembrar que, como o valor da força interna foi adicionado de um incremento, a sua matriz de rotação deve ser referente ao deslocamento com essa parcela incluída, o que justifica a utilização da matriz de rotação atualizada armazenada em  $Ra$ . Na linha 35,  $FiG$  é transposta para a garantia de cálculos posteriores.

Realizados os cálculos que definem o vetor de forças internas do elemento, o mesmo deve ser inserido no vetor de forças internas da estrutura. Nas linhas 36 e 37 são armazenados, na variável *vetGrau*, os graus de liberdade, extraídos de *MgrauLib1*, do elemento. A seguir, a estrutura de repetição iniciada na linha 38 e finalizada na linha 42, é utilizada para percorrer as posições do vetor de forças internas do elemento. Dentro desse laço, encontra-se uma estrutura de decisão (linhas 39 a 41) utilizada para verificar se o grau de liberdade de cada posição do vetor de forças internas do elemento é ou não igual a zero ( $if\ vetGrau(j) = 0$ ). Pode-se observar, na linha 40, que as regras de inserção do vetor de forças internas do elemento no vetor de forças internas da estrutura são as mesmas utilizadas na montagem da matriz de rigidez.

Após a saída de todas as estruturas de repetição (linha 43), o vetor de forças internas da estrutura (sem as condições de contorno) é transposto e passado para a variável *FI* (linha 44), que é retornada pela função para o código principal, juntamente com a matriz de tensões dos elementos armazenada em *PMM1*.

### 3.6.2.2 Função *criaVetDuNat*

A função *criaVetDuNat*, ao contrário da maioria, é chamada de dentro da função *CriaVetForceInt*, sendo representada pela linha de comando a seguir:

```
[vetDuNat]=criaVetDuNat(AL,L,Uloc);
```

Pode-se observar que a função recebe como argumentos os comprimentos dos elementos no incremento e iteração anterior, respectivamente. Recebe, também, o vetor de deslocamentos *Uloc*.

Observando o corpo da função, representado pela Figura 3.33, pode-se notar, na linha 2, que a variável *d* recebe a dimensão do vetor de deslocamentos do elemento, representado por *vet*, para a definição do vetor de deslocamentos naturais *vetDuNat*, que será preenchido com valores nulos (linha 3). As linhas 4 e 5 são utilizadas para o cálculo e armazenamento do deslocamento axial na posição 4 de *vetDuNat*. Nas linhas 6 a 9, são calculados os dados que serão inseridos na posição 3 da variável *vetDuNat* (linha 10). Na linha 11 é complementado o cálculo para inserção do resultado na posição 6 da variável *vetDuNat* (linha 12). Finalmente, na linha 13, o vetor de deslocamentos naturais armazenado em *vetDuNat* é transposto, para a realização de cálculos futuros, para ser retornado pela função *criaVetDuNat* para a função *CriaVetForceInt*.



```

1  function [vetDuNat]=criaVetDuNat (L1,L2,vet)
2  -   d=length(vet);
3  -   vetDuNat(1:d)=0;
4  -   delta=L2-L1;
5  -   vetDuNat(4)=delta;
6  -   u=vet(4)-vet(1);
7  -   v=vet(5)-vet(2);
8  -   psi=atan(v/(L1+u));
9  -   phi1= vet(3)-psi;
10 -   vetDuNat(3)=phi1;
11 -   phi2=vet(6)-psi;
12 -   vetDuNat(6)=phi2;
13 -   vetDuNat=vetDuNat';
14 -   end

```

Figura 3.33: Função criaVetDuNat.

### 3.6.3 Verificação da Convergência - Função compara

A verificação da convergência consiste na comparação entre a força interna e a força aplicada. Esta comparação é realizada pela função *compara*, cuja chamada é realizada pelo código principal e pode ser observada na linha de comando a seguir:

```
[ret,g,norma]=compara(FRef,FI,deltaForce,limite);
```

Pode-se observar que a função *compara* recebe como argumentos o vetor de forças internas *FI*, assim como o vetor de cargas de referência *FRef* e o carregamento acumulado *deltaForce*, os dois últimos sendo utilizados para a criação do vetor de forças externas. a variável *limite* é passada como argumento da função para definir a tolerância de convergência aceita pela análise para a saída do ciclo iterativo.

```

1  function [ret,g,norma]=compara (Fant,FI,deltaForce,limite)
2  -   Fext=deltaForce*Fant;
3  -   g= Fext-FI;
4  -   norma=norm(g)/norm(Fext);
5  -   if norma<=limite
6  -       ret=1;
7  -   else
8  -       ret=0;
9  -   end
10 -   end

```

Figura 3.34: Função compara.

Observando o corpo da função, representado na Figura 3.34, nota-se que, na linha 2, a variável *Fext* recebe o cálculo do vetor de referência pelo carregamento acumulado.

A seguir, a variável  $g$  recebe o vetor diferença entre a força externa e a força interna (linha 3). A variável  $norma$  recebe a normalização dessa diferença (linha 4), que será comparada ao limite de convergência na estrutura de decisão apresentada nas linhas 5 a 9. Esta estrutura altera o valor da variável  $ret$  de zero para 1 quando a norma for menor que a tolerância (linha 6).

Após a verificação da convergência, são retornadas para o código principal as variáveis  $ret$ ,  $g$  e  $norma$ . A variável  $ret$  é utilizada para a definição de entrada na estrutura de decisão onde as atualizações do incremento de força e do deslocamento são realizadas. A variável  $g$  é utilizada para a criação do vetor de deslocamentos residual  $deltaG$ , conforme representado na linha de comando a seguir:

```
deltaG=K\g;
```

A variável  $norma$  é devolvida pela função para o código principal para ser utilizada após a saída do ciclo iterativo.

### 3.6.4 Atualização Iterativa

A estrutura de decisão representada dentro do trecho de código contido na Figura 3.30 define a etapa de atualização dos incrementos de carregamento e deslocamento que serão utilizados na criação do novo vetor de forças internas. A estratégia iterativa adotada neste trabalho foi a estratégia de comprimento de arco cilíndrico, cuja implementação será abordada a seguir. Esta estratégia é representada pela função  $arc$ , cujo funcionamento será abordado no tópico 3.6.4.1.

Realizados os cálculos, o vetor de deslocamentos retornado pela função, armazenado na variável  $desl$ , é remontado, sendo incluídas suas condições de contorno. A função  $MontaVetDes$ , definida no tópico 3.5.5.1, é a responsável por essa montagem.

A variável  $deltaForce$  é atualizada também, através da soma dela mesma com a variável  $carga$ , retornada pela função  $arc$  para o código principal, possuindo o incremento do carregamento para o processo iterativo.

#### 3.6.4.1 Função $arc$

A função  $arc$  é utilizada para a atualização dos deslocamentos e carregamentos durante o ciclo iterativo. Sua chamada é realizada pelo código principal, podendo ser observada

na linha de comando a seguir:

```
[ret,desl,carga]=arc(URef1,U, deltaG, dl*dl,VgrauLib,ret);
```

Pode-se observar que a função *arc* recebe como argumentos de entrada, a variável *URef1*, definida na fase predita da análise (Seção 3.5.3), o vetor de deslocamentos armazenado na variável *U* e a variável *deltaG* (calculada anteriormente). A função recebe também o comprimento de arco elevado ao quadrado ( $dl * dl$ ), assim como o vetor de graus de liberdade de a variável *ret*, utilizada para verificações referentes à convergência.

A função retorna a variável *ret*, já conhecida, assim como as variáveis *desl* e *carga* que armazenam, respectivamente, o vetor de deslocamentos atualizado e o incremento a ser realizado no carregamento.

```

1  function [ret,desl,carga]=arc(deltaUr,deltaUant, deltaUg,deltaL2,VgrauLib,ret)
2  -   [deltaUant]=ReduzV2(VgrauLib,deltaUant);
3  -   d=length(deltaUant);
4  -   A=deltaUr'*deltaUr;
5  -   B=2*deltaUr'*(deltaUant+deltaUg);
6  -   C=(deltaUant+deltaUg)'*(deltaUant+deltaUg)-deltaL2;
7  -   DELTA=B^2-4*A*C;
8  -   if DELTA>=0
9  -       gama1=(-B+sqrt(DELTA))/(2*A);
10 -       gama2=(-B-sqrt(DELTA))/(2*A);
11 -       deltaU1=deltaUant+deltaUg+gama1*deltaUr;
12 -       deltaU2=deltaUant+deltaUg+gama2*deltaUr;
13 -       cosAng1=(deltaUant'*deltaU1)/(deltaL2);
14 -       cosAng2=(deltaUant'*deltaU2)/(deltaL2);
15 -       if cosAng1>cosAng2
16 -           desl=deltaU1;
17 -           carga=gama1;
18 -       else
19 -           desl=deltaU2;
20 -           carga=gama2;
21 -       end
22 -   else
23 -       ret=2;
24 -       desl=deltaUant;
25 -       carga=0;
26 -   end
27 - end

```

Figura 3.35: Função arc.

Observando o corpo da função, retratado na Figura 3.35, percebe-se, na linha 2, que são removidas as condições de contorno do vetor de deslocamentos *U*, agora conhecido por *deltaUant*, através da função *ReduzV2*, anteriormente abordada no tópico 3.5.4.1. Em seguida, a variável *d* recebe o tamanho de *deltaUant* (linha 3).

As linhas 4 a 6 são utilizadas para o armazenamento nas variáveis  $A$ ,  $B$  e  $C$  os coeficientes a serem utilizados no cálculo da equação do segundo grau utilizada na estratégia de comprimento de arco.

Na linha 7 é armazenado na variável  $DELTA$  o resultado da aplicação da fórmula de Bhaskara. A seguir, uma estrutura de decisão (linhas 8 a 26) é utilizada para a verificação das existência de raízes complexas ( $DELTA < 0$ ). As linhas 22 a 26 dizem respeito às medidas a serem tomadas quando  $DELTA < 0$ . O valor de  $ret$  será alterado para 2, indicando a ocorrência de raízes complexas (linha 23), o que implicará na saída do laço iterativo. A variável  $desl$  armazenará o vetor de deslocamentos  $deltaUant$  (linha 24) e o incremento de carregamento a ser adicionado ao incremento anterior será igualado a zero, o que indica que não haverá alterações no deslocamento ou no carregamento.

As linhas 9 a 14 tratam do restante dos cálculos para as duas raízes obtidas como soluções do problema, armazenadas nas variáveis  $gama1$  e  $gama2$  (linhas 9 e 10), referentes ao incremento de carregamento. As linhas 11 e 12 são utilizadas para a realização do cálculo do deslocamento para as diferentes soluções de carregamento. As linhas 13 e 14 são utilizadas para calcular os cossenos dos ângulos que serão utilizados na escolha de uma das soluções da equação. A estrutura de decisão compreendida entre as linhas 15 e 21 é responsável pela escolha de uma das raízes calculadas. As variáveis de carregamento de deslocamento referentes ao maior cosseno serão utilizadas para a próxima iteração. As variáveis  $ret$ ,  $desl$  e  $carga$  são, finalmente, retornadas para o código principal.

## 3.7 Ciclo Incremental - Atualizações para o próximo incremento

Finalizado o ciclo iterativo, alguns procedimentos devem ser realizados antes do retorno ao início do laço incremental, caracterizando assim a entrada no próximo passo incremental.

### 3.7.1 Função *AtualizaVar*

a função *AtualizaVar* é utilizada para a atualização das variáveis antes do próximo passo incremental. Sua chamada é realizada pelo código principal e pode ser observada pela linha de comando a seguir:

```
[PMM, ACoord, Xacum, Yacum, THETAcum] = . . .
```

```
AtualizaVar(PMM,PMM1,Coord,Xacum,Yacum,THETAcum,U,carr);
```

Observando a linha de comando, pode-se notar que são passadas como argumentos da função as variáveis  $PMM$ ,  $PMM1$ ,  $Coord$ ,  $Xacum$ ,  $Yacum$ ,  $THETAcum$ ,  $U$  e  $carr$ . São retornadas as variáveis  $PMM$ ,  $ACoord$ ,  $Xacum$ ,  $Yacum$  e  $THETAcum$ , devidamente atualizadas.

```

1  function [PMM,ACoord,Xacum,Yacum,THETAcum]=...
2  AtualizaVar(PMM,PMM1,Coord,Xacum,Yacum,THETAcum,U,carr)
3  - PMM=PMM+PMM1;
4  - ACoord=Coord;
5  - Xacum=Xacum+U(carr(1)*3-2);
6  - Yacum=Yacum+U(carr(1)*3-1);
7  - THETAcum = THETAcum + U(carr(1)*3);
8  - end

```

Figura 3.36: Função AtualizaVar.

Observando o corpo da função na Figura 3.36, pode-se perceber na linha 3 que a variável  $PMM$  é atualizada somando-se o conteúdo de  $PMM1$  ao valor já existente de  $PMM$ . Na linha 4, o valor de  $Coord$  é passado para  $ACoord$ . Nas linhas 5 a 7, são acumulados os valores dos deslocamentos axial, transversal e rotação referentes ao primeiro nó carregado descrito no arquivo de entrada (Figura 3.7) e armazenados nas variáveis  $Xacum$ ,  $Yacum$  e  $THETAcum$ , respectivamente.

### 3.7.2 Função fatCorrDLDES

A função *fatCorrDLDES* atualiza o valor do comprimento de arco desejado,  $dldes$ , que será utilizado posteriormente no cálculo do incremento do carregamento no ciclo incremental seguinte. Sua chamada é realizada pelo código principal e pode ser observada na linha de comando a seguir:

```
[dldes]=fatCorrDLDES(ret, limite, norma,Id,itera,dl);
```

Pode-se observar que a função *fatCorrDLDES* recebe como parâmetros de entrada as variáveis  $ret$ ,  $limite$ ,  $norma$ ,  $Id$ ,  $itera$  e  $dl$ . Vale observar a importância da variável  $ret$ , pois é através dela que pode ser detectado se durante o ciclo iterativo houve ocorrência de raízes complexas ( $ret = 2$ ) e também se a estrutura convergiu ( $ret = 1$ ).

A função retorna para o código principal o comprimento de arco desejado devidamente atualizado, que é recebido pela variável  $dldes$ .

```
1 function [dldes]=fatCorrDLDES(ret, limite, norma,Id,itera,dl)
2 -     if ret==1
3 -         fac=1000;
4 -         if itera>1
5 -             fac=sqrt(Id/(itera-1));
6 -         end
7 -     else
8 -         fac=limite/norma;
9 -         if fac>0.5
10 -             fac=0.5;
11 -         end
12 -         if fac<0.1
13 -             fac=0.1;
14 -         end
15 -     end
16 -     dldes=fac*dl;
17 - end
```

Figura 3.37: Função fatCorrDLDES.

O corpo da função ilustrada na Figura 3.37 é iniciado com uma estrutura de decisão (linha 2 à 15) utilizada para a verificação da convergência. Na linha 3, a variável *fac* recebe o valor 1000 que será utilizado no caso de a estrutura ter convergido com apenas uma entrada no ciclo iterativo, ou seja, caso os deslocamentos estejam ocorrendo linearmente.

A seguir, a estrutura de decisão compreendida entre as linhas 4 a 6 é utilizada na maioria dos casos, quando a estrutura convergiu necessitando de mais de uma iteração para tal. Nesses casos, o fator de correção *fac* é calculado na linha 5.

As linhas 7 a 14 referem-se aos casos em que a estrutura não convergiu, ou seja, quando há ocorrência de raízes complexas durante o ciclo iterativo. Outros casos podem, posteriormente, ser adicionados a esta estrutura.

A linha 8 define o fator de correção *fac* como a razão entre *limite* e *norma*. A seguir, são definidas duas estruturas de decisão, que garantirão que esse valor deverá permanecer entre 0.5 (linhas 9 a 11) e 0.1 (linhas 12 a 14).

Finalmente, na linha 16, a variável *dldes* recebe o cálculo que define o novo comprimento de arco desejado, através do produto do fator de correção *fac* pelo comprimento de arco corrente *dl*.

As funções *AnimaEst* e *ArmazenaDados* serão abordadas na Seção 3.8 devido à natureza de pós-processamento e saída de dados das mesmas.

## 3.8 Pós-Processamento

A etapa de pós processamento consiste no tratamento e armazenamento dos dados gerados pela análise para a geração dos resultados da análise. Neste trabalho pode ser gerada uma animação das deformações da estrutura durante os passos do ciclo incremental, para observação do usuário (função *AnimaEst*).

É gerada, também, a matriz *MATDATA* com os dados obtidos durante o processo da análise não linear (função *ArmazenaDados*) e, após a saída do ciclo incremental, seu conteúdo é salvo em arquivo. Finalmente, é utilizada a função *grafico2* que permite a geração de gráficos com base nos dados armazenados na matriz *MATDATA* e na leitura de arquivos externos. Vale lembrar que as funções *AnimaEst* e *ArmazenaDados* são chamadas ao final do ciclo incremental, para garantia da captura dos dados da estrutura já em equilíbrio, enquanto a função *grafico2* é chamada pelo código principal após o encerramento do ciclo incremental.

### 3.8.1 Função AnimaEst

A Função *AnimaEst* é chamada pelo código principal ao final do ciclo incremental, sendo representada pela linha de comando a seguir:

```
[Anima]=AnimaEst(Nelem,Coord,connect,Anima,r,Napoio,Xi,Xj,Yi,Yj);
```

Pode-se observar que a função *AnimaEst* recebe como parâmetros as variáveis *Nelem*, *Coord*, *connect*, *r* e *Napoio*, cujos valores foram obtidos do arquivo de entrada de dados. Já as variáveis *Anima*, *Xi*, *Xj*, *Yi* e *Yj* foram retornadas pela função *CriaAnima* definida na fase de entrada de dados. Após o processamento da imagem para a atualização da animação, o resultado é retornado pela função para a variável *Anima*.

Observando o corpo da função na Figura 3.38, percebe-se que a estrutura de repetição compreendida entre as linhas 2 e 11 é utilizada para percorrer todos os elementos da estrutura. As linhas 3 a 6 são utilizadas para o armazenamento das coordenadas (x,y) dos nós iniciais e finais de cada elemento nas variáveis *x1*, *x2*, *y1* e *y2* que serão organizadas em vetores axiais (linha 7) e transversais (linha 8) a serem utilizados na geração da imagem (linha 9). A linha 10 consiste na utilização de um comando do Matlab utilizado para manter a imagem gerada pela função *plot* (própria do Matlab) quando a mesma função for chamada novamente. Esta linha de comando garante que toda a estrutura seja

```

1  function [Anima]=AnimaEst (Nelem, Coord, conect, Anima, r, Napoio, Xi, Xj, Yi, Yj)
2  -  for i=1:Nelem
3  -      x1 = Coord(conect(i,1),1);
4  -      y1 = Coord(conect(i,1),2);
5  -      x2 = Coord(conect(i,2),1);
6  -      y2 = Coord(conect(i,2),2);
7  -      x=[x1 x2];
8  -      y=[y1 y2];
9  -      plot(x,y,'b','linewidth',2);
10 -      hold on;
11 -  end
12 -  for i=1:Napoio
13 -      apx(i) = Coord(r(i,1),1);
14 -      apy(i) = Coord(r(i,1),2);
15 -  end
16 -  plot(apx,apy,'ro','linewidth',2);
17 -  axis([Xi Xj Yi Yj]);
18 -  hold on
19 -  Fanima = getframe;
20 -  Anima = addframe (Anima, Fanima);
21 -  hold off;
22 -  end

```

Figura 3.38: Função AnimaEst.

representada na imagem e não somente o último elemento.

A estrutura de repetição seguinte (linhas 12 a 15) é utilizada para a geração dos vetores com as coordenadas (x,y) referentes aos apoios da estrutura, que também serão inseridos na Figura (linha 16). A linha 17 é utilizada para garantir que as dimensões da Figura não sejam alterados, pois as variáveis passadas como componentes do vetor passado como argumento da função *axis* (também parte integrante da biblioteca do Matlab) após definidas pelo usuário na função *CriaAnima*, não são alteradas durante a execução do código. A linha 18 é utilizada para garantia da permanência das informações quando a função *getframe* (própria do Matlab) for utilizada para a captura da imagem e armazenamento na variável *Fanima* (linha 19). A seguir, a função *addframe*, também própria do Matlab é utilizada para a inclusão da imagem armazenada em *Fanima* na variável *Anima*, onde são armazenadas as informações do filme. Finalmente, na linha 21, o comando *hold off* é utilizado para liberação das informações utilizadas na geração da imagem.



### 3.8.2 Função ArmazenaDados

A função *ArmazenaDados* é utilizada para o armazenamento de informações referentes às deformações da estrutura ocorridas durante a análise não linear. Sua chamada é realizada ao final do ciclo incremental pelo código principal e pode é representada pela linha de comando a seguir:

```
[MATDATA]=ArmazenaDados(inc,itera,deltaForceInc,...
    deltaForce,Xacum,Yacum,THETAcum,U,carr,Coord,MATDATA);
```

Pode-se observar que os dados a serem gravados na matriz são passados como argumentos da função e retornados pela função para o código principal inseridos na matriz representada pela variável MATDATA.

```
1 function [MATDATA]=ArmazenaDados(inc,itera,deltaForceInc,...
2     deltaForce,Xacum,Yacum,THETAcum,U,carr,Coord,MATDATA)
3 - MATDATA(inc,:)= [inc deltaForce deltaForceInc Coord(length(Coord),2)...
4     Coord(length(Coord),1) U(carr(1)*3-1) U(carr(1)*3-2)...
5     U(carr(1)*3) Yacum Xacum THETAcum itera];
6 - end
```

Figura 3.39: Função ArmazenaDados.

Observando o corpo da função, representado na Figura 3.39, pode-se perceber que há apenas uma linha de comando (representada pelas linhas 3 a 5), onde todas as variáveis são passadas como argumento. Optou-se por esta abordagem para o caso da necessidade de tratamento de dados antes da inserção dos mesmos na matriz.

### 3.8.3 Função grafico2

A função *grafico2* é utilizada para a geração de gráficos, podendo ser incluídas informações exteriores, passadas por arquivos. Nesta função, os valores dos deslocamentos utilizados na geração dos gráficos podem também, complementados por valores utilizados para adimensionalizar os gráficos. Sua chamada é realizada pelo código principal, após a saída do ciclo incremental, sendo representada pela última linha do código, representada a seguir:

```
grafico2(MATDATA,nome);
```

Pode-se observar que a função não possui retorno, ela apenas recebe como argumentos de entrada as variáveis *MATDATA*, já conhecida e *nome*, que possui o nome a ser utilizado quando os gráficos forem salvos. O corpo da função pode ser observado na Figura 3.40.

vários comandos e funções utilizados nesta função, anteriormente abordados durante este capítulo, são utilizados aqui para a geração dos gráficos das trajetórias de equilíbrio das estruturas analisadas, devendo o seu código ser, conforme o caso, alterado para maior eficiência na geração dos gráficos.

Vale observar que esta função pode ser suprimida do código, sendo substituída pelo comando de plotagem de gráficos do Matlab *plot* recebendo como argumentos as posições de coluna da matriz *MATDATA* referentes às trajetórias de equilíbrio axial, transversal ou de rotação combinadas às posições referentes às forças aplicadas (inseridas na mesma matriz).

```

1  function grafico2(MATDATA,nome)
2  -   clc;
3  -   exibe=input('Deseja gerar um gráfico? S ou N--> ','s');
4  -   if exibe=='s'
5  -       contenc=input('DIGITE A VERSÃO PARA GERAÇÃO DE GRAFICOS OU (c)','s');
6  -       if contenc~='c'
7  -           nome=strcat(nome,contenc);
8  -       end
9  -   end
10 -   cont=97;
11 -   nome1=nome;
12 -   while exibe=='s'
13 -       nomeX=input('Digite o nome do rótulo do eixo X do gráfico:', 's');
14 -       nomeY=input('Digite o nome do rótulo do eixo Y do gráfico:', 's');
15 -       filename=input('arquivo para comparação ou N se não houver:', 's')
16 -       if filename~='n'
17 -           cor=input('letra para a cor e traçado dos dados do arquivo:', 's');
18 -           fid = fopen(filename, 'r');
19 -           [X, Y] = textread(filename, '%f %f', 'headerlines', 1);
20 -           fclose(fid);
21 -           plot(X,Y,cor);
22 -           hold on;
23 -           pause;
24 -       end
25 -       GrafX=input('Coluna para eixo x do gráfico:');
26 -       SignX=input('escalar que acompanha os valores do eixo x:');
27 -       GrafY=input('Coluna para eixo y do gráfico:');
28 -       SignY=input('escalar que acompanha os valores do eixo y:');
29 -       cor=input('Entre com a cor e traçado:', 's');
30 -       op=input('Digite (s) para definir os eixos do gráfico', 's');
31 -       if op=='s'
32 -           Xi=input('Entre com a margem inicial de x: ');
33 -           Xj=input('Entre com a margem final de x: ');
34 -           Yi=input('Entre com a margem inicial de y: ');
35 -           Yj=input('Entre com a margem final de y: ');
36 -       end
37 -       t=plot(MATDATA(:,GrafX)*SignX, (MATDATA(:,GrafY)*SignY),cor);
38 -       if op=='s'
39 -           axis([Xi Xj Yi Yj]);
40 -       end
41 -       ylabel(nomeY);
42 -       xlabel(nomeX);
43 -       nome=strcat('C:\Exemplo de Local\',nome1,cont, '.jpg');
44 -       saveas(t,nome);
45 -       nome=strcat('C:\Exemplo de Local\',nome1,cont, '.jpg');
46 -       saveas(t,nome);
47 -       cont=cont+1;
48 -       exibe=input('novo gráfico(N), complementar gráfico(C) ou sair(Q)? ','s');
49 -       if exibe=='n'
50 -           hold off;
51 -           exibe='s';
52 -       end
53 -       if exibe=='c'
54 -           hold on;
55 -           exibe='s';
56 -       end
57 -       if exibe=='q'
58 -           exibe='c';
59 -       end
60 -   end
61 -   hold off;
62 -   close all;
63 -   end

```

Figura 3.40: Função grafico2.

## Capítulo 4

### Validação e Resultados

Neste Capítulo serão apresentados alguns exemplos encontrados na literatura, visando a validação dos resultados do código desenvolvido neste trabalho.

Para a geração desses resultados foi utilizado um notebook Vaio VPC-F12GXB, que possui as seguintes configurações:

Processador Intel(R) Core(TM) i7 CPU Q740 de 1.73GHz 1.73GHz.;

Memória RAM de 6.00 GB;

Placa de vídeo NVIDIA GFORCE with CUDA<sup>TM</sup>;

Disco Rígido de 500 GB.

Os testes foram realizados em um Sistema Operacional Windows 7 home premium 64 Bits, que vem *default* no computador. O software utilizado no desenvolvimento do código foi o Matlab R2010a.

Deve-se observar que todas as análises realizadas com o código desenvolvido neste trabalho utilizaram-se das estratégias incrementais e iterativas de comprimento de arco cilíndrico com estratégia de análise de sinal por parâmetro GSP.

Poderão ser observadas, nas Figuras ilustradas neste Capítulo, as definições de carregamento pontual externo  $P$ , deslocamento axial  $u$ , deslocamento transversal  $w$  e rotação  $\theta$ .

Na Seção 4.1 serão analisados e comparados casos clássicos de exemplos estruturais, cujos resultados numéricos obtidos através deste código serão comparados com resultados analíticos e numéricos encontrados na literatura. Na Seção 4.2 serão apresentados também alguns exemplos fortemente não lineares, cujos resultados podem ser encontrados na

literatura, com o objetivo de comprovar a capacidade de análise do código desenvolvido. Na Seção 4.3 serão apresentados outros exemplos.

Na Seção 4.4 deste Capítulo serão também abordados aspectos computacionais entre o código desenvolvido em Matlab e o código CS-ASA(Silva, 2009), desenvolvido em Fortran. Estas comparações têm o objetivo de se observar as distinções mais relevantes entre os dois tipos de programação.

## 4.1 Exemplos Clássicos

Nesta Seção serão abordados dois exemplos clássicos que possuem solução analítica encontrada na literatura. A Viga Engastada-Livre possui solução analítica apresentada por Timoshenko e Gere (1982). A Coluna Engastada-Livre possui solução analítica apresentada em Southwel(1941). As soluções de ambas serão apresentadas a seguir.

### 4.1.1 Viga Engastada Livre

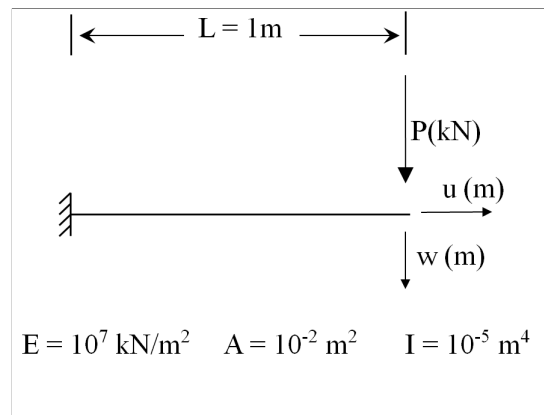


Figura 4.1: Viga Engastada-Livre.

O primeiro exemplo apresentado neste Capítulo tem por objetivo verificar a capacidade da formulação não linear utilizada neste trabalho em resolver problemas com grandes deslocamentos e rotações. Seus resultados analíticos são frequentemente utilizados para validar modelos numéricos.

A Viga Engastada-Livre representada na Figura 4.1 possui comprimento  $L = 1\text{m}$ , sendo discretizada em 10 elementos finitos. A mesma foi submetida a um carregamento pontual vertical negativo  $P$  em sua extremidade livre (nó 11, referente ao último elemento).

A estrutura possui, ainda, módulo de elasticidade  $E = 10^7 kN/m^2$ , área da seção  $A = 10^{-2} m^2$  e momento de inércia  $I = 10^{-5} m^4$ . Foram definidos, para o processo de análise incremental-iterativa, parâmetro inicial de carga  $\Delta\lambda_1^0 = 0.025$ , com tolerância  $\zeta = 10^{-3}$  e quantidade de iterações desejadas  $Id = 2$ .

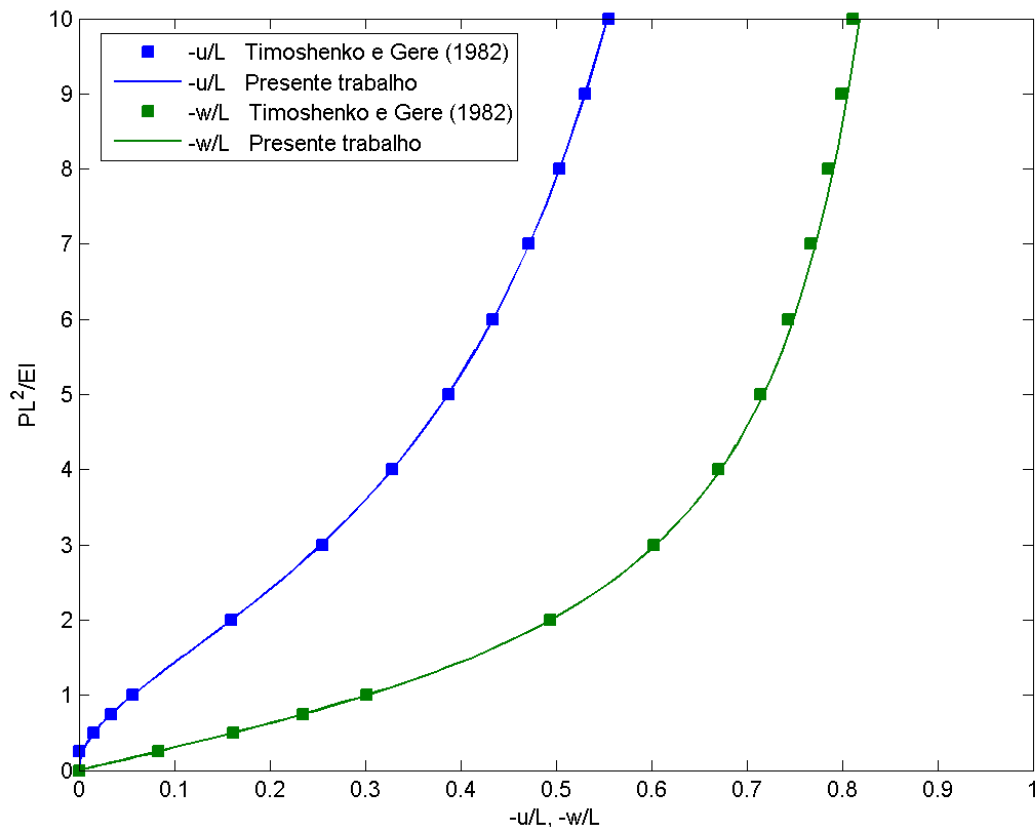


Figura 4.2: trajetória de equilíbrio (10 elem).

Foram necessários 118 incrementos com um tempo de processamento total de  $7.203(10^{-1})$  segundos, neste caso, para a estrutura gerar o gráfico ilustrado na Figura 4.2.

Pode-se observar, no gráfico, os pontos utilizados para comparação dos resultados encontrados por Timoshenko e Gere(1982), cujos valores adimensionalizados podem ser observados na Tabela 4.1. As curvas das linhas referem-se aos dados coletados pelo código desenvolvido neste trabalho, comprovando a eficiência do mesmo.

Pode-se observar, também, que as curvas da trajetória de equilíbrio desta estrutura não apresentam pontos críticos (pontos de bifurcação ou pontos limite). Por este motivo, estes mesmos resultados podem ser obtidos através de implementações de análises não lineares mais simplificadas, como a iteração à carga constante e incremento constante do

$PL^2/EI$	$-u/L$	$-w/L$
0	0	0
0.25	0.0004	0.083
0.5	0.016	0.162
0.75	0.034	0.235
1	0.056	0.302
2	0.16	0.494
3	0.255	0.603
4	0.329	0.670
5	0.388	0.714
6	0.434	0.744
7	0.472	0.767
8	0.504	0.785
9	0.531	0.799
10	0.555	0.811

Tabela 4.1: Dados pontuais: Timoshenko e Gere (1982).

parâmetro de carga, sendo o tratamento de sinal dispensável.

No entanto, deve-se observar que, com a aplicação de estratégias otimizadas, como a estratégia incremental-iterativa de comprimento de arco cilíndrico, a análise pode ser realizada com redução de custo computacional e economia de tempo, como pode ser observado na Tabela 4.2.

<b>Iterações Desejadas</b>	<b>Incrementos</b>	<b>Tempo de Processamento(s)</b>
1	300	1.159
2	118	0.799
3	90	0.751
5	50	0.571
10	30	0.551

Tabela 4.2: Tempo de processamento por Incrementos e Id.

A Tabela 4.2 relaciona o tempo com a variação das iterações desejadas, mostrando que o tempo de processamento pode ser reduzido, assim como a quantidade de incrementos, visando uma análise não linear mais eficiente da estrutura.

O gráfico apresentado na Figura 4.3 indica, além das duas trajetórias de equilíbrio para deslocamento axial e transversal, a curva da trajetória de equilíbrio da rotação  $\theta$ . Para a geração deste gráfico foram utilizadas 10 iterações desejadas, sendo necessários apenas 30 incrementos.

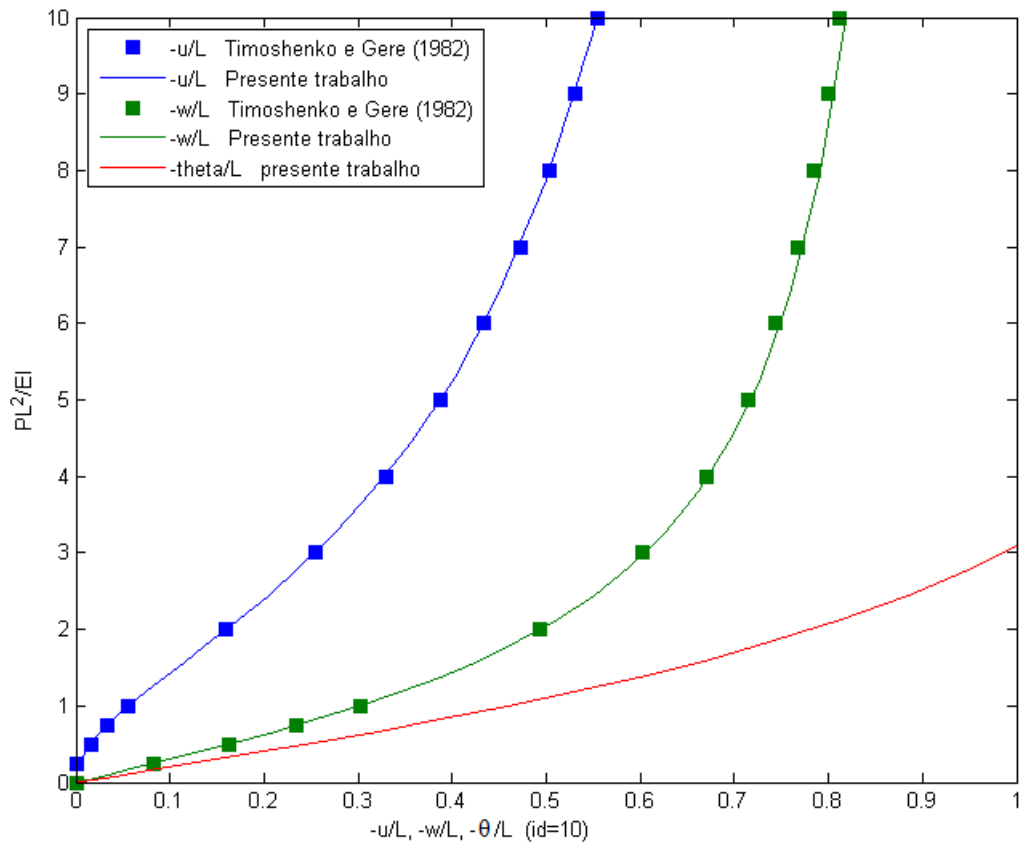


Figura 4.3: trajetória de equilíbrio com 30 incrementos.

Pode-se observar na Tabela 4.2 que a qualidade dos resultados contidos no gráfico é mantida, apesar da redução de 300 (1 iteração desejada) para 30 incrementos (10 iterações desejadas), o que equivale a 1/10 da quantidade de incrementos necessária para a geração das curvas de equilíbrio. Observando novamente a Tabela 4.2 percebe-se, inclusive, uma economia de tempo de processamento de aproximadamente 0.608 segundos, o que equivale à quase metade do tempo necessário para a realização da análise.



### 4.1.2 Coluna Engastada Livre

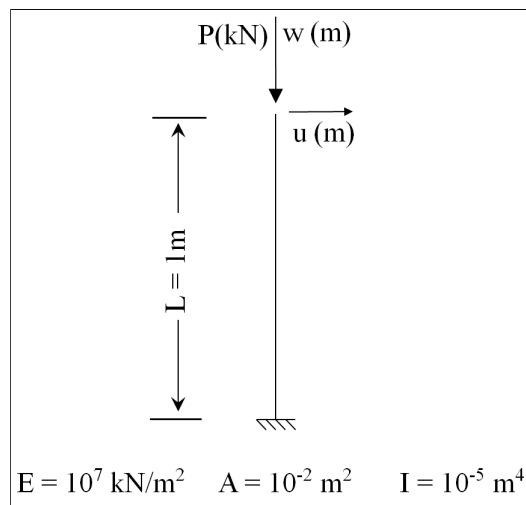


Figura 4.4: Coluna Engastada-Livre.

O exemplo a seguir consiste em uma coluna engastada em sua extremidade inferior e livre na superior, onde há um carregamento pontual vertical negativo,  $P(kN)$ , e comprimento  $L = 1m$ .

O primeiro problema a ser observado, neste caso, é a existência de um ponto de bifurcação. Para eliminar a bifurcação, a estratégia de solução adotada foi inserir na estrutura da coluna discretizada inicialmente em 10 elementos, um elemento de excentricidade de tamanho  $0.0002m$  no sentido axial na extremidade livre da coluna. A estrutura, portanto, passa a ter 11 elementos, sendo sua carga aplicada ao nó livre do elemento de excentricidade, conforme pode ser observado na Figura 4.5. Com isso, um pequeno momento passa a ser associado à carga aplicada, o que levará a coluna a se deformar na direção em que esse elemento foi introduzido.

Pode-se observar melhor, na Figura 4.5, o elemento de excentricidade inserido no topo da coluna, o que não era possível observando-se a Figura 4.4.

As propriedades físicas e geométricas ( $E$ ,  $A$  e  $I$ ) da estrutura são as mesmas definidas no exemplo da Viga Engastada-Livre.

Para o processo de análise incremental-iterativa foram definidos como parâmetro inicial de carga  $\Delta\lambda_1^0 = 0.2$ , com tolerância  $\zeta = 10^{-4}$  e Iterações Desejadas  $Id = 5$ .

Foram necessários 5450 incrementos e gasto um tempo de processamento total de 51.072 segundos para a geração do gráfico observado na Figura 4.6.

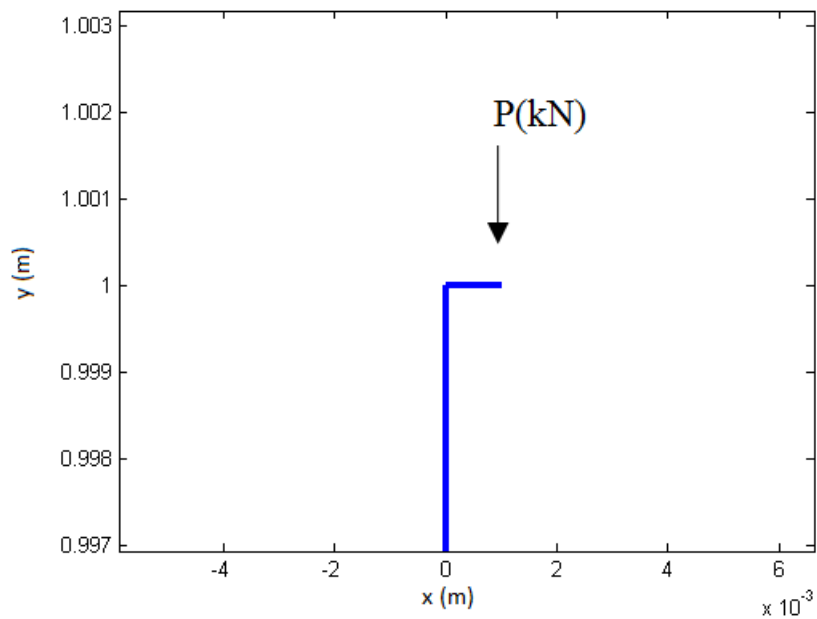


Figura 4.5: Matlab: Carregamento utilizando elemento de excentricidade.

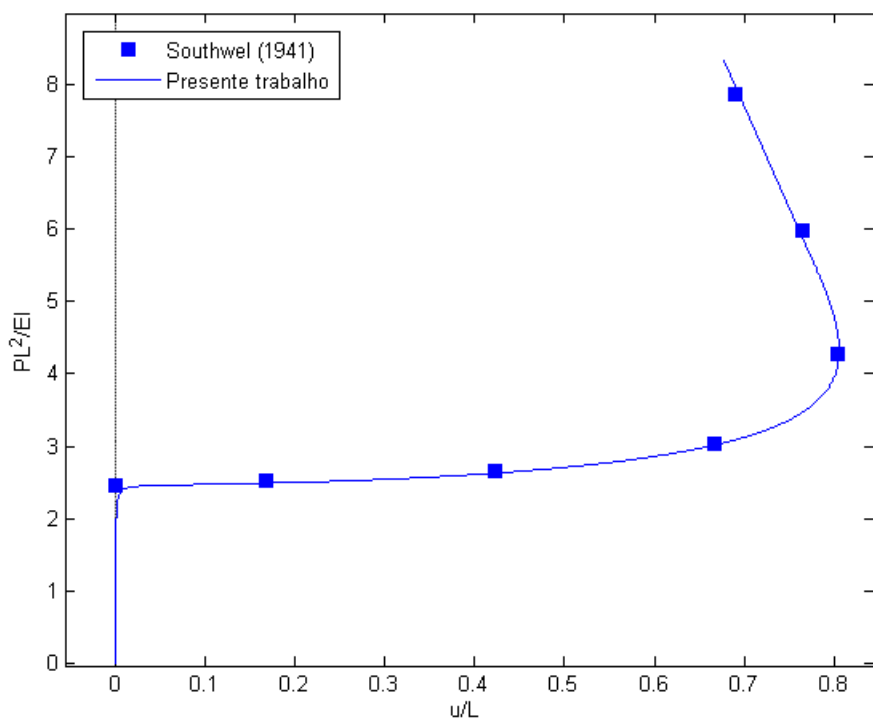


Figura 4.6: Trajetória de Equilíbrio com 11 elementos.

O gráfico exibido na Figura 4.6 possui solução analítica apresentada por Southwel(1941) representada nos pontos, cujos valores podem ser observados na Tabela 4.3.

$PL^2/EI$	$u/L$
2.4674	0
2.518	0.167
2.652	0.422
3.036	0.666
4.266	0.804
5.982	0.765
7.857	0.69

Tabela 4.3: Dados pontuais: Southwel (1941).

Pode-se observar no gráfico que até o carregamento alcançar o valor crítico de  $PL^2/EI = 2.2$  não há praticamente deslocamento algum. A partir dessa região, ocorrem grandes deslocamentos apesar do pouco acréscimo de carga. Deve-se observar também, a existência de um ponto limite no deslocamento em torno de  $u = 0.8$ .

## 4.2 Exemplos Fortemente Não Lineares

Nesta Seção serão estudados alguns exemplos de estruturas de pórticos planos encontrados na literatura que possuem forte não linearidade geométrica. O objetivo é verificar a eficiência computacional do código desenvolvido no presente trabalho. Soluções numéricas encontradas na literatura serão utilizadas para confirmar os resultados fornecidos pelas análises.

Na Seção 4.2.1 serão abordados dois exemplos de pórticos em L: o pórtico de Lee (Galvão, 2000) e o pórtico de Roorda (Galvão, 2004), cuja solução necessita da utilização de um elemento de excentricidade para a realização das análises com geração das trajetórias de equilíbrio nos dois sentidos.

A seguir, será analisado o caso do arco birrotulado, cujos resultados serão comparados aos resultados numéricos encontrados por Yang e Kuo(1994). Duas análises serão realizadas para esse pórtico: carregamento centrado e carregamento excêntrico.

### 4.2.1 Pórticos em L

Pórticos em L, também conhecidos como L-Frames, são amplamente utilizados na validação de formulações fortemente não lineares, devido às suas trajetórias de equilíbrio compostas por curvas acentuadas e pontos críticos de carregamento e deslocamento. Um

breve estudo com alterações do apoio superior será realizado nesta Seção, com o objetivo de se observar as alterações de sua trajetória de acordo com diferentes condições de contorno.

#### 4.2.1.1 Pórtico de Lee

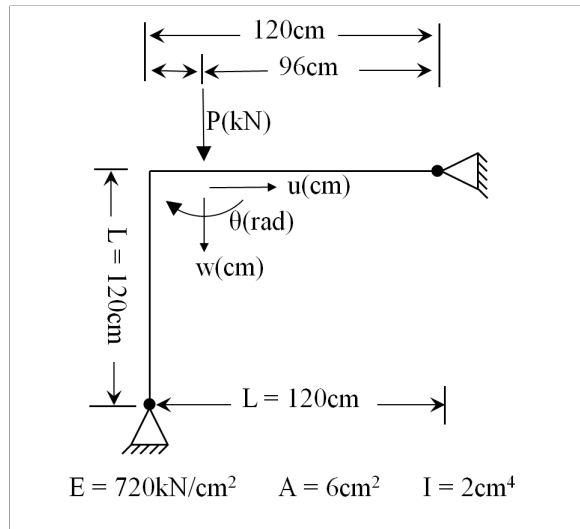


Figura 4.7: Pórtico de Lee.

O exemplo ilustrado na Figura 4.7, também conhecido como Pórtico de Lee, é frequentemente utilizado por pesquisadores para validar estratégias de solução não linear. Isso se deve ao fato de sua trajetória de equilíbrio ser marcada por pontos limites de carga e deslocamento. Para validação dos resultados, serão utilizados os resultados encontrados por Schweizerhof e Wriggers (1986), que também utilizaram elementos finitos na obtenção de suas soluções.

Esta estrutura foi definida com dimensão 120x120, discretizada em 20 elementos, possuindo carregamento pontual vertical negativo localizada no 13º nó, de coordenadas cartesianas 24x120, conforme ilustrado na Figura 4.7.

A estrutura possui, ainda, módulo de elasticidade  $E = 720 \text{ kN/cm}^2$ , área da seção  $A = 6 \text{ cm}^2$  e momento de inércia  $I = 2 \text{ cm}^4$ . Foram definidos, para o processo de análise incremental-iterativa, parâmetro inicial de carga  $\Delta\lambda_1^0 = 5(10^{-1})$ , com tolerância  $\zeta = 10^{-3}$  e quantidade de iterações desejadas  $Id = 5$ . Foram necessários 592 incrementos e gasto um tempo de processamento de 13.917 segundos para a geração do gráfico ilustrado na Figura 4.8.

Podem ser observadas no gráfico da Figura 4.8 as trajetórias de equilíbrio de deslo-

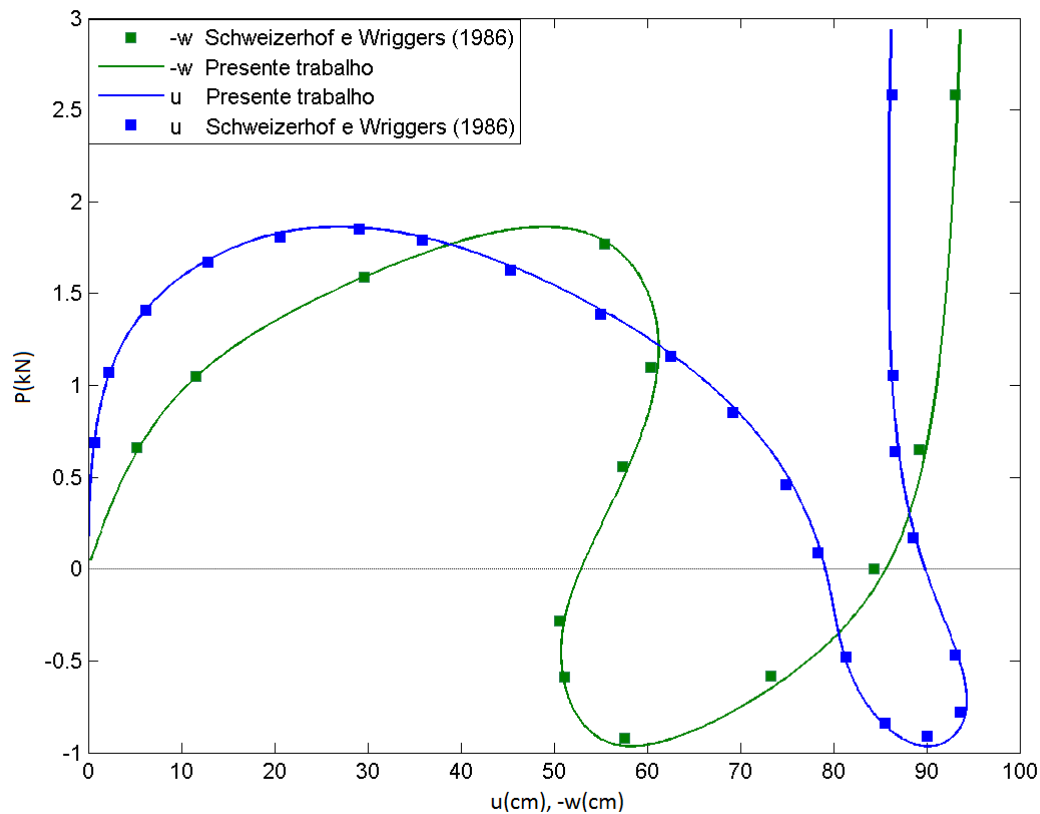


Figura 4.8: Trajetória de equilíbrio do deslocamento axial e transversal do pórtico de Lee.

camento axial e transversal. Os pontos marcados no gráfico correspondem às soluções numéricas de Schweizerhof e Wriggers(1986), cujos valores se encontram nas Tabelas 4.4 e 4.5

O gráfico ilustrado na Figura 4.9 nos dá a trajetória de equilíbrio da rotação. Pode-se verificar a mesma curva em Galvão (2000), o que, mais uma vez, valida os resultados da análise.

O gráfico ilustrado na Figura 4.10 mostra os pontos limites de carregamento e deslocamento, cujos valores podem ser encontrados na Tabela 4.6.

As deformadas da estruturas nos pontos A, B, C e D podem ser observadas nas Figuras extraídas do código representados pela Figura 4.11.

Pode-se observar que os pontos limites A e D são pontos limites de carregamento, após os quais o sinal do parâmetro de incremento de carga muda, alterando, portanto, o sentido do carregamento. Para que esta modificação no sinal ocorra adequadamente, utilizamos a estratégia de mudança de sinal através do parâmetro GSP.

$-u(cm)$	$P(kN)$
0.72	0.69
2.205	1.07
6.13	1.41
12.86	1.67
20.61	1.81
29.09	1.851
35.81	1.79
45.28	1.63
55	1.39
62.48	1.16
69.14	0.85
74.93	0.46
78.33	0.088
81.31	-0.48
85.5	-0.84
90	-0.91
93.55	-0.78
93.09	-0.47
88.5	0.17
86.6	0.64
86.41	1.056
86.27	2.58

Tabela 4.4: Deslocamento Axial: Schweizerhof e Wriggers(1986).

$-w(cm)$	$P(kN)$
5.21	0.66
11.52	1.05
29.62	1.59
55.39	1.77
60.35	1.096
57.33	0.56
50.57	-0.28
51.1	-0.59
57.6	-0.92
73.27	-0.58
84.31	0
89.22	0.65
93.046	2.58

Tabela 4.5: Deslocamento Transversal: Schweizerhof e Wriggers(1986).

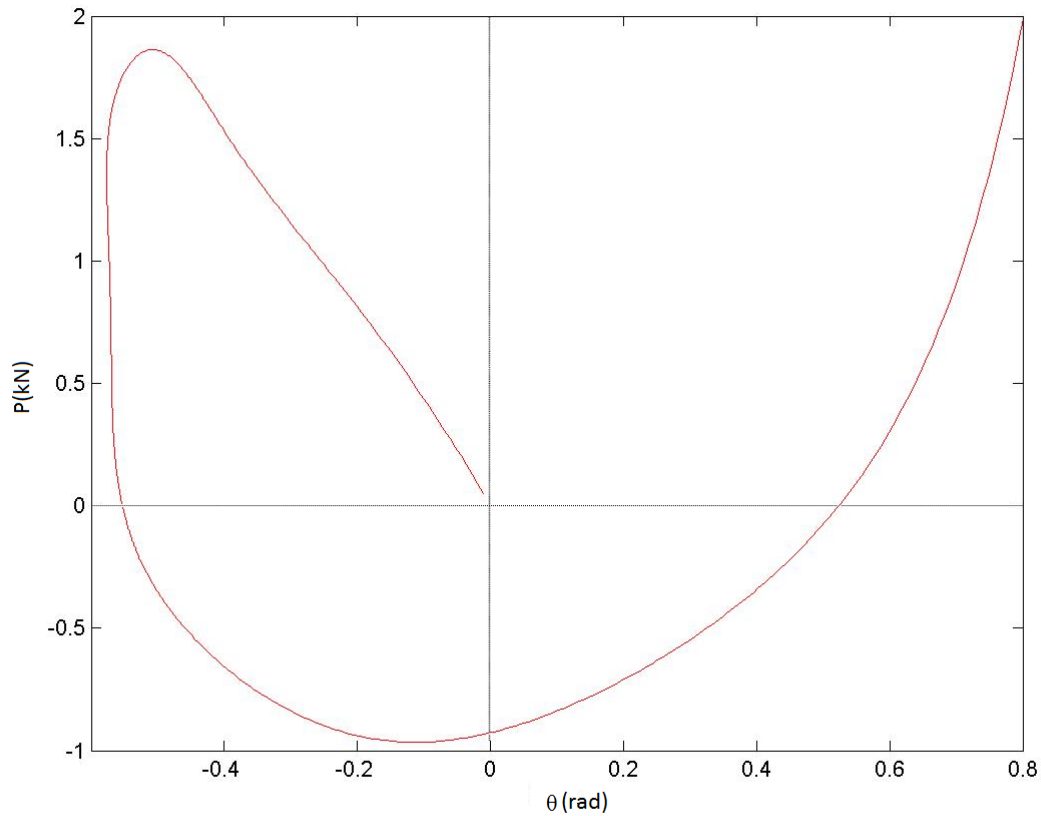


Figura 4.9: Trajetória de Equilíbrio da rotação  $\theta$ .

<i>Pontos</i>	$-w(cm)$	$P(kN)$
A	48.791	1.856
B	61.006	1.192
C	50.749	-0.438
D	58.188	-0.942

Tabela 4.6: Pontos Limites - Galvão(2000).

Os pontos limites B e C referem-se aos pontos limites de deslocamento.

Outro aspecto que merece atenção reside na relação entre a quantidade de incrementos e iterações necessárias para convergência da resposta, dada uma determinada tolerância.

O gráfico indicado na Figura 4.12 mostra a relação entre a quantidade de incrementos e, conseqüentemente, de iterações acumuladas durante a análise para uma variação do limite de tolerância para a convergência da análise para um determinado ponto da curva de trajetória de equilíbrio.

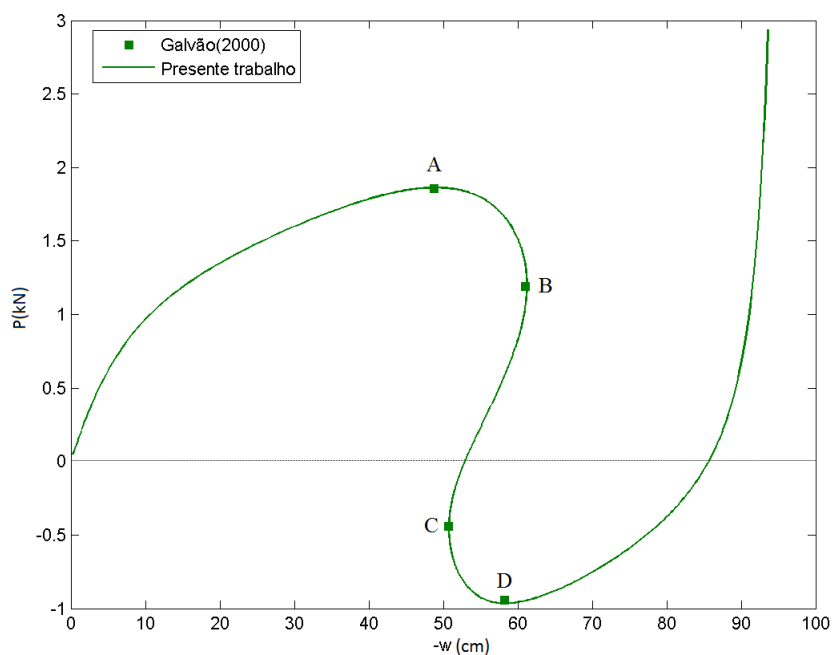


Figura 4.10: Pontos Limites de Carregamento e Deslocamento.

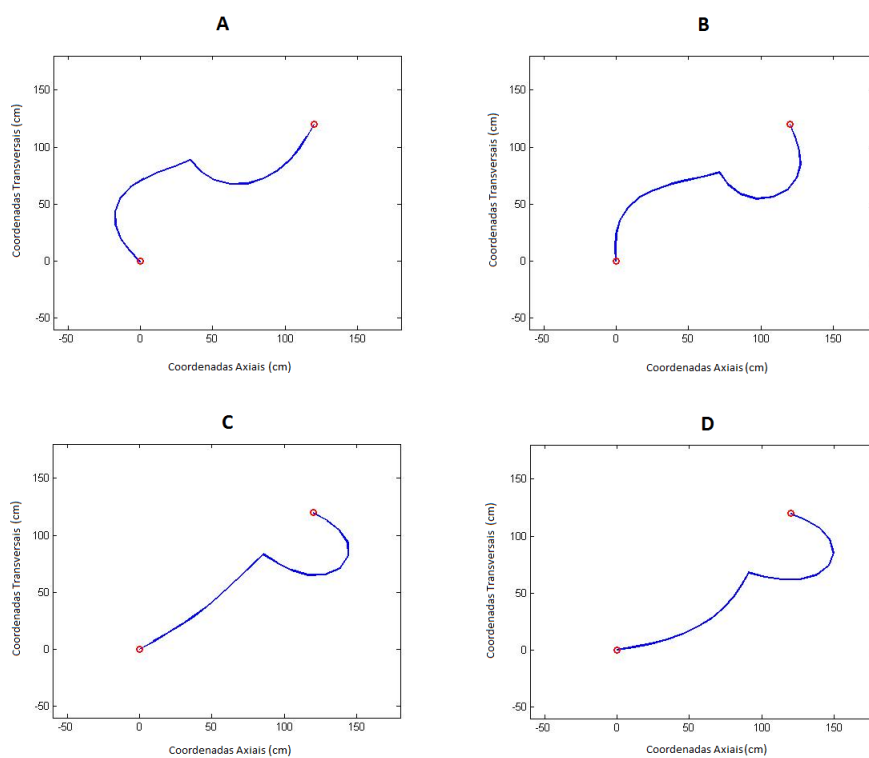


Figura 4.11: Deformação da Estrutura nos Pontos Limites.



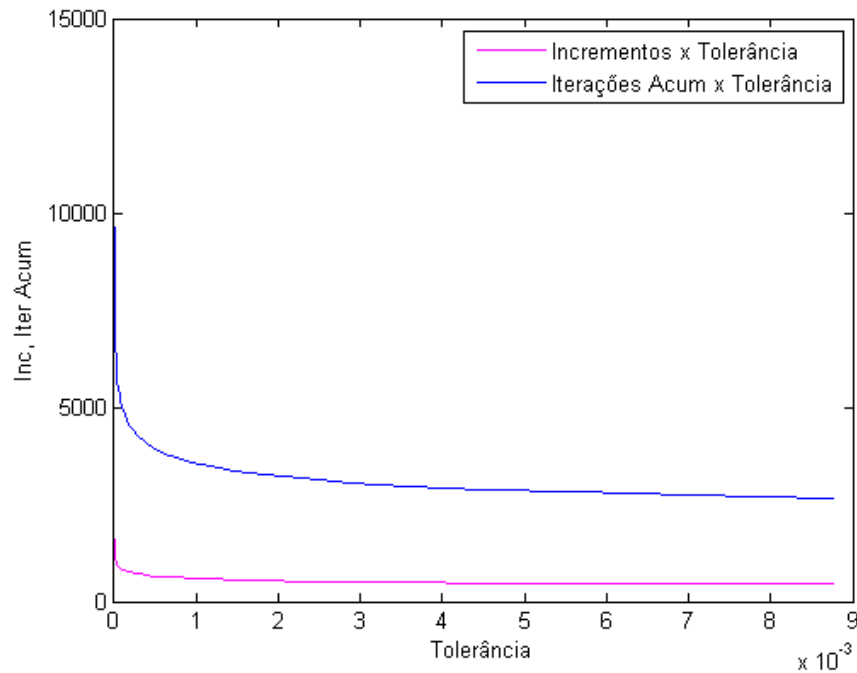


Figura 4.12: Pórtico de Lee: Tolerância x Incrementos e Iterações Acumuladas.

Pode-se observar que, à medida que a tolerância diminui, se aproximando de zero, a quantidade de incrementos e iterações cresce. Vale observar que, a partir de aproximadamente  $0.5(10^{-3})$  a curva se acentua consideravelmente, revelando um crescimento acentuado na quantidade de incrementos e iterações necessários para a realização da análise. Isso mostra que, quanto mais próxima de zero a tolerância, maior será o esforço computacional para a geração dos gráficos. Isso acarreta em um aumento da precisão da análise. No entanto, vale lembrar que a precisão aumenta muito pouco nesses casos, não sendo possível observar alterações relevantes nas curvas dos gráficos das trajetórias de equilíbrio, o que denota um gasto computacional que pode ser considerado desnecessário.

## 4.2.1.2 Roorda Frame

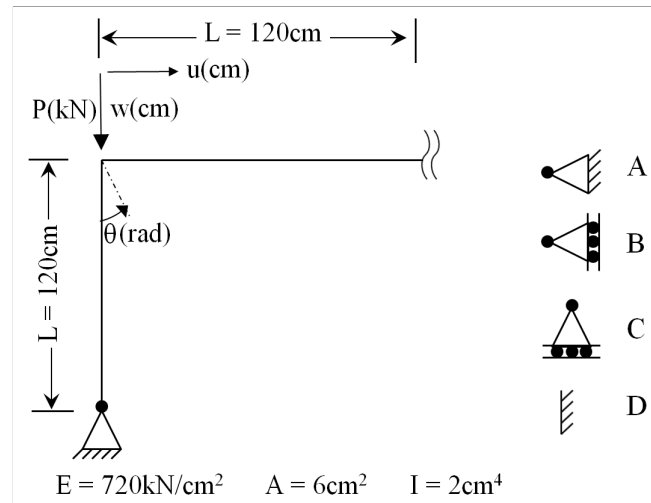


Figura 4.13: Pórtico de Roorda.

O segundo pórtico em L a ser analisado é conhecido como Pórtico de Roorda (Roorda, 1965). O exemplo utilizado possui as mesmas propriedades físicas e geométricas do exemplo apresentado no tópico 4.2.1.1 deste trabalho, o pórtico de Lee.

Esta estrutura possui bifurcação assimétrica, que será obtida através da inserção de um elemento de excentricidade no carregamento no valor de  $0.12\text{cm}$ , utilizado para obter as duas regiões da trajetória de equilíbrio.

Ao contrário das colunas, que apresentam bifurcação simétrica estável, os pórticos apresentam bifurcações assimétricas instáveis. Será realizada, nesta análise, um breve estudo sobre as variações das condições de apoio no pórtico de Roorda.

Para a geração do gráfico, a estrutura foi discretizada em 21 elementos finitos (sendo o 21º definido para a aplicação da excentricidade). O incremento inicial de carregamento foi definida como  $\Delta\lambda_1^0 = 5(10^{-3})$  para todos os casos, exceto o caso de apoio com grau de liberdade axial e à rotação, cujo valor definido foi de  $\Delta\lambda_1^0 = 5(10^{-4})$ . Foram definidos, em todos os casos, 300 incrementos para a geração de cada trajetória de equilíbrio.

O gráfico ilustrado na Figura 4.14 exibe as trajetórias de equilíbrio estável e instável do pórtico de Roorda para os quatro diferentes casos de condições de apoio na parte superior da estrutura, representados na Figura 4.13, a saber:  $A$  = liberdade na rotação;  $B$  = rotação com deslocamento transversal;  $C$  = rotação com deslocamento axial e  $D$  = engaste. Para tal, foram necessárias oito análises estruturais, sendo necessárias duas (equilíbrio estável e instável) para cada diferente apoio. Pode-se observar na Figura 4.14

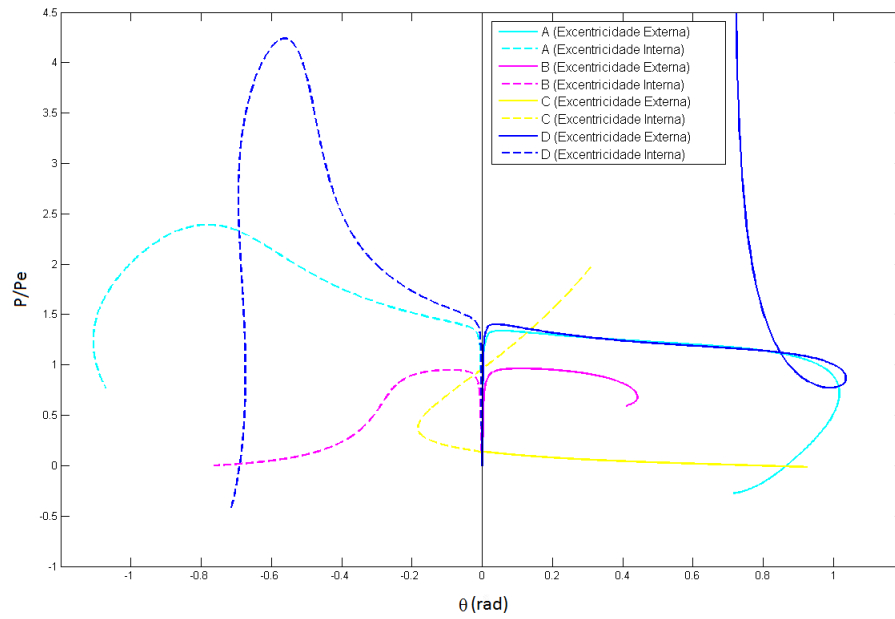


Figura 4.14: Roorda Frame: trajetórias de equilíbrio.

a variação do parâmetro de carga  $P/P_e$ , onde  $P_e = (\pi^2 EI)/(L^2)$ , com a rotação do nó de ligação entre as duas barras.

### 4.2.2 Arco Circular Birrotulado

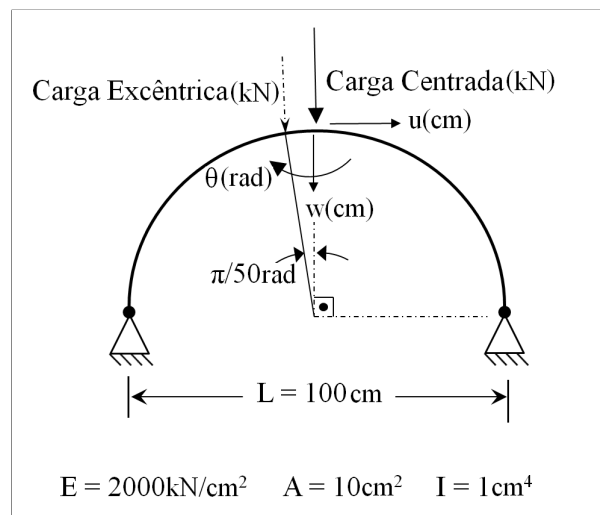


Figura 4.15: Arco Circular Birrotulado.

O próximo exemplo a ser abordado, o arco circular birrotulado, pode ser observado na Figura 4.15 e será analisado sob duas situações: o carregamento centrado, conhecido como

modelo perfeito, e o carregamento excêntrico, também conhecido como modelo imperfeito. Este exemplo visa verificar e comparar a eficiência computacional do código desenvolvido com a formulação desenvolvida por Yang e Kuo (1994).

A estrutura representada na Figura 4.15 possui propriedades físicas e geométricas representadas pelos valores  $E = 2000kN/cm^2$ ,  $A = 10cm^2$  e  $I = 1cm^4$ . Essas propriedades serão as mesmas para as duas análises em questão. Seus dois apoios possuem restrição aos deslocamentos axial e transversal, sendo livres para rotação.

#### 4.2.2.1 Carga Centrada

O arco utilizado neste exemplo possui carga aplicada centrada. O arco em questão foi discretizado em 26 elementos finitos possuindo sua carga transversal negativa aplicada no 14º nó da estrutura.

Foram definidos, para a realização da análise, tolerância  $\zeta = 10^{-5}$ , parâmetro inicial do incremento de carga  $\Delta\lambda_1^0 = 5(10^{-2})$  e número de iterações desejadas  $Id = 2$ .

Para a geração do gráfico ilustrado na Figura 4.16 foram necessários 10385 incrementos, e gasto um tempo total de processamento de 176.458 segundos.

Os dados pontuais apresentados na Figura 4.16 referem-se às análises de Yang e Kuo(1994), cujos valores podem ser observados na Tabela 4.7

$-w(cm)$	$P(kN)$
37.059	8.186
79.412	-21.928
15	48.648
89.845	-82.9
12.353	129.841
89.845	-182.003

Tabela 4.7: Carregamento Centrado: Yang e Kuo (1994).

Pode-se observar, no gráfico, que o arco apresenta um comportamento cíclico onde, após o término de cada ciclo, os resultados tornam-se menos precisos, devido ao fato de os elementos tornarem-se menos eficientes à medida que a estrutura se subdivide. O problema pode ser melhorado à medida que se discretize mais a estrutura.

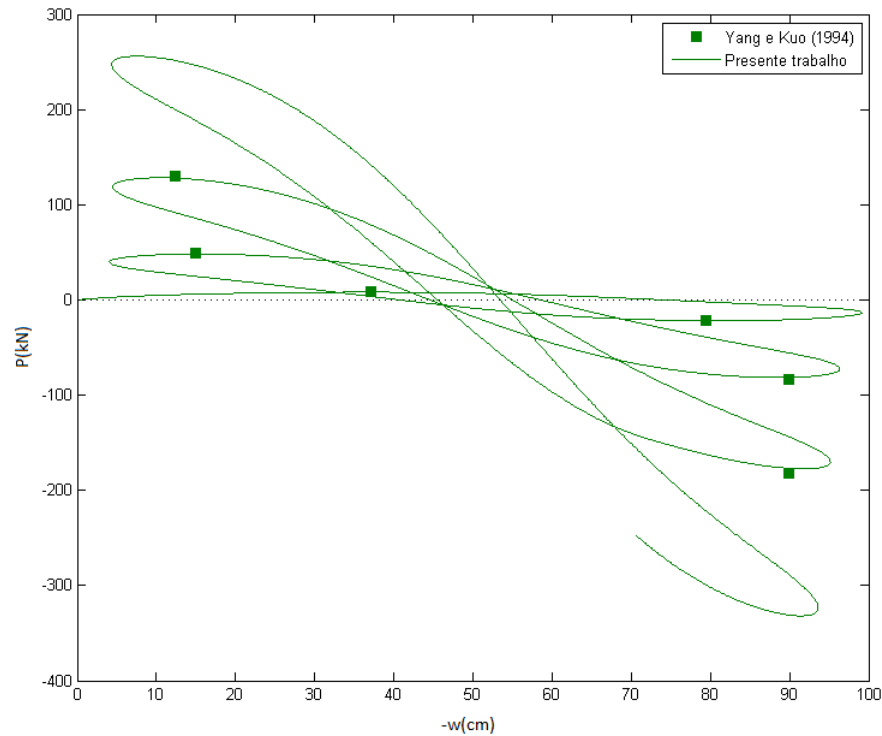


Figura 4.16: Carga centrada:  $P \times w$ .

#### 4.2.2.2 Carga Excêntrica

Para a aplicação do carregamento excêntrico, foi utilizada a mesma estrutura discretizada em 26 elementos, sendo o carregamento alterado para a posição referente ao nó 15.

Foram definidos, para a realização da análise, tolerância  $\zeta = 10^{-4}$ , parâmetro inicial do incremento de carga  $\Delta\lambda_1^0 = 5(10^{-1})$  e número de iterações desejadas  $Id = 2$ .

Para a geração do gráfico ilustrado na Figura 4.17 foram necessários 12000 incrementos, e gasto um tempo total de processamento de 212.024 segundos.

Os dados pontuais apresentados na Figura 4.17 referem-se às análises de Yang e Kuo(1994), cujos valores podem ser observados na Tabela 4.8

No caso do carregamento excêntrico, ao contrário do carregamento concentrado, pode-se observar os gráficos gerados das trajetórias de equilíbrio do deslocamento axial e da rotação nas Figuras 4.18 e 4.19, respectivamente.

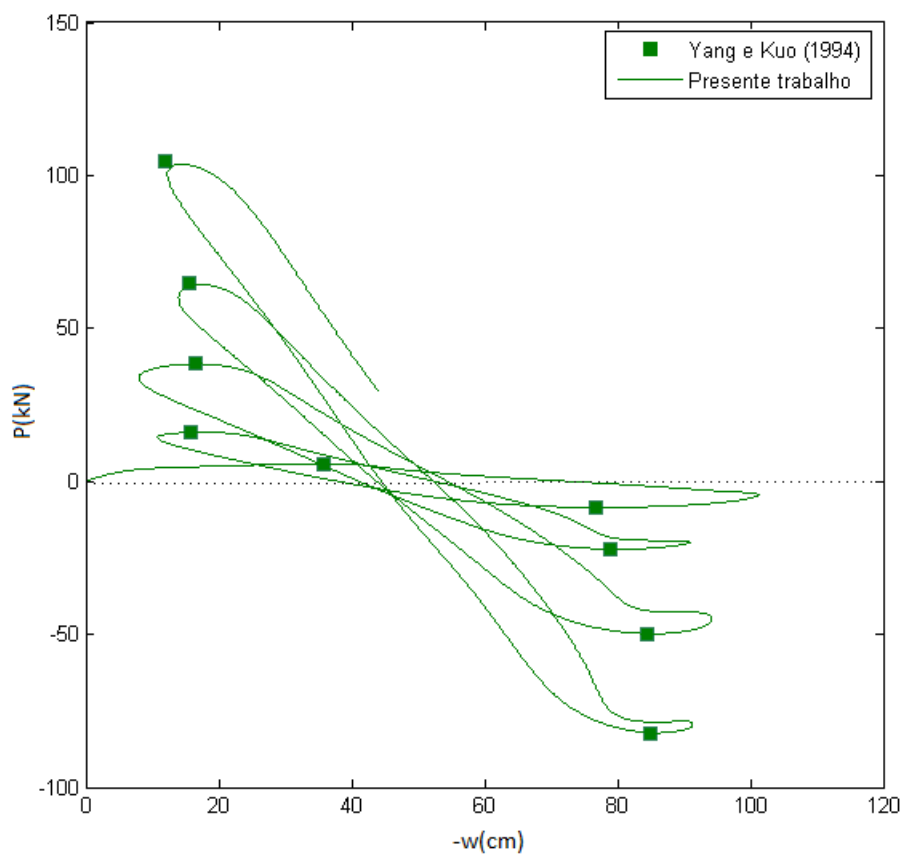


Figura 4.17: Carga Excêntrica:  $P \times w$ .

$-w(cm)$	$P(kN)$
35.735	5.813
76.8	-8.498
15.8	16.149
79.02	-22.162
16.6	38.566
84.4	-49.896
15.48	64.875
84.87	-82.42
11.91	104.611

Tabela 4.8: Carregamento Excêntrico: Yang e Kuo(1994).

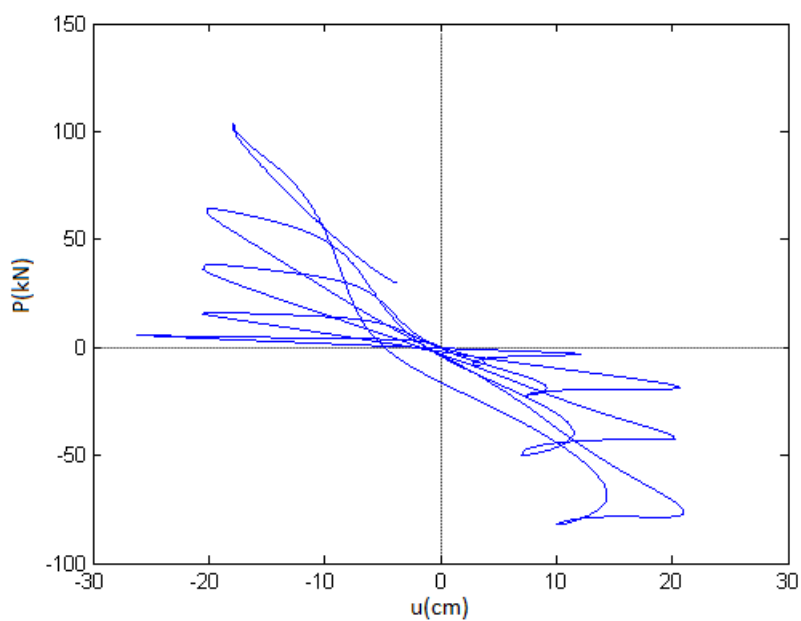


Figura 4.18: Carga Excêntrica:  $P \times u$ .

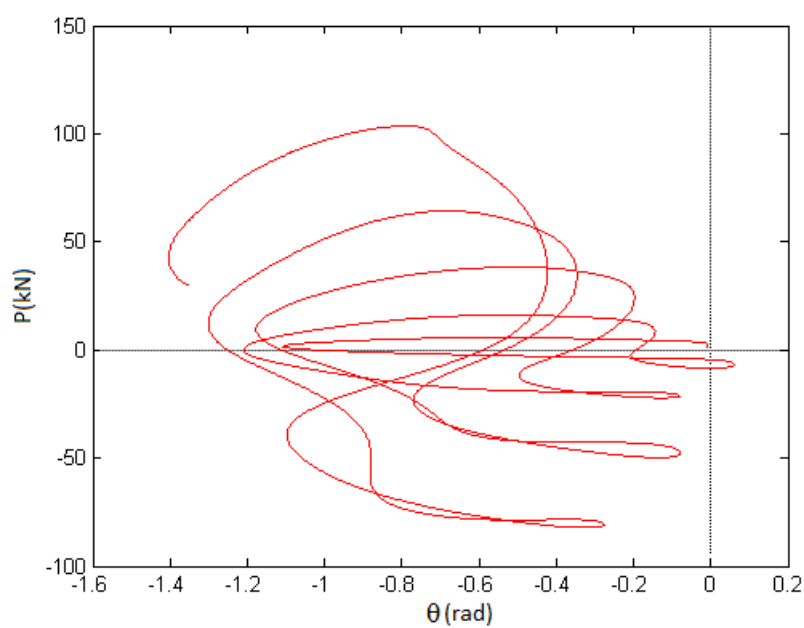


Figura 4.19: Carga Excêntrica:  $P \times \theta$ .

### 4.3 Outros exemplos

Nesta Seção serão apresentados os casos de estruturas de galpão, sendo observados os comportamentos do galpão simples e do galpão duplo, ambos possuindo carregamentos pontuais. Suas propriedades físicas e geométricas não possuirão as mesmas definições, assim como o comprimento de suas barras não serão os mesmos.

Para a realização destas análises, devido à existência de múltiplos carregamentos, as trajetórias de equilíbrio serão realizadas para cada ponto de carregamento aplicado, com o objetivo de melhor observação do comportamento das estruturas através do estudo de suas trajetórias de equilíbrio. Nesta análise, as vigas das estruturas de galpão não serão tratadas individualmente como elementos finitos, sendo as mesmas discretizadas em elementos menores. A estrutura de galpão simples possuirá, portanto, 20 elementos finitos e a do galpão duplo será discretizada em 35 elementos. Pode-se verificar que cada barra de cada estrutura será discretizada, portanto, em 5 elementos finitos.

#### 4.3.1 Galpão

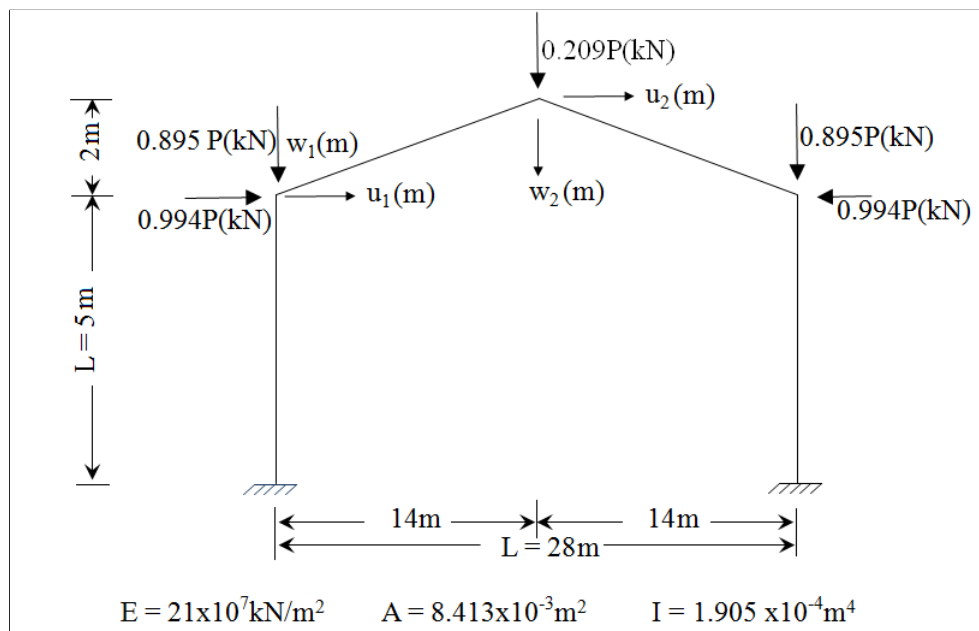


Figura 4.20: Estrutura tipo Galpão.

A estrutura apresentada na Figura 4.20, também conhecida como pórtico tipo Galpão, possui 3 carregamentos pontuais e foi discretizada em 20 elementos finitos. A estrutura possui 2 engastes como apoios para a estrutura. Suas propriedades físicas e geométricas são distribuídas igualmente por toda a estrutura, possuindo  $E = 21(10^7)\text{ kN/m}^2$ ,  $A =$



$8.413(10^{-3})m^2$  e  $I = 1.905(10^{-4})m^4$ . Foram aplicados 3 carregamentos, cujos valores e localizações podem ser observados na Tabela 4.9.

Nó	<i>CargaAxial(kN)</i>	<i>CargaTransversal(kN)</i>
6	0.994521895368	-0.895471536732
11	0	-0.209056926536
16	-0.994521895368	-0.895471536732

Tabela 4.9: Galpão - Carregamentos.

Pode-se observar que a combinação dos carregamentos axial e transversal nos nós 6 e 16 correspondem, na realidade, a um carregamento inclinado.

Para a realização da análise, foi definida tolerância  $\zeta = 10^{-5}$ , parâmetro inicial de incremento de carga  $\Delta\lambda_1^0 = 5(10^{-2})$  e iterações desejadas  $Id = 2$ . Para a geração dos gráficos foram necessários 300 incrementos e gasto um tempo de processamento total de 4.016 segundos.

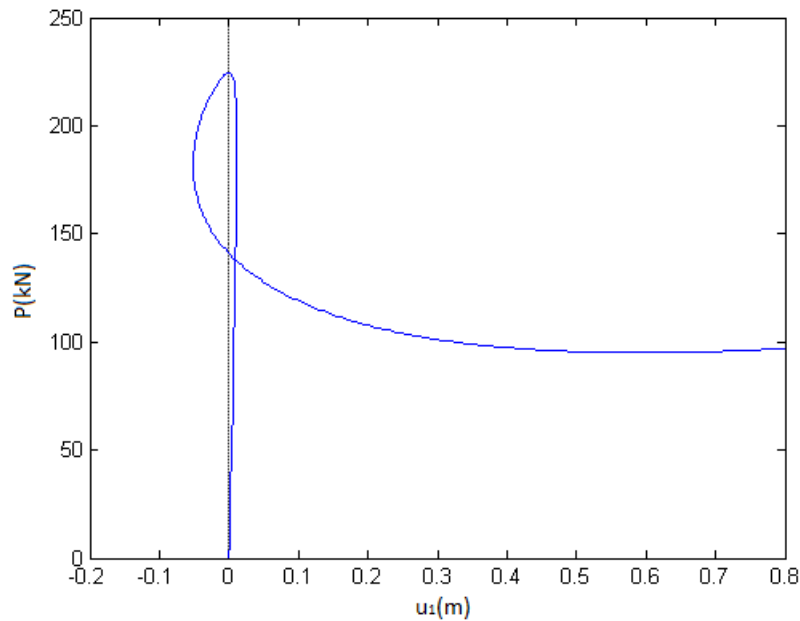


Figura 4.21: Galpão: Trajetória de Equilíbrio axial.

Os gráficos representados nas Figuras 4.21, 4.22 e 4.23 retratam as trajetórias de equilíbrio dos deslocamentos axial, transversal e da rotação, referentes ao nó 6 da estrutura (topo da parede lateral esquerda), cujas coordenadas são  $(x, y) = (0m, 5m)$ . Pode-se observar que em todos os casos há muito pouco deslocamento para um considerável aumento no carregamento. A seguir, percebe-se que a estrutura atinge rapidamente o ponto crítico

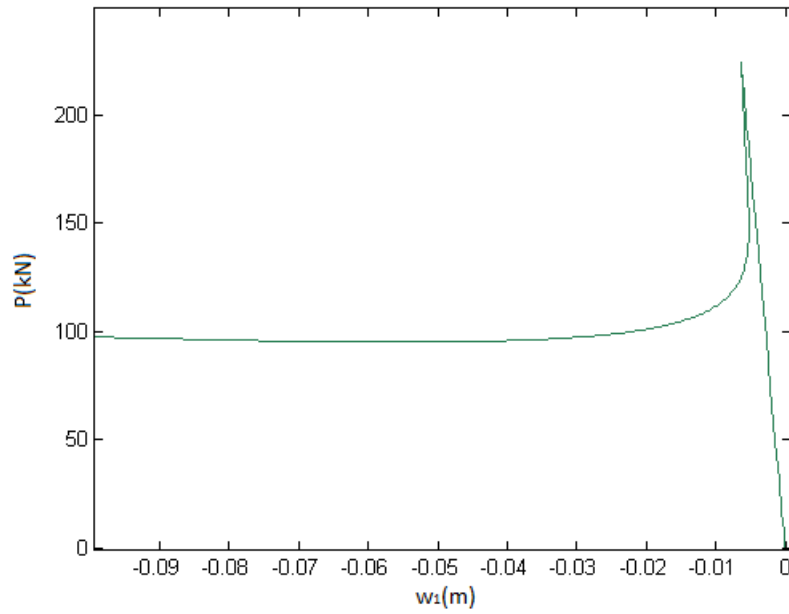


Figura 4.22: Galpão: Trajetória de Equilíbrio Transversal.

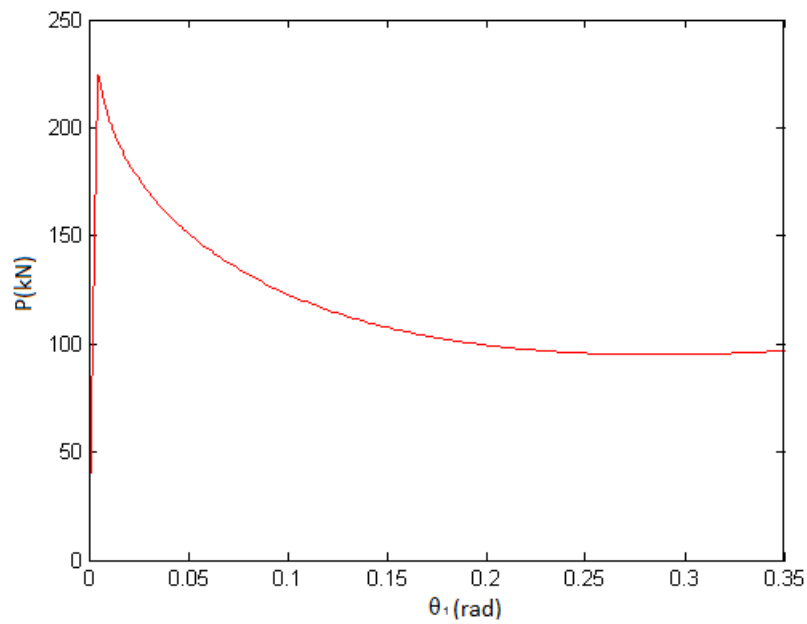
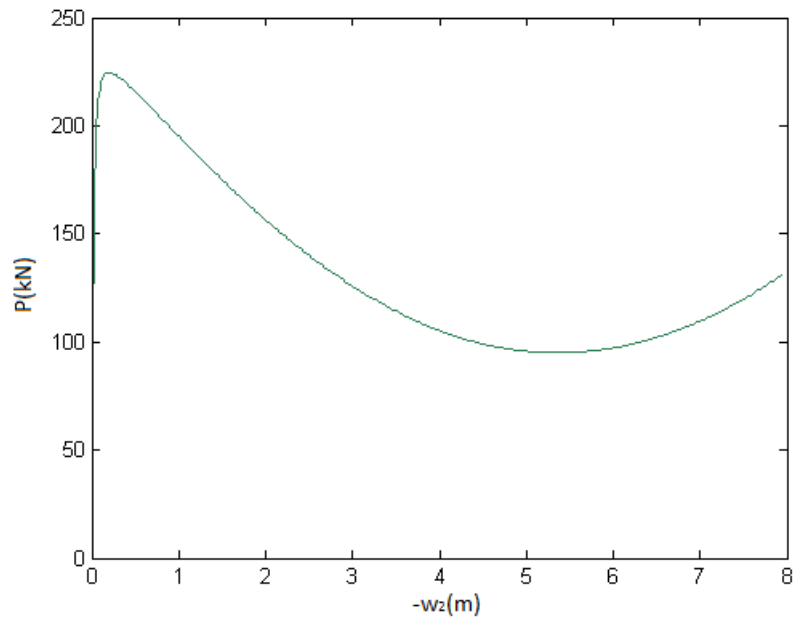


Figura 4.23: Galpão: Trajetória de Equilíbrio da Rotação.

de carregamento máximo, sofrendo em seguida uma perda de carregamento que atinge novo ponto crítico, agora de carregamento mínimo. Observando a escala no eixo horizontal dos gráficos, percebe-se que não há grandes deslocamentos em nenhum dos casos, sendo o axial mais significativo que os outros.



O gráfico 4.24 exibe a trajetória de equilíbrio referente ao topo do galpão - nó 11, coordenadas  $(x, y) = (14m, 6.5m)$ . Pode-se observar, na Figura 4.24, que o primeiro ponto limite de carregamento é atingido em  $P = 224.435kN$ . Pode-se observar também que, até esse ponto, quase não há deslocamento transversal. Após o limite do carregamento, a estrutura sofre grandes deslocamentos transversais.

### 4.3.2 Galpão Duplo

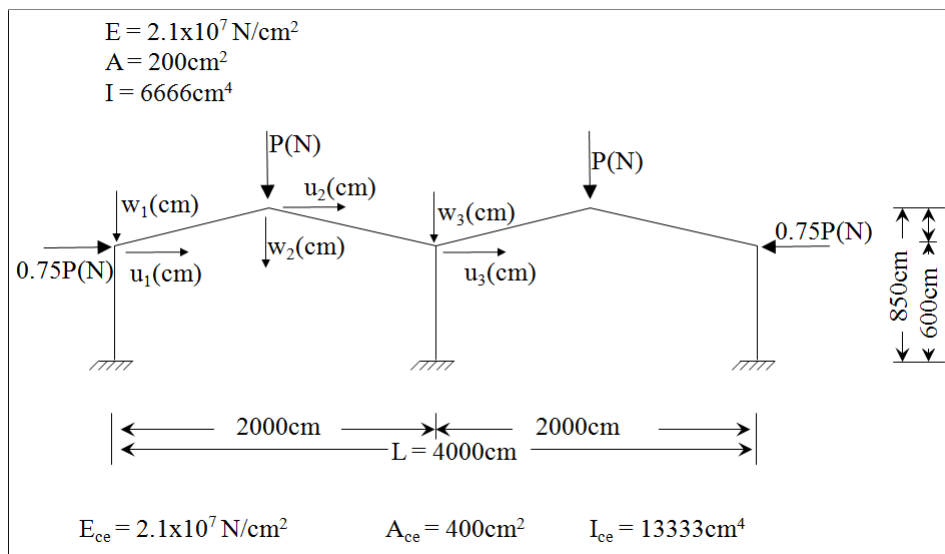


Figura 4.25: Galpão Duplo.

O galpão duplo, representado pela Figura 4.25, corresponde a dois galpões simples que compartilham a mesma coluna central. E estrutura foi discretizada em 35 elementos, possuindo carregamentos em sua estrutura, conforme ilustrado na Tabela 4.10. A estrutura possui 3 apoios do tipo engaste.

Nó	<i>Carga Axial(N)</i>	<i>Carga Transversal(N)</i>
6	0.75	0
11	0	-1
26	0	-1
31	-0.75	0

Tabela 4.10: Galpão Duplo - Carregamentos.

A estrutura possui, para suas colunas externas, propriedades físicas e geométricas nos valores  $E = 2.1(10^7)N/cm^2$ ,  $A = 400cm^2$  e  $I = 13333cm^4$ . O resto da estrutura possui propriedades  $E = 2.1(10^7)N/cm^2$ ,  $A = 200cm^2$  e  $I = 6666cm^4$ . Para a realização da análise foi definida tolerância  $\zeta = 10^{-3}$ , com parâmetro inicial de incremento de carga  $\Delta\lambda_1^0 = 50$  e iterações desejadas  $Id = 1$ . Para a geração dos gráficos, foram necessários 2080 incrementos e gasto um tempo de processamento total de 33.223 segundos.

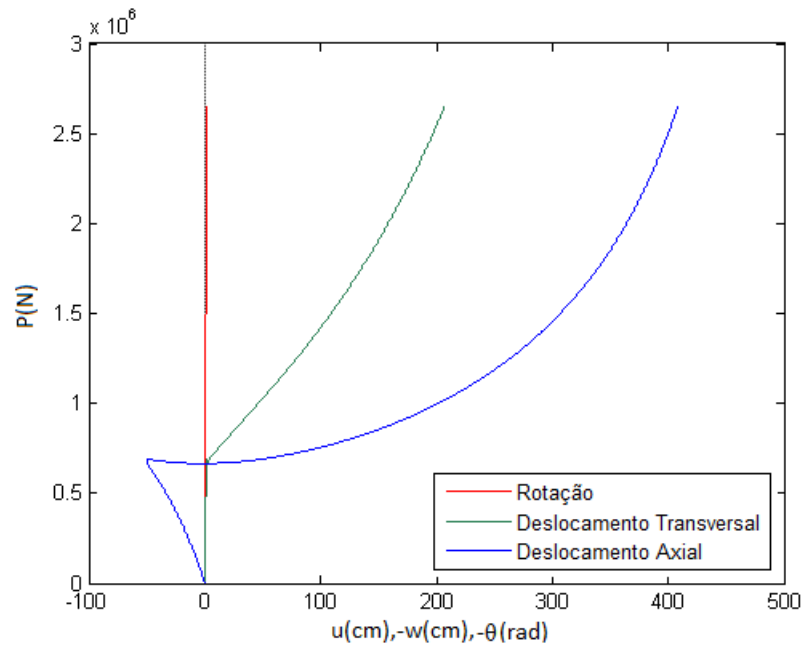


Figura 4.26: Trajetórias de equilíbrio do galpão Duplo.

O gráfico da Figura 4.26 apresenta as 3 trajetórias de equilíbrio referentes ao nó 6 da estrutura em questão. Aproximando a imagem, como observado na Figura 4.27, pode-se observar o ponto em comum onde as três trajetórias se encontram.

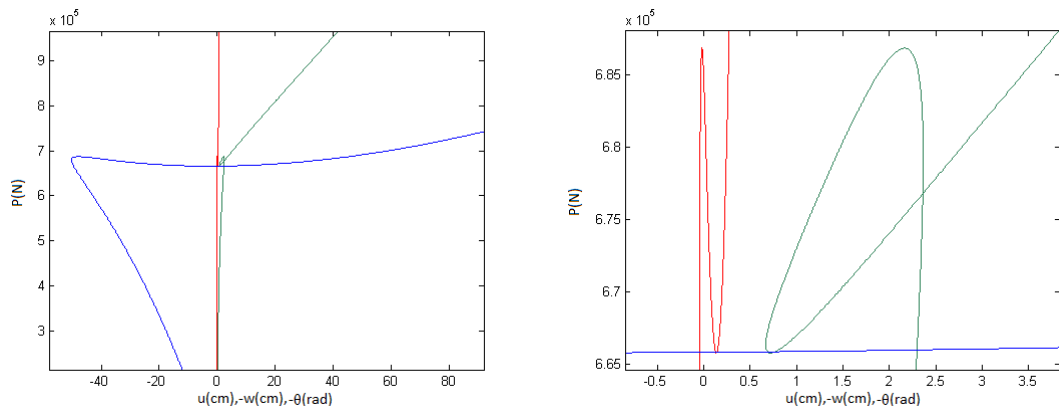


Figura 4.27: Aproximação das Trajetórias de equilíbrio.

Pode-se observar, nos gráficos da Figura 4.27, que o carregamento atinge seu ponto limite em  $P = 686859.853$ , onde as trajetórias de equilíbrio sofrem variações em seu comportamento. A Figura 4.28 mostra a semelhança entre a trajetória de equilíbrio do deslocamento axial e da rotação, respeitadas as escalas das mesmas.

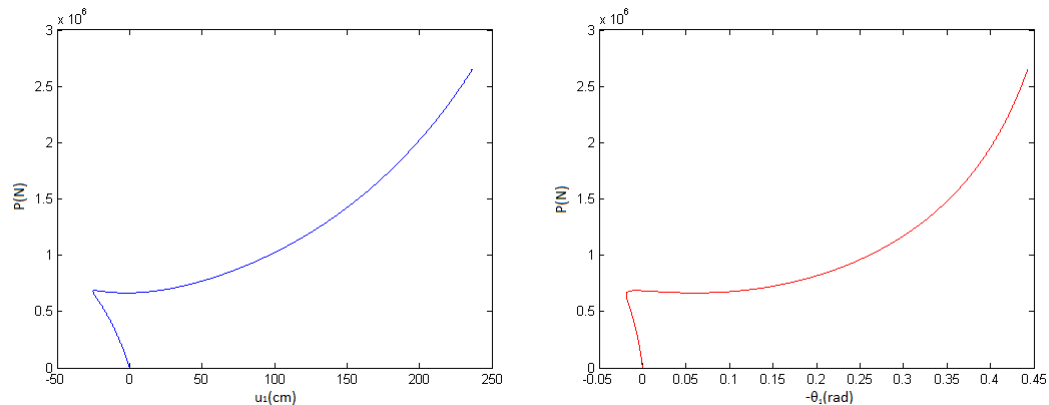


Figura 4.28: Trajetórias de equilíbrio do deslocamento axial e da Rotação.

Analisando a trajetória de equilíbrio do nó 11, pode-se observar que a trajetória que sofreu grande alteração de comportamento refere-se ao deslocamento transversal, como pode ser observado na Figura 4.29.

Como os esforços aplicados na estrutura são simétricos, assim como a própria estrutura e suas propriedades físicas e geométricas, as trajetórias de equilíbrio dos outros pontos que sofrem carregamento possuem o mesmo comportamento, variando em alguns casos apenas o sentido do gráfico.

Analisando a trajetória de equilíbrio do nó central da estrutura, pode-se observar que o nó 11 sofre uma pequena variação transversal, que pode ser interpretada como uma

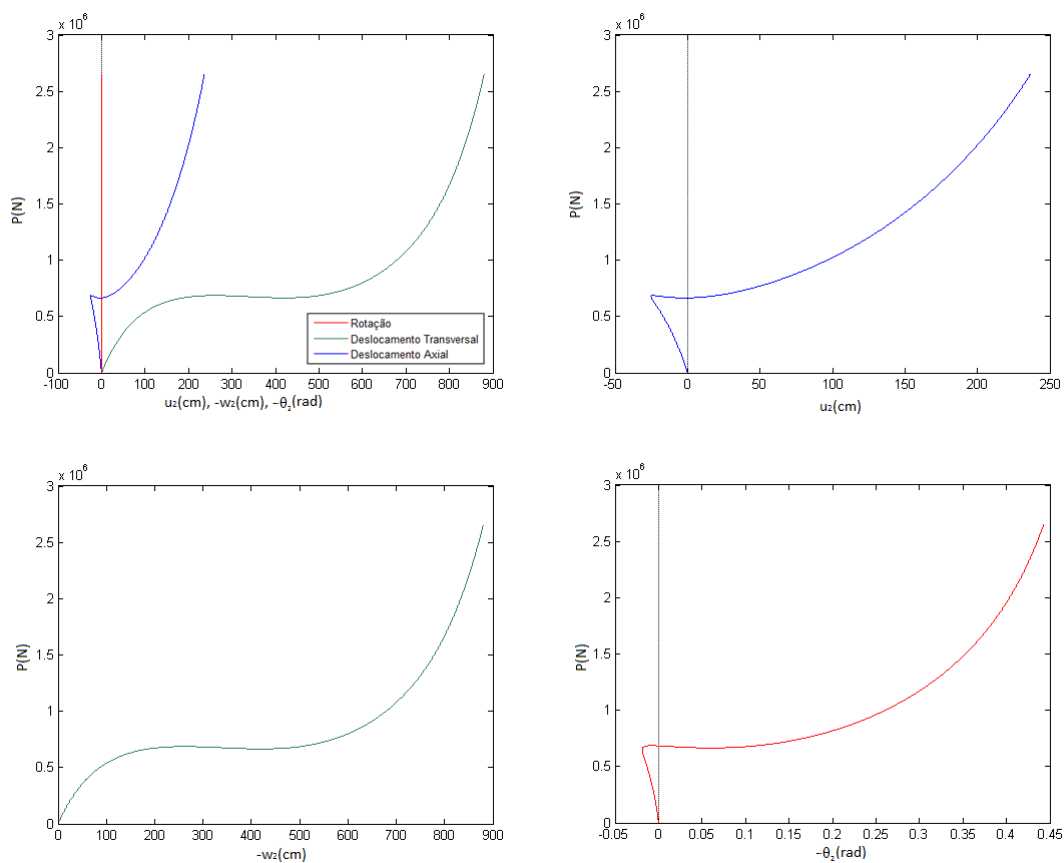


Figura 4.29: Trajetórias de equilíbrio para o nó 11.

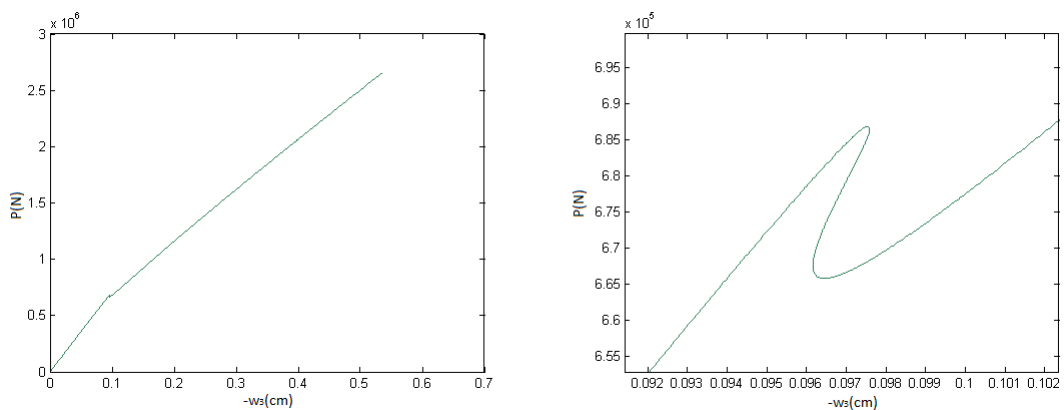


Figura 4.30: Trajetórias de equilíbrio transversal para o nó 16.

espécie de achatamento, representado pela Figura 4.30. Pode ser observado, também, que nos pontos críticos de carregamento ela também sofre variações.

## 4.4 Comparações

O presente trabalho teve como objetivo o desenvolvimento de um código computacional em ambiente Matlab que realize análises estáticas de pórticos planos, sem perda de qualidade de resultados em relação ao código CS-ASA (Silva, 2009).

Pôde ser percebido, durante o desenvolvimento deste trabalho, que se justifica o uso da linguagem Matlab em substituição à linguagem Fortran, na qual o CS-ASA (Silva, 2009) foi desenvolvido. Vários motivos levaram a esta escolha, destacando-se, principalmente, a simplicidade da linguagem Matlab aliada à sua proximidade com a linguagem matemática e a facilidade do ambiente Matlab na geração de saídas de dados otimizadas como, por exemplo, a geração de animações e gráficos.

O código computacional CS-ASA (Silva, 2009), por sua vez, apenas gera arquivos de dados de saída, cujos dados deverão ser, posteriormente, utilizados em outros códigos para a geração de gráficos. Quanto à geração de animações, o referido código não gera esse tipo de resultado e em seu arquivo de saída não há informações que permitam que a mesma seja gerada externamente ao código.

Outros aspectos abordado foram a organização e clareza do código. Buscou-se, neste trabalho, desenvolver um código computacional cuja aplicação possuísse a visualização mais próxima possível da teoria da análise estática não linear de pórticos planos. Através dessa abordagem, os códigos retratam com mais fidelidade a teoria estudada, sendo mais simples para o usuário enxergar as etapas da teoria na implementação do código. as etapas da análise foram representadas em sua maioria por funções e algumas estruturas de repetição, sendo agrupadas de acordo com a teoria da análise não linear.

Um tema de grande relevância a ser abordado se refere à generalização em substituição às estruturas de repetição. Algumas funções no Matlab foram criadas com conceitos de generalização de código, visando a extensão da análise contida nesse código para a análise de estruturas em geral. Essa abordagem é válida e de grande utilidade, pois é permitida a utilização da mesma função para diversas estruturas. Vale lembrar que essas generalizações não abrangem todas as funções do código, necessitando, portanto, de complementação em trabalhos futuros.

Obsevando novamente o CS-ASA (Silva, 2009) percebe-se a grande utilidade desta generalização, tendo em vista que muitas subrotinas (equivalentes às funções no Matlab) referem-se à mesma etapa da análise, alterando-se apenas a estrutura a ser analisada. A partir do momento que se pode utilizar a mesma função para todas as estruturas, sendo

modificados apenas alguns poucos parâmetros, justifica-se a generalização das funções, pois diminuem e simplificam consideravelmente o código.

A utilização, pelo código computacional desenvolvido neste trabalho, de apenas uma estratégia de iteração (incremento do comprimento de arco), incremento de carga (comprimento de arco cilíndrico) e análise de sinal (parâmetro GSP), implica diretamente em mais uma considerável redução no tamanho do código. Esta medida se justifica devido à alta qualidade dos resultados obtidos através dos gráficos gerados pelas análises. Isso pode ser observado comparando-se os resultados obtidos em estudos anteriores, que utilizaram o CS-ASA (Galvão, 2000; Galvão, 2004; Silva, 2009).

Testes foram realizados através da geração dos gráficos de uma mesma análise para a observação das discrepâncias. O resultado foi a sobreposição dos mesmos, o que valida o resultado da análise implementada neste trabalho. Outro aspecto importante reside na qualidade computacional da combinação dessas estratégias, que acabam por alcançar os mesmos resultados necessitando em muitos casos, de uma quantidade muito menor de incrementos, o que acarreta em menor custo computacional e, conseqüentemente, maior desempenho e economia de tempo.

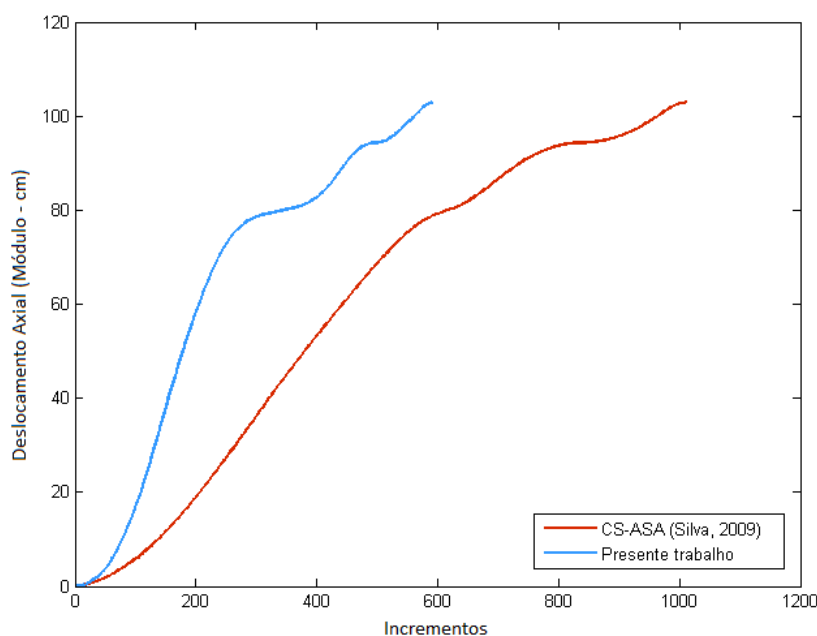


Figura 4.31: Pórtico de Lee: Deslocamento Axial x Incrementos.

Observando atentamente os gráficos representados nas Figuras 4.31 e 4.32, pode-se notar que eles representam a relação entre o deslocamento absoluto acumulado das tra-



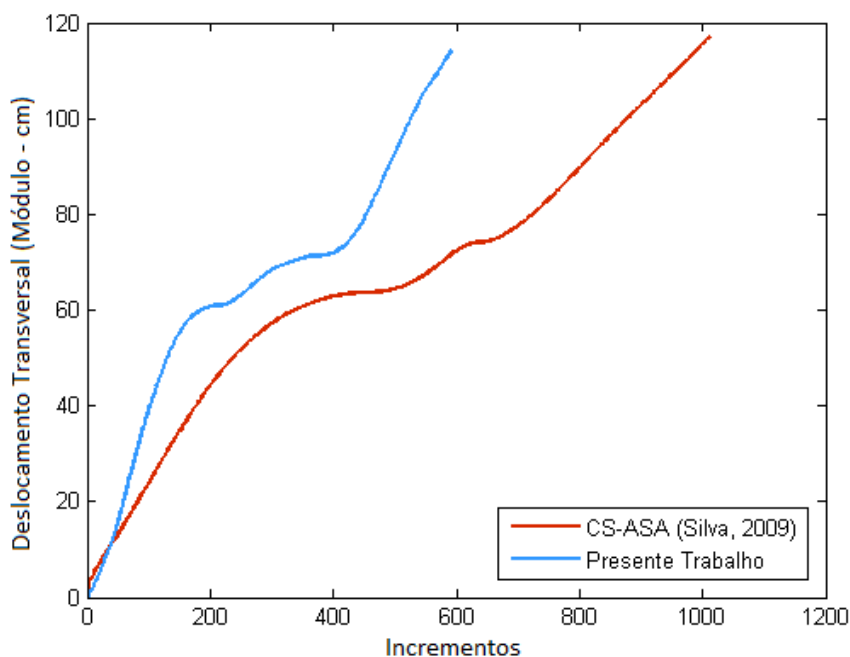


Figura 4.32: Pórtico de Lee: Deslocamento Transversal x Incrementos.

jetórias de equilíbrio axial e transversal e a quantidade de incrementos necessária para o alcance da solução, equivalente ao gráfico gerado na Figura 4.8. O código desenvolvido no presente trabalho necessitou (como observado na Seção 4.2.1.1) de 592 incrementos para a geração das trajetórias de equilíbrio enquanto o CS-ASA utilizou pouco mais de mil incrementos. Observe que o valor máximo das curvas no eixo referente ao deslocamento acumulado é o mesmo, enquanto o valor máximo das curvas no eixo referente à quantidade de incrementos difere consideravelmente. Esta divergência provavelmente se deve à utilização de limites máximos e mínimos para o cálculo do incremento de comprimento de arco no CS-ASA.

Nota-se, portanto que, no caso do pórtico de Lee, o código desenvolvido neste trabalho precisou de uma quantidade menor de incrementos para a geração dos mesmos deslocamentos axial e transversal que o CS-ASA, usando a mesma metodologia de solução.

A opção por funções generalizadas aliadas à escolha de estratégias eficazes adicionadas a escolha de uma linguagem mais simplificada em sua escrita pode levar a uma redução de código considerável. O código utilizado no presente trabalho é constituído de 23 funções desenvolvidas neste trabalho, possuindo um total de apenas 510 linhas, ao passo que o CS-ASA possui cerca de 200 funções próprias e aproximadamente 20 mil linhas.

## Capítulo 5

# Conclusões e Sugestões de Trabalhos Futuros

### 5.1 Introdução

Muitos pesquisadores têm se empenhado e direcionado suas pesquisas para o desenvolvimento de metodologias práticas e eficientes para uma análise não linear de sistemas estruturais. Todas estas pesquisas deram origem a códigos estruturados em Fortran que foram compilados e unidos em um grande sistema computacional, o CS-ASA (Silva, 2009) que, apesar de ser uma ferramenta poderosa na análise de estabilidade de estruturas esbeltas, é um código que pode ser de difícil aplicabilidade por usuários e novos desenvolvedores.

No presente trabalho foi desenvolvido um código computacional em ambiente Matlab que realiza análise estática não linear de pórticos planos, buscando uma abordagem mais didática, com códigos mais simples e organizados, onde o usuário possa compreender melhor a implementação computacional desta teoria, esclarecendo pontos obscuros da mesma e explicitando melhor sua organização e aplicação. Esta conexão entre metodologia e aplicação computacional foi exposta em maiores detalhes no Capítulo 3.

A Seção 5.2 apresenta algumas conclusões referentes ao desempenho e eficácia do código desenvolvido, levando em conta as análises realizadas pelo mesmo, que foram apresentadas no Capítulo 4.

Algumas sugestões para trabalhos futuros são fornecidas na Seção 5.3, objetivando a continuidade do desenvolvimento do presente trabalho, assim como a otimização em próximos estudos.

## 5.2 Conclusões

O código computacional desenvolvido neste trabalho utilizou, para todas as análises realizadas no Capítulo 4, as mesmas metodologias incrementais e iterativas.

As análises realizadas no referido Capítulo mostram que não houve a necessidade da implementação de outros métodos para que os resultados das análises fossem alcançados com sucesso.

Pode-se observar, nos gráficos do Capítulo 5, que os pontos encontrados na literatura coincidem com os gráficos gerados pelo código desenvolvido neste trabalho, sem alteração dos métodos. Isso mostra que o código computacional foi desenvolvido em concordância com a teoria de análise de estruturas de pórticos planos, validando assim a eficiência do código desenvolvido, tanto para problemas clássicos como para problemas fortemente não lineares.

Observando a tabela 4.2 na Seção 4.1.1, conclui-se que através das metodologias implementadas no código em questão, foi alcançada uma considerável redução no tempo de processamento e, conseqüentemente, no custo computacional, apenas através da alteração de informações constantes nos dados de entrada.

Com base nas informações anteriores, pode-se concluir que a utilização das estratégias incrementais e iterativas de comprimento de arco cilíndrico aliadas à estratégia de análise de sinal do incremento de carga por parâmetro GSP, possibilitam a realização das análises estáticas de pórticos planos com sucesso.

A validação das análises realizadas neste trabalho também podem ser observadas comparando-se os gráficos da Seção 4.1 e das subseções 4.2.1 e 4.2.2 com outros encontrados na literatura. Isso implica em uma considerável simplificação do código computacional desenvolvido e, conseqüentemente, um aumento significativo na qualidade da apresentação didática deste trabalho, o que acarreta em um melhor aprendizado da implementação deste tipo de análise por parte de novos usuários.

Outro aspecto que merece atenção reside na linguagem utilizada para o desenvolvimento deste código (Matlab), que reduziu consideravelmente o tamanho do código, bem como melhorou sua legibilidade.

A utilização de funções pré-existentes, assim como a própria estrutura da linguagem, criada com base matricial, simplificam o código, de forma que se torna mais visível ao usuário a estrutura da teoria da análise propriamente dita. Com isso, é possível ao usuário

prestar mais atenção à aplicação computacional da teoria, a qual na maioria dos casos acaba por permanecer em segundo plano em relação às estruturas inerentes à programação, que acabam por capturar a maior parte da atenção do usuário, dificultando assim a sua compreensão da implementação da análise.

### 5.3 Sugestões de Trabalhos Futuros

A implementação computacional da metodologia apresentada neste trabalho caracteriza uma base sólida para o desenvolvimento da análise de estruturas em geral.

Através da generalização de algumas funções pode-se ampliar a aplicação deste tipo de análise para outras estruturas além de pórticos planos, sem a necessidade de um aumento significativo deste código computacional. Esta alteração no código computacional permite que estruturas como treliças planas podem ser analisadas sem prejuízo computacional, assim como treliças e pórticos espaciais podem ser analisados sem a necessidade da inclusão de novas funções.

O código pode ser usado de base para novas pesquisas e futuras implementações que permitam realizações de análises mais realistas de modelos envolvendo elementos finitos reticulados. Sugere-se a implementação das demais formulações e metodologias do CS-ASA: Semirrigidez das ligações, análise não linear dinâmica, não linearidade física, problemas de restrições de contato, entre outros.

Uma interface amigável pode ser desenvolvida em ambiente Matlab, com o objetivo de se aumentar a interatividade do sistema com o usuário.

Otimizações no código propriamente dito, como vetorização das estruturas de repetição e paralelização de funções também podem ser realizadas para que análises mais robustas possam ser realizadas.

Podem ser realizados estudos de nanorressonadores utilizando-se o presente código como base computacional, assim como outras estruturas microscópicas.

Pode ser desenvolvido um projeto educacional envolvendo a utilização de uma plataforma educacional em ambiente web, com a utilização do presente código devidamente aprimorado, visando o desenvolvimento acadêmico.

## Referências

- [1] Allaire, G. e Pantz, O., 2006. Structural optimization with FreeFem++. *Struct Multidisc Optim* 32(3):173181. doi:10.1007/s00158-006-0017-y
- [2] Alves, R.V., 1995. Instabilidade Não Linear Elástica de Estruturas Reticuladas Espaciais. Tese de Doutorado, COPPE/UFRJ.
- [3] Andreassen, E., Clausen, A., Schevenels, M., Lazarov, B. e Sigmund, O., 2011. Efficient topology optimization in MATLAB using 88 lines of code. *Struct Multidisc Optim* 43(1):116. doi:10.1007/s00158-010-0594-7
- [4] Aranha Jr, Gandhi Yeddo da Rocha, 2003. A formulação de um Elemento Finito de Barra para Análise Dinâmica Não Linear Geométrica, com aplicação a cabos de Linhas Aéreas de Transmissão de Energia Elétrica, Universidade Federal do Pará. Centro Tecnológico, Belém.
- [5] Argyris, J. H., 1964. *Recent Advances in Matrix Methods of Structural Analysis*. Pergamon Press.
- [6] Batoz, J.L. e Dhatt, G., 1979. Incremental Displacement Algorithms for Nonlinear Problems. *Int. J. Numer. Methods Eng.*, Vol. 14, p. 1262-1267.
- [7] Bergan, P. G., Horrigmoe, G., Krakeland, B. e Soreide, T., 1978. Solution Techniques for Non-Linear Finite Element Problems. *Int. J. Numer. Methods Eng.*, Vol. 12, p. 1677-1696.
- [8] Bergan, P.G., 1980. Solution Algorithms for Nonlinear Structural Problems. *Computers Structures*, Vol. 12, p. 497-509.
- [9] Brito, G. L. R.; Detenborn, R. e Veloso, G., 2011. Ambiente Expresso para Processamento Distribuído com Matlab. Tocantins: S/UFTO
- [10] Carvalho, M. F. M. S., 2010. Formulação corrotacional para análise de vigas com elementos finitos. Lisboa: faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa (Dissertação de Mestrado)
- [11] Celes, W., Paulino, G. H., and Espinha, R., 2005. A compact adjacency-based topological data structure for finite element mesh representation. *International Journal for Numerical Methods in Engineering*, 64(11), pp. 1529-1556.
- [12] Chajes, A. e Churchill, J. E., 1987. Nonlinear Frame Analysis by Finite Element Methods. *Journal of Structural Engineering*, Vol. 113, No. 6, p. 1221-1235.
- [13] Challis, V.J., 2010. A discrete level-set topology optimization code written in matlab. *Struct Multidisc Optim* 41(3):453464. doi:10.1007/s00158-009-0430-0

- [14] Coelho, K. O.; Lages, E. N. e Martins, M. A. L., 2011. Linear Programming Applied to the Definition of the Maximum Load Capacity and Minimum Volume of a Truss Structure. Alagoas: LCCV/CTEC/UFAL
- [15] Cook, R.D., Malkus, D.S., e Plesha, M.E., 1989. Concepts and Applications of Finite Element Analysis, 3rd ed., New York, John Wiley Sons, Inc.
- [16] Crisfield, M.A., 1981. A Fast Incremental/Iterative Solution Procedure That Handles "Snap-Through", Computers Structures, Vol. 13, pp. 52-62.
- [17] Crisfield, M.A., 1991. Non-Linear Finite Element Analysis of Solids and Structures, Vol 1, John Wiley \* Sons.
- [18] Crisfield, M.A., 1997. Non-Linear Finite Element Analysis of Solids and Structures, Vol 2, John Wiley \* Sons.
- [19] Ebner, A. M., 1972. A Theoretical and Numerical Comparison of Elastic Nonlinear Finite Element Methods. Computers Structures, Vol. 2, P 1043-1061.
- [20] Galvão, A.S., 2000. Formulações geometricamente não lineares de elementos finitos para análise de sistemas estruturais metálicos reticulados planos. Dissertação de Mestrado, Ouro Preto: Programa de Pós-Graduação em Engenharia Civil, DE-CIV/Escola de Minas/UFOP.
- [21] Galvão, A.S., 2004. Estabilidade estática e dinâmica de pórticos planos com ligações semi-rígidas. Rio de Janeiro: Programa de Pós-Graduação em Engenharia Civil, Departamento de Engenharia Civil, PUC-Rio. (Tese de Doutorado).
- [22] Goto, Y. e Chen, W., 1987. Second-Order Elastic Analysis for Frame Design. Journal of the Structural Engineering, Vol. 113, No 7, p. 1500-1519.
- [23] Heijer, C. D. e Rheinboldt, W.C., 1981. On Steplength Algorithms for a Class of Continuation Methods. SIAM J. Num. Analysis, Vol. 18, p. 925-948.
- [24] Jennings, A., 1968. Frame Analysis Including Change of Geometry. Journal of the Structural Division. ASCE, Vol. 94, p. 627-644.
- [25] Krenk, S., 1993. Dual Orthogonality Procedure for Nonlinear Finite Element Equations. Engineering Mechanics. Department of Building Technology and Structural Engineering, Aalborg University, Denmark, No. 12, p. 01-18.
- [26] Krenk, S., 1995, An Orthogonal Residual Procedure for Non-Linear Finite Element Equations, Int. J. Numer. Methods Eng., vol. 38, p. 823-839.
- [27] Lavall, A. C. C.; Fakury, R. H.; Silva, R. G. L. e Leandro Oliveira, L. A. R., 2011. Análise Avançada de Pórticos de Aço com Ligações Semirrígidas conforme as prescrições da ABNT NBR 8800: 2008. Minas Gerais: DEES/EE/UFMG
- [28] Leon, S. E.; Paulino, G. H.; Pereira, A.; Menezes, I. F. M.; Lages, E. N., 2011. A Unified Library of Nonlinear Solution Schemes. Rio de Janeiro: Tecgraf/PUC-Rio
- [29] Liu, Z., Korvink, J.G. e Huang, R., 2005. Structure topology optimization: fully coupled level set method via FEMLAB. Struct Multidisc Optim 29:407417

- [30] Machado, F.C.S., 2005. Análise Inelástica de Segunda-ordem de Sistemas Estruturais Metálicos. Dissertação de Mestrado, Programa de Pós-Graduação em Engenharia Civil, Deciv/EM/UFOP, Ouro Preto, MG, Brasil.
- [31] Maciel, F. V., 2012. Equilíbrio e Estabilidade de Elementos Estruturais com Restrições Bilaterais Impostas por Bases Elásticas. Dissertação de Mestrado. Minas Gerais: UFOP.
- [32] Mallett, R. H. e Marcal, P. V., 1968. Finite Element Analysis of Nonlinear Structures. Journal of the Structural Division. Proc. ASCE, Vol. 94, No. ST9, p. 2081-2103.
- [33] Martin, H. C., 1965. On The Derivation of Stiffness Matrices for The Analysis of Large Deflection and Stability Problems. Conference of Matrix Methods in Structural Mechanics, Wright-Patterson Air Force Base, Ohio.
- [34] Meek, J.L. e Tan, H.S., 1984. Geometrically Nonlinear Analysis of Space Frames by an Incremental Iterative Technique. Comp. Methods Appl. Mech. Eng., Vol. 47, p. 261-282.
- [35] Neuenhofer, A. e Filippou, F.C., 1998. Geometrically Nonlinear Flexibility-Based Frame Finite Element. Journal of the Structural Engineering, Vol. 124, No. 6, p. 704-711.
- [36] Pacoste, C. e Eriksson, A., 1997. Beam elements in instability problems. Comput. Methods Appl. Mech. Engrg., No 144, p. 163-197.
- [37] Pinheiro, L., 2003. Análises não lineares de sistemas estruturais metálicos rotulados e semi-rígidos. Dissertação de Mestrado. Ouro Preto: PROPEC/EM/UFOP.
- [38] Powell, G. H., 1969. Theory of nonlinear elastic structures. J. struct. Div., ASCE, Vol 95, No 12, p. 2687-2701.
- [39] Prado, I. M., 2012. CS-ASA Preprocessor: Sistema Gráfico Interativo de Pré-processamento para Análise Avançada de Estruturas. Dissertação de Mestrado. Minas Gerais: UFOP.
- [40] Queiros, L. O. A., 2007. Análise estrutural de galpões pré-moldados em contrato considerando a influência da rigidez nas ligações viga-pilar. Dissertação de Mestrado. Universidade Federal de Alagoas. Centro de Tecnologia, Maceió.
- [41] Ramm, E., 1981. Strategies for Tracing the Non-Linear Response Near Limit-Points, Non-linear Finite Element Analysis in Structural Mechanics. Springer-Verlag, Berlim, pp. 63-89.
- [42] Riks, E., 1972. The Application of Newton's Methods to the Problems Elastic Stability. Int. J. Solids Structures, ASME Journal of Applied Mechanics, Vol. 39, pp. 1060-1066.
- [43] Riks, E., 1979. An Incremental Approach to the Solution of Snapping and Buckling Problems. Int. J. Solids Structures, Vol. 15, p. 529-551.

- [44] Rocha, G., 2000. Estratégias de Incremento de Carga e de Iteração para Análise Não-Linear de Sistemas Estruturais, Dissertação de Mestrado, Deciv-UFOP, Ouro Preto.
- [45] Rocha, P.A.S., 2006. Análise inelástica de Segunda Ordem de Estruturas Metálicas com Ligações Semi-rígidas. Dissertação de Mestrado, Programa de Pós-Graduação em Engenharia Civil, Deciv/EM/UFOP, Ouro Preto, MG, Brasil.
- [46] Santos, M.N., 2007. Emprego de Elemento Finito Híbrido na Análise Não-linear de Estruturas Metálicas. Dissertação de Mestrado, Programa de Pós-Graduação em Engenharia Civil, Deciv/EM/UFOP, Ouro Preto, MG, Brasil.
- [47] Schweizerhof, K. H. e Wriggers, P., 1986. Consistent Linearization for path Following Methods in Nonlinear FE Analysis. . *Comp. Methods Appl. Mech. Eng.*, Vol. 59, p. 269-279.
- [48] Sigmund, O., 2001. A 99 line topology optimization code written in Matlab. *Struct Multidisc Optim* 21(2):120127
- [49] Silva, A.R.D., 2009. Sistema Computacional para Análise Avançada Estática e Dinâmica de Estruturas Metálicas. Tese de Doutorado. Minas Gerais: UFOP.
- [50] Silva, S. S. e Silva, W. T. M., 2011. Nonlinear Analysis of Plane Frames using a Corotational Formulation and Beam Elements 2D. PECC/DCEE/UnB.
- [51] Silveira, R.A.M., 1995. Análise de elementos estruturais esbeltos com restrições unilaterais de contato. Tese de Doutorado. Rio de Janeiro: PUC-Rio.
- [52] Silveira, R.A.M., Rocha, G., e Gonçalves, P.B., 1999. Estratégias numéricas para análises geometricamente não-lineares. Anais do XV Congresso Brasileiro de Engenharia Mecânica (COBEM/99), Águas de Lindóia/SP, Brasil, Novembro, 22-26/11/1999, 10 páginas, CD-ROM, ISBN: 85-85769-03-3.
- [53] Silveira, R.A.M., Rocha, G., e Gonçalves, P.B., 1999. Estratégias de incremento de carga e iteração para análise não-linear de estruturas. Anais do XX Congresso Ibero Latino Americano sobre Métodos Computacionais para Engenharia (XX CILAMCE), Universidade de São Paulo, São Paulo/SP, Brasil, Novembro, pp. 213.1-213.20, CD-ROM, ISBN: 85-901027-1-8.
- [54] Southwell, R. V., 1941. *An introduction to the Theory of Elasticity for Engineers and Physicists*, 2a edn., Oxford University Press, Oxford, England.
- Suresh K (2010) A 199-line matlab code for Pareto-optimal tracing in topology optimization. *Struct Multidisc Optim* 42(5):665 679. doi:10.1007/s00158-010-0534-6
- [55] Timoshenko, S.P. e Gere, J.E., 1982. *Mecânica dos Sólidos*. Livros Técnicos e Científicos, Vol 01.
- [56] Torkamani, M.A.M.; Sonmez, M. e Cao, J., 1997. Second-Order Elastic Plane-Frame Analysis Using Finite-Element Method. *Journal of Structural Engineering*, Vol 12, No 9, p. 1225-1235.



- 
- [57] Wen, R.K.; e Rahimzadeh J., 1983. Nonlinear Elastic Frame Analysis by Finite Element. *Journal of the Structural Engineering*, Vol. 109, No 8, p. 1952-1971.
  - [58] Wong, M.B. e Tin-Loi, F., 1990. Geometrically Nonlinear Analysis of Elastic Framed Structures. *Computers \* Structures*, Vol. 34, No. 4, p. 633-640.
  - [59] Yang, Y. B. e Kuo, S. B., 1994. *Theory Analysis of Nonlinear Framed Structures*, Prentice Hall.
  - [60] Zienkiewicz, O.C., 1971. *The Finite Element in Engineering Science*, McGraw-Hill, London.