# ANÁLISE ESTÁTICA E DINÂMICA POR SUBESTRUTURAÇÃO E PROGRAMAÇÃO ORIENTADA POR OBJETOS

10/90

Este exemplar corresponde à redação final da tese defendida por Marco Lúcio Bittencourt e aprovada pela comissão julgadora em 21/02/90.

Cos Lyntt Cd.

p/ Fernando Iguti

UNIVERSIDADE ESTADUAL DE CAMPINAS
FACULDADE DE ENGENHARIA MECÂNICA
DEPARTAMENTO DE PROJETO MECÂNICO

# ANÁLISE ESTÁTICA E DINÂMICA POR SUBESTRUTURAÇÃO E PROGRAMAÇÃO ORIENTADA POR OBJETOS

MARCO LÚCIO BITTENCOURT

Tese apresentada à Faculdade de Engenharia Mecânica - UNICAMP, como parte dos requisitos exigidos para obtenção do título de MESTRE EM ENGENHARIA MECÂNICA.

Campinas, 21 de Fevereiro de 1990

**UNIVERSIDADE ESTADUAL DE CAMPINAS**
**FACULDADE DE ENGENHARIA MECÂNICA**
**DEPARTAMENTO DE PROJETO MECÂNICO**

Tese de: Mestrado

Título da Tese: Análise Estática e Dinâmica por Subestruturação e Programação Orientada por Objetos

Autor: Marco Lúcio Bittencourt

Orientador: Prof. Dr. Fernando Iguti

Aprovado por

Prof. Dr. Fernando Iguti, Presidente

Prof. Dr. Raul Antonino Feijóo

Prof. Dr. Paulo Roberto G. Kurka

Campinas, 21 de Fevereiro de 1990

# Agradecimentos

Ao Prof. Dr. Fernando Iguti pela orientação do trabalho, incentivo, amizade e dedicação dispensados.

Ao amigo Prof. Janito Vaqueiro Ferreira pela colaboração indispensável na implementação dos algorítmos de análise estrutural, preparação dos exemplos e apresentação dos resultados.

Ao amigo Prof. Ilmar Ferreira Santos pelo incentivo, amizade sincera e revisão do trabalho.

Ao Prof. Dr. Raul Feijóo pelas críticas e sugestões ao trabalho.

Ao Armando Arruda pela gentileza na confecção dos desenhos.

Aos amigos Rogério C. Bastos Duarte e Waldemar Scudeller Jr. pela colaboração na apresentação dos resultados e revisão do trabalho.

A FAPESP - Fundação de Amparo à Pesquisa do Estado de São Paulo, pela concessão da bolsa de estudos.

A todo pessoal do GEPROM pelo ambiente agradável proporcionado, essencial para o bom andamento do trabalho.

*Aos meus pais Fábio e Terezinha;*
*aos meus irmãos Magdeline e Murilo;*
*e a Maristela pelo apoio e carinho.*

*ANÁLISE ESTÁTICA E DINÂMICA POR SUBESTRUTURAÇÃO E PROGRAMAÇÃO ORIENTADA POR OBJETOS*

# RESUMO

Este trabalho apresenta, inicialmente, algorítmos para análise linear estática e dinâmica de estruturas modeladas por elementos finitos, considerando as técnicas de subestruturação e ciclo-simetria. Posteriormente, especificam-se as características gerais de um programa computacional implementando estas técnicas, utilizando-se para isso, os conceitos provenientes da engenharia de programas e do paradigma orientado por objetos.

*STATIC AND DYNAMIC SUBSTRUCTURING ANALYSIS AND OBJECT-ORIENTED PROGRAMMING*

# ABSTRACT

This work presents firstly some algorithms for static and dynamic analysis of finite elements modelled structures, using substructuring and cyclic symmetry techniques. The general characteristics of a computer program is then described, which implements such techniques, using software enginnering and object-oriented concepts.

# Conteúdo

# Lista de Figuras

# Lista de Tabelas

# Capítulo 1

# INTRODUÇÃO

Este capítulo tem por objetivo introduzir os vários conceitos abordados neste trabalho. Inicialmente, considera-se a técnica de análise por subestruturação e um algoritmo específico para análise de estruturas ciclo-simétricas. Na segunda parte do trabalho, discutem-se alguns conceitos de engenharia de programas e do modelo orientado por objetos, os quais são aplicados na especificação do programa FAE – Ferramenta de Análise Estrutural.

## 1.1 Análise por Subestruturação

O MEF possui uma grande versatilidade, permitindo a sua aplicação nos mais variados tipos de problemas. No entanto, em muitos casos, o número de graus de liberdade necessário para a modelagem estrutural excede a capacidade de memória dos computadores disponíveis. Assim, torna-se conveniente uma partição da estrutura em subestruturas para que a análise possa ser efetuada. Esta partição pode ser realizada de forma arbitrária, porém é interessante realizá-la fisicamente.

A técnica de subestruturação pode ser resumida, de maneira geral, em três passos básicos [1]:

- divisão da estrutura global em uma série de subestruturas.
- determinação das propriedades (neste caso massa e rigidez) de cada subestrutura e montagem das respectivas matrizes.
- obtenção da equação global de movimento da estrutura pela superposição das matrizes de cada subestrutura, e solução desta equação para a determinação das características de interesse no estudo da estrutura.

Segundo [2,3], há várias vantagens em se utilizar a técnica de subestruturação em relação a técnica convencional, podendo-se citar:

- a demanda computacional envolvida na análise de uma estrutura modelada por subestruturas é, geralmente, inferior àquela necessária em uma análise global.
- a preparação e a verificação dos dados do modelo estrutural é bastante simplificada.
- subestruturas iguais necessitam ser especificadas e reduzidas apenas uma vez.
- modificações no modelo são efetuadas de maneira local, ou seja, apenas nas subestruturas correspondentes as partes a serem reanalisadas.
- a precisão numérica obtida é superior em relação a técnica convencional.

Uma subestrutura pode ser especificada de maneira recursiva, permitindo a construção de um modelo estrutural com vários níveis de subestruturação [4,5]. Os elementos finitos são os componentes básicos desta hierarquia, podendo, no entanto, estar presentes em qualquer um dos níveis. Ressalta-se que esta hierarquia não necessita ser uniforme, ou seja, uma subestrutura pode ser constituída por outras pertencentes a quaisquer outros níveis, e não somente ao nível imediatamente inferior. A Figura 1.1 ilustra um modelo por subestruturação em vários níveis.

A técnica de subestruturação na análise estática de estruturas pode ser encontrada em Przemieniecki [6], estando por sua vez implementada em vários programas comerciais [2,7]. Neste caso, as subestruturas, a partir da determinação das suas propriedades de rigidez ou flexibilidade, podem ser consideradas como elementos estruturais complexos. Os métodos matriciais de deslocamento ou força podem então, ser aplicados à estrutura fracionada. Uma vez que os deslocamentos ou forças nos contornos da subestrutura são determinados, pode-se analisar cada subestrutura separadamente sob deslocamentos ou forças nos contornos conhecidos, dependendo do método empregado. Neste trabalho, considera-se apenas o método dos deslocamentos ou da rigidez.

Na análise estática, a técnica de subestruturação é exata, consistindo, basicamente, de uma condensação das variáveis internas de cada subestrutura em função das variáveis de contorno.

No Capítulo 2, apresenta-se a técnica de subestruturação em vários níveis para análise linear estática. Discute-se então, o desenvolvimento de um programa protótipo implementando este algoritmo. Aplicam-se estes conceitos na análise de uma estrutura reticulada, verificando-se que os resultados obtidos são os mesmos daqueles provenientes da análise global [8]. Neste caso, procura-se apenas validar a formulação considerada, não se preocupando em avaliar a performance computacional das técnicas convencional e por subestruturação.

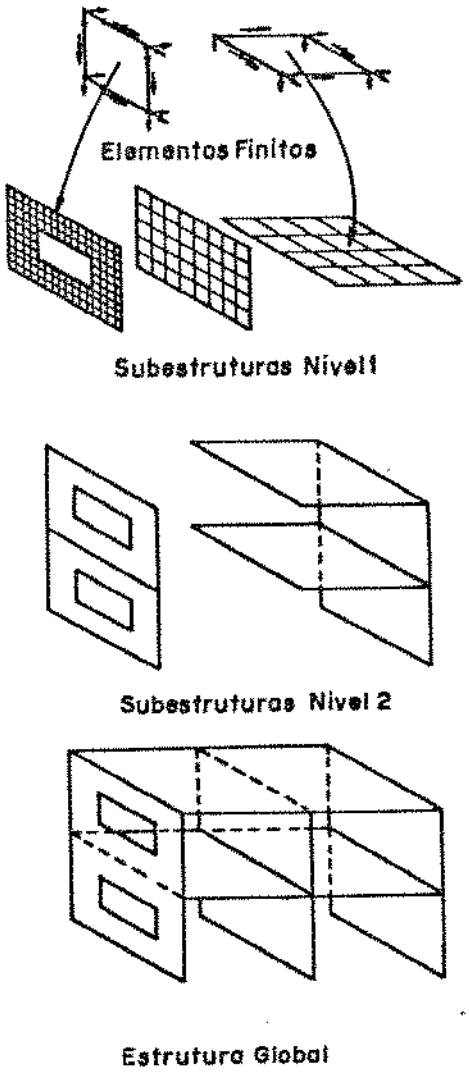


Figura 1.1: Modelo por subestruturação em vários níveis [3].

## 1.2 Estruturas Ciclo-simétricas

Uma grande quantidade de estruturas pode ser modelada utilizando uma concepção axissimétrica, podendo-se citar, como exemplo, os vasos de pressão. No entanto, quando este tipo de modelamento não é possível, deve-se analisar a estrutura globalmente. Assim, torna-se interessante aplicar outros métodos que explorem a característica de simetria de uma estrutura.

O conceito de simetria é frequentemente, empregado em diversas áreas de Engenharia. Pode-se citar, como exemplo, a utilização de simetria reflexiva, a qual permite reduzir o tamanho de um modelo pela metade para cada plano de simetria do sistema físico em estudo. No entanto, existem outros tipos de simetria, tal como a simetria cíclica ou rotacional.

Na simetria cíclica, as propriedades de um corpo são repetidas em intervalos iguais ao redor de um eixo de simetria. Este conceito foi inicialmente, empregado no estudo de redes elétricas polifásicas [9]. Posteriormente, estendeu-se a formulação matemática de simetria cíclica para análise estrutural utilizando o MEF [10,11]. Desta forma, reduziu-se não só a demanda computacional exigida na análise de estruturas ciclo-simétricas, mas também a tarefa de preparação do modelo discreto.

Uma estrutura ciclo-simétrica de ordem $N$ é aquela constituída de $n$ $(n = 1, \ldots, N)$ regiões simetricamente dispostas em torno de um eixo a cada intervalo de $2\pi/N$ radianos. A Figura 1.2 ilustra o modelo discreto de uma estrutura ciclo-simétrica de ordem 6. Assim, esta estrutura é constituída de 6 regiões giradas entre si de um ângulo de 60° em torno do eixo $z$ (plano $xy$) no sentido anti-horário. Denomina-se de região fundamental o segmento correspondente a $n = 1$.

Verifica-se que a aplicação do conceito de simetria cíclica em análise estrutural apresenta as seguintes vantagens [2,10]:

- o analista necessita especificar o modelo estrutural apenas para a região fundamental, o qual é, geralmente menor que o modelo obtido por técnicas convencionais.
- o tempo computacional envolvido é reduzido em relação aquele necessário em uma análise global.
- o condicionamento numérico é superior comparado ao método convencional, devido ao melhor condicionamento das matrizes em relação à resolução numérica dos sistemas de equações e problemas de autovalor envolvidos.

No entanto, para realizar uma análise estática ou dinâmica de estruturas ciclo-simétricas, deve-se empregar algumas transformações baseadas na série de Fourier discreta. Desta forma, torna-se possível manipular o modelo matemático da região funda-

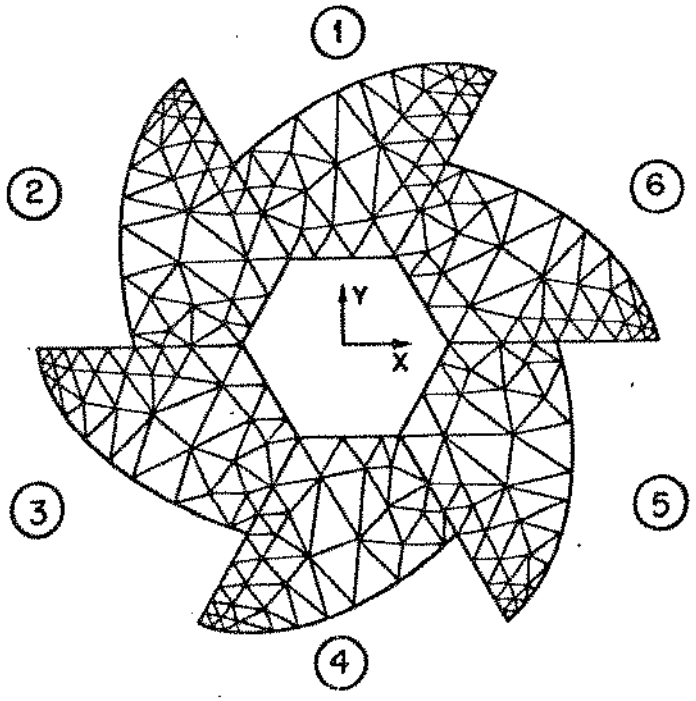


Figura 1.2: Modelo discreto de uma estrutura ciclo-simétrica [11].

mental em termos de componentes cíclicos. Assim, a sequência de passos mostrada na Figura 1.3 é empregada neste tipo de análise.

Observa-se então, que a partir do modelo discreto da região fundamental em grandezas físicas, representa-se este modelo em componentes cíclicos. Os sistemas de equações ou problemas de autovalor provenientes são resolvidos, obtendo-se os parâmetros de interesse expressos em componentes cíclicos, os quais são, finalmente, transformados para grandezas físicas.

As transformações cíclicas envolvidas são apresentadas no Capítulo 3, juntamente com a equação de equilíbrio estático para estruturas ciclo-simétricas expressa em componentes cíclicos. Neste caso, os graus de liberdade internos do modelo discreto da região fundamental são condensados ao longo do contorno, possibilitando uma redução do número de variáveis do modelo. Implementou-se este algoritmo no programa ANAFIN [12], aplicando-se o mesmo na análise estática de uma estrutura reticulada.

Na análise dinâmica, é muito comum o problema de determinação das frequências naturais e modos de vibrar de uma estrutura. Pode-se aplicar as transformações cíclicas para análise dinâmica de estruturas ciclo-simétricas. Assim, a partir da equação de vibração livre da estrutura [13,14], obtém-se os problemas de autovalor em componentes cíclicos, os quais podem ser resolvidos pelo método de iteração por subespaço [15].

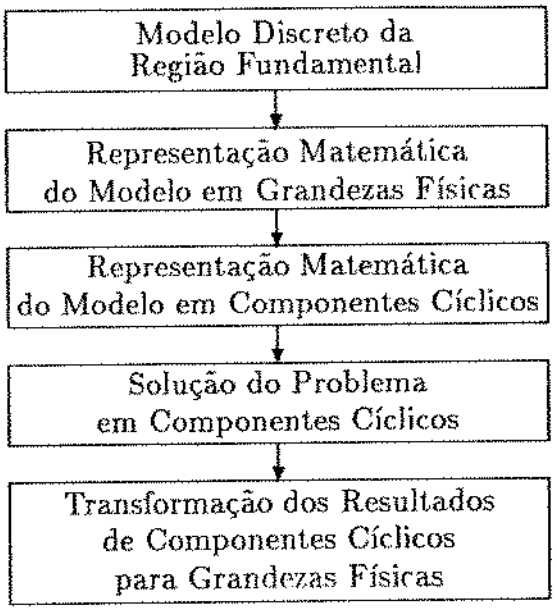


Figura 1.3: Algoritmo para análise de estrutura ciclo-simétrica.

Este algoritmo é discutido no Capítulo 4, observando-se que a formulação matemática utilizada considera apenas grandezas reais. No entanto, usando-se a aritmética complexa, a descrição do método é simplificada, além de permitir uma interpretação física de forças aplicadas na estrutura [16]. Estes conceitos foram implementados no programa ANAFIN [12] e aplicados no estudo do comportamento dinâmico de uma estrutura reticulada.

Para casos gerais de estruturas modeladas por um conjunto de subestruturas, esta técnica de ciclo-simetria não pode ser aplicada. Desta forma, desenvolveram-se outros métodos para análise dinâmica por subestruturação.

Pode-se diminuir o número de graus de liberdade do modelo global, utilizando-se algumas técnicas de redução [17,18]. Uma outra possibilidade, é analisar cada subestrutura separadamente, e manipular os resultados destas análises para se obter as características da estrutura global. Vários métodos foram desenvolvidos com este objetivo [19,20,21,22,23], verificando-se, no entanto, a necessidade de se resolver um problema de autovalor para cada subestrutura. Observa-se que alguns métodos não possuem esta restrição [24,25,26]. Apresenta-se no Capítulo 3 uma breve discussão dos métodos de subestruturação dinâmica. No Apêndice B, considera-se o método de Síntese por Subestruturação, assim como uma técnica iterativa para a síntese discreta [27].

## 1.3 Engenharia de Programas

Ao longo dos últimos anos, a importância dos programas em um sistema de informação vem aumentando substancialmente, devido ao fato que os programas representam, atualmente, o elemento principal nestes sistemas, diferenciando um produto de seus similares.

No entanto, os programas tornaram-se complexos e ambiciosos, de difícil gerenciamento e manutenção, de custos elevados e de baixa confiabilidade e qualidade. Além disso, as características dos sistemas de informação nem sempre refletem as necessidades de seus usuários. Isto implicou na *crise de software* [28].

Por outro lado, a introdução de tecnologias de empacotamento de componentes na indústria eletrônica permitiu um crescimento exponencial no seu desenvolvimento. Equipamentos cada vez mais sofisticados estão sendo produzidos e o nível de integração e especificação dos circuitos tem crescido em curtos períodos de tempo. Assim, computadores com grande capacidade de processamento estão disponíveis a número crescente de usuários. No entanto, as técnicas tradicionais de desenvolvimento de programas não são capazes de explorar todos estes recursos.

Enquanto na indústria de componentes eletrônicos verificou-se uma revolução nos modos de produção, os programas continuam sendo desenvolvidos a partir de declarações simples de uma linguagem de programação. Isto pode ser comparado ao modo como os circuitos integrados eram construídos inicialmente, ou seja, a partir de componentes discretos, como por exemplo transistores, resistores e capacitores.

Portanto, novas metodologias de desenvolvimento de programas devem ser empregadas, visando aproximações sistemáticas e confiáveis. Estas soluções passam necessariamente pelas técnicas de Engenharia de Programas.

No Capítulo 5, aborda-se, inicialmente, os vários aspectos da crise de *software*, verificando-se que não há uma solução única para todos os problemas provenientes desta crise. No entanto, através da aplicação sistemática de métodos da Engenharia de Programas nas várias fases de desenvolvimento de um programa, pode-se obter resultados satisfatórios.

Discute-se então, os paradigmas de programação, destacando-se o paradigma de construção de protótipos, pois este pode ser utilizado para a definição dos requerimentos de um programa. Observa-se que esta metodologia foi adotada neste trabalho, visando identificar as principais características dos algoritmos de análise estrutural a serem implementados no programa FAE.

Segundo Presmann [28], independente do paradigma empregado, a implantação de um programa compreende as fases de definição, desenvolvimento e manutenção. As características mais relevantes destas fases são discutidas no Capítulo 5, enfatizando-se as etapas de especificação de sistemas e análise de requerimentos da fase de definição

de um programa.

O principal objetivo da análise de sistemas é elaborar uma definição clara do sistema de informação a partir das necessidades gerais especificadas pelo usuário, alocando funções para os vários elementos deste sistema. Já na análise de requerimentos, procura-se representar o domínio de informação definido para os elementos de programa, caracterizando-se as funções, performance e interfaces requeridas.

Estes conceitos são aplicados para a definição das características principais do programa FAE. Considera-se, neste caso, o modelo orientado por objetos, discutido na próxima seção.

## 1.4 Programação Orientada por Objetos

O estilo de programação convencional é baseado no modelo dados/procedimentos. Assim, um sistema é desenvolvido baseado nas funções que o mesmo executa, sendo a estrutura de dados compartilhadas por estas funções. Portanto, as funções assumem, previamente, os tipos de dados a serem manipulados.

Nos últimos anos, o conceito de programação orientada por objetos tem sido aplicado no desenvolvimento de programas. Neste modelo, o projeto conceitual de um sistema de informação é baseado nos objetos que o mesmo manipula, ao invés das funções que executa.

Um objeto é um módulo constituído de uma estrutura de dados e um conjunto de operações que manipula estes dados. A comunicação entre os objetos é feita através do envio de mensagens, as quais especificam apenas a tarefa a ser realizada. O objeto deve decifrar as mensagens e selecionar a função a ser executada.

O mecanismo de herança permite construir uma hierarquia de objetos, onde cada objeto situado em um nível inferior da hierarquia herda as características daqueles localizados nos níveis superiores. Assim, através de um conceito geral, novas características podem ser acrescentadas num processo de especialização da informação.

Segundo Rentsch [29], a programação orientada por objetos será nos próximos anos, o que a programação estruturada foi nos anos 70. O termo programação orientada por objetos é derivado do conceito de classe introduzido pela linguagem *Simula 67*. Entretanto, foi a partir do desenvolvimento do sistema de programação *Smalltalk-80* [30,31] que o paradigma orientado por objetos foi apresentado de modo mais uniforme. Assim, o sistema *Smalltalk* tornou-se o elemento principal na programação orientada por objetos [29].

Entretanto, a linguagem *Smalltalk* apresenta baixa eficiência computacional, inibindo a sua aplicação no desenvolvimento de sistemas complexos. Ressalta-se porém, que arquiteturas dedicadas ao estilo de programação *Smalltalk* vêm sendo desenvolvidas,

assim como novas versões da linguagem, o que permitirá uma maior utilização deste sistema de programação.

O conceito de programação por objetos tem sido incorporado à várias linguagens convencionais. Pode-se citar, como exemplo, a linguagem híbrida *Objective-C* [32], a qual contém todas as características da linguagem C [33], incorporando ainda, os conceitos de orientação por objetos ao estilo de *Smalltalk-80* [31]. A linguagem C++ [34] constitui-se numa redefinição da linguagem C, visando implementar algumas características de programação por objetos. No entanto, para se desenvolver sistemas baseados no modelo de objetos pode-se utilizar uma linguagem convencional, podendo-se tomar como exemplo o sistema de radioterapia desenvolvido com Pascal padrão [35]. Verifica-se que este modelo apresenta algumas limitações sob alguns aspectos relacionados à programação por objetos.

No Capítulo 6, apresentam-se alguns conceitos gerais, tais como modulação e abstração de dados [28,36], assim como definições específicas do modelo orientado por objetos [37,38]. Compara-se então, as técnicas de programação convencional e por objetos. Descrevem-se, de forma geral, duas linguagens de programação no Apêndice C.

Observa-se que as várias metodologias de desenvolvimento de programas empregadas não tem possibilitado a construção sistemática e confiável de programas. Uma das qualidades desejáveis em um programa é o reaproveitamento dos seus elementos em outras aplicações. Visando este objetivo, algumas técnicas vêm sendo desenvolvidas e aplicadas [39]. O emprego do modelo orientado por objetos representa uma aproximação realista para a produção de componentes de programa reaproveitáveis [40,41].

Segundo Cox [32] e Ledbetter [40], os elementos de programas devem ser desenvolvidos de maneira análoga à construção de circuitos integrados, ou seja, a partir do reaproveitamento de componentes básicos com funções e interfaces bem definidas. Estes módulos são denominados *Software-ICs*.

A Figura 1.4 ilustra um problema, frequentemente, encontrado ao se reaproveitar componentes de programas já implementados. O desenvolvimento de programas possui uma natureza repetitiva. Assim, um programador sempre desenvolve uma nova versão sobre os mesmos temas básicos: *Nem sempre é o mesmo, nem sempre é um outro* [41].

No Capítulo 6, discute-se a reutilização de código, comparando-se as técnicas de desenvolvimento de circuitos integrados e programas. Ao final, verifica-se como o modelo por objetos pode ser aplicado visando o reaproveitamento de programas.

## 1.5 FAE - Ferramenta de Análise Estrutural

Os métodos de Engenharia de Programas permitem um desenvolvimento disciplinado e confiável para os programas. Por sua vez, os conceitos de orientação por objetos

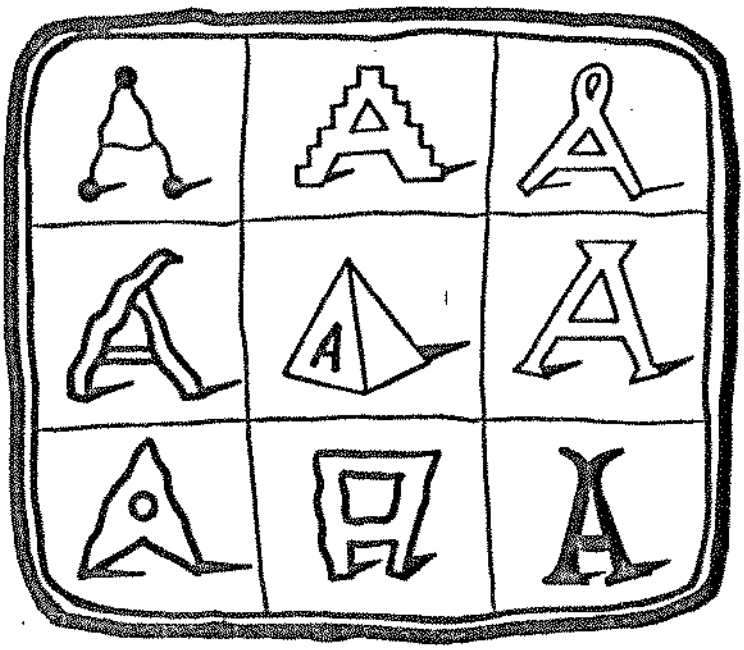

Figura 1.4: *Nem sempre é o mesmo, nem sempre é um outro* [41].

possibilitam a obtenção de programas modulares, onde a independência entre os objetos do programa é uma das características principais. Além disso, o paradigma de objetos representa uma aproximação realista para o reaproveitamento de código. As técnicas de análise estrutural abordadas neste trabalho, reduzem, de forma significativa, a ordem dos sistemas de equações e problemas de autovalor provenientes dos processos de análise.

Assim, o programa FAE (Ferramenta de Análise Estrutural) tem por objetivo implementar algoritmos para análise linear (estática e dinâmica) de estruturas modeladas pelo MEF, empregando-se as técnicas convencional, por subestruturação e ciclo-simétrica. Para isso, aplicam-se os conceitos de Engenharia de Programas para a definição e documentação das características gerais do programa. A decomposição dos domínios funcional e de informação do programa é baseada no modelo por objetos, considerando assim, os conceitos de modulação e abstração de dados. Desta forma, novas técnicas de análise estrutural podem ser implementadas de maneira simples, sem alterar a estrutura básica do programa. Além disso, pode-se elaborar programas de acordo com as necessidades específicas de um usuário, devido a característica modular dos objetos.

Confeccionou-se os documentos de Especificação do Sistema [42] e Análise de Requerimentos [43] para o programa FAE. No Capítulo 7, discutem-se as principais características destes documentos, assim como as necessidades gerais do programa FAE. Os objetos identificados, para esta primeira versão do programa, são apresentados,

destacando-se os seus atributos e suas operações básicas. Desenvolveu-se um documento para cada um destes objetos descrevendo os seus requisitos [44]. O modelo adotado para este documento pode ser visto no Apêndice D.

# Capítulo 2

# ANÁLISE ESTÁTICA POR SUBESTRUTURAÇÃO

## 2.1 Introdução

Neste capítulo, apresenta-se, inicialmente, a formulação matemática da técnica de subestruturação estática baseada no método dos deslocamentos. Posteriormente, discute-se o modelo por subestruturação em vários níveis e a construção de um programa protótipo implementando esta técnica. Finalmente, aplicam-se estes conceitos na análise estática de uma estrutura reticulada.

## 2.2 Análise Estática por Subestruturação utilizando o Método dos Deslocamentos

A formulação matemática apresentada a seguir considera a técnica de subestruturação em vários níveis [4], sendo uma extensão daquela abordada em [6].

No método dos deslocamentos, cada subestrutura é analisada separadamente das demais, assumindo-se que todos os contornos comuns às subestruturas adjacentes estão completamente fixos. Os contornos são então, relaxados simultaneamente e os deslocamentos presentes são determinados a partir das equações de equilíbrio de forças nos contornos. Desta maneira, cada subestrutura pode ser analisada separadamente sob carregamentos e deslocamentos conhecidos nos contornos [6].

O conjunto completo de equações de equilíbrio para uma estrutura pode ser escrito na seguinte forma matricial,

$$[K]\{U\} = \{P\} \tag{2.1}$$

onde $[K]$ é a matriz de rigidez, $\{U\}$ é o vetor dos deslocamentos correspondentes às forças externas $\{P\}$.

A estrutura é então, dividida em $n$ subestruturas, introduzindo-se contornos internos. O nível de subestruturação é indicado pelo índice $m$ ($m = 1, 2, \ldots, M$), sendo $M$ o último nível. A notação $n, m$ indica a n-ésima subestrutura do m-ésimo nível. Portanto, pode-se particionar a equação matricial (2.1) em,

$$\begin{bmatrix} [K_{n,m}]_{cc} & [K_{n,m}]_{ci} \\ [K_{n,m}]_{ci}^T & [K_{n,m}]_{ii} \end{bmatrix} \begin{Bmatrix} \{U_{n,m}\}_c \\ \{U_{n,m}\}_i \end{Bmatrix} = \begin{Bmatrix} \{P_{n,m}\}_c \\ \{P_{n,m}\}_i \end{Bmatrix} \tag{2.2}$$

onde o vetor $\{U_{n,m}\}_c$ denota os deslocamentos, geralmente, comuns a duas ou mais subestruturas; $\{U_{n,m}\}_i$ é o vetor de deslocamentos internos; $\{P_{n,m}\}_c$ e $\{P_{n,m}\}_i$ são os vetores de forças externas correspondentes a $\{U_{n,m}\}_c$ e $\{U_{n,m}\}_i$, respectivamente.

Os deslocamentos globais da estrutura podem ser calculados a partir da superposição de dois vetores,

$$\{U_{n,m}\} = \{U_{n,m}\}^{(\alpha)} + \{U_{n,m}\}^{(\beta)} \tag{2.3}$$

onde $\{U_{n,m}\}^{(\alpha)}$ denota o vetor dos deslocamentos devido a $\{P_{n,m}\}_i$ com $\{U_{n,m}\}_c = 0$, ou seja, com contornos internos fixos. O vetor $\{U_{n,m}\}^{(\beta)}$ representa as correções necessárias aos deslocamentos $\{U_{n,m}\}^{(\alpha)}$ para permitir deslocamentos $\{U_{n,m}\}_c$ nos contornos das subestruturas, com $\{P_{n,m}\}_i = 0$. Logo, a equação (2.3) torna-se,

$$\{U_{n,m}\} = \begin{Bmatrix} \{U_{n,m}\}_c \\ \{U_{n,m}\}_i \end{Bmatrix} = \underbrace{\begin{Bmatrix} \{U_{n,m}\}_c^{(\alpha)} \\ \{U_{n,m}\}_i^{(\alpha)} \end{Bmatrix}}_{\substack{\text{contornos} \\ \text{internos} \\ \text{fixos}}} + \underbrace{\begin{Bmatrix} \{U_{n,m}\}_c^{(\beta)} \\ \{U_{n,m}\}_i^{(\beta)} \end{Bmatrix}}_{\substack{\text{contornos} \\ \text{internos} \\ \text{relaxados}}} \tag{2.4}$$

onde $\{U_{n,m}\}_c^{(\alpha)} = 0$.

Analogamente, as forças externas são separadas em,

$$\{P_{n,m}\} = \{P_{n,m}\}^{(\alpha)} + \{P_{n,m}\}^{(\beta)} \tag{2.5}$$

ou seja,

$$\{P_{n,m}\} = \begin{Bmatrix} \{P_{n,m}\}_c \\ \{P_{n,m}\}_i \end{Bmatrix} = \underbrace{\begin{Bmatrix} \{P_{n,m}\}_c^{(\alpha)} \\ \{P_{n,m}\}_i^{(\alpha)} \end{Bmatrix}}_{\substack{\text{contornos} \\ \text{internos} \\ \text{fixos}}} + \underbrace{\begin{Bmatrix} \{P_{n,m}\}_c^{(\beta)} \\ \{P_{n,m}\}_i^{(\beta)} \end{Bmatrix}}_{\substack{\text{contornos} \\ \text{internos} \\ \text{relaxados}}} \tag{2.6}$$

onde $\{P_{n,m}\}_i^{(\alpha)} = \{P_{n,m}\}_i$ e $\{P_{n,m}\}_i^{(\beta)} = 0$.

Assim, pode-se escrever, a partir de (2.2), as seguintes equações

$$\begin{bmatrix} [K_{n,m}]_{cc} & [K_{n,m}]_{ci} \\ [K_{n,m}]_{ci}^T & [K_{n,m}]_{ii} \end{bmatrix} \begin{Bmatrix} \{U_{n,m}\}_c^{(\alpha)} \\ \{U_{n,m}\}_i^{(\alpha)} \end{Bmatrix} = \begin{Bmatrix} \{P_{n,m}\}_c^{(\alpha)} \\ \{P_{n,m}\}_i^{(\alpha)} \end{Bmatrix} \tag{2.7}$$

$$\begin{bmatrix} [K_{n,m}]_{cc} & [K_{n,m}]_{ci} \\ [K_{n,m}]_{ci}^T & [K_{n,m}]_{ii} \end{bmatrix} \begin{Bmatrix} \{U_{n,m}\}_c^{(\beta)} \\ \{U_{n,m}\}_i^{(\beta)} \end{Bmatrix} = \begin{Bmatrix} \{P_{n,m}\}_c^{(\beta)} \\ \{P_{n,m}\}_i^{(\beta)} \end{Bmatrix} \tag{2.8}$$

Quando os contornos das subestruturas são fixados tem-se, a partir de (2.7), que

$$\{U_{n,m}\}_i^{(\alpha)} = [K_{n,m}]_{ii}^{-1} \{P_{n,m}\}_i \tag{2.9}$$

$$\{P_{n,m}\}_c^{(\alpha)} = [K_{n,m}]_{ci} [K_{n,m}]_{ii}^{-1} \{P_{n,m}\}_i = \{R_{n,m}\}_c \tag{2.10}$$

onde $\{P_{n,m}\}_c^{(\alpha)}$ representa as reações nos contornos internos necessárias para manter $\{U_{n,m}\}_c = 0$ quando as forças $\{P_{n,m}\}_i$ são aplicadas no interior de cada subestrutura.

Com os contornos internos livres, vem através de (2.8) que os deslocamentos $\{U_{n,m}\}_i^{(\beta)}$ são dados por,

$$\{U_{n,m}\}_i^{(\beta)} = -[K_{n,m}]_{ii}^{-1} [K_{n,m}]_{ci}^T \{U_{n,m}\}_c^{(\beta)} \tag{2.11}$$

$$\{U_{n,m}\}_c^{(\beta)} = [K_{n,m}]_c^{-1} \{P_{n,m}\}_c^{(\beta)} \tag{2.12}$$

onde,

$$[K_{n,m}]_c = [K_{n,m}]_{cc} - [K_{n,m}]_{ci} [K_{n,m}]_{ii}^{-1} [K_{n,m}]_{ci}^T \tag{2.13}$$

representa a matriz de rigidez do contorno das subestruturas. A matriz $\{P_{n,m}\}_c^{(\beta)}$ é determinada a partir de (2.5) e (2.10)

$$\{P_{n,m}\}_c^{(\beta)} = \{P_{n,m}\}_c - \{P_{n,m}\}_c^{(\alpha)} = \{P_{n,m}\}_c - [K_{n,m}]_{ci} [K_{n,m}]_{ii}^{-1} \{P_{n,m}\}_i = \{S_{n,m}\}_c \tag{2.14}$$

Observa-se que a matriz $[K_{n,m}]_c$ é obtida a partir da superposição das matrizes de rigidez de contorno das subestruturas pertencentes aos outros níveis $r$ ($r = 1, \ldots, m-1$) de subestruturação. Determina-se o vetor $\{P_{n,m}\}_c^{(\beta)}$, de maneira análoga. Estes conceitos são abordados na próxima seção.

## 2.3 Subestruturação Estática em Vários Níveis

Para se aplicar a técnica de subestruturação em vários níveis, deve-se inicialmente, analisar cada subestrutura, efetuando-se uma redução da matriz de rigidez e do vetor de carregamento relacionados com os graus de liberdade de contorno internos especificados, aplicando-se as equações (2.13) e (2.14), respectivamente.

O processo de obtenção das matrizes de rigidez das subestruturas inicia-se no primeiro nível de subestruturação e possui uma característica recursiva. Considerando então, a n-ésima subestrutura do primeiro nível, determina-se a sua matriz de rigidez global $[K_{n,1}]_g$ através da superposição das matrizes de rigidez dos elementos finitos que constituem esta subestrutura. Assim, obtém-se as matrizes $[K_{n,1}]_{ii}$, $[K_{n,1}]_{cc}$ e $[K_{n,1}]_{ci}$, reduzindo-se a matriz de rigidez global para a de contorno interno $[K_{n,1}]_c$. Este processo é realizado para todas as subestruturas distintas do primeiro nível. O vetor de carregamento é determinado por um processo de superposição e condensado em função dos graus de liberdade de contorno interno.

Para o segundo nível de subestruturação, determina-se a matriz global de uma subestrutura através da superposição das matrizes de rigidez de contorno das subestruturas do primeiro nível, e dos elementos finitos que eventualmente estejam presentes neste nível. O vetor de carga é determinado de maneira análoga, ou seja, pela superposição do carregamento de contorno das subestruturas do primeiro nível e das forças externas aplicadas no contorno. Cada subestrutura é então, reduzida para as variáveis de contorno especificadas para o segundo nível. Observa-se que vários graus de liberdade de contorno do primeiro nível são agora considerados como internos, sendo portanto, condensados.

Aplica-se então, este processo recursivamente até que se atinja o topo da hierarquia de subestruturas. Desta maneira, calculam-se os deslocamentos de contorno deste nível através da resolução de um sistema de equações. Conhecendo-se os deslocamentos de contorno, determinam-se os deslocamentos internos. A partir daí, tem-se, novamente, um processo recursivo para o cálculo dos deslocamentos nodais das subestruturas em todos os níveis, mas em sentido inverso aquele da montagem das matrizes de rigidez. Os deslocamentos de contorno das subestruturas, pertencentes ao nível imediatamente inferior ao topo da hierarquia, são idênticos aos deslocamentos internos correspondentes já calculados para as subestruturas do nível mais alto, bastando apenas atribuir estes valores aos deslocamentos de contorno das subestruturas deste nível. Desta forma, calculam-se todos os deslocamentos nodais da estrutura, modelada como um conjunto de subestruturas, até que se atinja o primeiro nível de subestruturação.

Ressalta-se que todas as restrições físicas de contorno devem ser consideradas em todos os níveis de subestruturação em que os graus de liberdade correspondentes estiverem presentes. Eliminam-se as linhas e colunas da matriz de rigidez, assim como a

linha do vetor de carregamento, para estes graus de liberdade restritos no nível onde foram especificados. Assim, obtém-se, ao final do processo de montagem, uma matriz global de contorno das subestruturas não singular.

Considere a n-ésima subestrutura do m-ésimo nível. Suponha que esta subestrutura seja constituída de $s$ subestruturas, pertencentes a qualquer um dos níveis inferiores $r$ ($r = 1, \ldots, m-1$), e de $l$ elementos finitos. A matriz de rigidez global $[K_{n,m}]_g$ desta subestrutura é obtida pela superposição das matrizes de rigidez de contorno $[K_{p,r}]_c$ ($p = 1, \ldots, s$) das subestruturas e das matrizes de rigidez $[K^e_{q,m}]$ ($q = 1, \ldots, l$) dos elementos finitos. Portanto,

$$[K_{n,m}]_g = \left[ [K_{1,r}]_c \ \ldots \ [K_{s,r}]_c \ \left[K^e_{1,m}\right] \ \ldots \ \left[K^e_{l,m}\right] \right] \tag{2.15}$$

Conhecendo-se os graus de liberdade de contorno da subestrutura $n$, obtém-se, por inspeção de $[K_{n,m}]_g$, as matrizes $[K_{n,m}]_{ii}$, $[K_{n,m}]_{cc}$ e $[K_{n,m}]_{ci}$. Utilizando-se a equação (2.13), chega-se à matriz de rigidez de contorno $[K_{n,m}]_c$. Eliminam-se então, as linhas e as colunas correspondentes aos graus de liberdade de contorno restritos.

O vetor de carregamento de contorno da subestrutura é determinado através da equação,

$$\{S_{n,m}\}_c = -\sum_{p=1}^{s} \{R_{p,r}\}_c + \{P_{n,m}\}_c \tag{2.16}$$

onde a somatória é feita sob as reações de contorno $\{R_{p,r}\}_c$ presentes nas $s$ subestruturas e $\{P_{n,m}\}_c$ é o vetor das forças externas aplicadas no contorno. O sinal negativo implica que as reações nos contornos são consideradas como forças externas aplicadas.

Este processo é aplicado a todas as subestruturas em todos níveis. Quando se atinje o topo da hierarquia, denotado por $M$, tem-se o sistema de equações global em termos apenas das variáveis de contorno das subestruturas deste nível, podendo ser expresso pela equação,

$$[K_{1,M}]_c \{U_{1,M}\}_c = \{S_{1,M}\}_c \tag{2.17}$$

Desta forma, calculam-se os deslocamentos de contorno $\{U_{1,M}\}_c$ para este nível. A partir daí, determinam-se os deslocamentos internos, empregando-se a equação (2.2). Escrevendo-se de maneira geral, tem-se que,

$$\{U_{n,m}\}_i = [K_{n,m}]_{ii}^{-1} \left( \{P_{n,m}\}_i - [K_{n,m}]_{ic} \{U_{n,m}\}_c \right) \tag{2.18}$$

Atribuem-se os deslocamentos de contorno para as subestruturas do nível inferior e utiliza-se a equação (2.18) para o cálculo dos deslocamentos internos deste nível. Aplica-se este procedimento recursivamente até se atingir o primeiro nível de subetruturação. Desta forma, determinam-se todos deslocamentos nodais do modelo estrutural.

## 2.4 Implementação de um Programa Protótipo

Visando implementar a técnica de análise por subestruturação em vários níveis, deve-se considerar as vantagens apresentadas na Seção 1.1. O algoritmo é essencialmente recursivo, no sentido que o processo se repete até que uma condição especificada seja encontrada. Neste caso, um nível de recursão termina, durante o processo de montagem das matrizes de rigidez e vetores de carregamento de uma subestrutura, quando se atinge um elemento finito, visto que estes elementos não são definidos de uma forma recursiva. Assim, basta acessar um procedimento para o cálculo da matriz de rigidez deste elemento finito. Desta forma, torna-se interessante tratar de maneira uniforme as subestruturas e os elementos finitos.

Foi implementado um programa protótipo para análise estática por subestruturação de estruturas reticuladas, visando verificar a aplicação do algoritmo discutido nas seções anteriores, não havendo, no entanto, a preocupação com a eficiência geral do programa.

Para a estruturação dos dados deste programa, utilizou-se um modelo similar aquele apresentado em [5]. A especificação das subestruturas é realizada de forma consistente, permitindo simplificar o modelamento da estrutura considerada, mesmo em casos mais complexos. Além disso, as subestruturas são especificadas em relação a um sistema de referência local, possibilitando o seu reaproveitamento nos demais níveis da hierarquia, assim como a construção de uma biblioteca de subestruturas visando o modelamento de outras estruturas. Observa-se, no entanto, que ao se passar de um nível de subestruturação para outro, deve-se efetuar uma transformação de coordenadas.

A montagem e condensação das matrizes de rigidez e vetores de carregamento é realizada através de uma operação recursiva. Ressalta-se que a matriz de rigidez necessita ser montada e reduzida apenas uma vez, e armazenada em disco para uso posterior. Já o vetor de carregamento não possui esta característica, sendo, portanto, necessário condensá-lo sempre que houver a presença de qualquer carregamento em uma subesturutra. Esta operação, denominada *monta*, pode ser melhor entendida através do seguinte fluxo:

```
monta (subestrutura)
  if (número de subestruturas componentes)
    for i = 1 : número de subestruturas
      leitura dos dados da subestrutura i
      if (subestrutura reduzida)
        leitura da matriz de rigidez de contorno
        montagem recursiva do vetor de carregamento
      else
        monta (subestrutura i)
      end
      superposição da matriz de rigidez e vetor de carregamento
```

```
      da subestrutura no nível superior
    end
  else
    monta matriz de rigidez e vetor de carregamento de elemento finito
    return
  end
  superposição do carregamento presente no nível considerado
  if (número de nós de contorno)
    condensação da matriz de rigidez e vetor de carregamento
  end
  armazena em disco matriz de rigidez condensada
end
```

Assim, a operação descrita recebe como entrada os dados da subestrutura, ou seja, as subestruturas de outros níveis que a compõem, as coordenadas e incidência nodais, nós de contorno, carregamento e condições de contorno. No caso em que a subestrutura for especificada em função de outras subestruturas $i$, realiza-se a leitura da matriz de rigidez do disco e a montagem recursiva do carregamento, ou executa-se novamente a operação *monta* com os dados da i-ésima subestrutura. Verifica-se que as matrizes de rigidez e vetores de carregamento de elementos finitos são calculados sempre que forem necessários, pois estas matrizes podem variar para diferentes ocorrências dos elementos na hierarquia de subestruturas.

Considere a n-ésima subestrutura do nível $m$. Observa-se que após a obtenção da matriz de rigidez e vetor de carregamento globais, deve-se superpor o carregamento especificado para esta subestrutura. A partir daí, se a subestrutura possuir nós de contorno, condensam-se os graus de liberdade internos aplicando as equações (2.13) e (2.14). Armazena-se então, a matriz de rigidez da subestrutura para posterior utilização.

Ao final da operação, obtém-se a matriz de rigidez e o vetor de carregamento de contorno para a subestrutura no topo da hierarquia. Calculam-se então, os deslocamentos nodais de contorno, e através da aplicação recursiva da equação (2.18) os deslocamentos nodais para as demais subestruturas da hierarquia especificada, como apresentado na Seção 2.3.

## 2.5 Análise Estática de uma Estrutura Reticulada

Considere a estrutura reticulada plana ilustrada na Figura 2.1. Os seguintes valores foram utilizados para o carregamento indicado e as propriedades físicas e geométricas dos elementos de barra:

- $F_1 = -10{,}0$ N $\quad F_2 = -8{,}0$ N $\quad F_3 = -8{,}0$ N $\quad F_4 = -6{,}0$ N.

- $F_5 = -10{,}0$ N $\quad F_6 = -6{,}0$ N $\quad F_7 = 6{,}0$ N $\quad F_8 = 8{,}0$ N.
- Módulo de elasticidade longitudinal: $E = 20{,}0 \times 10^5$ N/m$^2$.
- Área da secção transversal: $A = 0{,}01$ m$^2$.

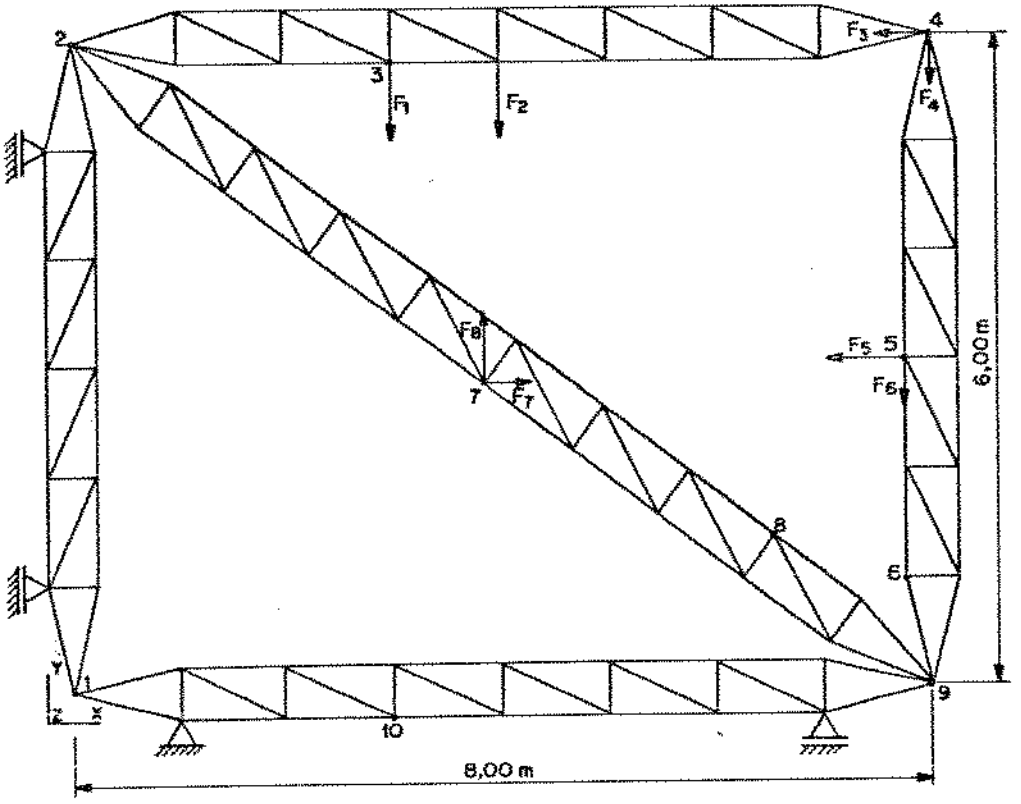


Figura 2.1: Estrutura reticulada analisada.

Esta treliça foi, inicialmente, discretizada e analisada utilizando-se o método convencional matricial, empregando-se 135 graus de liberdade [8]. Analisou-se então, a mesma estrutura através de um modelo por subestruturação em seis níveis. A hierarquia de subestruturas considerada é mostrada na Figura 2.2, e a topologia destas subestruturas na Figura 2.3.

No topo desta hierarquia, tem-se a subestrutura *ESTR*, a qual é constituída de duas subestruturas *TREL6*, duas *TREL8* e uma *TREL10*. Já no nível básico, têm-se elementos finitos de barra com dois graus de liberdade por nó. Verifica-se porém, que

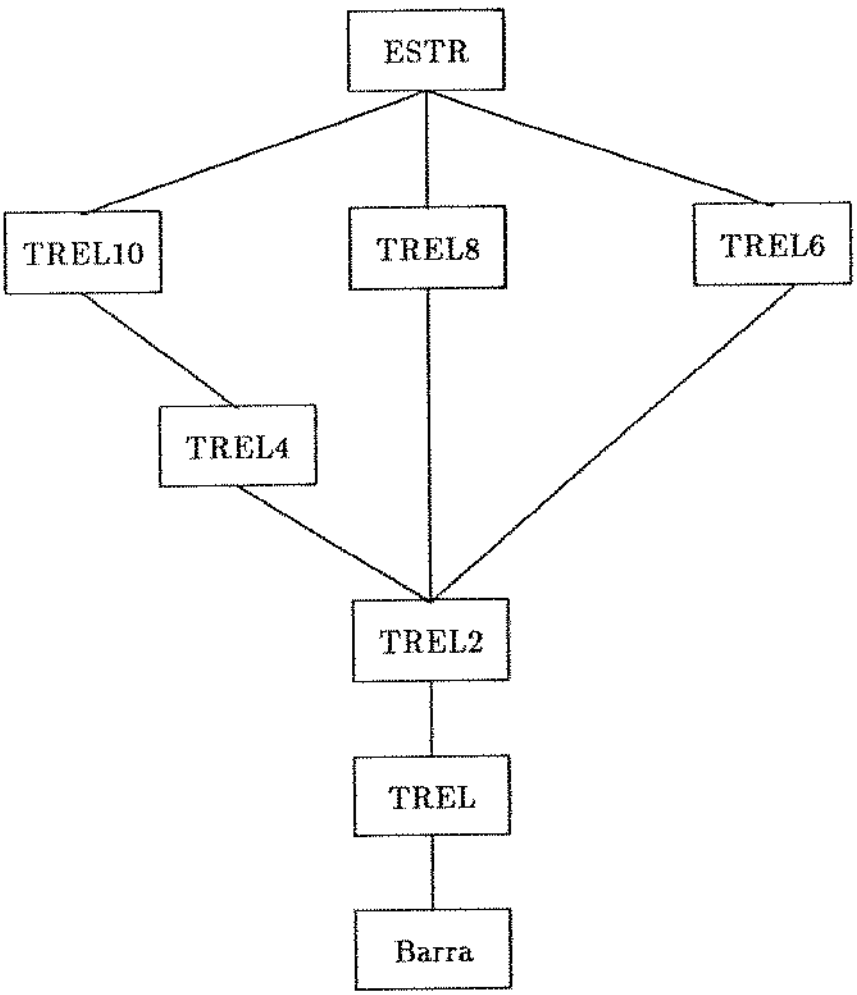


Figura 2.2: Hierarquia de subestruturas.

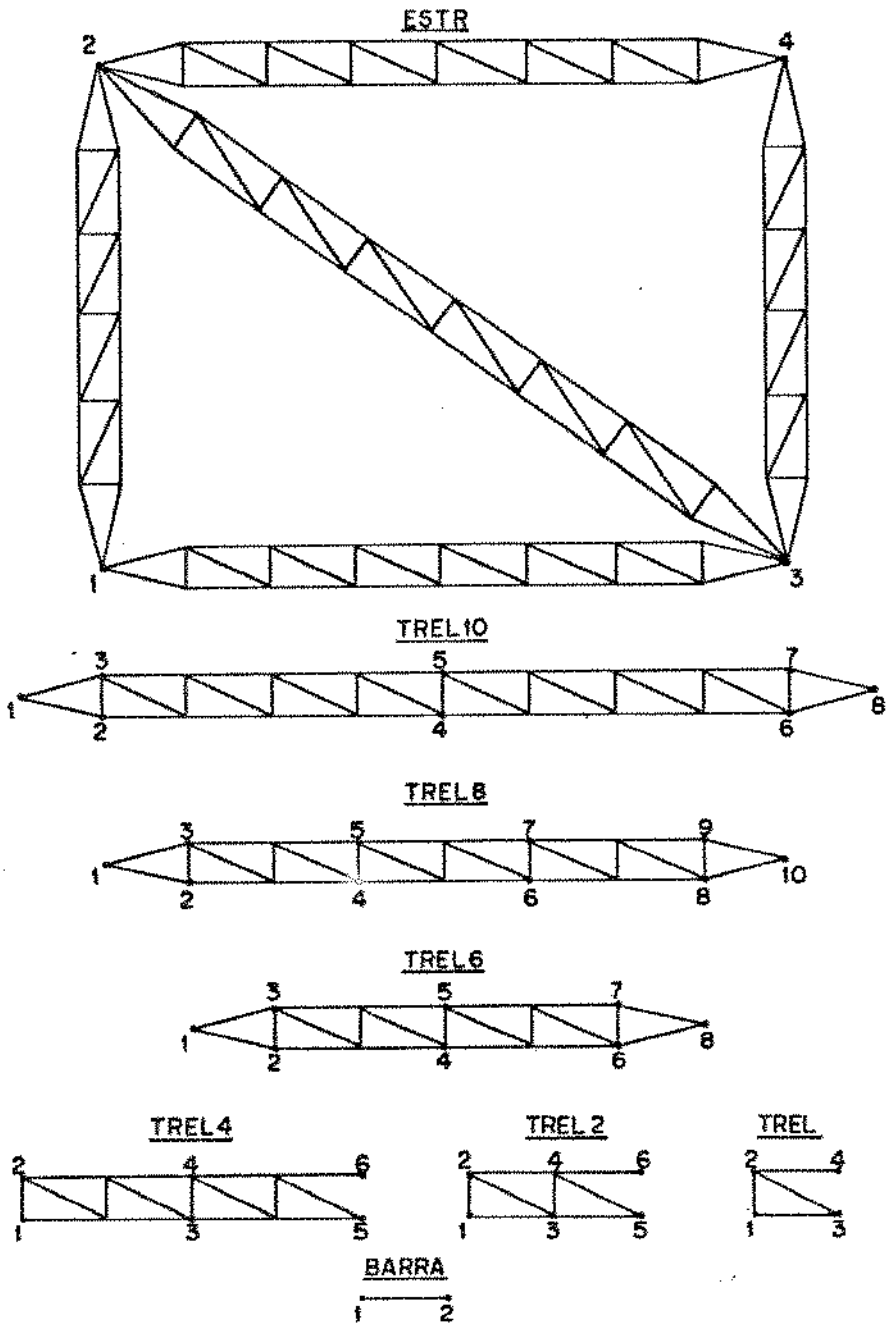


Figura 2.3: Topologia das subestruturas.

uma subestrutura situada num certo nível pode ser especificada em função de subestruturas de qualquer nível inferior, como por exemplo *TREL8*. A Tabela 2.1 apresenta a composição das subestruturas, assim como seus nós de contorno.

| Subestrutura | Componentes | Nós de contorno |
|---|---|---|
| ESTR | 2 TREL6 | |
| | 2 TREL8 | |
| | 1 TREL10 | |
| TREL10 | 2 TREL4 | 1 8 |
| | 5 Barras | |
| TREL8 | 3 TREL2 | 1 10 |
| | 5 Barras | |
| TREL6 | 2 TREL2 | 1 8 |
| | 5 Barras | |
| TREL4 | 2 TREL2 | 1 2 5 6 |
| TREL2 | 2 TREL | 1 2 5 6 |
| TREL | 4 Barras | 1 2 5 6 |

Tabela 2.1: Subestruturas: constituição e nós de contorno.

Como mencionado anteriormente, as subestruturas são modeladas utilizando um sistema de coordenadas locais. Para ilustrar este fato, considere a subestrutura *TREL10* mostrada na Figura 2.4. As coordenadas nodais foram especificadas em relação ao sistema de referência local indicado, sendo os valores destas coordenadas apresentados na Tabela 2.2.

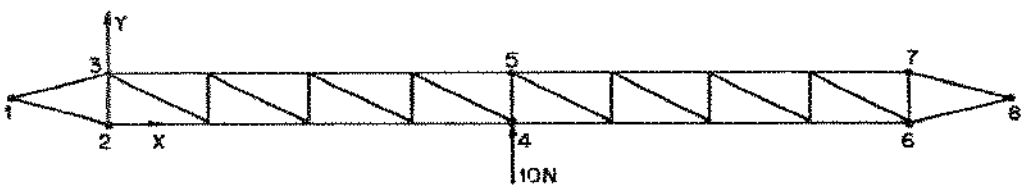


Figura 2.4: Subestrutura TREL10.

Observa-se na Figura 2.4 que o carregamento foi especificado segundo o sistema de referência local. Portanto, as forças $F_7$ e $F_8$ na Figura 2.1 foram obtidas a partir da força de 10 N, aplicada ao nó 4 de *TREL10*, devido a transformação de coordenadas para o sistema de referência de *ESTR*.

| Nó | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| x | −1,0 | 0,0 | 0,0 | 4,0 | 4,0 | 8,0 | 8,0 | 9,0 |
| y | 0,25 | 0,0 | 0,5 | 0,0 | 0,5 | 0,0 | 0,5 | 0,25 |

Tabela 2.2: Coordenadas nodais da subestrutura TREL10.

Os resultados obtidos foram os mesmos tanto na análise global quanto pela técnica de subestruturação. A Tabela 2.3 apresenta os deslocamentos dos nós indicados na Figura 2.1.

| Nó | u [cm] | v [cm] |
|---|---|---|
| 1 | −0.3375 | −0,1810 |
| 2 | −0,6837 | 1,9750 |
| 3 | −1,1480 | −9,9720 |
| 4 | −0,9432 | −2,1740 |
| 5 | −3,0470 | −2,1360 |
| 6 | −1,4870 | −2,180 |
| 7 | 5,040 | 5,374 |
| 8 | 2,989 | 1,803 |
| 9 | −0,4506 | −1,919 |
| 10 | −0,2504 | 1,815 |

Tabela 2.3: Deslocamentos dos nós indicados para ESTR.

A geometria deformada da estrutura analisada pode ser vista na Figura 2.5.

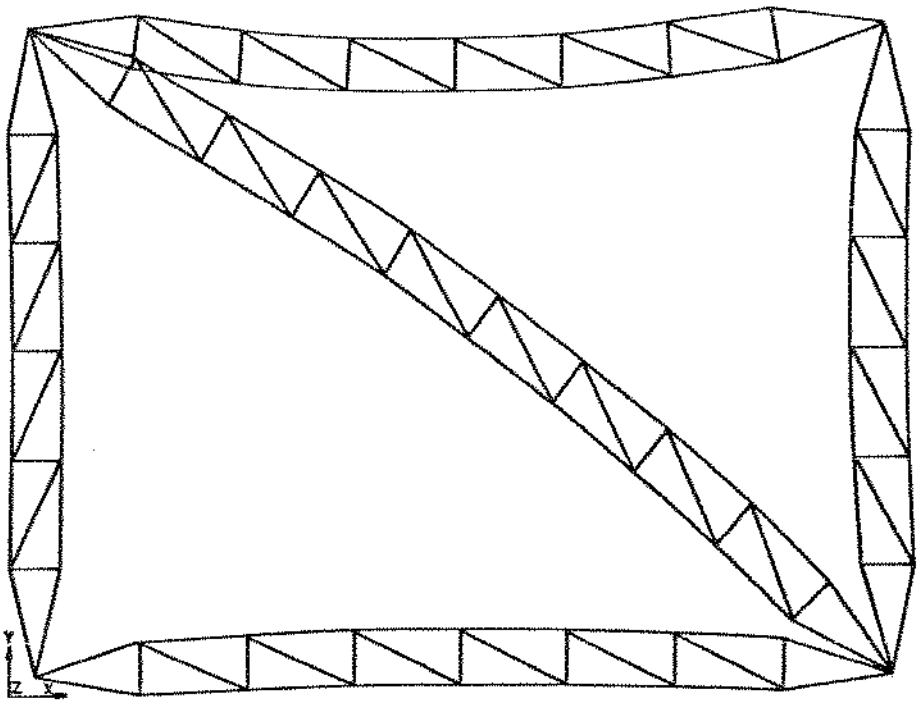

Figura 2.5: Geometria deformada de ESTR.

# Capítulo 3

# ANÁLISE ESTÁTICA DE ESTRUTURAS CICLO-SIMÉTRICAS

## 3.1 Introdução

Este capítulo tem por objetivo apresentar o conceito de simetria cíclica aplicado na análise estática de estruturas ciclo- simétricas modeladas pelo MEF. Inicialmente, discutem-se as transformações envolvidas neste tipo de análise, as quais são baseadas na série de Fourier discreta. Para o caso estático, condensam-se os graus de liberdade internos do modelo discreto da região fundamental ao longo do contorno. Obtém-se então, a equação de equilíbrio estático da estrutura, expressa em componentes cíclicos, utilizando-se as transformações apresentadas. Finalmente, consideram-se as condições de compatibilidade entre os segmentos, aplicando-se as equações resultantes na análise de uma estrutura reticulada.

## 3.2 Componentes Cíclicos

Considere um conjunto de $N$ vetores $\{U_n\}$ $(n = 1, \ldots, N)$ de ordem $m$ representando uma grandeza física qualquer. A partir deste conjunto de vetores, pode-se gerar outro, aplicando-se a série de Fourier discreta, ou seja [11],

$$\{\chi^k\} = \frac{1}{N} \sum_{n=1}^{N} \{U_n\} \exp\left(-\imath \frac{2\pi k(n-1)}{N}\right) \quad (k = 0, \ldots, N-1) \tag{3.1}$$

onde,

$$\exp\left(-\imath\frac{2\pi k(n-1)}{N}\right) = \cos\left(\frac{2\pi k(n-1)}{N}\right) + \imath\mathrm{sen}\left(\frac{2\pi k(n-1)}{N}\right)$$

é a função exponencial complexa de período $2\pi$.

Observa-se que quando $k = 0$, $\{\chi^0\}$ é a média aritmética dos vetores $\{U_n\}$ $(n = 1,\ldots,N)$, isto é,

$$\{\chi^0\} = \frac{1}{N}(\{U_1\} + \{U_2\} + \ldots + \{U_{N/2}\} + \ldots + \{U_N\})$$

e quando $N$ for par, tem-se para $k = N/2$,

$$\{\chi^{N/2}\} = \frac{1}{N}(\{U_1\} - \{U_2\} + \ldots + \{U_{N/2}\} - \ldots + (-1)^{N-1}\{U_N\})$$

Como a série de Fourier discreta é periódica, tem-se que os vetores $\{\chi^{N-k}\}$ são conjugados de $\{\chi^k\}$ $(k = 0,\ldots,N/2)$. Verifica-se que todos os vetores $\{\chi^k\}$ $(k = 0,\ldots,N-1)$ possuem ordem $2m$, excetuando-se os casos onde $k = 0$ e $k = N/2$ ($N$ par). Os $m$ primeiros componentes destes vetores correspondem aos termos em co-seno e os demais aos termos em seno. A série de Fourier discreta inversa é dada pela seguinte equação:

$$\{U_n\} = \sum_{k=0}^{N-1}\{\chi^k\}\exp\left(\imath\frac{2\pi k(n-1)}{N}\right) \quad (n = 1,\ldots,N) \tag{3.2}$$

Assim, as expressões (3.1) e (3.2) definem um par de transformações lineares, unitário e finito, não envolvendo nenhuma aproximação na sua aplicação para o problema discreto considerado, em relação à técnica convencional [11].

A teoria de componentes cíclicos estabelece que um conjunto de $N$ vetores com relações de fases arbitrárias pode ser decomposto em $N$ conjuntos de $N$ vetores de mesmas fases e magnitudes [10]. Considerando três vetores $\{U_1\}$, $\{U_2\}$ e $\{U_3\}$, tem-se que estes podem ser decompostos em três conjuntos de vetores $\{\chi_1^k\}$, $\{\chi_2^k\}$ e $\{\chi_3^k\}$ $(k = 0,1,2)$, como ilustrado na Figura 3.1. Observa-se que este fato, é análogo à aplicação da série discreta de Fourier aos vetores $\{U_1\}$, $\{U_2\}$ e $\{U_3\}$, como pode ser visto pela equação (3.2).

Desta forma, estabelecem-se as seguintes relações vetoriais [10]:

$$\{U_1\} = \{\chi_1^0\} + \{\chi_1^1\} + \{\chi_1^2\}$$

$$\{U_2\} = \{\chi_2^0\} + \{\chi_2^1\} + \{\chi_2^2\} \tag{3.3}$$

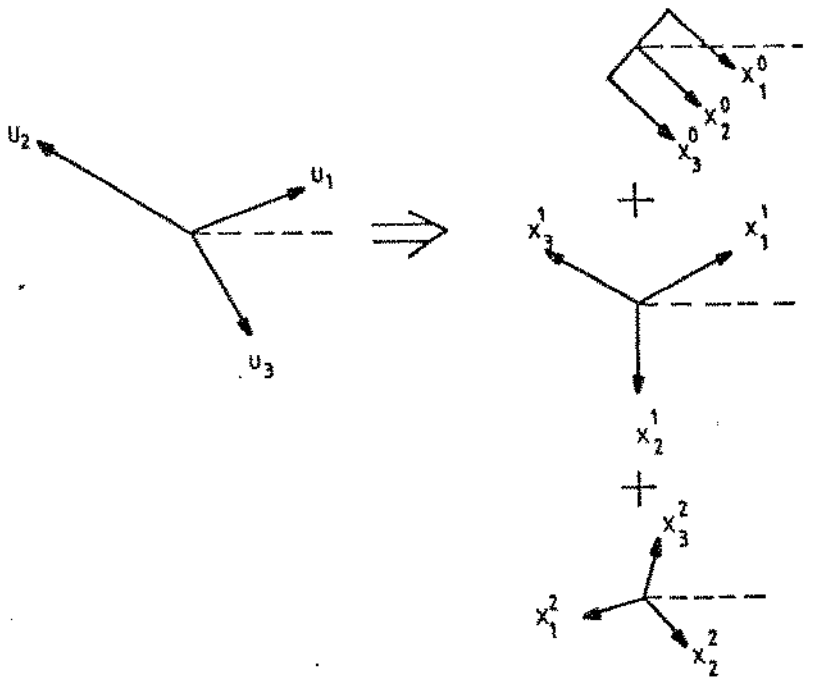


Figura 3.1: Conjunto de vetores e sua representação em componentes cíclicos [10] .

$$\{U_3\} = \{\chi_3^0\} + \{\chi_3^1\} + \{\chi_3^2\}$$

onde os vetores $\{U_n\}$ e $\{\chi_n^k\}$ $(n = 1, 2, 3)$ representam as grandezas físicas e os componentes cíclicos, respectivamente. O índice $k$ indica a ordem do componente cíclico e o índice $n$ o segmento considerado. Denomina-se de região fundamental o segmento correspondente a $n = 1$.

### 3.2.1 Transformação de Grandezas Físicas para Componentes Cíclicos

Como apresentado na seção anterior, a partir de um conjunto de vetores $\{U_n\}$ $(n = 1, \ldots, N)$, representando uma grandeza física qualquer, pode-se originar outro conjunto, aplicando-se ao primeiro a série de Fourier discreta. Assim, para transformar os vetores $\{U_n\}$ $(n = 1, \ldots, N)$ em componentes cíclicos $\{\chi_1^k\}$ $(k = 0, \ldots, N/2)$ da região fundamental $(n = 1)$, utiliza-se a expressão,

$$\{\chi_1^k\} = \frac{1}{N} \sum_{n=1}^{N} \{U_n\} \exp(-\imath(n-1)ka) \tag{3.4}$$

onde $a = 2\pi/N$ é o ângulo entre os $N$ segmentos presentes no modelo estrutural.

Introduzem-se então, as seguintes variáveis,

$$\{U^{kc}\} = (\{\chi_1^k\} + \{\chi_1^{N-k}\}) \tag{3.5}$$

$$\{U^{ks}\} = -\imath(\{\chi_1^k\} - \{\chi_1^{N-k}\}) \tag{3.6}$$

A partir das equações (3.4) a (3.6), pode-se escrever as seguintes relações para os componentes cíclicos [10],

$$\{\chi_1^0\} = \frac{1}{N}\sum_{n=1}^{N}\{U_n\}$$

$$\{U^{kc}\} = \frac{2}{N}\sum_{n=1}^{N}\{U_n\}\cos(n-1)ka \tag{3.7}$$

$$\{U^{ks}\} = \frac{2}{N}\sum_{n=1}^{N}\{U_n\}\mathrm{sen}(n-1)ka$$

$$\{\chi_1^{N/2}\} = \frac{1}{N}\sum_{n=1}^{N}(-1)^{n-1}\{U_n\}$$

onde a última expressão só deve ser utilizada quando $N$ for par.

Portanto, empregam-se estas relações para transformar grandezas físicas, tais como deslocamentos e carregamentos, para componentes cíclicos. As equações em (3.7) podem ser escritas matricialmente, ou seja,

$$\begin{Bmatrix} \{\chi_1^0\} \\ \{U^{1c}\} \\ \{U^{1s}\} \\ \vdots \\ \{U^{k_L s}\} \\ \{\chi_1^{N/2}\} \end{Bmatrix} = \begin{bmatrix} \frac{1}{N} & \frac{1}{N} & \frac{1}{N} & \cdots & \frac{1}{N} \\ \frac{2}{N} & \frac{2}{N}\cos a & \frac{2}{N}\cos 2a & \cdots & \frac{2}{N}\cos(N-1)a \\ 0 & \frac{2}{N}\mathrm{sen}a & \frac{2}{N}\mathrm{sen}2a & \cdots & \frac{2}{N}\mathrm{sen}(N-1)a \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & \frac{2}{N}\mathrm{sen}k_L a & \frac{2}{N}\mathrm{sen}2k_L a & \cdots & \frac{2}{N}\mathrm{sen}(N-1)k_L a \\ \frac{1}{N} & -\frac{1}{N} & \frac{1}{N} & \cdots & \frac{1}{N}(-1)^{N-1} \end{bmatrix} \begin{Bmatrix} \{U_1\} \\ \{U_2\} \\ \{U_3\} \\ \vdots \\ \{U_N\} \end{Bmatrix} \tag{3.8}$$

ou ainda,

$$\{\chi_1^k\} = [T_{\chi U}]\{U_n\} \tag{3.9}$$

Na equação (3.8) o índice $k_L$ depende de $N$ ser par ou ímpar. Portanto,

- $k_L = \frac{N-1}{2}$ para $N$ ímpar.
- $k_L = \frac{N-2}{2}$ para $N$ par.

### 3.2.2 Transformação de Componentes Cíclicos para Grandezas Físicas

Para se efetuar a transformação de componentes cíclicos para grandezas físicas, aplica-se a série de Fourier discreta inversa dada em (3.2). Assim, tem-se que,

$$\{U_n\} = \sum_{k=0}^{N-1} \{\chi_1^k\} \exp(-i(n-1)ka) \tag{3.10}$$

Pode-se escrever esta equação em função das variáveis definidas em (3.5) e (3.6), obtendo-se a seguinte expressão geral [10],

$$\{U_n\} = \{\chi_1^0\} + \sum_{k=1}^{k_L} [\{U^{kc}\}\cos(n-1)ka + \{U^{ks}\}\text{sen}(n-1)ka] + (-1)^{n-1}\{\chi_1^{N/2}\} \ (n = 1, 2, \ldots, N) \tag{3.11}$$

onde o termo $\{\chi_1^{N/2}\}$ deve ser incluído apenas quando $N$ for par.

O vetor $\{U_n\}$ $(n = 1, \ldots, N)$ representa uma grandeza física no n-ésimo segmento em termos dos componentes cíclicos da região fundamental. A equação (3.11) possui a seguinte forma matricial,

$$\begin{Bmatrix} \{U_1\} \\ \{U_2\} \\ \{U_3\} \\ \vdots \\ \{U_N\} \end{Bmatrix} = \begin{bmatrix} 1 & 1 & 0 & \ldots & 0 & 1 \\ 1 & \cos a & \text{sen}a & \ldots & \text{sen}k_L a & -1 \\ 0 & \cos 2a & \text{sen}2a & \ldots & \text{sen}2k_L a & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & \cos(N-1)a & \text{sen}(N-1)a & \ldots & \text{sen}(N-1)k_L a & (-1)^{N-1} \end{bmatrix} \begin{Bmatrix} \{\chi_1^0\} \\ \{U^{1c}\} \\ \{U^{1s}\} \\ \vdots \\ \{U^{k_L s}\} \\ \{\chi_1^{N/2}\} \end{Bmatrix} \tag{3.12}$$

ou seja,

$$\{U_n\} = [T_{U\chi}]\{\chi^k\} \tag{3.13}$$

As matrizes de transformação $[T_{\chi U}]$ e $[T_{U\chi}]$ são ortogonais [10]. Portanto,

$$[T_{\chi U}][T_{U\chi}] = [I] \tag{3.14}$$

## 3.3 Condensação Estática dos Graus de Liberdade Internos

A equação de equilíbrio estático para a região fundamental, em grandezas físicas, é dada por,

$$[K_1]\{U_1\} = \{P_1\} \tag{3.15}$$

onde,

- $[K_1]$ = matriz de rigidez da região fundamental,
- $\{U_1\}$ = vetor de deslocamentos nodais,
- $\{P_1\}$ = vetor de carregamento aplicado à região. fundamental

A Figura 3.2 ilustra a região fundamental de uma estrutura ciclo-simétrica. Os nós desta região podem ser convenientemente separados em nós do lado direito ($d$), do lado esquerdo ($e$) e internos ($i$). Portanto, neste caso os nós $1,\ldots,7$ pertencem ao lado direito da região fundamental, os nós $8,\ldots,15$ ao lado esquerdo e os demais são internos.

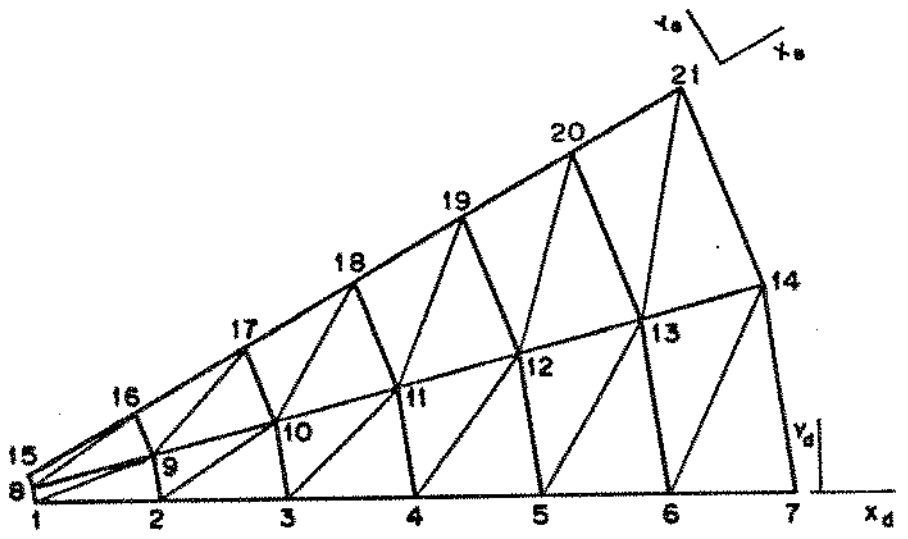


Figura 3.2: Região fundamental e seus nós.

De maneira análoga, particiona-se a equação (3.15), obtendo-se,

$$\begin{bmatrix} [K_{dd}] & [K_{de}] & [K_{di}] \\ [K_{de}]^T & [K_{ee}] & [K_{ei}] \\ [K_{di}]^T & [K_{ei}]^T & [K_{ii}] \end{bmatrix} \begin{Bmatrix} \{U_d\} \\ \{U_e\} \\ \{U_i\} \end{Bmatrix} = \begin{Bmatrix} \{P_d\} \\ \{P_e\} \\ \{P_i\} \end{Bmatrix} \tag{3.16}$$

ou na forma de um sistema de equações,

$$\begin{cases} [K_{dd}]\{U_d\} + [K_{de}]\{U_e\} + [K_{di}]\{U_i\} = \{P_d\} \\ [K_{de}]^T\{U_d\} + [K_{ee}]\{U_e\} + [K_{ei}]\{U_i\} = \{P_e\} \\ [K_{di}]^T\{U_d\} + [K_{ei}]^T\{U_e\} + [K_{ii}]\{U_i\} = \{P_i\} \end{cases} \tag{3.17}$$

onde,

- $[K_{dd}]$ = matriz de rigidez, graus de liberdade do lado direito,
- $[K_{ee}]$ = matriz de rigidez, graus de liberdade do lado esquerdo,
- $[K_{ii}]$ = matriz de rigidez, graus de liberdade internos,
- $[K_{de}]$ = matriz de rigidez, acoplamento dos graus de liberdade dos lados direito e esquerdo,
- $[K_{di}]$ = matriz de rigidez, acoplamento dos graus de liberdade do lado direito e internos,
- $[K_{ei}]$ = matriz de rigidez, acoplamento dos graus de liberdade do lado esquerdo e internos,
- $\{U_d\}$ = deslocamentos dos nós do lado direito,
- $\{U_e\}$ = deslocamentos dos nós do lado esquerdo,
- $\{U_i\}$ = deslocamentos dos nós internos,
- $\{P_d\}$ = carregamento aplicado aos nós do lado direito,
- $\{P_e\}$ = carregamento aplicado aos nós do lado esquerdo,
- $\{P_i\}$ = carregamento aplicado aos nós internos.

O modelo discreto da região fundamental pode ser reduzido, condensando-se os graus de liberdade internos. Para isso, isola-se o vetor $\{U_i\}$ a partir da terceira relação do sistema de equações (3.17), ou seja,

$$\{U_i\} = [K_{ii}]^{-1} \left( \{P_i\} - [K_{di}]^T \{U_d\} - [K_{ei}]^T \{U_e\} \right) \tag{3.18}$$

Substituindo esta relação nas duas equações restantes em (3.17), chega-se ao seguinte sistema de equações reduzido,

$$\begin{cases} [\bar{K}_{dd}]\{U_d\} + [\bar{K}_{de}]\{U_e\} = \{\bar{P}_d\} \\ \left[\bar{K}_{de}\right]^T \{U_d\} + [\bar{K}_{ee}]\{U_e\} = \{\bar{P}_e\} \end{cases} \tag{3.19}$$

ou em forma matricial,

$$\begin{bmatrix} [\bar{K}_{dd}] & [\bar{K}_{de}] \\ \left[\bar{K}_{de}\right]^T & [\bar{K}_{ee}] \end{bmatrix} \begin{Bmatrix} \{U_d\} \\ \{U_e\} \end{Bmatrix} = \begin{Bmatrix} \{\bar{P}_d\} \\ \{\bar{P}_e\} \end{Bmatrix} \tag{3.20}$$

onde,

- $[\bar{K}_{dd}] = [K_{dd}] - [K_{di}] [K_{ii}]^{-1} [K_{di}]^T$

- $[\bar{K}_{ee}] = [K_{ee}] - [K_{ei}][K_{ii}]^{-1}[K_{ei}]^T$
- $[\bar{K}_{de}] = [K_{de}] - [K_{di}][K_{ii}]^{-1}[K_{ei}]^T$
- $\{\bar{P}_d\} = \{P_d\} - [K_{di}][K_{ii}]^{-1}\{P_i\}$
- $\{\bar{P}_e\} = \{P_e\} - [K_{ei}][K_{ii}]^{-1}\{P_i\}$

Assim, obtém-se um sistema de equações em função apenas dos graus de liberdade dos lados direito e esquerdo, o qual pode ser escrito de maneira análoga a (3.15), ou seja,

$$[\bar{K}_1]\{\bar{U}_1\} = \{\bar{P}_1\} \tag{3.21}$$

Resolve-se então, o sistema de equações (3.19), determinando-se os deslocamentos $\{U_d\}$ e $\{U_e\}$ dos lados direito e esquerdo, respectivamente. A partir daí, calculam-se os deslocamentos internos $\{U_i\}$ através da equação (3.18).

Observando-se a Figura 3.2, verifica-se que os sistemas de referência local dos lados direito $(x_d, y_d)$ e esquerdo $(x_e, y_e)$ estão defasados entre si. Torna-se então, necessário compatibilizar estes sistemas de referência para que se possa aplicar as condições de contorno entre os segmentos. Para isso, basta girar o sistema de referência do lado esquerdo no sentido anti-horário, utilizando-se a matriz de rotação (A.4) (ver Apêndice A). Ressalta-se que esta transformação pode ser realizada durante a montagem da matriz de rigidez global $[K_1]$ da região fundamental, bastando aplicar (A.4) para os termos das matrizes dos elementos relacionados aos graus de liberdade do lado esquerdo. Assim, chega-se a seguinte equação a partir de (3.20):

$$\begin{bmatrix} [\bar{K}_{dd}] & [\bar{K}_{de}][T] \\ \left([\bar{K}_{de}][T]\right)^T & [T]^T[\bar{K}_{ee}][T] \end{bmatrix} \begin{Bmatrix} \{U_d\} \\ \{U_e\} \end{Bmatrix} = \begin{Bmatrix} \{\bar{P}_d\} \\ [T]^T\{\bar{P}_e\} \end{Bmatrix} \tag{3.22}$$

Visando simplificar a notação empregada, a equação (3.20) será utilizada no restante do texto, assumindo-se, no entanto, que as transformações indicadas em (3.22) foram efetuadas.

## 3.4 Equação de Equilíbrio em Componentes Cíclicos

A equação de equilíbrio estático de uma estrutura constituída de $N$ réplicas da região fundamental, expressa em grandezas físicas, é análoga a (3.21). Portanto,

$$[\bar{K}_n]\{\bar{U}_n\} = \{\bar{P}_n\} \quad (n = 1, \ldots, N) \tag{3.23}$$

O vetor de deslocamentos $\{\bar{U}_n\}$ pode ser escrito em componentes cíclicos, empregando-se a transformação indicada em (3.13). Assim, utilizando-se esta relação, e pré-multiplicando a equação (3.23) por $[T_{U\chi}]^T$, tem-se a representação da equação de equilíbrio em função de componentes cíclicos. Logo,

$$[K^k]\{\chi^k\} = \{F^k\} \tag{3.24}$$

onde,

- $[K^k] = [T_{U\chi}]^T[\bar{K}_n][T_{U\chi}]$
- $\{F^k\} = [T_{U\chi}]^T\{\bar{P}_n\}$

Como ainda não foram aplicadas as condições de compatibilidade entre os $N$ segmentos, a matriz de rigidez $[\bar{K}_n]$ é constitída de $N$ submatrizes idênticas ao longo da sua diagonal principal [10]. Estas submatrizes representam as equações de equilíbrio de cada um dos $N$ segmentos. Determina-se a seguinte relação geral a partir de (3.24) [10],

$$\begin{cases} [K^k] = N[\bar{K}_1] & \text{para } k = 0, N/2 \\ [K^k] = \frac{N}{2}[\bar{K}_1] & \text{para } 1 \leq k \leq k_L \end{cases} \tag{3.25}$$

Logo, observa-se que as equações de equilíbrio estático para os segmentos, em componentes cíclicos, são $N$ ou $N/2$ vezes as equações correspondentes em grandezas físicas. Portanto, a equação de equilíbrio global da estrutura desacoplada em componentes cíclicos, considerando os $N$ segmentos, é dada por [10],

$$[\bar{K}_1]\begin{bmatrix} N[I] & & & & & \\ & \frac{N}{2}[I] & & & & \\ & & \frac{N}{2}[I] & & & \\ & & & \ddots & & \\ & & & & \frac{N}{2}[I] & \\ & & & & & N[I] \end{bmatrix}\begin{Bmatrix} \{\chi_1^0\} \\ \{U^{1c}\} \\ \{U^{1s}\} \\ \vdots \\ \{U^{k_L s}\} \\ \{\chi_1^{N/2}\} \end{Bmatrix} = \begin{Bmatrix} \{F_1^0\} \\ \{F^{1c}\} \\ \{F^{1s}\} \\ \vdots \\ \{F^{k_L s}\} \\ \{F_1^{N/2}\} \end{Bmatrix} \tag{3.26}$$

onde $[I]$ é a matriz identidade de mesma ordem que $[\bar{K}_1]$.

## 3.5 Aplicação das Condições de Contorno

A expressão (3.26) representa a equação de equilíbrio de uma estrutura ciclo-simétrica em componentes cíclicos. No entanto, torna-se necessário aplicar ainda, as condições de contorno entre os segmentos, as quais ocorrem ao longo dos nós dos lados direito e

esquerdo, como mostrado na Figura 3.3. Assim, as seguintes condições de compatibilidade, expressas em grandezas físicas, devem ser satisfeitas para os graus de liberdade dos lados direito e esquerdo dos segmentos:

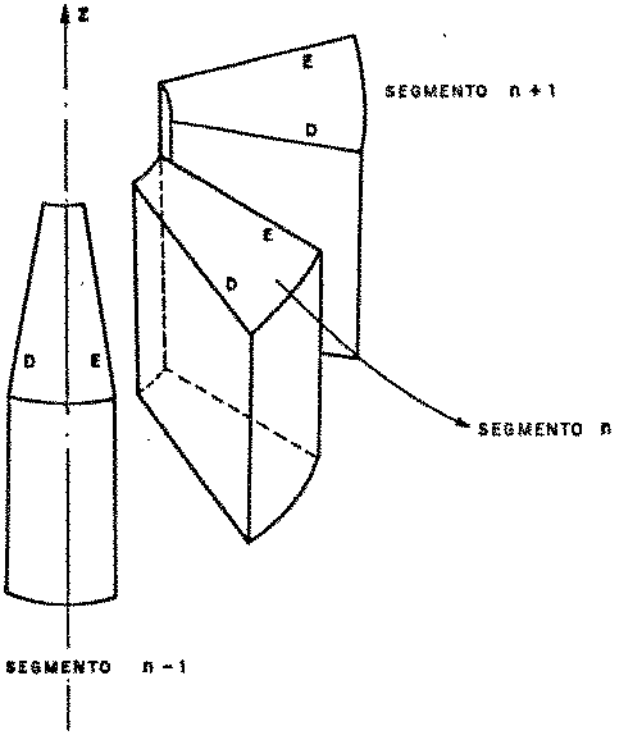


Figura 3.3: Segmentos e suas condições de contorno.

$$\{U_{n-1}\}_e = \{U_n\}_d \tag{3.27}$$

$$\{U_n\}_e = \{U_{n+1}\}_d \tag{3.28}$$

Como o sistema de referência utilizado é cartesiano, a compatibilização dos sistemas de referência dos lados direito e esquerdo dos segmentos, efetuada através da transformação indicada na equação (3.22), é necessária. Ressalta-se que as relações (3.27) e (3.28) poderiam ser aplicadas diretamente, caso se utilizasse um sistema de referência cilíndrico.

As equações (3.27) e (3.28) podem ser representadas em componentes cíclicos, aplicando-se a transformação em (3.4). Obtém-se então, a seguinte expressão geral [10],

$$\{\chi^k\}_e = \{\chi^k\}_d \exp(-i(n-1)ka) \tag{3.29}$$

Considerando $k = 0$, deduz-se a partir de (3.29) que,

$$\{\chi^0\}_e = \{\chi^0\}_d \tag{3.30}$$

e para $k = N/2$ tem-se que,

$$\{\chi^{N/2}\}_e = -\{\chi^{N/2}\}_d \tag{3.31}$$

A relação (3.29) pode ser expressa em termos das variáveis $\{U^{kc}\}$ e $\{U^{ks}\}$ definidas em (3.5) e (3.6), respectivamente. Portanto, vem que para $k = 1, \ldots, k_L$ [10],

$$\begin{Bmatrix} \{U^{kc}\}_e \\ \{U^{ks}\}_e \end{Bmatrix} = \begin{bmatrix} \cos ka[I] & \text{sen}ka[I] \\ -\text{sen}ka[I] & \cos ka[I] \end{bmatrix} \begin{Bmatrix} \{U^{kc}\}_d \\ \{U^{ks}\}_d \end{Bmatrix} \tag{3.32}$$

onde $[I]$ é a matriz identidade de ordem igual a $\{U^{kc}\}_d$.

Assim, verifica-se que as variáveis do lado esquerdo dos segmentos podem ser escritas em função das variáveis do lado direito, ou vice-versa.

As equações (3.30), (3.31) e (3.32) permitem aplicar as condições de contorno entre os segmentos à equação de equilíbrio (3.26). Considerando o segmento de ordem $k = 0$, tem-se a partir de (3.26) que,

$$N[\bar{K}_1]\{\chi_1^0\} = \{F_1^0\} \tag{3.33}$$

e utilizando (3.30) vem que,

$$\{\chi_1^0\} = \begin{Bmatrix} \{\chi_1^0\}_d \\ \{\chi_1^0\}_e \end{Bmatrix} = \begin{bmatrix} [I] \\ [I] \end{bmatrix} \{\chi_1^0\}_d \tag{3.34}$$

onde $[I]$ é a matriz identidade de mesma ordem que $\{\chi_1^0\}_d$.

Substituindo (3.34) em (3.33), e pré-multiplicando (3.33) por $[[I]\ [I]]$, chega-se a seguinte equação,

$$N \begin{bmatrix} [I] & [I] \end{bmatrix} \begin{bmatrix} [\bar{K}_{dd}] & [\bar{K}_{de}] \\ [\bar{K}_{de}]^T & [\bar{K}_{ee}] \end{bmatrix} \begin{bmatrix} [I] \\ [I] \end{bmatrix} \{\chi_1^0\}_d = \begin{bmatrix} [I] & [I] \end{bmatrix} \begin{Bmatrix} \{F_1^0\}_d \\ \{F_1^0\}_e \end{Bmatrix}$$

obtendo-se então, a relação final,

$$N([\bar{K}_{dd}] + [\bar{K}_{ee}] + [\bar{K}_{de}] + [\bar{K}_{de}]^T)\{\chi_1^0\}_d = \{F_1^0\}_d + \{F_1^0\}_e \tag{3.35}$$

Para o segmento de ordem $k = N/2$, determina-se a sua equação de equilíbrio a partir de (3.26). Logo,

$$N[\bar{K}_1]\{\chi_1^{N/2}\} = \{F_1^{N/2}\} \tag{3.36}$$

e usando (3.31),

$$\{\chi_1^{N/2}\} = \begin{Bmatrix} \{\chi_1^{N/2}\}_d \\ \{\chi_1^{N/2}\}_e \end{Bmatrix} = \begin{bmatrix} [I] \\ -[I] \end{bmatrix} \{\chi_1^{N/2}\}_d \tag{3.37}$$

Pré-multiplicando (3.36) por $[[I] \ -[I]]$ e aplicando (3.37) tem-se que,

$$N \begin{bmatrix} [I] & -[I] \end{bmatrix} \begin{bmatrix} [\bar{K}_{dd}] & [\bar{K}_{de}] \\ \left[\bar{K}_{de}\right]^T & [\bar{K}_{ee}] \end{bmatrix} \begin{bmatrix} [I] \\ -[I] \end{bmatrix} \{\chi_1^{N/2}\}_d = \begin{bmatrix} [I] & -[I] \end{bmatrix} \begin{Bmatrix} \{F_1^{N/2}\}_d \\ \{F_1^{N/2}\}_e \end{Bmatrix}$$

a partir da qual determina-se,

$$N([\bar{K}_{dd}] + [\bar{K}_{ee}] - [\bar{K}_{de}] - \left[\bar{K}_{de}\right]^T)\{\chi_1^0\}_d = \{F_1^0\}_d - \{F_1^0\}_e \tag{3.38}$$

A equação de equilíbrio para os demais segmentos $k = 1, \ldots, k_L$ é dada por.

$$\frac{N}{2} \begin{bmatrix} [\bar{K}_1] & [0] \\ [0] & [\bar{K}_1] \end{bmatrix} \begin{Bmatrix} \{U^{kc}\} \\ \{U^{ks}\} \end{Bmatrix} = \begin{Bmatrix} \{F^{kc}\} \\ \{F^{ks}\} \end{Bmatrix}$$

a qual pode ser rearranjada na seguinte forma,

$$\frac{N}{2} \begin{bmatrix} [\bar{K}_{dd}] & [0] & [\bar{K}_{de}] & [0] \\ [0] & [\bar{K}_{dd}] & [0] & [\bar{K}_{de}] \\ \left[\bar{K}_{de}\right]^T & [0] & [\bar{K}_{ee}] & [0] \\ [0] & \left[\bar{K}_{de}\right]^T & [0] & [\bar{K}_{ee}] \end{bmatrix} \begin{Bmatrix} \{U^{kc}\}_d \\ \{U^{ks}\}_d \\ \{U^{kc}\}_e \\ \{U^{ks}\}_e \end{Bmatrix} = \begin{Bmatrix} \{F^{kc}\}_d \\ \{F^{ks}\}_d \\ \{F^{kc}\}_e \\ \{F^{ks}\}_e \end{Bmatrix} \tag{3.39}$$

A dependência entre as variáveis dos lados esquerdo e direito dos segmentos ocorre segundo (3.32). Assim,

$$\begin{Bmatrix} \{U^{kc}\}_d \\ \{U^{ks}\}_d \\ \{U^{kc}\}_e \\ \{U^{ks}\}_e \end{Bmatrix} = \begin{bmatrix} [I] & [0] \\ [0] & [I] \\ \cos(ka)[I] & \text{sen}(ka)[I] \\ -\text{sen}(ka)[I] & \cos(ka)[I] \end{bmatrix} \begin{Bmatrix} \{U^{kc}\}_d \\ \{U^{ks}\}_d \end{Bmatrix} \tag{3.40}$$

Substituindo (3.40) em (3.39), e pré-multiplicando (3.39) pela transposta da matriz de transformação indicada em (3.40), chega-se a equação de equilíbrio final para os segmentos $k = 1, \ldots, k_L$,

$$\frac{N}{2} \begin{bmatrix} [C] & [S] \\ -[S] & [C] \end{bmatrix} \begin{Bmatrix} \{U^{ks}\}_d \\ \{U^{kc}\}_d \end{Bmatrix} = \begin{Bmatrix} \{F_C\} \\ \{F_S\} \end{Bmatrix} \tag{3.41}$$

onde,

- $[C] = [\bar{K}_{dd}] + [\bar{K}_{ee}] + ([\bar{K}_{de}] + \left[\bar{K}_{de}\right]^T)\cos ka$
- $[S] = (\left[\bar{K}_{de}\right]^T - [\bar{K}_{de}])\text{sen}ka$
- $\{F_C\} = \{F^{kc}\}_d + \cos ka\{F^{kc}\}_e - \text{sen}ka\{F^{ks}\}_e$
- $\{F_S\} = \{F^{ks}\}_d + \text{sen}ka\{F^{kc}\}_e + \cos ka\{F^{ks}\}_e$

Observa-se que o sistema de equação (3.41) pode ser reduzido, condensando-se as variáveis $\{U^{kc}\}_d$. Logo,

$$\{U^{kc}\}_d = [C]^{-1}(\frac{2}{N}\{F_S\} + [S]\{U^{ks}\}_d) \tag{3.42}$$

Substituindo (3.42) na primeira equação de (3.41) resulta,

$$[\bar{C}]\{\bar{U}^{ks}\}_d = \{\bar{F}_C\} \tag{3.43}$$

onde,

- $[\bar{C}] = \frac{N}{2}([C] + [S][C]^{-1}[S])$
- $\{\bar{F}_C\} = \{F_C\} + [S][C]^{-1}\{F_S\}$

Assim, chega-se a um sistema de equações em função apenas das variáveis $\{U^{ks}\}_d$ para os segmentos $k = 1, \ldots, k_L$.

Portanto, as equações (3.35), (3.38) e (3.43) constituem as equações de equilíbrio estático para uma estrutura ciclo-simétrica, em componentes cíclicos, com as condições de compatibilidade aplicadas entre os segmentos.

## 3.6 Aplicação do Algoritmo a uma Estrutura Reticulada

Analisou-se a estrutura treliçada da Figura 3.4 com o objetivo de verificar os conceitos apresentados. A região fundamental desta estrutura é mostrada na Figura 3.2. O algoritmo descrito foi implementado no programa *ANAFIN* [12], utilizando-se assim, a estrutura de dados e os métodos de cálculo já implementados.

Considerou-se o diâmetro externo igual a $30\,m$ e o interno $4\,m$. As demais circunferências estão igualmente espaçadas. A altura da estrutura é $15\,m$. Adotou-se os seguintes valores para as propriedades físicas e geométricas do modelo da região fundamental:

- módulo de elasticidade longitudinal: $E = 2{,}1 \times 10^{11}\,N/m^2$,

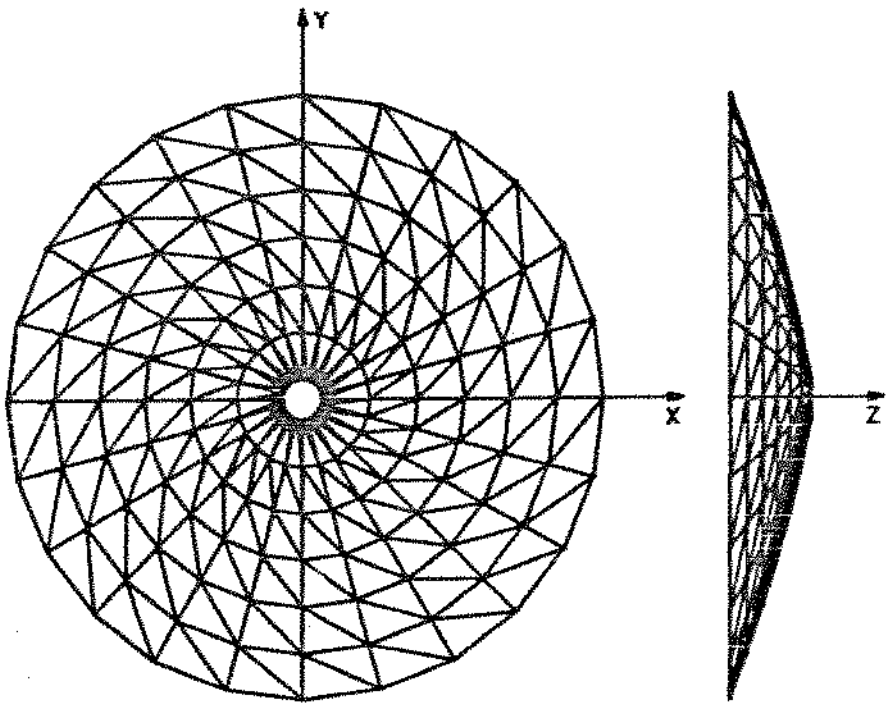


Figura 3.4: Estrutura ciclo-simétrica analisada.

- diâmetro da secção transversal das barras: $d = 20\,mm$.
- área da secção transversal das barras: $A = 3,14 \times 10^{-4}\,m^2$.

Restringiu-se todos os graus de liberdade dos nós da base da estrutura e os graus de liberdade na direção $z$ dos nós presentes no topo da estrutura.

Aplicou-se cargas concentradas para os nós indicados na Figura 3.5. As direções e as magnitudes destas forças podem ser vistas na Tabela 3.1.

| Nó | $F_x$ [KN] | $F_y$ [KN] |
|---|---|---|
| 1 | 0,0 | −50,0 |
| 2 | 35,0 | −35,0 |
| 3 | 50,0 | 0,0 |
| 4 | 35,0 | 35,0 |
| 5 | 0,0 | 50,0 |
| 6 | −35,0 | 35,0 |
| 7 | −50,0 | 0,0 |
| 8 | −35,0 | −35,0 |

Tabela 3.1: Valores do carregamento aplicado à estrutura.

O modelo global foi discretizado com as seguintes características:

- 456 elementos de barra tridimensional.
- 168 nós → 504 graus de liberdade.
- 408 graus de liberdade livres.

Já para o modelo discreto da região fundamental considerou-se:

- 44 elementos de barra tridimensional.
- 21 nós → 63 graus de liberdade.
- 51 graus de liberdade livres
  - 17 graus de liberdade do lado direito.
  - 17 graus de liberdade do lado esquerdo.
  - 17 graus de liberdade internos.

Na análise por componentes cíclicos, utilizou-se 12 segmentos. Desta forma, a regiâo fundamental foi composta de nós dos lados direito e esquerdo, além dos nós internos. Ressalta-se que devido a condensação estática dos graus de liberdade internos, reduziu-se o número de variáveis do modelo para 34, e finalmente para 17 após a aplicação das

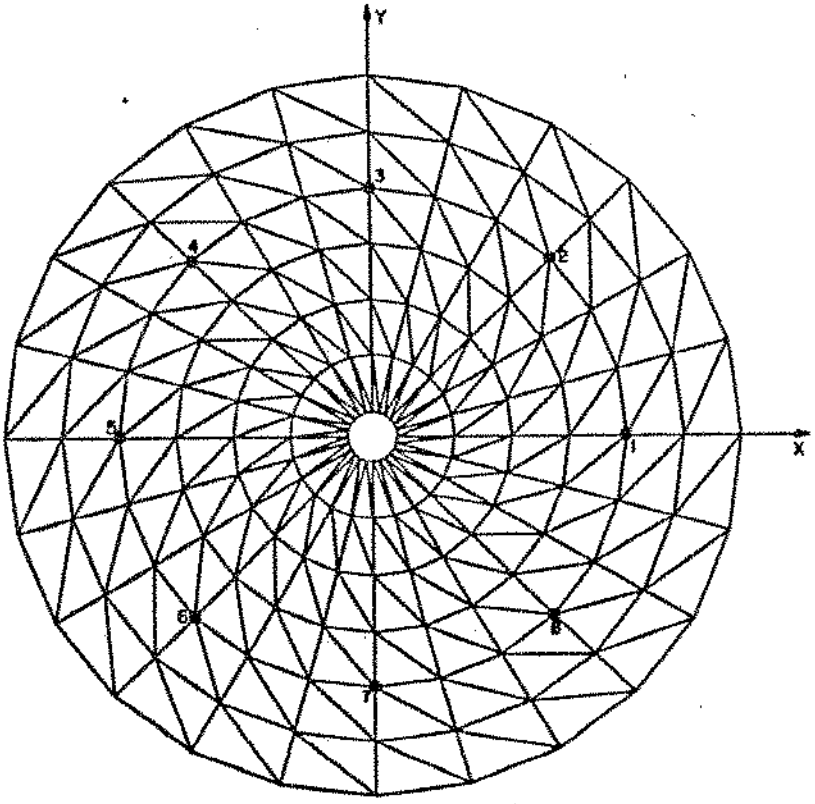


Figura 3.5: Carregamento aplicado à estrutura.

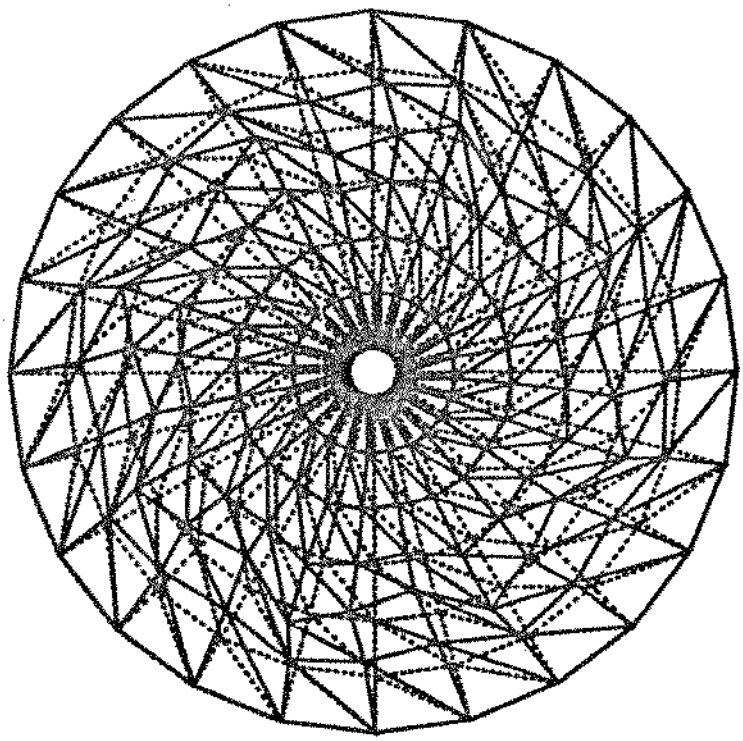

Figura 3.6: Geometria deformada da estrutura analisada.

condições de compatibilidade entre os segmentos. Observa-se ainda, que se deve considerar pela metade os valores da área da secção transversal, ou o módulo de elasticidade, para os elementos de barra pertencentes à interface de dois segmentos. Os resultados obtidos foram os mesmos em ambas análises. No entanto, a tarefa de preparação do modelo discreto da região fundamental foi menos trabalhosa que aquela para o modelo global. A Tabela 3.2 apresenta os valores dos deslocamentos de alguns nós da região fundamental, e a Figura 3.6 a geometria deformada da estrutura.

| Nó | u [mm] | v [mm] | w [mm] |
|---|---|---|---|
| 2 | 0,0755 | −1,1110 | −0,2295 |
| 5 | −1,7000 | −4,2350 | −7,1170 |
| 11 | −0,8422 | −3,0230 | −6,4790 |
| 13 | −3,9130 | −2,8120 | −13,3000 |
| 17 | 1,2270 | −1,5490 | −0,2501 |
| 20 | −0,1034 | −1,7850 | −3,3110 |

Tabela 3.2: Deslocamentos nodais da região fundamental.

Finalmente, ressalta-se que neste tipo de análise as condições de contorno também devem ser ciclo-simétricas. Além disso, o modelo discreto da estrutura a ser analisada deve possuir um furo na direção do eixo de simetria, permitindo assim, o desacopla-

mento das equações de equilíbrio dos vários segmentos e a aplicação das condições de compatibilidade entre os graus de liberdade dos nós dos lados direito e esquerdo.

# Capítulo 4

# ANÁLISE DINÂMICA DE ESTRUTURAS CICLO-SIMÉTRICAS

## 4.1 Introdução

Este capítulo considera a aplicação das transformações cíclicas, discutidas no Capítulo 3, para a equação de vibração livre de uma estrutura ciclo-simétrica. Apresentam-se então, as equações obtidas ao se aplicar as condições de compatibilidade entre os segmentos. Estes conceitos são empregados na análise dinâmica de uma estrututura reticulada. Finalmente, discute-se, de forma geral, alguns métodos de subestruturação dinâmica.

## 4.2 Equação de Movimento em Componentes Cíclicos

A equação de vibração livre para uma estrutura constituída de $N$ réplicas da região fundamental, em função de grandezas físicas, é dada por

$$[M_n]\{\ddot{u}_n\} + [K_n]\{u_n\} = \{0\} \quad (n = 1, \ldots, N) \tag{4.1}$$

onde $[M_n]$ e $[K_n]$ denotam, respectivamente, as matrizes de massa e rigidez dos segmentos, ambas de ordem $m$. Da mesma forma, $\{\ddot{u}_n\}$ e $\{u_n\}$ são os vetores de aceleração e deslocamento. Adotando-se uma solução do tipo exponencial para a equação diferencial (4.1), chega-se ao seguinte problema de autovalor [13,14],

$$([K_n] - \lambda_{n_i}[M_n])\{U_{n_i}\} = \{0\} \quad (i = 1, \ldots, m) \tag{4.2}$$

onde $\lambda_{n_i}$ são os autovalores e $\{U_{n_i}\}$ os autovetores.

Pode-se expressar os autovetores $\{U_{n_i}\}$ $(i = 1, \ldots, m)$ em componentes cíclicos, aplicando-se a transformação indicada em (3.13). Substituindo esta relação em (4.2), e pré-multiplicando por $[T_{U\chi}]^T$, obtém-se o problema de autovalor em componentes cíclicos, ou seja,

$$([K^k] - \lambda_i[M^k])\{\chi_i^k\} = \{0\} \quad (i = 1, \ldots, m) \tag{4.3}$$

onde,

- $[K^k] = [T_{U\chi}]^T[K_n][T_{U\chi}]$
- $[M^k] = [T_{U\chi}]^T[M_n][T_{U\chi}]$

Obtém-se portanto, a seguinte relação geral,

$$\begin{cases} [K^k] = N[K_1], \quad [M^k] = N[M_1] & \text{para } k = 0, N/2 \\ [K^k] = \frac{N}{2}[K_1], \quad [M^k] = \frac{N}{2}[M_1] & \text{para } 1 \leq k \leq k_L \end{cases} \tag{4.4}$$

Como as condições de contorno entre os segmentos não foram ainda consideradas, determina-se para o caso dinâmico uma equação de movimento análoga a (3.26), ou seja,

$$([K_1] - \lambda[M_1]) \begin{bmatrix} N[I] & & & & & \\ & \frac{N}{2}[I] & & & & \\ & & \frac{N}{2}[I] & & & \\ & & & \ddots & & \\ & & & & \frac{N}{2}[I] & \\ & & & & & N[I] \end{bmatrix} \begin{Bmatrix} \{\chi_1^0\} \\ \{U^{1c}\} \\ \{U^{1s}\} \\ \vdots \\ \{U^{k_L s}\} \\ \{\chi_1^{N/2}\} \end{Bmatrix} = \begin{Bmatrix} \{0\} \\ \{0\} \\ \{0\} \\ \vdots \\ \{0\} \\ \{0\} \end{Bmatrix} \tag{4.5}$$

onde $[I]$ é a matriz identidade de mesma ordem que as matrizes de rigidez $[K_1]$ e de massa $[M_1]$ da região fundamental.

## 4.3 Aplicação das Condições de Contorno

O problema de autovalor para uma estrutura em função de componentes cíclicos é descrito pela equação (4.5). Deve-se no entanto, aplicar as condições de compatibilidade entre os nós dos lados direito e esquerdo dos segmentos, como indicado pelas relações

do lado direito, lado esquerdo e internos. Neste caso, observa-se a presença dos graus de liberdade internos, ao contrário da análise estática onde estes foram condensados ao longo do contorno.

Para o caso $k = 0$, ao se aplicar a condição de compatibilidade indicada em (3.30), chega-se a seguinte relação,

$$\{\chi_1^0\} = \begin{Bmatrix} \{\chi_1^0\}_d \\ \{\chi_1^0\}_e \\ \{\chi_1^0\}_i \end{Bmatrix} = \begin{bmatrix} [I] & [0] \\ [I] & [0] \\ [0]^T & [I_i] \end{bmatrix} \begin{Bmatrix} \{\chi_1^0\}_d \\ \{\chi_1^0\}_i \end{Bmatrix} \tag{4.6}$$

onde $[I]$ e $[I_i]$ são, respectivamente, as matrizes identidade de mesma ordem que $\{\chi_1^0\}_d$ e $\{\chi_1^0\}_i$. Denotando-se os números de variáveis do lado direito e interno por $m_d$ e $m_i$, respectivamente, tem-se que $[0]$ é uma matriz nula de ordem $m_d \times m_i$. Através da equação (4.5), determina-se o problema de autovalor para o segmento de ordem $k = 0$, ou seja,

$$N([K_1] - \lambda[M_1])\{\chi_1^0\} = \{0\} \tag{4.7}$$

Substituindo (4.6) em (4.7), e pré-multiplicando pela transposta da matriz de transformação indicada em (4.6), vem que,

$$N \begin{bmatrix} [I] & [I] & [0] \\ [0]^T & [0]^T & [I_i] \end{bmatrix} \left( \left( \begin{bmatrix} [K_{dd}] & [K_{de}] & [K_{di}] \\ [K_{de}]^T & [K_{ee}] & [K_{ei}] \\ [K_{di}]^T & [K_{ei}]^T & [K_{ii}] \end{bmatrix} - \right.$$

$$\left. -\lambda \begin{bmatrix} [M_{dd}] & [M_{de}] & [M_{di}] \\ [M_{de}]^T & [M_{ee}] & [M_{ei}] \\ [M_{di}]^T & [M_{ei}]^T & [M_{ii}] \end{bmatrix} \right) \begin{bmatrix} [I] & [0] \\ [I] & [0] \\ [0]^T & [I_i] \end{bmatrix} \begin{Bmatrix} \{\chi_1^0\}_d \\ \{\chi_1^0\}_e \\ \{\chi_1^0\}_i \end{Bmatrix} = \begin{Bmatrix} \{0\} \\ \{0\} \\ \{0\} \end{Bmatrix} \tag{4.8}$$

obtendo-se, então, a expressão final,

$$N \left( \begin{bmatrix} [K_{dd}] + [K_{ee}] + [K_{de}] + [K_{de}]^T & [K_{di}] + [K_{ei}] \\ ([K_{di}] + [K_{ei}])^T & [K_{ii}] \end{bmatrix} - \right.$$

$$\left. -\lambda \begin{bmatrix} [M_{dd}] + [M_{ee}] + [M_{de}] + [M_{de}]^T & [M_{di}] + [M_{ei}] \\ ([M_{di}] + [M_{ei}])^T & [M_{ii}] \end{bmatrix} \right) \begin{Bmatrix} \{\chi_1^0\}_d \\ \{\chi_1^0\}_i \end{Bmatrix} = \begin{Bmatrix} \{0\} \\ \{0\} \end{Bmatrix} \tag{4.9}$$

Considerando o caso $k = N/2$ ($N$ par), deduz-se de maneira análoga, a partir de (3.31), a relação,

$$\{\chi_1^{N/2}\} = \left\{ \begin{array}{c} \{\chi_1^{N/2}\}_d \\ \{\chi_1^{N/2}\}_e \\ \{\chi_1^{N/2}\}_i \end{array} \right\} = \left[ \begin{array}{cc} [I] & [0] \\ -[I] & [0] \\ [0]^T & [I_i] \end{array} \right] \left\{ \begin{array}{c} \{\chi_1^{N/2}\}_d \\ \{\chi_1^{N/2}\}_i \end{array} \right\} \tag{4.10}$$

Assim, através da equação (4.5), obtém-se o problema de autovalor para o segmento de ordem $k = N/2$,

$$N([K_1] - \lambda[M_1])\{\chi_1^{N/2}\} = \{0\} \tag{4.11}$$

Substituindo (4.10) em (4.11), e pré-multiplicando pela transposta da matriz de transformação indicada em (4.10), tem-se que,

$$N \left[ \begin{array}{ccc} [I] & -[I] & [0] \\ [0]^T & [0]^T & [I_i] \end{array} \right] \left( \left[ \begin{array}{ccc} [K_{dd}] & [K_{de}] & [K_{di}] \\ [K_{de}]^T & [K_{ee}] & [K_{ei}] \\ [K_{di}]^T & [K_{ei}]^T & [K_{ii}] \end{array} \right] -\lambda \left[ \begin{array}{ccc} [M_{dd}] & [M_{de}] & [M_{di}] \\ [M_{de}]^T & [M_{ee}] & [M_{ei}] \\ [M_{di}]^T & [M_{ei}]^T & [M_{ii}] \end{array} \right] \right) \left[ \begin{array}{cc} [I] & [0] \\ -[I] & [0] \\ [0]^T & [I_i] \end{array} \right] \left\{ \begin{array}{c} \{\chi_1^{N/2}\}_d \\ \{\chi_1^{N/2}\}_e \\ \{\chi_1^{N/2}\}_i \end{array} \right\} = \left\{ \begin{array}{c} \{0\} \\ \{0\} \\ \{0\} \end{array} \right\} \tag{4.12}$$

Simplificando-se (4.12) obtém-se,

$$N \left( \left[ \begin{array}{cc} [K_{dd}] + [K_{ee}] - ([K_{de}] + [K_{de}]^T) & [K_{di}] - [K_{ei}] \\ ([K_{di}] - [K_{ei}])^T & [K_{ii}] \end{array} \right] -\lambda \left[ \begin{array}{cc} [M_{dd}] + [M_{ee}] - ([M_{de}] + [M_{de}]^T) & [M_{di}] - [M_{ei}] \\ ([M_{di}] - [M_{ei}])^T & [M_{ii}] \end{array} \right] \right) \left\{ \begin{array}{c} \{\chi_1^{N/2}\}_d \\ \{\chi_1^{N/2}\}_i \end{array} \right\} = \left\{ \begin{array}{c} \{0\} \\ \{0\} \end{array} \right\} \tag{4.13}$$

Finalmente, para os segmentos de ordem $k = 1, \ldots, k_L$ pode-se escrever, utilizando-se a equação (3.32), a dependência entre as variáveis dos lados direito e esquerdo dos segmentos na seguinte maneira,

$$\left\{ \begin{array}{c} \{U^{kc}\}_d \\ \{U^{ks}\}_d \\ \{U^{kc}\}_e \\ \{U^{ks}\}_e \\ \{U^{kc}\}_i \\ \{U^{ks}\}_i \end{array} \right\} = \left[ \begin{array}{cccc} [I] & [0] & [0] & [0] \\ [0] & [I] & [0] & [0] \\ \cos(ka)[I] & \mathrm{sen}(ka)[I] & [0] & [0] \\ -\mathrm{sen}(ka)[I] & \cos(ka)[I] & [0] & [0] \\ [0]^T & [0]^T & [I_i] & [0] \\ [0]^T & [0]^T & [0] & [I_i] \end{array} \right] \left\{ \begin{array}{c} \{U^{kc}\}_d \\ \{U^{ks}\}_d \\ \{U^{kc}\}_i \\ \{U^{ks}\}_i \end{array} \right\} \tag{4.14}$$

Da mesma forma, determina-se o problema de autovalor através de (4.5), ou seja,

$$\frac{N}{2}\begin{bmatrix} (|K_1| - \lambda|M_1|) & |0| \\ |0| & (|K_1| - \lambda|M_1|) \end{bmatrix}\begin{Bmatrix} \{U^{kc}\} \\ \{U^{ks}\} \end{Bmatrix}\begin{Bmatrix} \{0\} \\ \{0\} \end{Bmatrix} \tag{4.15}$$

Substituindo (4.14) em (4.15), e pré-multiplicando pela transposta da matriz de transformação indicada em (4.14), chega-se a seguinte relação final,

$$\frac{N}{2}\left(\begin{bmatrix} |K_1| & |K_2| & |K_3| & |K_4| \\ |K_2|^T & |K_1| & -|K_4| & |K_3| \\ |K_3|^T & -|K_4| & |K_{ii}| & |0| \\ |K_4|^T & |K_3|^T & |0| & |K_{ii}| \end{bmatrix} - \right.$$

$$\left. -\lambda\begin{bmatrix} |M_1| & |M_2| & |M_3| & |M_4| \\ |M_2|^T & |M_1| & -|M_4| & |M_3| \\ |M_3|^T & -|M_4| & |M_{ii}| & |0| \\ |M_4|^T & |M_3|^T & |0| & |M_{ii}| \end{bmatrix}\right)\begin{Bmatrix} \{U^{kc}\}_d \\ \{U^{ks}\}_d \\ \{U^{kc}\}_i \\ \{U^{ks}\}_i \end{Bmatrix} = \begin{Bmatrix} \{0\} \\ \{0\} \\ \{0\} \\ \{0\} \end{Bmatrix} \tag{4.16}$$

onde,

- $|K_1| = |K_{dd}| + |K_{ee}| + \cos(ka)(|K_{de} + |K_{de}|^T)$
- $|K_2| = \text{sen}(ka)(|K_{de}| - |K_{de}|^T)$
- $|K_3| = |K_{di}| + \cos(ka)|K_{ei}|$
- $|K_4| = -\text{sen}(ka)|K_{ei}|$
- $|M_1| = |M_{dd}| + |M_{ee}| + \cos(ka)(|M_{de}| + |M_{de}|^T)$
- $|M_2| = \text{sen}(ka)(|M_{de}| - |M_{de}|^T)$
- $|M_3| = |M_{di}| + \cos(ka)|M_{ei}|$
- $|M_4| = -\text{sen}(ka)|M_{ei}|$

Portanto, as equações (4.9), (4.13) e (4.16) representam os problemas de autovalor em componentes cíclicos com as condições de compatibilidade aplicadas entre os segmentos. Observa-se que o número de autovetores obtidos para os segmentos de ordem $k = 1, \ldots, k_L$ é o dobro daquele proveniente dos casos $k = 0$ e $k = N/2$. Deve-se aplicar a transformação dada (3.9) para expressar os autovetores em função de grandezas físicas.

A maioria dos modos de vibração de estruturas ciclo-simétricas ocorrem em pares ortogonais degenerados. Assim, se um modo possui uma amplitude máxima, em um certo ponto da estrutura, pode-se efetuar uma rotação, de um ângulo $a = 2\pi/N$, para esta forma modal sem alterar a frequência natural de vibração, devido a propriedade de ciclo-simetria da estrutura. Verifica-se no entanto, que os modos obtidos para os casos

$k = 0$ e $k = N/2$ não são degenerados, exceto quando ocorrer dois modos distintos com a mesma frequência natural [16].

Considerando um autovetor $\{u\}$ da estrutura global, pode-se expressá-lo na seguinte maneira,

$$\{u\} = \{\{u_1\}\{u_2\}\ldots\{u_N\}\}^T$$

onde $\{u_n\}$ $(n = 1,\ldots,N)$ é um vetor real de ordem $m$ contendo os deslocamentos associados com o n-ésimo segmento.

Tomando-se $k = 0$ tem-se que,

$$\{u\} = \{\{u_1\}\{u_1\}\ldots(-1)^{N-1}\{u_1\}\}^T$$

ou seja, a forma modal não se altera quando se efetua uma rotação de um número inteiro de segmentos. Para $k = N/2$ ($N$ par) escreve-se o vetor $\{u\}$ como,

$$\{u\} = \{\{u_1\} - \{u_1\}\ldots - \{u_1\}\}^T$$

não se verificando nenhuma alteração ao se efetuar uma rotação da forma modal de um número par de segmentos. No entanto, se o número de segmentos for ímpar, tem-se uma troca de fase.

Finalmente, observa-se que os demais modos devem ser decompostos em componentes ortogonais através da combinação linear de senos e co-senos.

## 4.4 Análise Dinâmica de uma Estrutura Reticulada

Efetuou-se a análise dinâmica da estrutura reticulada mostrada na Figura 4.1. Utilizou-se elementos finitos de viga espacial na especificação do modelo discreto. Considerou-se os seguintes valores para as propriedades físicas e geométricas destes elementos:

- módulo de elasticidade longitudinal: $E = 68070\,N/mm^2$.
- coeficiente de Poisson: $\nu = 0,20$.
- densidade: $\mu = 2,65 \times 10^{-6}\,Kg/mm^3$.
- diâmetro da seção transversal: $d = 2\,mm$.
- área da seção transversal: $A = 3,14\,mm^2$.

Restringiu-se todos os graus de liberdade dos nós situados na base da estrutura. O modelo global foi discretizado com as seguintes características:

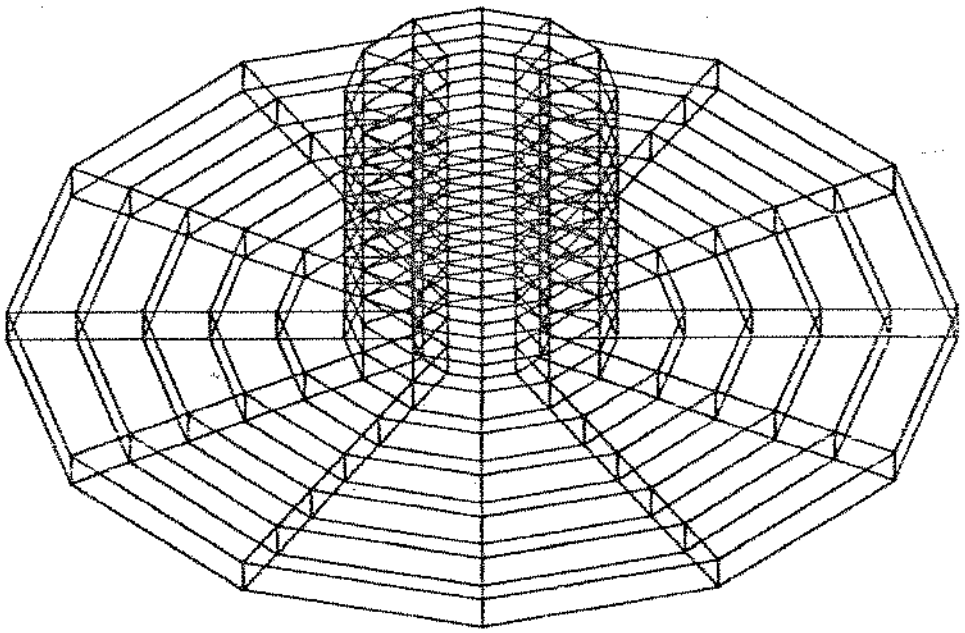

Figura 4.1: Estrutura ciclo-simétrica analisada.

- 876 elementos de viga espacial.
- 360 nós.
- 2088 graus de liberdade.

Na análise por componentes cíclicos, considerou-se 12 segmentos. Para o modelo discreto da região fundamental, ilustrado na Figura 4.2, empregou-se:

- 116 elementos de viga espacial.
- 60 nós.
- 348 graus de liberdade.

A Tabela 4.1 indica os valores obtidos, na análise global [8], para as 8 primeiras frequências naturais da estrutura. Já as Tabelas 4.2, 4.3 e 4.4 contém os deslocamentos de alguns nós da região fundamental, mostrados na Figura 4.2, obtidos para 3 modos de vibração. Nesta análise, efetuou-se uma condensação de Guyan [17], considerando 60 graus de liberdade principais. Na análise cíclica, resolveu-se 7 problemas de autovalor, correspondentes aos segmentos de ordem $k = 0$, $k = N/2$ e $k = 1, \ldots, k_L$ $(k_L = 5)$.

As Figuras 4.3, 4.4 e 4.5 ilustram as formas modais para os 3 modos de vibração considerados.

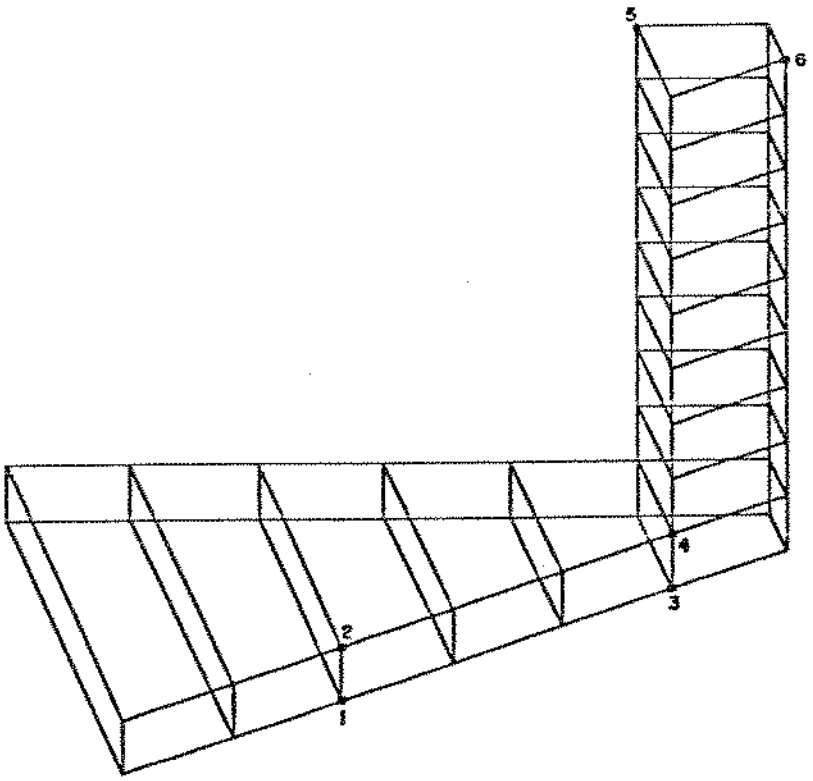


Figura 4.2: Região fundamental da estrutura analisada.

| Número | Frequência [Hz] |
|---|---|
| 1 | 7,10 |
| 2 | 9,23 |
| 3 | 9,25 |
| 4 | 12,86 |
| 5 | 26,47 |
| 6 | 28,61 |
| 7 | 28,69 |
| 8 | 30,78 |

Tabela 4.1: Frequências naturais calculadas para a estrutura.

| Nó | u [mm] | v [mm] | w [mm] |
|---|---|---|---|
| 1 | $-0,0055$ | $0,0006$ | $-1,1780$ |
| 2 | $0,0662$ | $0,0153$ | $-1,1780$ |
| 3 | $-0,0949$ | $0,0193$ | $-2,0500$ |
| 4 | $0,1865$ | $-0,0410$ | $-2,0530$ |
| 5 | $5,8030$ | $-0,4912$ | $-2,5220$ |
| 6 | $5,8500$ | $-0,5281$ | $-1,0610$ |

Tabela 4.2: Deslocamentos nodais - frequência $\omega = 9,25\,Hz$.

| Nó | u [mm] | v [mm] | w [mm] |
|---|---|---|---|
| 1 | $-1,0990$ | $1,9540$ | $-0,0013$ |
| 2 | $-1,0910$ | $2,0510$ | $-0,0013$ |
| 3 | $-1,3869$ | $2,4923$ | $0,0083$ |
| 4 | $-1,1450$ | $2,6730$ | $0,0085$ |
| 5 | $0,0577$ | $4,4000$ | $-0,0114$ |
| 6 | $-1,0660$ | $1,8690$ | $0,0028$ |

Tabela 4.3: Deslocamentos nodais - frequência $\omega = 12,86\,Hz$.

| Nó | u [mm] | v [mm] | w [mm] |
|---|---|---|---|
| 1 | $0,0018$ | $0,0288$ | $4,5550$ |
| 2 | $0,0248$ | $0,0482$ | $4,5540$ |
| 3 | $0,0534$ | $0,0281$ | $-1,1850$ |
| 4 | $-0,0153$ | $-0,0105$ | $-1,1910$ |
| 5 | $0,0018$ | $0,0032$ | $-1,4260$ |
| 6 | $-0,0188$ | $0,0024$ | $-1,2460$ |

Tabela 4.4: Deslocamentos nodais - frequência $\omega = 30,78\,Hz$.

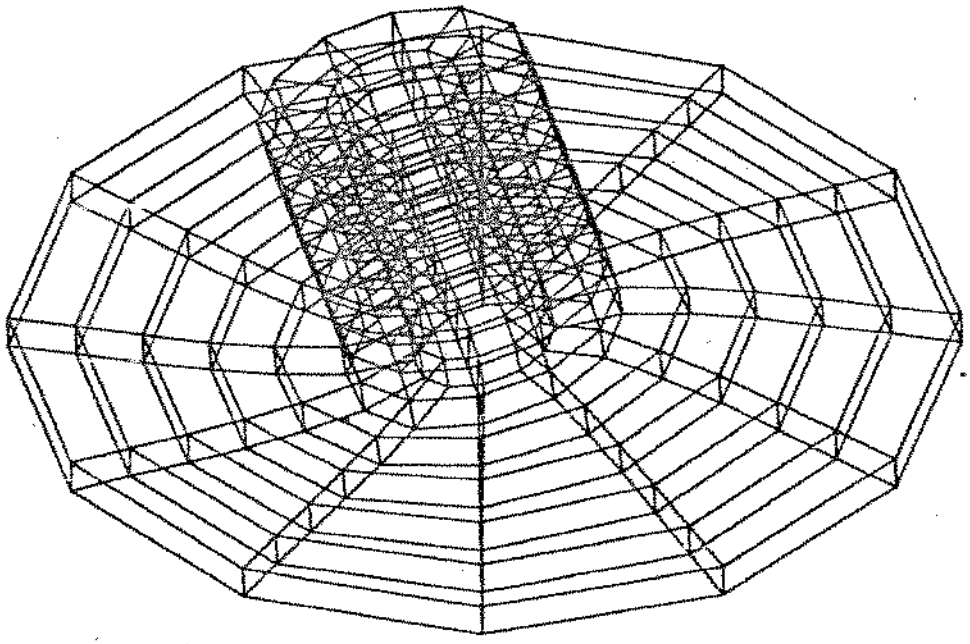

Figura 4.3: Forma modal - frequência $\omega = 9,25\,Hz$.

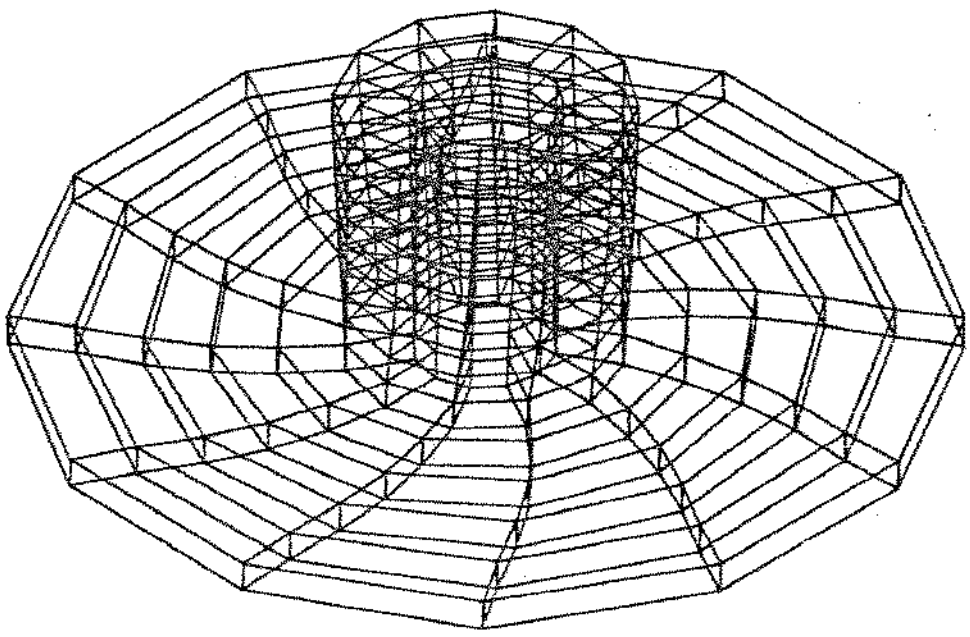

Figura 4.4: Forma modal - frequência $\omega = 12,86\,Hz$.

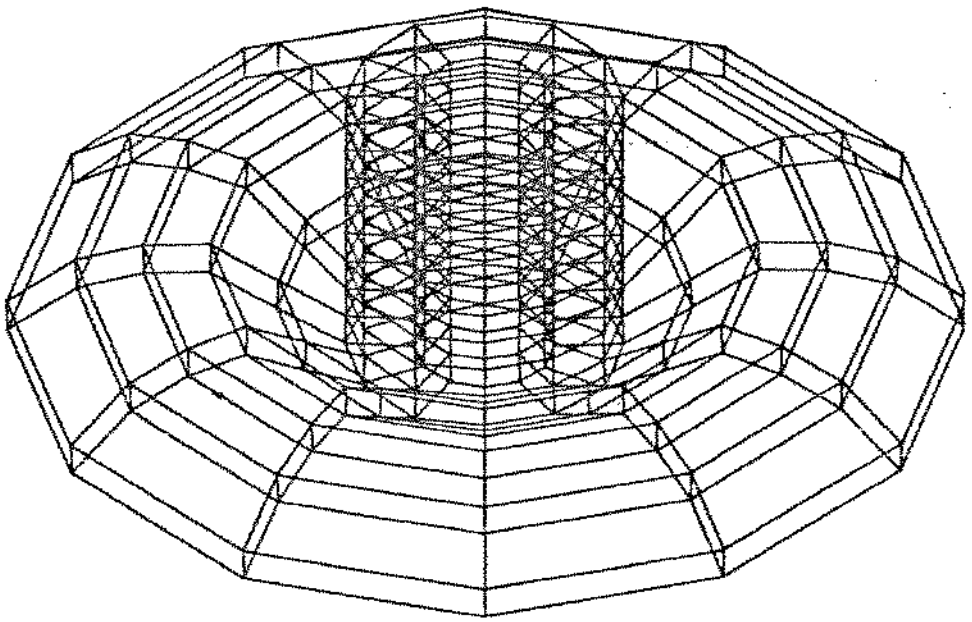

Figura 4.5: Forma modal - frequência $\omega = 30,78\ Hz$.

## 4.5 Métodos de Subestruturação Dinâmica

O algoritmo apresentado nas seções anteriores pode ser utilizado de maneira eficiente para a obtenção das frequências naturais e modos de vibração de estruturas ciclo-simétricas. No entanto, este método é limitado, no sentido que se aplica a uma classe particular de estruturas.

Assim, no caso geral, torna-se necessário utilizar outros algoritmos para efetuar a análise dinâmica de estruturas modeladas como um conjunto de subestruturas. Duas alternativas têm sido usadas para esta finalidade: condensação ou redução de coordenadas; e síntese modal. Na primeira, a estrutura é modelada globalmente, efetuando-se então, uma redução de coordenadas. Neste caso, pode-se realizar, por exemplo, uma condensação estática de variáveis [18] ou aplicar a redução de Guyan [17,18]. Na segunda alternativa, obtém-se a redução do modelo estrutural através da divisão da estrutura em subesturutras, as quais são analisadas separadamente e os resultados manipulados de forma adequada para a obtenção das frequências naturais e modos de vibração da estrutura global.

Neste sentido, um dos primeiros métodos desenvolvidos foi o de Síntese dos Modos Componentes [20]. Neste caso, o movimento de cada subestrutura é representado por dois conjuntos de modos de vibração: os modos de restrição, os quais relacionam as

coordenadas internas e de contorno; os modos normais, obtidos quando os contornos são totalmente fixados. Dentro deste contexto, várias outras técnicas foram elaboradas [19,21,22,23].

Em todos estes métodos, o movimento de cada subestrutura é representado por uma combinação linear dos modos de vibrar da subestrutura, diferindo apenas na definição destes modos. As condições nos contornos internos entre as subestruturas adjacentes podem ser livres, fixas ou consistir de carregamentos inerciais ou rígidos. Para se obter um conjunto de modos de vibrar adequados, deve-se resolver um problema de autovalor para cada subestrutura. Entretanto, frequentemente, isto não é uma tarefa simples, e em muitos casos o problema de autovalor para uma subestrutura não pode ser definido unicamente, devido à impossibilidade de se considerar de modo exato o efeito das subestruturas adjacentes.

Observa-se que em alguns métodos não é necessário a resolução de um problema de autovalor para cada subestrutura. Segundo [24], pode-se utilizar apenas as matrizes de rigidez das subestruturas, efetuando-se uma condensação estática das variáveis internas ao longo do contorno. Determina-se a matriz de massa da estrutura global a partir da contribuição de cada elemento finito. Assim, a partir de uma base inicial de vetores, aplica-se o método numérico de iteração por subespaço [15] para a obtenção de vetores de iteração melhorados, e ao final do processo as frequências naturais e modos de vibração da estrutura. Portanto, calcula-se a matriz de massa global a cada iteração, sendo, no entanto, possível determiná-la inicialmente e armazená-la para posterior acesso. Ressalta-se que este método de subestruturação não envolve, praticamente, nenhum erro de aproximação quando se compara os seus resultados com aqueles obtidos pela técnica global.

No entanto, pode-se considerar a técnica de subestruturação como um método de discretização de Rayleigh-Ritz, tornando possível representar o movimento das subestruturas através de funções admissíveis. No conjunto das funções admissíveis estão os polinômios de baixa ordem, os quais são facilmente manipulados pelo computador. Ressalta-se que os modos naturais são também funções admissíveis adequadas. Para acoplar as subestruturas adjacentes seleciona-se um conjunto finito de pontos, e exige-se que as condições de compatibilidade geométricas sejam satisfeitas apenas nestes pontos, utilizando-se para isso, o método de resíduos ponderados. Define-se então, o conceito de estrutura intermediária como aquela onde as condições de contorno são apenas aproximadas. Este método é denominado Síntese por Subestruturação [25,26], e é apresentado no Apêndice B para sistemas discretos, juntamente com um processo iterativo para a síntese discreta [27].

# Capítulo 5

# ENGENHARIA DE PROGRAMAS

## 5.1 Introdução

Neste capítulo, caracteriza-se de maneira global vários aspectos da crise de programas. A partir daí, apresenta-se os paradigmas de programação, enfatizando-se o paradigma de construção de protótipos. Discute-se, então, as fases de implementação de um programa, em especial os passos de análise de sistema e requerimentos. Finalmente, dois métodos de análise de requisitos são abordados.

## 5.2 A Crise de Programas

Durante os primórdios da indústria de informática, a principal preocupação estava relacionada com a obtenção de computadores com maior capacidade de processamento e armazenamento. Nesta mesma época, os programas eram desenvolvidos de modo artesanal, não existindo métodos sistemáticos de projeto. Assim, os programas eram implementados visando aplicações específicas e com distribuição limitada, possuindo uma característica pessoal ou local a uma organização.

Devido aos grandes avanços da eletrônica, sistemas multi-usuários introduziram novas técnicas de interação homem-máquina, possibilitando o surgimento de novas áreas de aplicações para os computadores. Com este progresso, criou-se uma grande demanda por programas aplicativos, permitindo uma maior produção e distribuição destes programas a vários usuários. No entanto, métodos formais para o projeto de programas não foram aplicados, fazendo com que o custo de manutenção atingisse recursos alarmantes. Iniciou-se assim, uma crise no processo de desenvolvimento de programas.

Nos últimos anos, tem-se verificado a implantação de sistemas distribuídos, onde vários computadores executam tarefas concorrentemente comunicando-se entre si, au-

mentando a complexidade dos sistemas de informação. Além disso, observa-se uma grande disseminação de microprocessadores e computadores pessoais em várias áreas. Isto viabilizou, o surgimento de empresas especializadas no desenvolvimento de programas, com o objetivo de explorar este mercado aberto pelos avanços da informática. Atualmente, os programas consomem a maior parte dos recursos aplicados na implantação de um sistema de informação. Entretanto, a crise de programas vem intensificando-se ainda mais, podendo-se descrevê-la através dos seguintes argumentos [28]:

- a sofisticação dos computadores ultrapassou em muito a capacidade de desenvolvimento de programas que explorem totalmente o potencial disponível.
- a demanda por novos programas é bem superior à capacidade de desenvolvimento.
- técnicas de projeto inadequadas dificultam a manutenção de programas existentes.

Os processos de desenvolvimento de programas possuem algumas características que os diferenciam daqueles empregados nos projetos de computadores. Isto se deve ao fato que um programa é um elemento lógico, e não físico, de um sistema. Algumas características relevantes podem ser citadas [28]:

- um programa é desenvolvido e não manufaturado, no sentido clássico. Embora várias semelhanças possam existir nos projetos de *hardware* e *software*, a natureza destas atividades são fundamentalmente distintas. Os custos de um programa estão concentrados essencialmente no projeto, inviabilizando a aplicação de técnicas de gerenciamento tradicionais de manufatura.
- um programa se deteriora mas não se desgasta, ou seja, um programa não é suscetível ao ambiente físico onde atua. Portanto, quando um componente eletrônico se danifica, substitui-se por um outro. Já uma falha em um programa indica um erro de projeto ou implementação. Assim, a manutenção de um programa é bem mais complexa.
- os programas são desenvolvidos visando solucionar necessidades específicas de um cliente, não sendo construídos a partir de componentes já existentes. Enquanto os circuitos são projetados a partir de componentes disponíveis em catálogos, não existem componentes de programas que possam ser reaproveitados.

A crise de programas engloba vários aspectos relacionados ao desenvolvimento e a manutenção de programas, assim como aspectos de produtividade frente a uma demanda crescente. Vários problemas de gerenciamento tem sido encontrados, tais como: estimativa de custos e cronograma imprecisos, baixa produtividade, insatisfação dos usuários com a qualidade e confiabilidade. No entanto, as principais dificuldades enfrentadas são [28]:

- a inexistência de dados sobre experiências de aplicação de métodos de desenvolvimento de programas, inviabiliza a realização de estimativas com o objetivo de avaliar a produtividade e a eficácia destas técnicas, as ferramentas e os padrões utilizados.
- a comunicação entre os clientes e a equipe responsável pelo desenvolvimento é, frequentemente, vaga e deficiente, fazendo com que as necessidades reais dos usuários não sejam atendidas.
- a qualidade dos programas é questionável, devido a pouca importância dada ao desenvolvimento de técnicas para avaliar a confiabilidade e a qualidade dos programas.
- as dificuldades e o alto custo de manutenção dos programas.

Talvez a maior causa desta crise venha ser a oposição das pessoas responsáveis pelo projeto em relação a introdução de novas técnicas de desenvolvimento, criando-se os *mitos de programação* [28]. No entanto, aplicando-se soluções de engenharia estes problemas podem ser superados. Os conceitos de Engenharia de Programas visam disciplinar o desenvolvimento de programas em todas as suas etapas, desde a especificação, passando pelo projeto, até a manutenção. Na próxima seção, discutem-se os elementos principais de Engenharia de Programas, os quais constituem os paradigmas de programação.

## 5.3 Paradigmas da Engenharia de Programas

Não há uma solução única e satisfatória para os problemas provenientes da *crise de software*. Entretanto, através da combinação de métodos definidos em todas as fases de implementação de um programa, incluindo técnicas de gerenciamento e avaliação de qualidade, pode-se alcançar os objetivos provenientes da Engenharia de Programas.

A Engenharia de Programas é composta de passos englobando três elementos principais para o controle do processo de desenvolvimento de programas, bem como para permitir a implementação produtiva de programas confiáveis e de alta qualidade. Estes passos são, geralmente, denominados paradigmas de programação. Os elementos considerados são os seguintes [28]:

**Métodos** definem como o programa deve ser desenvolvido, abrangendo várias etapas, incluindo: planejamento e estimativa de projeto; análise de requerimentos; projeto da estrutura de dados, arquitetura e procedimentos do programa; codificação; testes; e manutenção.

**Ferramentas** provêm suporte automatizado ou semi-automatizado para a aplicação dos métodos em todas as suas etapas. As ferramentas podem ser integradas per-

mitindo o desenvolvimento de ambientes de Engenharia de Programas Assistida por Computador.

**Procedimentos** são responsáveis pelo acoplamento dos métodos e ferramentas, possibilitando o desenvolvimento racional e progressivo de um programa. Os procedimentos definem a sequência na qual os métodos serão aplicados, os documentos requeridos, os critérios de controle visando assegurar a qualidade e as características relevantes na avaliação do desenvolvimento do programa.

Um paradigma é escolhido baseado na natureza do projeto e da aplicação, os métodos e ferramentas a serem empregados e os resultados exigidos. Três paradigmas têm sido usados extensivamente [28]: Ciclo de Vida Clássico; Construção de Protótipos; e Técnicas de Linguagens de Quarta Geração. A seguir, descrevem-se as principais características do paradigma de Construção de Protótipos, empregado na definição de requerimentos dos algoritmos de análise estrutural a serem implementados no programa FAE.

### 5.3.1 Construção de Protótipos

O paradigma de Construção de Protótipos é, preferencialmente, aplicado em situações onde o usuário possui apenas uma visão geral de suas necessidades, sem identificar detalhadamente os requerimentos de entrada/saída e processamento. Em alguns casos, deseja-se verificar a eficiência de um algoritmo, a adaptabilidade a um sistema operacional e a forma de interação homem-máquina.

O processo de construção de protótipos permite a criação de um modelo do programa a ser construído. Idealmente, este paradigma deve ser empregado como um mecanismo de identificação dos requerimentos de um programa.

Para aplicar este paradigma, utiliza-se os passos ilustrados na Figura 5.1. Inicialmente, deve-se definir os objetivos do programa e identificar os requerimentos conhecidos pelo cliente. A partir daí, realiza-se um projeto rápido, obtendo-se um protótipo, o qual pode ser avaliado pelo cliente/usuário visando refinar os requerimentos para o projeto final. Assim, através de um processo iterativo, o cliente define suscintamente suas necessidades, permitindo ao analista uma visão mais clara dos objetivos do programa.

Portanto, emprega-se o paradigma de construção de protótipos para a obtenção de um primeiro sistema, o qual deve ser parcial ou totalmente descartado após a identificação dos requerimentos. A partir daí, realiza-se o projeto do programa final onde a qualidade e a manutenção são características importantes a serem consideradas.

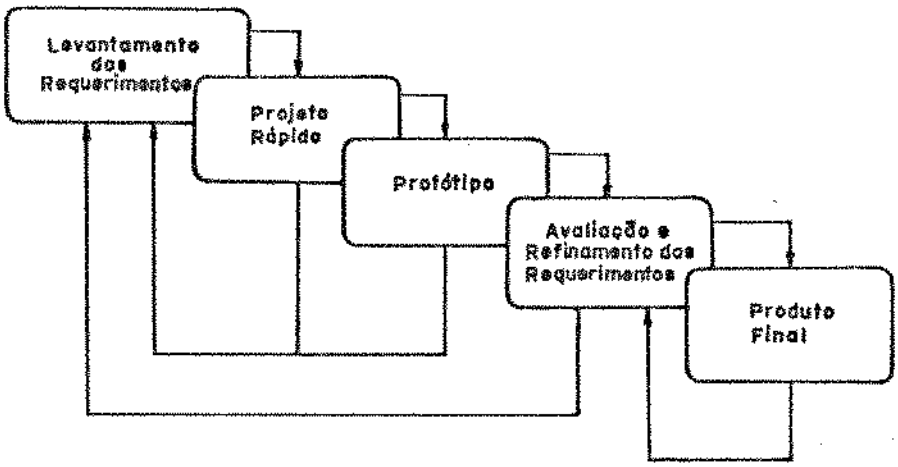


Figura 5.1: Sequência de passos aplicados no paradigma de construção de protótipos [28].

## 5.4 Fases de Implantação de um Programa

Independente do paradigma de programação escolhido, da área de aplicação, tamanho do projeto e complexidade do sistema de informação, pode-se identificar, como mostrado na Figura 5.2, três fases para implantação de um programa, ou seja [28]:

- a fase de definição
- a fase de desenvolvimento
- a fase de manutenção

Durante a fase de definição, deve-se identificar a informação a ser processada, a função e a performance desejadas, as interfaces a serem estabelecidas, as restrições de projeto e os critérios de validação necessários para a implementação do sistema. Três passos específicos ocorrem durante esta fase:

**análise de sistema** define-se, neste caso, o papel de cada elemento em um sistema de informação, alocando as respectivas funções para os programas. Estas características devem ser registradas num documento de Especificação do Sistema.

**planejamento do projeto** durante este passo, determinam-se os recursos requeridos para o desenvolvimento do projeto, assim como estabelecem-se as estimativas de custos e cronograma. Geralmente, confecciona-se um documento de Planejamento do Programa, apresentando estas características e a viabilidade do projeto.

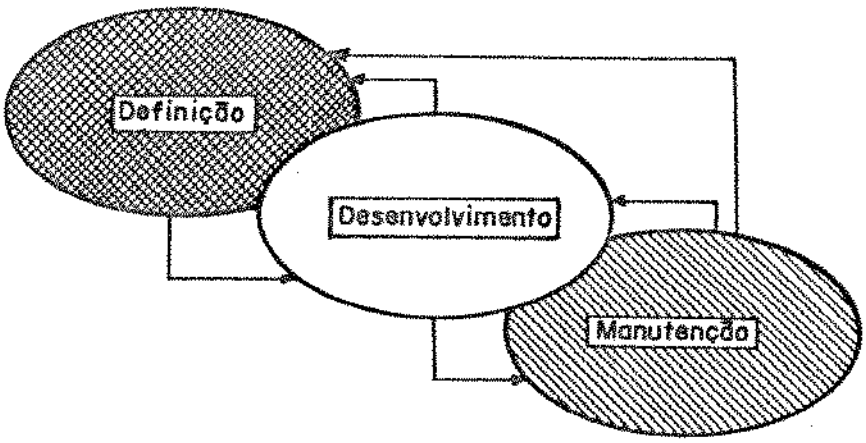


Figura 5.2: Fases de implantação de um programa [28].

**análise de requerimentos** as funções alocadas para os programas na análise de sistemas são detalhadas, definindo-se o domínio de informação a ser representado e os critérios de validação. Um documento de Análise de Requerimentos é normalmente produzido.

Ao final, deve-se avaliar tecnicamente o documento de requisitos e compatibilizar as estimativas presentes no documento de planejamento aos objetivos estabelecidos.

Na fase de desenvolvimento, especifica-se o projeto da estrutura de dados e da arquitetura do programa, bem como detalhes de procedimentos, codificação em uma linguagem computacional e os testes a serem realizados. De maneira análoga, três passos são aplicados:

**projeto do programa** os requerimentos estabelecidos são transformados em um conjunto de representações descrevendo a estrutura de dados, a arquitetura e os algoritmos empregados. Além disso, critérios são utilizados para avaliar a qualidade do projeto.

**codificação** as representações de projeto são traduzidas para uma linguagem computacional, resultando num programa executável. A codificação é vista como uma consequência natural de um bom projeto.

**testes** neste passo, empregam-se vários tipos de testes, procurando identificar falhas de projeto e de implementação. Os testes unitários validam a performance funcional de cada componente do programa, enquanto os testes de integração avaliam a arquitetura do programa, assim como as funções e as interfaces.

Nesta fase, confeccionam-se os documentos de Projeto do Programa e de Planejamento de Testes, os quais devem ser avaliados pelo cliente e pelo grupo de desenvolvimento do programa, procurando verificar se todos os requerimentos foram alcançados.

Finalmente, na fase de manutenção os erros encontrados são corrigidos, adaptações são efetuadas e melhoramentos são implementados. Os passos das fases de definição e desenvolvimento são reaplicados, mas no contexto de um programa já existente.

Além disso, efetuam-se revisões detalhadas ao final de cada passo, procurando assegurar a manutenção da qualidade do programa. Os vários documentos confeccionados são revisados, assegurando que todas as informações sobre o sistema estarão disponíveis para uso posterior. As alterações devem ser controladas e aprovadas pelo cliente e o analista.

Nas seções seguintes, apresentam-se os passos de análise de sistema e requerimentos da fase de definição de um programa, visto que estes conceitos serão aplicados no Capítulo 7. Observa-se que para o programa final a ser implementado, não será considerado o passo de planejamento do projeto, pois, neste caso, não estão envolvidos recursos que justifiquem realizar estimativas de custos. No entanto, o cronograma, os equipamentos exigidos e viabilidade do projeto estão incluídos no documento de Especificação do Sistema [42].

## 5.5 Análise de Sistemas

Um sistema informatizado congrega vários elementos que se interagem para executar as tarefas requeridas. Estes elementos, ilustrados na Figura 5.3, são os seguintes [28]:

**Software** são os programas básicos e aplicativos, a estrutura de dados, os algoritmos e os documentos descrevendo as características lógicas e de controle requeridas.

**Hardware** dispositivos eletrônicos (computadores, discos, memória) que provêm a capacidade computacional, assim como os dispositivos eletromecânicos (sensores, motores) que realizam as funções externas ao sistema.

**Pessoas** são os usuários e operadores do sistema de informação.

**Banco de Dados** coleção organizada de dados acessada pelos programas.

**Documentação** manuais descrevendo informações necessárias ao uso e operação do sistema.

**Procedimentos** são os passos que definem o uso específico de cada elemento do sistema.

Na análise de sistemas, a partir de uma especificação geral de um cliente, procura-se atingir os seguintes objetivos [28]:

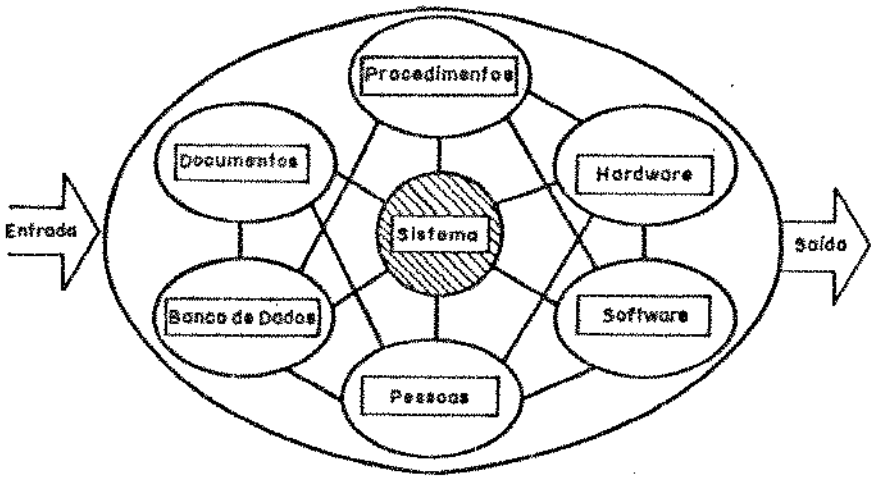


Figura 5.3: Elementos de um sistema de informação [28].

- identificar as necessidades do cliente.
- avaliar a viabilidade de implantação do sistema.
- realizar análises técnicas e econômicas.
- alocar funções para os vários elementos do sistema.
- estabelecer custos e cronogramas.
- criar uma definição clara e objetiva do sistema, constituindo-se numa base para todas as atividades ao longo da implantação do sistema.

Assim, transforma-se um conceito geral, ou em certos casos nebuloso, em uma representação real das funções do sistema, as quais são alocadas para cada um dos elementos da Figura 5.3. Uma intensa comunicação entre o cliente e o analista é fundamental para identificar, de maneira não ambígua, todas necessidades do sistema. Ao final, produz-se um documento de Especificação do Sistema, descrevendo todas as características e objetivos do sistema. Este documento é então, revisado sob os pontos de vista técnico e gerencial pelo analista e cliente, procurando verificar o escopo do projeto, as definições das funções, performance, interfaces e os riscos de desenvolvimento.

## 5.6 Análise de Requerimentos

A partir das funções alocadas aos elementos do sistema de informação durante a sua especificação, realiza-se a análise de requerimentos que consiste num passo intermediário entre a análise de sistema e o projeto do programa. Neste caso, o analista deve executar as seguintes tarefas [28]:

- especificar a função e a perfomance dos programas.
- definir as interfaces dos programas com os demais elementos do sistema.
- estabelecer as restrições de projeto relativas aos programas.
- refinar e representar o domínio da informação abordado pelos programas.

Desta forma, o analista pode representar as informações e as funções a serem traduzidas no projeto da estrutura de dados, arquitetura e procedimentos. Além disso, através da especificação de requerimentos, pode-se avaliar a qualidade do programa após a sua implementação. A análise de requerimentos possui quatro áreas de concentração de esforços [28]:

**reconhecimento do problema** o analista deve considerar e avaliar as funções alocadas aos programas no contexto do sistema. A partir daí, estabelece-se um canal de comunicação com os clientes e usuários, assim como, com a equipe de desenvolvimento. O interesse está no reconhecimento das necessidades básicas de acordo com a visão de cliente e usuários.

**avaliação e síntese** o analista deve avaliar o fluxo e a estrutura da informação, refinar em detalhe as funções alocadas aos programas, estabelecer as interfaces e restrições de projeto. Desta forma, descreve-se o problema procurando, a partir daí, sintetizar uma solução.

**especificação** neste caso, faz-se uma representação dos programas, a ser analisada e aprovada pelo cliente e usuários. Estabelecem-se critérios de validação, os quais são utilizados, posteriormente, como uma base para a realização dos testes. Produz-se um manual preliminar do usuário, permitindo ao analista considerar a visão do usuário.

**revisão** a partir dos documentos de especificação de requisitos e manual do usuário, o analista e o cliente propõem modificações para as funções, performance, representação da informação, restrições e critérios de validação. Além disso, estimativas anteriores devem ser reavaliadas visando a compatibilização com os novos conhecimentos adquiridos no processo de análise.

Vários métodos de análise de requisitos têm sido desenvolvidos [28]. No entanto, todos este métodos possuem alguns princípios fundamentais:

- os domínios de informação e funcional devem ser representados e entendidos.
- os problemas devem ser particionados numa forma hierárquica, permitindo um detalhamento gradativo da informação.
- as representações física e lógica do sistema devem ser desenvolvidas.

De forma geral, os programas são construídos para processar dados. Assim, o domínio da informação considera três visões diferentes dos dados, ou seja [28]:

**Fluxo da informação** representa as transformações efetuadas sobre os dados ao longo do sistema. Estas transformações consistem nas funções a serem executadas pelo programa. Assim, a interface de cada função é definida conhecendo-se os dados presentes entre duas tranformações consideradas.

**Conteúdo da informação** representa os dados individuais pertencentes a uma estrutura maior de informação, podendo-se citar como exemplo um campo de um registro de dados qualquer.

**Estrutura da informação** representa a organização lógica dos dados.

O particionamento dos domínios funcional e de informação é realizado visando tratar um problema em suas várias partes, facilitando o entendimento e gerenciamento das suas funções. Conceitualmente, estabelece-se uma representação hierárquica das funções e informações, particionando-se então, o elemento mais geral desta hierarquia de duas maneiras: detalhamento vertical da informação; ou decomposição horizontal das funções ao longo da hierarquia [28].

A visão lógica dos requerimentos de um programa representa as funções e as informações a serem executadas e processadas sem considerar detalhes de implementação. Já na visão física, as restrições impostas por outros elementos do sistema à estrutura da informação e às funções são relevantes.

Os métodos de análise de requerimentos combinam procedimentos sistemáticos e uma notação única para analisar os domínios funcional e de informação de um programa. Além disso, possuem regras para o particionamento do problema e meios para representar as visões lógica e física do problema. A principal característica destes métodos é auxiliar o analista na obtenção de uma descrição precisa e independente para os programas num sistema de informação [28].

A maioria dos métodos de análise de requisitos possuem mecanismos para a representação do domínio de informação do problema, a partir da qual derivam-se as funções e as demais características do programa. A seguir, apresentam-se alguns passos de dois métodos de análise de requerimentos aplicados no Capítulo 7. Ressalta-se que este métodos são aplicados nas várias fases de implantação de um programa e não apenas durante a análise de requisitos.

### 5.6.1 Método de Análise Orientado por Fluxo de Dados

Os métodos de análise por fluxo de dados baseiam a decomposição do domínio de informação nas transformações aplicadas aos dados desde a entrada até a saída do sistema

de informação. Vários tipos de mecanismos de entrada/saída, executados pelos vários elementos do sistema, podem estar presentes. Para representar o fluxo de informação ao longo do sistema, este métodos utilizam diagramas de fluxo de dados (DFD), como aquele ilustrado na Figura 5.4.

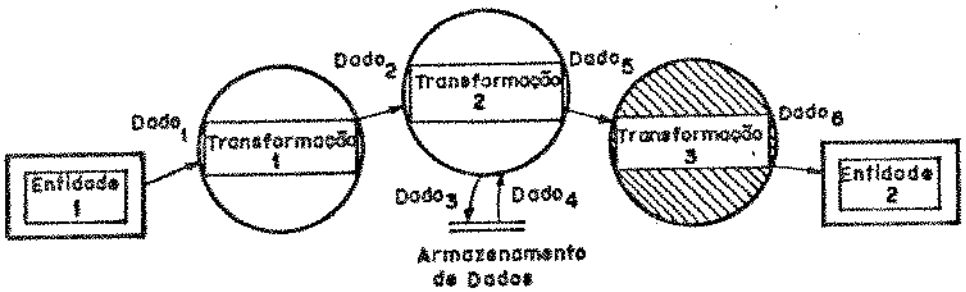


Figura 5.4: Exemplo de um diagrama de fluxo de dados [28].

O DFD é utilizado para descrever um sistema ou programa em qualquer nível de abstração. Assim, pode-se particionar esta representação em vários níveis, possibilitando uma especificação gradativa dos detalhes do fluxo de informação e das funções do sistema. Neste caso, denomina-se de modelo do sistema fundamental o primeiro nível do DFD, pois este representa todo o programa como um único elemento com várias entradas e saídas.

A simbologia empregada num DFD é simples. Um retângulo representa uma entidade externa, a qual pode ser um elemento do sistema ou até mesmo um outro sistema produzindo ou recebendo informação do sistema considerado. Um círculo representa um processo ou uma transformação aplicado aos dados, os quais são denotados por setas. Já as linhas duplas indicam o armazenamento de informação pelo sistema.

Observa-se que nenhuma indicação explícita da sequência de eventos é fornecida pelo diagrama. Um aspecto importante é a exigência de continuidade da informação. Logo, quando um processo for refinado em níveis inferiores do DFD, as mesmas entradas e saídas devem ser mantidas.

Além da representação gráfica, estes métodos descrevem o conteúdo da informação através do dicionário de dados. Esta técnica permite representar dados compostos através de mecanismos de sequência, seleção ou repetição. O dicionário de dados é expandido até que todos os dados compostos tenham sido descritos em função de ítens elementares, de forma definida e não ambígua. A Tabela 5.1 indica a sintaxe utilizada

no dicionário de dados. Exemplos de aplicação desta técnica podem ser encontrados no Capítulo 7. Finalmente, observa-se que as funções podem ser descritas através de uma linguagem natural ou estruturada [28].

| Mecanismo | Notação | Significado |
|---|---|---|
| | = | composto de |
| Sequência | + | e |
| Seleção | \|\| | ou |
| Repetição | $\{\}^n$ | n repetições de |
| | ( ) | dados opcionais |

Tabela 5.1: Sintaxe utilizada no dicionário de dados.

### 5.6.2 Método de Jackson

O método de Jackson representa o domínio de informação através de modelos do mundo real. Vários passos devem ser realizados para a aplicação deste método no processo de desenvolvimento de um sistema, ou seja [28]:

- passo de entidades e ações.
- passo da estrutura de entidades.
- passo do modelo inicial.
- passo funcional.
- passo de determinação de tempos no sistema.
- passo de implementação.

Apenas os dois primeiros passos serão discutidos neste trabalho, pois estes apresentam semelhança com a técnica de análise orientada por objetos aplicada no Capítulo 7.

O passo de entidades e ações inicia-se com uma descrição em linguagem natural do problema considerado. As entidades são então, selecionadas examinando-se todos os substantivos de interesse presentes nesta descrição. Uma ação ocorre em um ponto específico no tempo e é aplicada a uma entidade, sendo estas ações identificadas através dos verbos presentes na descrição. Este processo permite delimitar o escopo do sistema a ser implantado.

No segundo passo, descreve-se a estrutura da entidade, ou seja, o seu comportamento a medida que as ações são aplicadas. Para isso emprega-se a notação diagramática da Figura 5.5, onde as ações são descritas através de mecanismos de sequência, seleção ou

repetição. As restrições existentes podem ser indicadas através de notas que acompanham os diagramas.

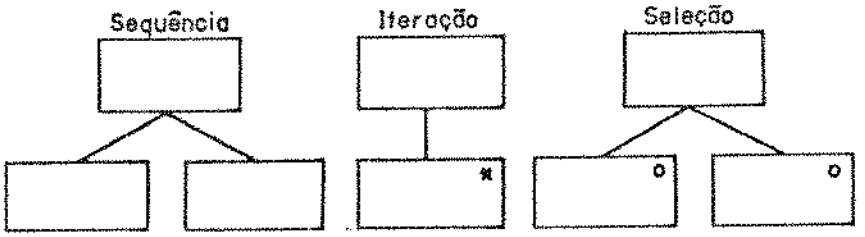


Figura 5.5: Notação gráfica empregada no método de Jackson [28].

Estes diagramas apresentam uma especificação ordenada das ações executadas sobre ou pela entidade, constituindo-se, assim, numa representação mais precisa do mundo real quando comparada a uma lista simples de ações e entidades. Deve-se criar um diagrama para cada entidade considerada [28].

# Capítulo 6

# PROGRAMAÇÃO ORIENTADA POR OBJETOS

## 6.1 Introdução

Este capítulo tem por objetivo discutir alguns aspectos relacionados à programação orientada por objetos. Inicialmente, são apresentados alguns conceitos gerais, tais como abstração e modulação, e conceitos relacionados ao modelo por objetos, como por exemplo objeto, classes e mensagens. Estes conceitos são ilustrados através da estrutura de dados de uma classe. Faz-se ainda, uma comparação entre programação convencional e programação por objetos. Posteriormente, discute-se o reaproveitamento de programas, fazendo-se uma analogia entre os modos de produção de componentes eletrônicos e programas, e como o modelo por objetos pode ser aplicado com maior sucesso. No Apêndice C são apresentadas, de maneira suscinta, as características de duas linguagens orientadas por objetos.

## 6.2 Conceitos Aplicados ao Modelo Orientado por Objetos

O modelo orientado por objetos, de maneira análoga aos vários paradigmas de programação, introduz alguns conceitos particulares, tais como objetos, mensagens, mecanismo de herança, dentre outros.

Ressalta-se, porém, que nem todas as linguagens orientadas por objetos implementam todos os conceitos a serem descritos. Assim, a apresentação feita a seguir tem um caráter mais geral, não procurando destacar as particularidades de uma linguagem.

## 6.3 Conceitos Gerais

### 6.3.1 Modulação

O conceito de modulação é de certa forma intuitivo, ou seja: dado um problema complexo, subdividi-se o mesmo em subproblemas menos complexos visando facilitar a solução do problema global.

No entanto, a aplicação do conceito de modulação não pode ser realizada indefinidamente no desenvolvimento de um programa. Ao mesmo tempo em que o esforço de desenvolvimento diminui substancialmente quando se aumenta o número de módulos, o esforço de interfaceamento deste módulos cresce na mesma proporção [28].

Alguns tipos de módulos são encontrados em linguagens de programação. O *módulo*, da linguagem *Módula*, e o *pacote*, proveniente de *Ada*, são componentes de programas que combinam abstrações de dados e procedimentos, incentivando assim, o desenvolvimento de programas modulares [28].

### 6.3.2 Ocultamento de Informação

A aplicação do conceito de ocultamento de informação no desenvolvimento de um sistema permite construir módulos onde a interdependência entre os mesmos é pequena. Além disso, a confiabilidade do sistema é aumentada e as modificações são efetuadas localmente dentro de cada módulo, preservando assim, a disseminação destas alterações ao longo de todo o sistema.

Para se alcançar uma modulação efetiva, define-se um conjunto de módulos interdependentes, os quais se comunicam entre si apenas através das informações necessárias para acessar uma característica do módulo ou executar uma de suas operações [28].

O estado de um módulo é descrito por variáveis locais, visíveis apenas dentro do escopo deste módulo, e um conjunto de procedimentos que manipulam estes dados. Assim, como os estados das variáveis internas não podem ser acessados diretamente por outros módulos, uma interface deve permitir que a estrutura de dados interna e procedimentos sejam alterados sem afetar a implementação de outros módulos [38].

O uso de abstrações de dados permite definir e utilizar o conceito de ocultamento de informação no desenvolvimento de programas.

### 6.3.3 Abstração

Para problemas onde se aplicam o conceito de modulação, vários níveis de abstração podem ser considerados. No nível mais alto de abstração, a solução adotada para o problema é colocada em termos de uma linguagem próxima ao ambiente do problema. Nos níveis mais baixos, descrevem-se os procedimetos a serem implementados. Assim, o uso

de abstração permite ao programador concentrar-se em um problema, considerando um nível de generalização qualquer sem se preocupar com detalhes irrelevantes ao problema [28].

Várias linguagens de programação, tais como *Ada*, *Modula* e *Smalltalk*, permitem a criação de tipos abstratos de dados, os quais consistem de uma representação interna para os dados e um conjunto de procedimentos para acessar e manipular os dados [36].

Um tipo abstrato de dados pode ser visto como uma classe de objetos, caracterizados pelas operações disponíveis sobre os mesmos e as propriedades abstratas destas operações [41].

### 6.3.4 Ligação Dinâmica

Nas linguagens convencionais, tais como *C*, *Pascal* e *FORTRAN*, a partir do conhecimento dos tipos de variáveis e constantes utilizadas em uma declaração, o compilador gera o código de máquina correspondente. Este processo de definir os tipos de dados aplicáveis a um operador em uma declaração, anterior a sua execução, é denominado *ligação estática* [32,38]. Além disso, algumas destas linguagens permitem que o usuário defina os seus próprios tipos de dados. Assim, supondo que um programa gráfico esteja sendo desenvolvido e que se deseje definir o tipo *vetor*, isto poderia ser feito, em linguagem *C*, através do seguinte conjunto de declarações:

```
typedef  struct {
   int  x, y;
} vetor;
```

No entanto, as linguagens convencionais não permitem alterar o significado dos seus operadores, impossibilitando a aplicação destes aos novos tipos definidos pelo usuário. Por exemplo, definindo dois vetores *origem* e *extensão*,

```
vetor origem, extensao;
```

e supondo que se desejasse somar estes dois vetores,

```
vetor  resultante = origem + extensao;
```

isto não poderia ser feito, mas somente através das declarações,

```
resultante.x = origem.x + extensao.x;
resultante.y = origem.y + extensao.y;
```

Entretanto, linguagens mais recentes, tais como *Ada*, *Modula* e *C++*, permitem esta flexibilidade de redefinir operadores convencionais para os tipos declarados pelo usuário. Assim, pode-se definir o tipo *vetor* onde o operador + é uma operação legal. No entanto, esta ligação entre tipos e operandos é realizada ainda, em tempo de compilação do código fonte.

A ligação dinâmica permite associar um tipo de dado a um operando em tempo de execução do programa. Em algumas linguagens orientadas por objeto, como um mesmo seletor de mensagem pode ser utilizado para objetos distintos, a ligação entre o tipo de dado e a operação é dinâmica, dependendo do receptor da mensagem.

O conceito de ligação dinâmica pode ser melhor entendido através do seguinte exemplo [32]. Considere um estojo onde devem ser guardados canetas, lápis e borrachas. Deseja-se saber o peso total do estojo quando for colocado em seu interior uma caneta, um lápis e uma borracha. Identificam-se quatro objetos de interesse: *estojo*, *caneta*, *lápis* e *borracha*. O objeto *estojo* é considerado como uma instância do objeto *caixa*.

Inicialmente, os objetos devem ser colocados no interior do estojo. Esta operação pode ser realizada pelas seguintes declarações de *Objective-C* :

```
extern   id Caixa, Caneta, Lapis, Borracha;
id Estojo = [Caixa new];                    // cria a instancia Estojo
[estojo adiciona: Caneta];                  // adiciona uma caneta ao estojo
[estojo adiciona: Lapis];                   // adiciona uma lapis ao estojo
[estojo adiciona: Borracha];                // adiciona uma borracha ao estojo
```

Para se obter o peso total do estojo deve-se calcular o peso de cada componente e adicionar estes valores ao peso do estojo vazio. Utilizam-se as declarações :

```
float PesoTotal = PesoEstojo;               // inicializa PesoTotal
int i, n = [self size];                     // n = numero de componentes
for  (i = 0; i < n; ++i) {
   id componente = [self at: i];            // proximo componente
   PesoTotal += [componente peso];          // acumula peso dos componentes
   }
return PesoTotal;
```

Observa-se então, que a mensagem *peso* é enviada para todos os objetos colocados no estojo. No entanto, o método que implementa esta mensagem é determinado, para cada objeto, em tempo de execução do programa.

## 6.4 Conceitos de Programação Orientada por Objetos

### 6.4.1 Objeto

Um objeto se constitui na unidade básica de modulação no modelo orientado por objetos, representando uma informação e a sua manipulação. Assim, visto como um conjunto de dados, um objeto pode ser manipulado, mas visto como um conjunto de procedimentos, um objeto descreve a sua própria manipulação [37]. Desta forma, um objeto pode ser considerado como um tipo abstrato de dados [36], ou seja, uma estrutura de dados e um conjunto de procedimentos que acessam e manipulam estes dados.

Para se manipular a informação representada por um objeto, deve-se solicitar ao mesmo que execute uma de suas operações, através do envio de uma mensagem. O objeto que recebe a mensagem é denominado receptor, devendo responder esta mensagem através da seleção da função correspondente, executar esta operação e retornar o controle para o objeto emissor da mensagem.

Portanto, um objeto pode descrever, em um único módulo, qualquer entidade do mundo real a ser representado num sistema de informação. Para isso, as características de uma entidade são descritas pelos dados e as ações por procedimentos acessados através do envio de mensagens.

Por exemplo, num sistema de projeto auxiliado por computador (PAC), uma peça mecânica pode ser considerada como um objeto. As características desta peça, tais como dimensões, material e processo de fabricação, são descritas pelos dados. Já o conjunto de operações, tais como análise estrutural e simulação do processo de fabricação, são representadas por procedimentos acessados pelo envio de mensagens.

### 6.4.2 Mensagens

As mensagens se constituem nas especificações das operações de um objeto. Assim, quando um objeto recebe uma mensagem, deve determinar como manipulá-la para obter a resposta requerida [37]. Uma mensagem inclui um seletor, descrevendo o tipo de manipulação desejada, e argumentos, os quais podem ser outros objetos ou valores das variáveis de um objeto.

A característica principal do mecanismo de mensagem é que o seletor é um nome de uma operação, descrevendo apenas a ação a ser executada. Entretanto, uma mesma mensagem pode ser interpretada de maneiras diferentes, pois é responsabilidade do receptor determinar a ação correta a ser executada. Assim, considerando dois objetos distintos, um mesmo seletor pode corresponder a operações distintas.

Portanto, no sistema PAC, exemplificado anteriormente, a mensagem para operação

de análise estrutural de uma turbina poderia ser a mesma daquela para um veículo, no entanto os métodos de análise empregados poderiam ser totalmente distintos.

### 6.4.3 Classes e Instâncias

Vários sistemas orientados por objetos fazem uma distinção entre um objeto e a sua descrição. Assim, definem-se os conceitos de classe e instância.

Uma classe é uma descrição geral de um conjunto de objetos semelhantes, provendo todas as informações necessárias para a criação e utilização dos objetos. Uma instância, por sua vez, é um objeto descrito por uma classe particular.

Cada objeto é instância de uma classe. Assim, uma classe descreve todas as similaridades de suas instâncias. Entretanto, cada instância contém as informações que as distingue das demais. Estas informações se constituem em um subconjunto determinado de variáveis, denominadas *variáveis de instância*. Todas as instâncias de uma classe tem o mesmo número de variáveis de instância, sendo o conteúdo destas variáveis distintos para cada uma delas [37].

Da mesma forma, todas as instâncias de uma classe utilizam o mesmo método para responder a uma mensagem particular. A diferença na resposta obtida para duas instâncias distintas é resultado dos diferentes valores armazenados nas variáveis. Além das variáveis de instância, tem-se, ainda as *variáveis de classe*, as quais são compartilhadas por todas as instâncias de uma classe.

Portanto, pode-se dizer que um sistema orientado por objetos é desenvolvido a partir da criação das classes que descrevem os objetos constituintes do sistema. Desta forma, uma classe pode ser vista como um objeto. Ressalta-se que outras denominações são atribuídas às classes, tais como *objetos de fábrica* [32].

Retomando o sistema PAC, várias classes poderiam ser definidas, tais como as classes correspondentes às peças mecânicas bidimensionais e tridimensionais, peças que podem ser analisadas por formulações simétricas, dentre outras.

### 6.4.4 Métodos

Os métodos são procedimentos invocados pelo envio de seletores de mensagens para as instâncias de uma classe. Portanto, um método, como um procedimento, é a descrição de uma sequência de ações a serem executadas, observando-se, no entanto, que para um método acessar outro, deve enviar uma mensagem. Assim, os métodos não podem ser separados dos objetos [37]. De forma análoga aos procedimentos, os métodos devem conhecer os tipos de dados que manipulam.

### 6.4.5 Mecanismo de Herança

O mecanismo de herança permite compartilhar as informações entre os objetos de um sistema. Supondo então, um sistema de informações descrito por uma hierarquia de objetos, tem-se que objetos situados em um nível inferior desta hierarquia herdam todas as características (dados e operações) dos objetos situados em níveis superiores.

A maioria dos sistemas orientados por objetos implementam o mecanismo de herança entre as classes do sistema. Uma classe pode ser alterada para criar uma outra. Nesta relação, a primeira classe é denominada *superclasse* e a segunda *subclasse*. Para uma subclasse, pode-se adicionar variáveis de instância e de classe, alterar métodos para mensagens presentes na superclasse e acrescentar métodos correspondentes a novas mensagens.

Considerando o exemplo do sistema PAC, uma classe denominada *peças mecânicas tridimensionais* poderia herdar as variáveis de instância $x$ e $y$, descrevendo as coordenadas de um ponto, da classe *peças mecânicas bidimensionais*. Além disso, acrescentaria uma variável $z$ e métodos para manipular e acessar esta variável.

### 6.4.6 Exemplo da Estrutura de Dados de uma Classe

A estrutura de dados apresentada na Figura 6.1 ilustra os conceitos de objetos, classes, instâncias, mensagens, métodos e mecanismo de herança, da maneira implementada pelo sistema *Smalltalk-80* [38].

A estrutura de dados da classe contém as informações necessárias para criação de instâncias. Os primeiros dois campos contém o número e o nome das variáveis de instância, respectivamente. O terceiro campo é uma chave indicando a existência ou não de variáveis de instância indexadas.

O campo de variável de classe armazena as variáveis compartilhadas por todas as instâncias, assim como os nomes destas variáveis. As informações comuns às instâncias de uma classe são, convenientemente, armazenadas neste campo. Tal fato, além de economizar espaço de memória, simplifica alterações realizadas nas variáveis compartilhadas pelas instâncias.

O dicionário de métodos contém pares de seletores e métodos. Quando uma mensagem é enviada a um objeto, o dicionário de métodos da classe é pesquisado para determinar o método a ser executado.

O campo da superclasse aponta para a superclasse correspondente. A estrutura de dados da superclasse contém os mesmos campos já apresentados, definindo as suas variáveis de instância e de classe e seus próprios métodos.

Quando um método, correspondente a uma mensagem enviada a uma instância, não é encontrado no dicionário de métodos da classe, efetua-se, então, uma pesquisa

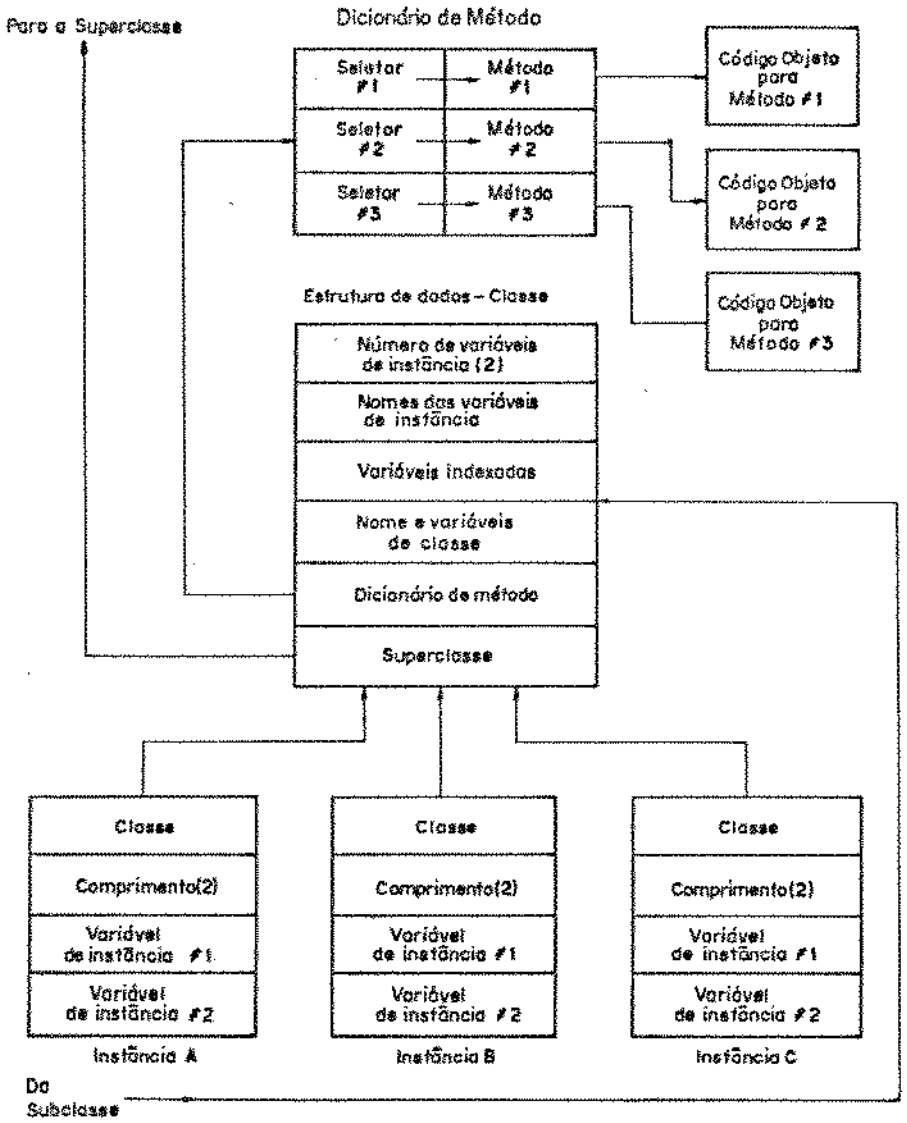


Figura 6.1: Estrutura de dados de uma classe [38].

na superclasse, e assim sucessivamente até que a raiz da hierarquia seja alcançada. Se o seletor do método não é encontrado ao longo de toda a hierarquia, retorna-se uma mensagem de erro.

Cada instância contém um apontador para a sua classe e um campo indicando o tamanho da instância. O apontador para a classe é necessário, pois este facilita a localização dos métodos de um objeto quando uma mensagem é enviada.

## 6.5 Programação Convencional e Programação por Objetos

A programação convencional é baseada no modelo dados/procedimento. Assim, sistemas de informação são compostos de uma coleção de dados, representando a informação, e um conjunto de procedimentos que manipulam estes dados. Portanto, um procedimento é executado através da passagem dos dados como argumentos.

Um problema relacionado com este paradigma é que dados e procedimentos são tratados de maneira independente. Assim, todos procedimentos assumem, previamente, os tipos de dados a serem manipulados [37].

No modelo orientado por objetos, uma entidade simples representa a informação e a sua manipulação. O acesso ao objeto é feito através do envio de uma mensagem, em vez de chamar, explicitamente, um procedimento. Uma mensagem então, nomeia a operação, enquanto um procedimento descreve os detalhes da operação.

Cada procedimento possui um nome que é usado quando se chama o mesmo. Portanto, o nome não apenas especifica o procedimento a ser chamado, mas também a ação a ser executada. Ao contrário, uma mensagem nomeia apenas a tarefa e não a maneira de executá-la, pois isto é uma responsabilidade do receptor [37]. A implementação das operações é feita através dos métodos.

Um objeto pode ser visto de uma maneira externa e interna. O lado externo de um objeto é a sua interface, descrevendo como o objeto é visto pelos demais com os quais se interage. Do ponto de vista externo, pode-se apenas requisitar uma tarefa a um objeto. O lado interno representa o objeto do ponto de vista de implementação. Neste caso, pode-se especificar a um objeto como executar uma tarefa.

A visão externa de um objeto representa o seu protocolo, ou seja, o conjunto de mensagens que o mesmo pode responder. A visão interna é similar à um modelo dados/procedimento. Os valores das variáveis locais de um objeto são análogos aos dados, enquanto, métodos são similares aos procedimentos. Esta distinção entre dados e procedimentos em um objeto é estritamente localizada na sua parte interna [37].

Assim, a programação por objetos representa uma *lei de inversão* [41]. Em vez de se construir módulos de operações e distribuir a estrutura de dados entre as rotinas resul-

tantes, o modelo por objetos utiliza uma estrutura de dados como base de modulação, associando cada rotina aos dados onde se aplica. Um novo sistema é desenvolvido a partir da criação das classes que descrevem o comportamento dos objetos, e não a partir das funções que o sistema executa.

Toma-se como exemplo, um sistema de gerenciamento de janelas. Estas janelas ocupam regiões retangulares na tela de vídeo de um computador, contendo textos e títulos. Várias operações podem ser aplicadas às janelas, como por exemplo, movê-las ao longo da tela, alterar os seus conteúdos, apagá-las, dentre outras.

Este sistema implementado através do modelo convencional teria dados representando a posição, tamanho, conteúdo, título, dentre outras características, para cada janela na tela de vídeo. Além disso, algumas operações seriam definidas para manipular a estrutura de dados. Para mover uma janela, por exemplo, seria necessário invocar o procedimento que move janelas, e fornecer ao mesmo a estrutura de dados representando a janela e sua nova posição na tela. O inconveniente desta representação é que os procedimentos assumem previamente os tipos de dados a serem manipulados.

Se o mesmo sistema fosse implementado através do modelo por objetos, este possuiria um conjunto de objetos representando as janelas. Assim, cada objeto descreveria uma janela através de uma estrutura de dados e um conjunto de procedimentos para manipulá-las em um único módulo. Para cada manipulação corresponderia um seletor de mensagens. Portanto, para mover uma determinada janela no vídeo seria necessário mandar uma mensagem para o objeto representando esta janela.

Assim, na programação convencional, uma alteração da estrutura de dados implica na modificação de todas as funções do sistema que utilizam estes dados. Já na programação por objetos, uma modificação ocorre localmente no objeto para os métodos que manipulam a estrutura de dados considerada.

Desta maneira, a programação por objetos apresenta uma maior flexibilidade em relação àquela encontrada no modelo convencional. Além disso, a característica modular de um objeto permite simplificar a manutenção de um programa, representando ainda, uma melhor aproximação para a reutilização de código.

## 6.6 Reaproveitamento de Código utilizando Programação Orientada por Objetos

Na seção seguinte apresenta-se uma comparação nos modos de produção de circuitos eletrônicos e programas, discutindo ainda, a necessidade de se utilizar técnicas que permitam a reutilização de elementos de programas. Posteriormente, discutem-se as limitações das técnicas empregadas no projeto e desenvolvimento de programas, sob o ponto de vista de reutilização de componentes, e como o modelo por objetos pode ser

aplicado de maneira mais mais razoável e confiável.

### 6.6.1 Comparação entre as Técnicas de Desenvolvimento de Circuitos Eletrônicos e de Programas

O progresso da indústria eletrônica nas últimas duas décadas é extraordinário. Isto se deve a utilização de tecnologias de empacotamento de componentes, o que permitiu um alto grau de reaproveitamento de projetos já desenvolvidos. Assim, foi possível reduzir substancialmente o esforço envolvido em novos projetos através da construção de módulos de componentes ou circuitos eletrônicos. Desta maneira, verificou-se um crescimento exponencial na indústria eletrônica.

Enquanto isso, os programas vem sendo desenvolvidos a partir de declarações simples de uma linguagem de programação. Estas declarações constituem a unidade básica de modulação, assim como os circuitos são a unidade básica para desenvolvimento de placas de circuitos integrados.

Logo, enquanto os projetistas de circuitos eletrônicos direcionaram as suas aplicações em termos de módulos, os programadores ainda idealizam programas através de declarações simples de uma linguagem. Apenas as ferramentas foram alteradas, como por exemplo as linguagens. Assim, o crescimento da produção de programas, no mesmo período de tempo considerado para a indústria de componentes eletrônicos, foi apenas aritmético.

O reaproveitamento de projetos e componentes na indústria eletrônica tornou-se a regra, enquanto no desenvolvimento de programas é uma exceção. A Figura 6.2 ilustra os vários estágios do desenvolvimento de *hardware* e *software* ao longo das duas últimas décadas.

Inicialmente, os circuitos eram construídos através de elementos discretos como resistores, transistores, capacitores, dentre outros. Com a integração destes componentes foi possível construir os circuitos integrados, facilitando o reaproveitamento dos módulos em várias aplicações. Os principais conceitos introduzidos pelas tecnologias de empacotamento podem ser resumidas na seguinte maneira [40]:

- os componentes executam funções únicas, acessíveis através de uma requisição. Portanto, o usuário não necessita conhecer os métodos utilizados, mas apenas como requisitar a um componente uma determinada função.
- a comunicação entre os componentes é feita através de mensagens, ou seja, do envio de sinais. Assim, a independência e o livre acoplamento dos componentes é alcançada.
- alto grau de reaproveitamento através do desenvolvimento de padrões. As funções

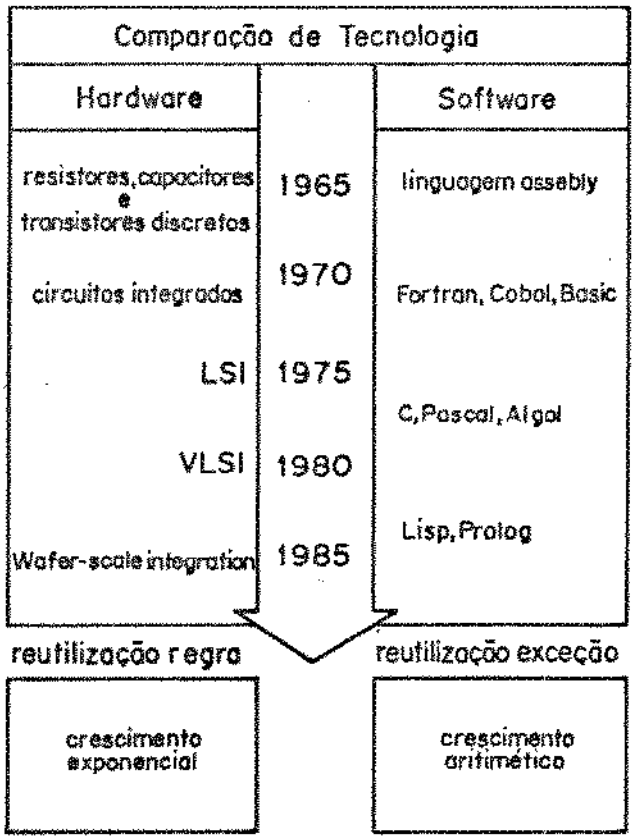


Figura 6.2: Comparação das técnicas empregadas no desenvolvimento de *Hardware* e *Software* [32].

padrões são facilmente identificáveis devido a correspondência direta entre o modelo real e o modelo do circuito.

- os componentes modificados herdam a maior parte das implementações anteriores, contendo apenas as alterações necessárias para suprir o novo comportamento desejado.

Dentro deste contexto, surge a questão se uma revolução nos modos de desenvolvimento de programas, comparável àquela ocorrida na eletrônica, é possível. Para isso, torna-se necessário aplicar técnicas de modulação análogas às tecnologias de empacotamento. A programação estruturada não permite uma aproximação viável para este problema, tornando-se necessário então, aplicar uma nova metodologia. Neste contexto, o modelo por objetos permite uma aproximação mais realista. Os conceitos básicos introduzidos pelo modelo de objetos, como técnica de modulação de programas, são as seguintes [32,40]:

- os tipos abstratos de dados permitem ao usuário manipular a estrutura de dados apenas através dos procedimentos, os quais constituem uma interface para os módulos de programas.
- os objetos se comunicam através de mensagens, especificando-se apenas a função a ser executada.
- o conceito de herança permite a difusão de código já desenvolvido ao longo da hierarquia do sistema. Assim, um programa pode ser desenvolvido a partir de outros módulos, de maneira análoga ao projeto de um circuito integrado.
- o mapeamento direto entre o modelo conceitual e modelo computacional.

Desta maneira, pode-se dizer que a metodologia de programação orientada por objetos representa um ponto de partida para o desenvolvimento de programas, onde o reaproveitamento de código possa ser uma regra a ser seguida [41].

### 6.6.2 Reaproveitamento de Componentes de Programas

As barreiras encontradas para a reutilização de código não são apenas de conteúdo gerencial, mas também de ordem técnica. Segundo Meyer [41], apenas 15% de códigos já desenvolvidos e testados são de aplicação geral. Assim, torna-se interessante discutir como a metodologia de programação por objetos pode ser aplicada visando o reaproveitamento de componentes de programas.

O desenvolvimento de programas, utilizando programação convencional, possui uma natureza repetitiva. Programadores comumente trabalham sobre o mesmo problema, desenvolvendo versões diferentes para o mesmo tema. Ao se elaborar uma nova versão

de um programa, o número de detalhes a ser alterado pode ser tão grande, que é possível questionar até que ponto as duas versões são semelhantes. Neste contexto, várias aproximações têm sido aplicadas visando a reutilização de código.

A técnica clássica compreende a construção de bibliotecas de rotinas, onde cada rotina implementa uma operação bem definida. Esta aproximação é adequada quando um conjunto de problemas individuais pode ser identificado. No entanto, possui as seguintes limitações [41]:

- qualquer instância de um problema deve ser identificável por um conjunto pequeno de parâmetros.
- os problemas individuais devem ser distintos.
- estruturas de dados complexas não devem ser utilizadas, pois as mesmas teriam de ser distribuídas entre as rotinas e a autonomia dos módulos seria perdida.

Uma outra solução está relacionada ao uso de linguagens modulares, tais como *Modula* e *Ada*. Um módulo pode conter mais de uma rotina, juntamente com declarações de tipos, constantes e variáveis. A limitação desta aproximação está em não reduzir significativamente a quantidade de código a ser escrita. Além disso, não é possível descrever as características comuns aos vários módulos.

Duas importantes características, presentes em algumas linguagens, devem ser consideradas para reaproveitamento de código. A primeira está relacionada à possibilidade de se atribuir significados distintos a um mesmo operador para tipos de dados diferentes. A segunda característica é a utilização de tipos de dados como parâmetros genéricos em um módulo. Isto permite a definição de módulos genéricos em vez de um grupo de módulos que diferem apenas no tipo de dados que manipulam.

Apesar de serem características úteis, estes conceitos não possuem uma grande flexibilidade, pois não permitem representar detalhes comuns entre grupos de implementações da mesma abstração de dados geral [41]. Este grau de flexibilidade pode ser alcançado através da metodologia orientada por objetos, que pode ser definida, assim, como uma técnica de modulação de sistemas. Como discutido anteriormente, a metodologia orientada por objetos baseia a decomposição de um sistema através das classes de objetos que este sistema manipula.

Na programação convencional, considera-se a decomposição através das funções que o sistema executa. Aproximações convencionais clássicas, como o desenvolvimento descendente, são adequadas se um programa deve resolver somente um problema bem determinado. Mas se as tarefas a serem executadas alteram-se ao longo do tempo de vida do programa, tal solução não é satisfatória.

No entanto, as categorias de objetos que o sistema trabalha são, praticamente, as mesmas. Portanto, se estas categorias são trazidas a um nível de abstração alto, torna-

se atrativo fazer a decomposição de um sistema em termos das categorias de objetos que o mesmo manipula.

Um conceito fundamental para que o modelo por objetos possa ser aplicado é o de tipo abstrato de dados, o qual descreve uma classe de objetos através de suas características externas, e não pela sua representação computacional. Assim, o desenvolvimento orientado por objetos pode ser visto como a construção de coleções estruturadas de implementações de tipos abstrato de dados [41].

Assim, a partir dos conceitos de modelo orientado por objetos, a reutilização de código torna-se possível. Construindo-se módulos que são especificados ao meio externo através de uma interface e permitindo a difusão de código através da hierarquia do sistema, torna-se possível representar as várias características de uma abstração de dados. Assim, pode-se construir módulos de aplicação geral, os quais permitem o desenvolvimento de um sistema particular acrescentando-se apenas os detalhes específicos desta aplicação.

# Capítulo 7

# FAE - FERRAMENTA DE ANÁLISE ESTRUTURAL

## 7.1 Introdução

Este capítulo tem por objetivo discutir as necessidades gerais do programa FAE – Ferramenta de Análise Estrutural. Para isso, apresentam-se as características identificadas no documento de Especificação do Sistema [42]. Discute-se então, um método de análise de requerimentos baseado nos conceitos do modelo orientado por objetos. Esta técnica é aplicada para a definição dos vários objetos a serem implementados no programa. Finalmente, apresenta-se a sintaxe gramatical a ser empregada na especificação do arquivo de dados do modelo estrutural considerado.

## 7.2 Análise de Sistema

Como discutido no Capítulo 5, a análise de sistema cria uma definição clara e objetiva de um sistema de informação, a partir da especificação de suas necessidades gerais. Desta forma, alocam-se as funções para os vários elementos do sistema, avalia-se a viabilidade para a sua implantação e estabelecem-se custos e programas.

Desenvolveu-se um documento de Especificação de Sistema para o programa FAE [42] com o objetivo de identificar, de maneira global, as necessidades para implantação de um programa para análises estática e dinâmica de estruturas modeladas pelo MEF. Utilizou-se o modelo de documentação dado em [28], o qual é dividido em várias seções, descritas a seguir:

**Introdução** descreve-se os objetivos e o ambiente de operação do sistema. Além disso,

especifica-se o escopo do sistema, a sua viabilidade, justificativa e recursos requeridos.

**Descrição Funcional** efetua-se a descrição das funções do sistema, devendo-se considerar os dados de entrada, tarefas a serem executadas, a informação resultante e a interface de dados.

**Alocação** as funções identificadas são distribuídas para os vários elementos do sistema.

**Restrições** apresentam-se as limitações técnicas e gerenciais que afetam o desenvolvimento do sistema. Várias restrições são consideradas, tais como as interfaces entre os vários elementos do sistema, projeto e implementação, recursos disponíveis e cronograma.

**Cronograma** nesta fase não é possível estabelecer um cronograma detalhado de trabalho. No entanto, realiza-se uma estimativa inicial, a qual deve ser reavaliada durante o planejamento do sistema.

A seguir apresentam-se as características gerais identificadas para o programa FAE.

### 7.2.1 Características Gerais do Programa FAE

O programa FAE foi dividido em três partes básicas: o pré-processador; o processamento; e o pós-processador. Deseja-se que estas partes sejam interativas com o usuário, permitindo facilidades e flexibilidade na execução das várias funções implementadas.

O pré-processador consiste na especificação do modelo discreto da estrutura em estudo. Inicialmente, utiliza-se um arquivo de dados contendo as características do modelo como entrada para o programa. A longo prazo, pretende-se utilizar um modelador de sólidos acoplado à algoritmos de geração automática de malhas para construção do modelo.

A partir dos dados de entrada, passa-se à fase de processamento, obtendo-se grandezas de interesse no estudo da estrutura, tais como deslocamentos e frequências naturais. Incialmente, considera-se os conceitos de subestruturação e simetria cíclica. Posteriormente, pretende-se implementar algoritmos adaptativos, permitindo que o processo de análise possa ser automatizado e refinado sob controle do usuário.

Finalmente, no pós-processamento dos dados, determina-se, a partir das grandezas básicas, outras características da estrutura, como por exemplo as distribuições de tensão. Neste caso, os resultados devem ser representados de forma gráfica, assim como através de tabelas com valores numéricos.

Duas outras funções devem ser consideradas na implementação do programa. A primeira é a interface gráfica com o usuário, a qual deve gerenciar o acesso e a manipulação do programa pelo usuário. Para isso, recebe comandos de entrada de um

periférico ou um arquivo de dados, devendo interpretá-los e executar as ações requisitadas. A outra função é o gerenciador da base de dados, o qual deve criar, atualizar e manipular a estrutura de dados ao longo da execução do programa.

Todas as características citadas anteriormente serão implementadas através de elementos de *software*. Observa-se que as funções discutidas representam macro-elementos do programa. Assim, torna-se necessário um processo de especificação mais aprofundado.

Algumas tarefas serão alocadas a elementos de *hardware*, tais como entrada de dados, processamento gráfico e aritmético, armazenamento de dados, dentre outros. No entanto, o controle de todas estas tarefas será executado por elementos de *software*. Na implementação do gerenciador da base de dados, serão considerados conceitos relativos a banco de dados, procurando dar flexibilidade no acesso dos dados do modelo e grandezas calculadas pelo programa. Por sua vez, a interface gráfica tem por objetivo criar um ambiente amigável com o usuário.

O ambiente adequado para o desenvolvimento do programa FAE é uma estação de trabalho com grande capacidade de processamento aritmético e gráfico, além de um conjunto de ferramentas de programação, tais como compiladores, editores e bibliotecas. Para a manipulação dos dados pelo usuário e apresentação dos resultados, torna-se interessante a presença de alguns periféricos, tais como *mouses*, traçadores gráficos e impressoras de alta qualidade.

O desenvolvimento deste programa é justificável, devido a vários fatores:

- os programas comerciais existentes para análise estrutural pelo MEF possuem uma concepção rígida, dificultando, em alguns casos, a manipulação do modelo e o acesso a suas várias funções. Além disso, o usuário adquire todo o programa, não sendo possível atender apenas as suas necessidades.
- devido aos avanços da eletrônica nos últimos anos, os computadores possuem um grande potencial operacional a ser explorado.
- novas metodologias de desenvolvimento de programas vêm sendo aplicadas, permitindo a construção sistemática de programas modulares e confiáveis.
- novos conceitos na área de análise pelo MEF têm sido introduzidas, oferecendo uma maior confiabilidade e facilidades de aplicação do método.

As principais restrições estão relacionadas ao conhecimento de todas as técnicas e algoritmos a serem implementados no programa. Assim, divide-se o desenvolvimento do trabalho em etapas, procurando inicialmente, implementar os métodos de análise estrutural discutidos neste trabalho. Pretende-se obter ainda, uma metodologia de desenvolvimento de programas baseada nos conceitos de orientação por objetos. A

estrutura do programa deve ser suficientemente flexível, para permitir que novas características possam ser acrescentadas, sem afetar globalmente os módulos já implementados. Na próxima seção, apresentam-se as principais características a serem consideradas na primeira versão do programa FAE.

### 7.2.2 Características Gerais da Primeira Versão do Programa FAE

Nesta etapa, serão implementados algoritmos para a análise linear (estática e dinâmica) utilizando as técnicas convencional, por subestruturação e ciclo-simétrica. Os sistemas de equações e problemas de autovalor, provenientes dos processos de análise, serão resolvidos, respectivamente, pelos métodos de eliminação de Gauss e iteração por subespaço [15].

Na análise por subestruturação estática, emprega-se o método de deslocamentos [6], e para a análise dinâmica o algoritmo de síntese por subestruturação [25].

Deve-se implementar os elementos finitos de barra e viga, visando aplicar os métodos de análise citados no estudo de estruturas reticuladas.

A especificação do modelo discreto da estrutura considerada é realizada através de um arquivo de dados. Assim, uma gramática e um interpretador para os comandos devem ser desenvolvidos. A estrutura de dados do programa é alocada dinamicamente, sendo inicializada com as características do modelo e o tipo de análise a ser efetuada. Os resultados obtidos são então, armazenados em arquivos de saída especificados pelo usuário.

A metodologia de programação utilizada, para a implantação destas características, deve possuir uma notação uniforme e objetiva, procurando simplificar não apenas o entendimento da estrutura do programa, como também facilitar a realização de qualquer modificação posterior.

Estima-se um período de 18 meses para a implementação desta primeira versão do programa FAE.

## 7.3 Análise de Requerimentos Orientada por Objetos

Um dos principais objetivos da análise de requerimentos é representar os domínios funcional e de informação alocados aos elementos de *software* durante a análise de sistema. Observa-se que os conceitos de orientação por objetos podem ser aplicados de maneira satisfatória na análise de requisitos de um programa, possibilitando efetuar uma partição natural dos domínios funcional e de informação.

A técnica de análise de requerimentos orientada por objetos pode ser resumida nos seguintes passos [28]:

- descreve-se em linguagem natural a solução do problema a ser resolvida pelos programas, utilizando níveis de detalhes consistentes com a especificação das funções obtidas na análise de sistema.
- os substantivos, presentes na descrição, são os objetos a serem considerados. Um objeto pertence ao *espaço de solução* do problema, se este objeto é necessário para implementar a solução considerada. Caso contrário, o objeto é utilizado apenas para descrever a solução, pertencendo ao *espaço do problema*.
- os atributos dos objetos são identificados pelos adjetivos, devendo-se associá-los aos seus respectivos objetos.
- as operações são determinadas pelos verbos contidos na descrição, devendo-se relacioná-las aos seus objetos.
- os atributos das operações são descritos pelos advérbios presentes no texto.

Portanto, elabora-se em linguagem natural uma descrição concisa e clara da solução do problema relacionada aos elementos de *software*. A partir daí, interpreta-se este texto, identificando os objetos de interesse, seus atributos e suas operações, assim como os atributos destas operações. Finalmente, constrôem-se uma tabela ilustrando os objetos, atributos e operações pertencentes ao espaço de solução do problema. Observa-se que esta técnica possui uma grande semelhança com o método de Jackson, discutido no Capítulo 5.

A análise orientada por objetos constitui-se num mecanismo simples para representar os elementos principais dos domínios funcional e de informação de um programa. Observa-se que cada objeto identificado, pode ser particionado e o processo de análise reaplicado iterativamente [28]. Exemplos de aplicação desta técnica podem ser encontrados na seção seguinte e no Apêndice D.

## 7.4 Análise de Requerimentos do Programa FAE

Confeccionou-se um documento de análise de requisitos [43], descrevendo os domínios funcional e de informação do problema, seguindo as especificações discutidas na Seção 7.2.2. Para isso, empregou-se o método de análise por objetos da seção anterior.

Primeiramente, procurou-se representar de maneira objetiva as necessidades gerais do programa FAE. Seguindo o método de análise por objetos da seção 7.3, elaborou-se a seguinte descrição para o problema [43]:

*Um programa para realizar a modelagem, análise (estática e dinâmica) e simulação de estruturas, baseado no MEF, deve ser desenvolvido.*

*A estrutura a ser estudada é especificada através de um modelo estrutural fornecido pelo usuário. Este modelo é então, armazenado na forma da estrutura de dados do programa.*

*A seguir, métodos de análise estrutural são empregados para se obter as grandezas de interesse no estudo da estrutura. Estes métodos devem acessar o modelo estrutural, para a obtenção de dados, e aplicar métodos de análise numérica, para a solução de sistemas de equações e problemas de autovalor provenientes dos algoritmos de análise. Os resultados são então, armazenados para posterior acesso.*

*Finalmente, pode-se obter a resposta da estrutura, sob condições pré-estabelecidas, através de métodos de simulação.*

Através da análise deste texto, identificou-se os objetos pertencentes ao espaço de solução do problema, assim como suas operações, ilustrados na Tabela 7.1. Observa-se que estes objetos representam conceitos mais gerais. Desta forma, os demais objetos identificados, para esta primeira versão do programa, estarão diretamente relacionados a estes conceitos gerais. Utilizou-se um diagrama de fluxo de dados, mostrado na Figura 7.1, para apresentar a interação entre os vários objetos do programa. O dicionário de dados, relativo a este DFD, é dado a seguir.

| Objetos | Operações Básicas |
|---|---|
| Modelo Estrutural | Modelar a estrutura |
| | Armazenar modelo na estrutura de dados |
| Análise Estrutural | Acessar modelo estrutural |
| | Analisar a estrutura |
| | Aplicar métodos de análise numérica |
| Análise Numérica | Solucionar sistemas de equações e problemas de autovalor |
| Métodos de Simulação | Simular a estrutura |

Tabela 7.1: Objetos e operações básicas.

- Características da Estrutura = modelo por elementos finitos.
- Modelo por Elementos Finitos = incidência + coordenadas nodais + elementos finitos + carregamento + propriedades físicas + condições de contorno.
- Dados do Modelo = características do modelo armazenadas na estrutura de dados do programa.

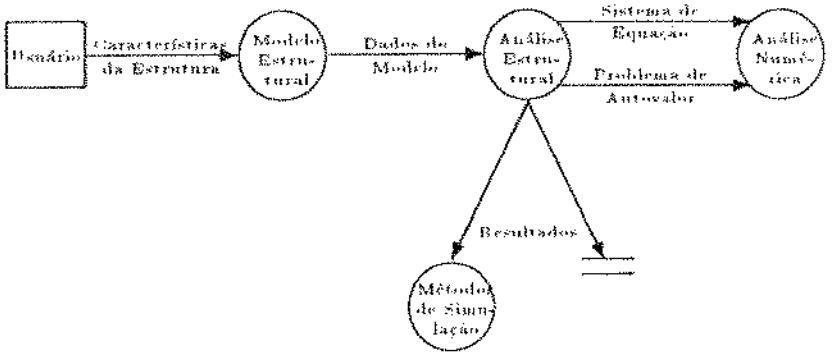


Figura 7.1: Diagrama de fluxo de dados.

- Sistema de Equação = conjunto de equações satisfeitas simultaneamente.
- Problemas de Autovalor = equação característica + sistema de equação.
- Equação Característica = $([K] - \omega[M])\{U\}$.
- Resultados da Análise = [deslocamentos | frequências naturais + modos de vibração].
- Deslocamentos = $[ux \mid uy \mid uz]$.
- Frequências naturais = $[\omega_1 \mid \omega_2 \mid \ldots \mid \omega_n]$.
- Modos de vibração = $[\{U_1\} \mid \{U_2\} \mid \ldots \mid \{U_n\}]$.

Os objetos da Tabela 7.1 devem possuir uma estrutura bastante flexível, permitindo que diferentes métodos de modelamento, análise e simulação possam ser incorporados ao programa. Os métodos numéricos, para solução de sistemas de equações e problemas de autovalor, devem ser implementados de maneira eficiente, procurando utilizar apenas a memória central do computador.

A partir da especificação destes objetos gerais, considerou-se as características requeridas para esta primeira versão do programa FAE, como discutido na Seção 7.2.2. Aplicou-se então, o método de análise da Seção 7.3, identificando-se os objetos apresentados na Tabela 7.2 [43]. O diagrama da Figura 7.2 ilustra como estes objetos se relacionam aos conceitos representados pelos objetos da Tabela 7.1.

A estrutura de dados deve ser alocada dinamicamente, permitindo assim, utilizar apenas a memória necessária para o armazenamento e análise do modelo considerado. No entanto, este tipo de alocação exige um gerenciamento mais complexo do espaço de memória, devido a criação de um grande número de apontadores.

Utiliza-se o armazenamento em banda para as matrizes obtidas pelos vários processos de análise, juntamente com um algoritmo para redução da largura de banda. Pretende-se aplicar, neste caso, os conceitos de matrizes esparsas.

| Objeto | Operações Básicas | Atributos |
|---|---|---|
| Modelo por Elementos Finitos | modelar estrutura pelo MEF | |
| Modelo por Subestruturação | criar hierarquia de subestruturas | recursivamente |
| Modelo por Ciclo-simétrico | armazenar e manipular características da região fundamental | recursivamente |
| Análise por Elementos Finitos | obter equação de movimento | através de um processo de superposição |
| Análise por Subestruturação | empregar métodos de análise | recursivamente |
| Análise Ciclo-simétrica | transformação físico-cíclico | |
| | transformação cíclico-físico | |
| | condensação dos g.l. internos | |
| Elementos Finitos | montar matrizes de rigidez e de massa | |
| Subestruturas | armazenar as submatrizes de rigidez e massa | |
| Matrizes | efetuar operações algébricas | |
| | acessar e manipular os elementos | |
| Método de Deslocamento | condensar g.l. internos | ao longo do contorno |
| | obter matriz de rigidez de contorno | por processo de superposição |
| | determinar deslocamentos de contorno | |
| | calcular deslocamentos internos | de cada subestrutura |
| Síntese por Subestruturação | representar movimento através de vetores admissíveis | |
| | obter vetores melhorados admissíveis | iterativamente |
| | empregar método de resíduos ponderados | |
| Eliminação de Gauss | reduzir matriz dos coeficientes | |
| | aplicar processo de substituição | |
| Iteração por Subespaço | empregar métodos de Jacobi | |
| | sequência de Sturm e | |
| | eliminação de Gauss | |

Tabela 7.2: Objetos, operações e atributos da primeira versão.

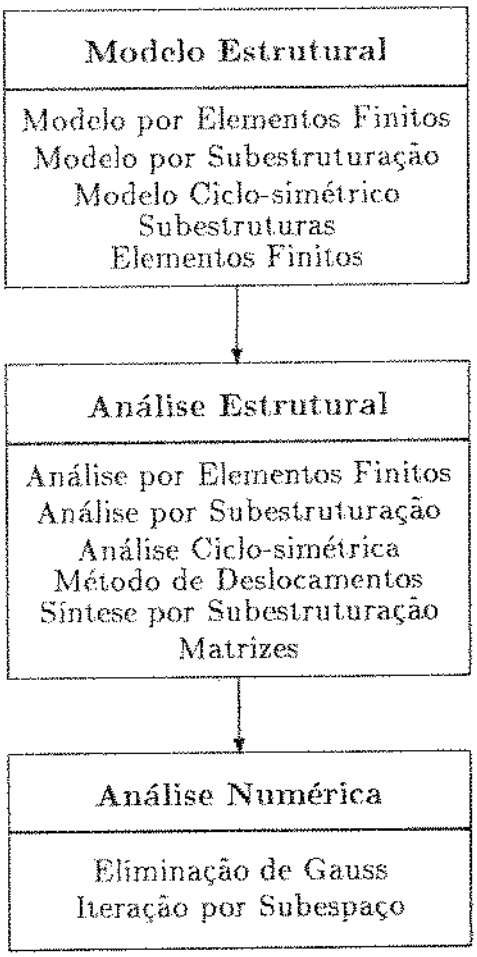


Figura 7.2: Relação entre os objetos identificados para a primeira versão e os objetos *Modelo Estrutural*, *Análise Estrutural* e *Análise Numérica*.

Para avaliar o desempenho do programa a ser implementado, deve-se efetuar alguns testes. Espera-se que os níveis de performance obtidos sejam comparáveis aos programas similares existentes. Assim, deve-se tomar uma estrutura reticulada e analisá-la através dos algoritmos implementados no programa.

Além disso, testes específicos e de consistência devem ser empregados a cada um dos objetos da Tabela 7.2. Estes testes serão especificados posteriormente para todos os objetos do sistema [44]. Da mesma maneira, testes de integridade para o programa como um todo devem ser aplicados, visando avaliar se os requisitos estabelecidos foram implementados satisfatoriamente.

Realizou-se então, a análise de requerimentos para os objetos da Tabela 7.2, confeccionando-se um documento individual para cada objeto [44]. O modelo destes documentos pode ser visto no Apêndice D, o qual apresenta o resultado da análise de requerimentos para o objeto *Matriz*.

Neste documento descreve-se, primeiramente, de maneira suscinta as características do objeto. Este texto é interpretado, identificando-se os atributos e operações do objeto, os quais são incluídos numa tabela. Utilizam-se os diagramas de Jackson da Figura 5.5 para representar graficamente os objetos e suas operações. Considera-se então, cada uma das operações, apresentando-se a sua definição, restrições, performance e restrições de projeto. Finalmente, especificam-se os critérios de validação para o objeto, considerando a performance desejada, os tipos de testes a serem empregados e a resposta esperada.

Ressalta-se que foram implementados alguns programas protótipos, procurando não apenas validar as técnicas de análise estrutural consideradas neste trabalho, mas também definir os seus requerimentos para a implementação no programa FAE. Esta metodologia seguiu os passos do paradigma de construção de protótipos, discutido no Capítulo 5.

## 7.5 Gramática para Especificação do Modelo Estrutural

As características do modelo estrutural a ser analisado devem ser especificadas através de um arquivo de comandos. Assim, torna-se necessário definir uma sintaxe gramatical para estes comandos. Esta sintaxe deve permitir uma especificação modular, possibilitando ainda, efetuar modificações de maneira simples e localizada.

Emprega-se uma sintaxe gramatical análoga àquela apresentada em [5], utilizando módulos com o seguinte formato geral:

```
[Módulo número/nome
  |
```

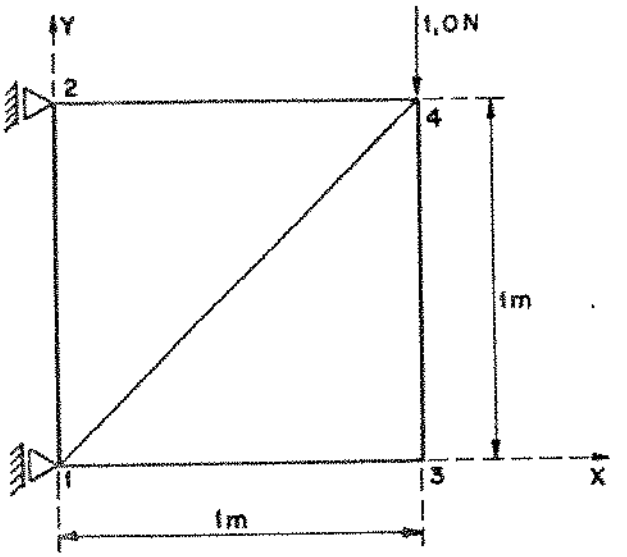


Figura 7.3: Estrutura reticulada especificada.

```
        cabeçalho
    ]
    [
        [registro de dados 1]
        [registro de dados 2]
        ⋮
        [registro de dados n]
    ]
]
```

onde

**Módulo** é o atributo do modelo estrutural a ser descrito.

**número/nome** é a denominação dada ao módulo ou o número de registro de dados.

A denominação *MÓDULO* pode ser vista como um operador que associa um nome com uma instância do objeto considerado. O cabeçalho é utilizado para descrever a estrutura dos atributos a serem especificados. Os colchetes ([,]) são usados como delimitadores, obtendo-se uma gramática não ambígua e livre de contexto.

Como exemplo, considere a seguinte especificação do modelo global para a estrutura ilustrada na Figura 7.3:

```
[MODELO TREL
```

```
[COMPONENTES 5
   [componente  nome]
      [
         [1  bar2]
         [2  bar2]
         [3  bar2]
         [4  bar2]
         [5  bar2]
      ]
]

[NOS 4
   [no  x  y]
      [
         [1  0.0 0.0]
         [2  0.0 1.0]
         [3  1.0 0.0]
         [4  1.0 0.0]
      ]
]

[INCIDENCIA 5
   [componente  incidencia]
      [
         [1  1 2]
         [2  1 3]
         [3  1 4]
         [4  2 4]
         [5  3 4]
      ]
]

[RESTRICOES 2
   [no  dx  dy]
      [
         [1  1 1]
         [2  0 1]
      ]
]
```

```
   [MATERIAL
      [E  A]
         [
            [2.1e5  0.0]
         ]
   ]

   [CARREGAMENTO 1
      [no  fx  fy]
         [
            [1  0.0 -1.0]
         ]
   ]
]

[ANALISE
   [GLOBAL  ESTATICA]
]
```

O módulo *MODELO* representa as características do modelo estrutural, sendo, portanto, constituído dos demais módulos, tais como *NÓS*, *INCIDÊNCIA*, *COMPONENTES*, *MATERIAL*, *NOS_CONTORNO*, *CARREGAMENTO*, *RESTRIÇÕES*, dentre outros. O atributo *TREL* é a denominação dada ao modelo em estudo. Alguns nomes são reservados para designar os elementos finitos implementados no programa, assim como as propriedades físicas e geométricas, sendo estes rótulos mostrados nas Tabelas 7.3 e 7.4, respectivamente.

| Designação | Elemento Finito |
|---|---|
| bar2 | barra plana |
| bar3 | barra espacial |
| viga2 | viga plana |
| viga3 | viga espacial |

Tabela 7.3: Denominações atribuídas aos elementos finitos.

Os graus de liberdade restritos são indicados pelo código 1 e os livres por 0. O módulo *ANÁLISE* especifica o tipo de análise a ser efetuada, estando disponíveis as análises estática e dinâmica global, por subestruturação e cíclica, indicadas, respectivamente, pelos rótulos *GLOBAL*, *SUBST* e *CICLICA*.

| Designação | Propriedade |
|---|---|
| $A$ | área da seção transversal |
| $E$ | módulo de elasticidade longitudinal |
| $mu$ | densidade |
| $nu$ | coeficiente de Poisson |
| $ix$, $iy$, $iz$ | momentos de inércia |

Tabela 7.4: Denominações utilizadas para especificação das propriedades físicas.

Observa-se portanto, a característica modular da gramática empregada. Os atributos do modelo estrutural e o tipo de análise a ser efetuada são especificados por módulos individuais. Desta forma, qualquer alteração do modelo pode ser realizada de maneira simples e localizada. Para o modelamento por subestruturação, tem-se o módulo *NOS-CONTORNO* para indicar os nós de contorno da subestruturas. Ressalta-se que cada subestrutura é especificada em relação a um sistema local de referência, sendo necessário efetuar uma transformação de coordenadas ao se passar para outro nível de subestruturação. Uma discussão mais detalhada desta gramática pode ser encontrada em [44].

# Comentários Finais

Os algoritmos de análise estrutural, apresentados neste trabalho, reduzem significativamente a tarefa de preparação do modelo discreto de uma estrutura. Além disso, através da implementação eficiente deste métodos, pode-se diminuir consideravelmente a demanda computacional envolvida na análise estática e/ou dinâmica do modelo estrutural.

A técnica de subestruturação consiste na divisão da estrutura em várias partes, as quais podem ser analisadas separadamente. No final, obtém-se a solução para a estrutura global através da manipulação dos resultados determinados para as várias subestruturas. Assim, grupos distintos podem analisar cada uma das partes de maneira independente. Além disso, numa reanálise do modelo, efetuam-se apenas modificações locais nas subestruturas consideradas. Ressalta-se que a formulação, apresentada no Capítulo 2, é exata, consistindo numa condensação estática das variáveis internas ao longo do contorno.

A recursividade do algoritmo por subestruturação em vários níveis simplifica a sua implementação, devido ao tratamento uniforme dos elementos presentes na hierarquia especificada para o modelo estrutural. Observa-se que esta técnica em vários níveis pode ser aplicada, de maneira similar, em uma análise empregando elementos finitos hierárquicos. Cada grau polinomial acrescentado às funções de forma dos elementos finitos, implica no acréscimo de uma linha e uma coluna na matriz de rigidez dos elementos. Pode-se tratar este grau polinomial de maneira análoga a um nível superior de subestruturação do modelo estrutural.

Várias estruturas mecânicas podem ser modeladas utilizando uma concepção ciclosimétrica. Assim, os métodos para análise estática e dinâmica, discutidos nos Capítulos 3 e 4, são muito úteis no estudo de problemas práticos. A aplicação das transformações, obtidas a partir da série de Fourier discreta, permite estudar uma estrutura através da solução de vários sistemas de equações ou problemas de autovalor de ordens reduzidas.

Na análise estática, através da condensação das variáveis internas, foi possível reduzir a ordem do modelo discreto da região fundamental. No entanto, não se efetuou esta redução na análise dinâmica, pois neste caso seriam introduzidas aproximações no cálculo das frequências naturais e modos de vibração da estrutura. Observa-se que as

transformações apresentadas possuem uma característica geral, podendo ser aplicadas em outros tipos de análise, como por exemplo no estudo de flambagem de uma estrutura ciclo-simétrica.

No Capítulo 4, apresentou-se, de maneira suscinta, as características gerais de alguns métodos de subestruturação dinâmica. Torna-se interessante implementar, em programas protótipos, os métodos onde não existe a necessidade da resolução de um problema de autovalor para cada subestrutura [24,27], visando avaliar a aplicação destes algoritmos.

Os conceitos de Engenharia de Programas, abordados no Capítulo 5, introduzem novas metodologias para o desenvolvimento sistemático e confiável de programas. Os requerimentos dos algoritmos de análise estrutural foram obtidos utilizando a filosofia do paradigma de construção de protótipos. Além disso, a confecção de documentos, em todas as fases de implementação, permitiu identificar e registrar as necessidades gerais e requerimentos dos vários elementos do programa. Da mesma forma, representou-se de maneira objetiva o domínio de informação considerado para os programas.

A principal característica do modelo orientado por objetos é possibilitar a construção de módulos interdependentes que se comunicam através de interfaces bem definidas. Assim, espera-se garantir a integridade dos objetos ao longo da hierarquia do programa. Um outro aspecto do modelo por objetos está relacionada à utilização dos conceitos de modulação, abstração de dados e mecanismo de herança, visando o reaproveitamento de código.

Portanto, o principal objetivo da aplicação dos conceitos oriundos da Engenharia de Programas foi obter um método sistemático para o desenvolvimento do programa FAE, identificando de maneira clara e uniforme os vários aspectos da primeira versão do programa. Através do modelo por objetos, procurou-se obter uma representação mais natural das várias características do programa. Neste caso, a aplicação do método de análise de requisitos, discutido no Capítulo 7, foi importante, devido a sua simplicidade e objetividade. Considera-se que os resultados obtidos foram satisfatórios.

Observa-se que a aplicação do modelo por objetos, para a implementação da técnica de subestruturação em vários níveis, permite explorar, de maneira natural, as arquitetutas paralelas de alguns computadores.

Visando a continuidade do trabalho, pretende-se realizar, a médio prazo, as seguintes atividades:

- utilizar os conceitos de ciclo-simetria para a determinação da resposta em frequência da estrutura.
- implementar os métodos de subestruturação dinâmica discutidos em [24,27].
- avaliar e revisar os vários documentos confeccionados para o programa FAE.

- realizar o projeto, a codificação e os testes para os objetos da primeira versão do programa FAE.

# Bibliografia


[1] **F.C. Nelson**. *Substructure Analysis of Vibrating Systems*. Shock and Vibration Digest, Vol.11, pp.3-9, 1979.

[2] **O. Egeland e P.O. Araldsen**. *SESAM-69 - A General Purpose Finite Element Method Program*. Computers & Structures, Vol.4, pp.41-68, 1974.

[3] **H. Petersson e E.P. Popov**. *Substructuring and Equation System Solutions in Finite Element Analysis*. Computers & Structures, Vol.7, pp.197-206, 1977.

[4] **T. Furuike**. *Computerized Multiple Level Substructuring Analysis*. Computers & Structures, Vol.2, pp.1063-1073, 1972.

[5] **A.L. Hale e L.V. Warren**. *Concepts of a General Substructuring System for Structural Dynamic Analysis*. Transactions of the ASME - Journal of Vibration, Acoustics, Stress and Reliability in Design, Vol.107, pp.2-12, 1985.

[6] **J.S. Przemieniecki**. *Theory of Matrix Structural Analysis*. McGraw Hill, 1968

[7] **A.K. Noor, H.A. Kamel e R.E. Fulton**. *Substructuring Techniques - Status and Projections*. Computers & Structures, Vol.8, pp.621-632, 1978.

[8] *ANSYS - PC/LINEAR 4.3 - Reference Manual*. Swanson Analysis Systems, Inc., 1988.

[9] **C.L. Fortescue**. *Method of Symmetrical Co-ordinates Applied to the Solution of Polyphase Networks*. Transactions AIEE, Vol.37, 1918.

[10] *MSC/NASTRAN - Aplication Manual*. The MacNeal-Schwendler Corporation, 1973.

[11] **H. Vold, H. Knapp e P. Hermann**. *ASKA: CS - Cyclic Symmetry: User's Guide ASKA UM 219*. University of Stuttgart, 1977.

[12] **A.L. Serpa, J.V. Ferreira e F. Iguti**. *ANAFIN - Manual do Usuário*. Campinas, UNICAMP/FEC/DPM/GEPROM, Versão 1.0, 1988.

[13] **L. Meirovitch.** *Analytical Methods in Vibration*. Macmillan Company, 1967.

[14] **L. Meirovitch.** *Elements of Vibration Analysis*. McGraw Hill, 1975.

[15] **K.J. Bathe e E.L. Wilson.** *Numerical Methods in Finite Element Analysis*. Prentice Hall, 1976.

[16] **D.L. Thomas.** *Dynamics of Rotationally Periodic Structures*. International Journal for Numerical Methods in Engineering, Vol.14, pp.81-102, 1979.

[17] **R.J. Guyan.** *Reduction of Stiffness and Mass Matrices*. AIAA Journal, Vol.3, No.2, pp.380, 1965.

[18] **L. Meirovitch.** *Computational Methods in Structural Dynamics*. Stijthoff & Noordhoff, 1980.

[19] **G.M.L. Gladwell.** *Branch Mode Analysis of Vibrating Systems*. Journal of Sound and Vibration, Vol.1, pp.41-59, 1964

[20] **W.C. Hurty.** *Dynamic Analysis of Structural System Using Component Modes*. AIAA Journal, Vol.3, No.4, pp.678-685, 1965.

[21] **R.R. Craig Jr. e M.C.C. Bampton.** *Coupling of Substructures for Dynamic Analyses*. AIAA Journal, Vol.6,No.7, pp.1313-1319, 1968.

[22] **W.A. Benfield e R.F. Hruda.** *Vibration Analysis of Structures by Component Mode Substitution*. AIAA Journal, Vol.9, No.7, pp.1255-1261, 1971.

[23] **R.M. Hintz.** *Analytical Methods in Component Modal Synthesis*. AIAA Journal, Vol.13, No.8, pp.1007-1016, 1975.

[24] **J.S. Arora e D.T. Nguyen.** *Eigensolution for Large Structural Systems with Substructures*. Journal for Numerical Methods in Engineering, Vol.15, pp.333-341, 1980.

[25] **A.L. Hale e L. Meirovitch.** *A General Substructuring Synthesis Method for the Dynamic Simulation of Complex Structures*. Journal of Sound and Vibration, Vol.69, No.2, pp.309-326, 1980.

[26] **A.L. Hale e L. Meirovitch.** *A General Dynamic Synthesis for Structures with Discrete Substructures*. Journal of Sound and Vibration, Vol.85, No.4, pp.445-457, 1982.

[27] **A.L. Hale e L. Meirovitch.** *A Procedure for Improving Discrete Substructure Representation in Dynamic Synthesis*. AIAA Journal, Vol.20, No.8, 1982.

[28] **R.S. Pressman.** *Software Engineering - A Practitioner's Approach.* McGraw-Hill. 1987.

[29] **T. Rentsch.** *Object Oriented Programming.* SIGPLAN Notices, Vol.17, No.9, pp.51-57, 1982.

[30] **D. Robson e A. Goldberg.** *The Smalltalk-80 System.* Byte, pp.74-86, August 1981.

[31] **A. Goldberg e D. Robson.** *Smalltalk-80: The Language and its Implementation.* Addison-Wesley, New York, 1983.

[32] **B. Cox.** *Object-Oriented Programming - An Evolutionary Approach.* Addison-Wesley, 1986.

[33] **B.W. Kernighan e D.M. Ritchie.** *C - A Linguagem de Programação.* Editora Campus, 1986.

[34] **B. Stroustrup.** *The C++ Programming Language.* Addison-Wesley, 1986.

[35] **J. Jacky e I. Kalet.** *An Object-Oriented Programming Discipline for Standard Pascal.* Communications of the ACM, Vol.30, No.9, pp.772-776, 1987.

[36] **B. Liskov e J. Guttag.** *Abstraction and Specification in Program Development.* MIT Press. 1986.

[37] **D. Robson.** *Object-Oriented Software Systems.* Byte, pp.74-86, August 1981.

[38] **G.A. Pascoe.** *Elements of Object-Oriented Programming.* Byte, pp.74-86, August 1986.

[39] **T.C. Jones.** *Reusability in Programming : A Survey of the State of the Art.* IEEE Transactions on Software Engineering, Vol.SE-10, No.5, pp.488-494, September 1984.

[40] **L. Ledbetter e B. Cox.** *Software Ics.* Byte, Vol.10, No.6, June 1985.

[41] **B. Meyer.** *Reusability: The Case for Object-Oriented Design.* IEEE Software, pp.50-64, March 1987.

[42] **M.L. Bittencourt.** *Especificação de Sistema.* Documentação do Programa FAE. UNICAMP/FEM/DPM, Fevereiro, 1990.

[43] **M.L. Bittencourt.** *Análise de Requerimentos.* Documentação do Programa FAE. UNICAMP/FEM/DPM, Fevereiro, 1990.

[44] **M.L. Bittencourt.** *Objetos do Sistema: Análise de Requerimentos, Projeto, Codificação e Testes.* Documentação do Programa FAE. UNICAMP/FEM/DPM, Fevereiro, 1990.

# Apêndice A

# Matriz de Rotação em Torno do Eixo de Simetria

Considere o vetor $\vec{x}$ com coordenadas $(u, v, 1)$ no sistema de referência $x_1y_1z_1$, como indicado na Figura A.1. Deseja-se expressar este vetor no sistema de referência $x_2y_2z_2$, o qual está girado de um ângulo $\theta$, no sentido anti-horário, em relação ao sistema $x_1y_1z_1$. Escreve-se o vetor $\vec{x}$ na seguinte forma,

$$\vec{x} = u\vec{x}_1 + v\vec{y}_1 + 1\vec{z}_1 = \begin{bmatrix} u & v & 1 \end{bmatrix} \begin{Bmatrix} \vec{x}_1 \\ \vec{y}_1 \\ \vec{z}_1 \end{Bmatrix} \tag{A.1}$$

onde $\vec{x}_1$, $\vec{y}_1$ e $\vec{z}_1$ são os versores nas direções dos eixos $x_1$, $y_1$ e $z_1$, respectivamente.

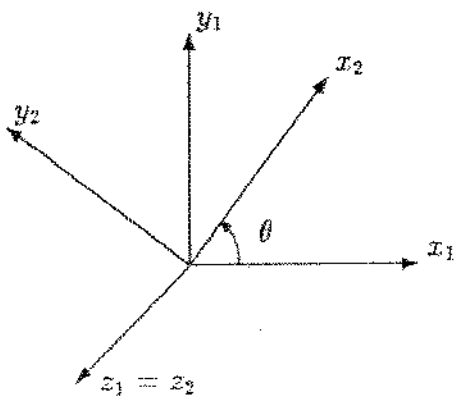


Figura A.1: Sistemas de referência $x_1y_1z_1$ e $x_2y_2z_2$.

Por sua vez os versores $\vec{x}_1$, $\vec{y}_1$ e $\vec{z}_1$ estão relacionados aos versores $\vec{x}_2$, $\vec{y}_2$ e $\vec{z}_2$ através

das relações,

$$\begin{aligned} \vec{x}_1 &= \cos\theta\vec{x}_2 - \mathrm{sen}\theta\vec{y}_2 \\ \vec{y}_1 &= \mathrm{sen}\theta\vec{x}_2 + \cos\theta\vec{y}_2 \\ \vec{z}_1 &= \vec{z}_2 \end{aligned} \tag{A.2}$$

ou em forma matricial,

$$\left\{ \begin{array}{c} \vec{x}_1 \\ \vec{y}_1 \\ \vec{z}_1 \end{array} \right\} = \left[ \begin{array}{ccc} \cos\theta & -\mathrm{sen}\theta & 0 \\ \mathrm{sen}\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{array} \right] \left\{ \begin{array}{c} \vec{x}_2 \\ \vec{y}_2 \\ \vec{z}_2 \end{array} \right\} \tag{A.3}$$

Substituindo (A.3) em (A.1), vem que,

$$\vec{r} = (\cos\theta - \mathrm{sen}\theta)u\vec{x}_2 + (\cos\theta - \mathrm{sen}\theta)v\vec{y}_2 + \vec{z}_2$$

Portanto, a matriz de rotação de um ângulo $\theta$ em torno do plano $x_1y_1$, no sentido anti-horário, é dada por,

$$[T] = \left[ \begin{array}{cc} \cos\theta & -\mathrm{sen}\theta \\ \mathrm{sen}\theta & \cos\theta \end{array} \right] \tag{A.4}$$

Observa-se que a matriz de rotação no sentido horário é a transposta da matriz $[T]$.

# Apêndice B

# Síntese por Subestruturação

## B.1 Síntese por Subestruturação para Sistemas Discretos

Neste caso, supõe-se que uma estrutura é consituída de $m$ subestrutuuras discretas. O modelo discreto para cada subestrutura possui um número de graus de liberdade elevado, criando dificuldades na solução do problema de autovalor para a estrutura global.

Considerando-se que cada subestrutura $s\,(s = 1, \ldots, m)$ se comporta independentemente das outras, as energias cinética e potencial têm a seguinte forma matricial, respectivamente, [26]

$$T_s = \frac{1}{2}\{\dot{u}_s\}^T[m_s]\{\dot{u}_s\}, \quad V_s = \frac{1}{2}\{u_s\}^T[k_s]\{u_s\} \tag{B.1}$$

onde $[m_s]$ e $[k_s]$ são as matrizes positivas definidas de massa e rigidez, respectivamente, e $\{u_s\}$ é o vetor de coordenadas generalizadas de dimensão $n_s$.

De acordo com o método de Rayleigh-Ritz, assume-se que o vetor de coordenadas generalizadas $\{u_s\}$ é aproximado por uma combinação linear dos vetores de Ritz para a subestrutura multiplicados por coordenadas generalizadas dependentes do tempo, ou seja,

$$\{u_s\} = \sum_{i=1}^{N_s} \phi_{si}\zeta_{si}(t) = [\Phi_s]\{\zeta_s\} \tag{B.2}$$

onde $[\Phi_s]$ é a matriz dos vetores de Ritz de ordem $n_s \times N_s$, e $\{\zeta_s\}$ é um vetor de dimensão $N_s$ representando as coordenadas generalizadas reduzidas. Substituindo (B.2) em (B.1), tem-se que,

$$T_s = \frac{1}{2}\{\dot{\varsigma}_s\}^T[M_s]\{\dot{\varsigma}\}_s, \quad V_s = \frac{1}{2}\{\varsigma_s\}^T[K_s]\{\varsigma_s\} \tag{B.3}$$

onde,

$$[M_s] = [\Phi_s]^T[m_s][\Phi_s], \quad [K_s] = [\Phi_s]^T[k_s][\Phi_s]$$

são as matrizes reduzidas de massa e rigidez para a subestrutura $s$, respectivamente, ambas de ordem $N_s \times N_s$. O quociente de Rayleigh para a estrutura completa, constituída de $m$ subestruturas independentes, é dado por [26],

$$R_d = \frac{\{\varsigma_d\}^T[K_d]\{\varsigma_d\}}{\{\varsigma_d\}^T[M_d]\{\varsigma_d\}} \tag{B.4}$$

onde $\{\varsigma_d\} = \{\{\varsigma_1\}^T \{\varsigma_2\}^T \ldots \{\varsigma_m\}^T\}^T$ é o vetor de configuração de dimensão $N = \sum_{s=1}^{m} N_s$; $[M_d] = \sum_{s=1}^{m}[M_s]$ e $[K_d] = \sum_{s=1}^{m}[K_s]$ $(s = 1, \ldots, m)$ são as matrizes de massa e rigidez da estrutura sem a aplicação das condições de contorno entre as subestruturas.

Deve-se satisfazer a compatibilidade geométrica entre os contornos internos comuns às subestruturas. Assim, tomando-se duas subestruturas adjacentes $r$ e $s$, tem-se que,

$$\{u_s\}_I = \{u_r\}_I \quad r, s = 1, 2, \ldots, m \quad s \neq r \tag{B.5}$$

onde $\{\}_I$ denota a parte do vetor de coordenadas generalizadas correspondentes aos deslocamentos e rotações nos pontos nodais comuns a $r$ e $s$. A partir destas restrições, obtém-se a seguinte relação entre o vetor de coordenadas generalizadas independentes $\{\varsigma\}$, de dimensão $n$, e o vetor $\{\varsigma_d\}$ [26],

$$\{\varsigma_d\} = [C]\{\varsigma\} \tag{B.6}$$

onde $[C]$ é uma matriz retangular $N \times n$.

Aplicando-se a equação de restrição (B.6) ao quociente de Rayleigh dado em (B.4), determina-se o seguinte problema de autovalor para a estrutura global,

$$[K][U] = [M][U][\Lambda] \tag{B.7}$$

onde,

$$[K] = [C]^T[K_d][C] \text{ e } [M] = [C]^T[M_d][C]$$

são as matrizes $n \times n$ de massa e rigidez da estrutura com as condições de compatibilidade nos contornos das subestruturas já aplicadas; $[U]$ é a matriz dos autovetores e $[\Lambda]$ são os autovalores da estrutura.

Para satisfazer as condições de compatibilidade geométricas (B.5), utiliza-se o método de Resíduos Ponderados na seguinte forma [26],

$$\{g_{rs}\}_i^T(\{u_s\} - \{u_r\})_I = \{0\} \quad r,s = 1,2,\ldots,m \quad s \neq r \tag{B.8}$$

onde $\{g_{rs}\}_i \quad (i = 1,2,\ldots,m_{rs})$ são os vetores de ponderação, e $m_{rs}$ é o número de coordenadas generalizadas no contorno interno entre as subestruturas $r$ e $s$. No entanto, requer-se que as condições de compatibilidade (B.8) sejam satisfeitas aproximadamente através de $M_{rs}$ vetores $\{g_{rs}\}_i \quad (i = 1,2,\ldots,M_{rs})$, onde $M_{rs} \ll m_{rs}$.

Define-se então, o conceito de estrutura intermediária, como àquela onde as condições de compatibilidade entre as subestruturas são satisfeitas de maneira exata apenas em algumas variáveis de contorno interno. O método de Rayleigh-Ritz é válido para a estrutura intermediária, verificando-se que os autovalores obtidos são inferiores àqueles da estrutura real. Portanto, o quociente de Rayleigh não fornece, necessariamente, limitantes superiores para os autovalores.

Outros aspectos do método, tais como a seleção dos vetores admissíveis e de ponderação, assim como os critérios de convergência, são apresentados em [25,26].

## B.2 Procedimento Iterativo para a Síntese Discreta

Considerando uma estrutura constituída de $m$ subestruturas discretas, a precisão da solução do problema de autovalor para a estrutura intermediária depende do conjunto de vetores admissíveis utilizado. A aproximação pode ser melhorada aumentando-se o número de vetores admissíveis usados para representar o movimento de cada subestrutura. Uma outra alternativa é melhorar o conjunto de vetores admissíveis, até atingir a precisão desejada, através de um processo iterativo, utilizando-se assim, o mesmo número de vetores. Este procedimento é discutido a seguir supondo as matrizes de massa e rigidez das subestruturas positivas definidas.

O problema de autovalor para uma subestrutura $s$ $(s = 1,\ldots,m)$ é descrito pela equação,

$$[k_s]\{u_s\} = \{\eta_s\} + \lambda[m_s]\{u_s\} \tag{B.9}$$

onde $\{\eta_s\}$ é um vetor de dimensão $n_s$ representando as forças exercidas por todas subestruturas adjacentes à subestrutura $s$. Logo, para cada subestrutura adjacente $r$, tem-se que,

$$\{\eta_s\}_{Ir} = \{\eta_{rs}\} \quad r,s = 1,2,\ldots,m \quad r \neq s \tag{B.10}$$

onde $\{\eta_s\}_{Ir}$ denota as entradas no vetor $\{\eta_s\}$ correspondentes às coordenadas generalizadas nos pontos do contorno interno $S_{rs}$, e $\{\eta_{rs}\}$ é um vetor desconhecido representando as forças exercidas pela subestrutura $r$ no contorno interno $S_{rs}$. Os dois vetores possuem dimensão $m_{rs}$.

Como a matriz de rigidez $[k_s]$ é positiva definida, a equação (B.9) admite a seguinte solução única,

$$\{u_s\} = \{f_s\} + \lambda[A_s]\{u_s\} \tag{B.11}$$

onde,

$$\{f_s\} = [k_s]^{-1}\{\eta_s\} = \sum_{r=1}^{m} [K_s]_{Ir}^{-1}\{\eta_{rs}\}, \quad [A_s] = [k_s]^{-1}[m_s] \tag{B.12}$$

onde $[k_s]_{Ir}^{-1}$ é uma matriz retangular $n_s \times m_{rs}$ formada pelas colunas de $[k_s]^{-1}$ que multiplicam as coordenadas generalizadas correspondentes aos pontos de contorno interno entre as subestruturas $r$ e $s$ $(r, s = 1, \ldots, m \quad r \neq s)$.

Seguindo o método de Síntese por Subestruturação, determina-se o quociente de Rayleigh para a estrutura completa a partir da união de $m$ subestruturas. Logo,

$$\lambda = R = \frac{\sum_{s=1}^{m} \{u_s\}^T [k_s]\{u_s\} - \sum_{s=1}^{m} \{u_s\}^T \{\eta_s\}}{\sum_{s=1}^{m} \{u_s\}^T [m_s]\{u_s\}} \tag{B.13}$$

Considerando a equação (B.10), vem que,

$$\lambda = R = \frac{\sum_{s=1}^{m} \{u_s\}^T [k_s]\{u_s\} - \sum_{s=1}^{m} \sum_{r=s+1}^{m} [\{u_s\}_{Ir} - \{u_r\}_{Is}]^T \{\eta_{rs}\}}{\sum_{s=1}^{m} \{u_s\}^T [m_s]\{u_s\}} \tag{B.14}$$

onde os pontos do contorno $S_{sr}$ são os mesmos de $S_{rs}$, e, portanto, $\{\eta_{rs}\} = -\{\eta_{sr}\}$.

As condições de compatibilidade ao longo dos contornos são dadas pela equação (B.8), podendo ser descritas como,

$$\{\eta_{rs}\}^T [\{u_s\}_{Ir} - \{u_r\}_{Is}] = \{0\} \quad r, s = 1, 2, \ldots, m \quad r \neq s \tag{B.15}$$

a qual deve ser satisfeita para todos os vetores de força $\{\eta_{rs}\}$ possíveis, anulando-se, assim, a somatória dupla em (B.14). Na síntese discreta, os vetores $\{\eta_{rs}\}$ são aproximados por,

$$\{\eta_{rs}\} = \sum_{i=1}^{M_{rs}} a_{rsi} \{g_{rs}\}_i \quad r, s = 1, 2, \ldots, m \tag{B.16}$$

onde $a_{rsi}$ são coeficientes desconhecidos, $\{g_{rs}\}_i$ são vetores linearmente independentes e $M_{rs} \ll m_{rs}$. Substituindo (B.16) em (B.15), vem que,

$$\{g_{rs}\}_i^T [\{u_s\}_{Ir} - \{u_r\}_{Is}] = \{0\} \quad r, s = 1, 2, \ldots, m \quad r \neq s \tag{B.17}$$

A equação (B.17) representa a satisfação das condições de compatibilidade geométricas pelo método de Resíduos Ponderados. Os vetores de ponderação $\{g_{rs}\}_i \quad (i = 1, \ldots, M_{rs})$ são utilizados como uma base de vetores para a expansão em série do vetor de força agindo no contorno interno $S_{rs}$, entre as subestruturas $r$ e $s$. A expressão (B.16) é exata para a estrutura intermediária, sendo aproximada para a estrutura real [27].

O vetor $\{f_s\}$, definido na equação (B.12), representa a resposta estática da subestrutura $s$, resultante das condições impostas nos contornos internos $S_{rs}$ $(r, s = 1, 2, \ldots, m;\ r \neq s)$. Substituindo (B.16) em (B.12), vem que,

$$\{f_s\} = \sum_{r=1}^{m} \sum_{i=1}^{M_{rs}} a_{rsi} [k_s]_{Ir}^{-1} \{g_{rs}\}_i = \sum_{i=1}^{M_{sc}} a_{si} \{F_s\}_i \tag{B.18}$$

sendo esta equação válida para as subestruturas positivas definidas, onde $M_{sc} = \sum_{r=1}^{m} M_{rs}$ é o número total de vetores de ponderação associados à subestrutura $s$. Os coeficientes $a_{si} \quad (i = 1, 2, \ldots, M_{sc})$ são valores desconhecidos, e $\{F_s\}_i \quad (i = 1, 2, \ldots, M_{sc})$ são os vetores definidos como a resposta estática resultante da aplicação de forças nos pontos de contorno interno.

Portanto, pode-se definir a iteração por subespaço para a estrutura intermediária como uma maneira de se produzir, implicitamente ou explicitamente, um operador global $[A] = [K]^{-1}[M]$ [27].

O operador implícito pode ser obtido, inicialmente, utilizando-se o operador $[A_s]$ da equação (B.11) para gerar vetores melhorados para a subestrutura, determinando-se então, os coeficientes $a_{si}$ satisfazendo as condições de compatibilidade (B.17). Logo, tem-se um processo iterativo onde as combinações lineares dos vetores estáticos $\{F_s\}_i$ são determinados automaticamente, com propriedades de convergência similares a iteração por subespaço. Além disso, os vetores melhorados para cada subestrutura podem ser produzidos de maneira independente [27].

Na síntese discreta, o movimento de cada subestrutura $s$ é representado por um conjunto de $N_s$ vetores admissíveis linearmente independentes $\{\phi_s\}_i \quad (i = 1, 2, \ldots, N_s)$. O processo iterativo baseia-se na escolha de um conjunto específico de vetores admissíveis $[\phi_s^p]$ a cada iteração $p$, baseados na equação (B.11). Portanto, cada subestrutura $s$ é representada pela seguinte soma de $N_s = M_{sc} + q$ vetores admissíveis na iteração $p$,

$$\{u_{s,p}\} = \sum_{i=1}^{N_s} \zeta_{si} \{\phi_s^p\}_i = \sum_{i=1}^{M_{sc}} \zeta_{si} \{f_s\}_i + \sum_{i=M_{sc}+1}^{N_s} \zeta_{si} \{\phi_s^p\}_i \tag{B.19}$$

onde $\zeta_{si} \quad (i = 1, 2, \ldots, N_s)$ são coeficientes desconhecidos a serem determinados pela síntese.

Observa-se que os primeiros $M_{sc}$ vetores admissíveis em (B.19) são os vetores de resposta estática $\{F_s\}_i \quad (i = 1, 2, \ldots, M_{sc})$ definidos em (B.18). Os demais $q$ vetores

admissíveis, linearmente independentes e arbitrários, são tomados durante a iteração inicial $p = 0$, sendo independentes dos vetores estáticos $\{F_s\}_i$. Para efeitos de simplificação, seleciona-se o mesmo número $q$ para todas as subestruturas.

A cada iteração $p = 0, 1, \ldots$, tem-se uma solução aproximada para a estrutura intermediária, produzida pela síntese discreta apresentada na seção anterior.

A precisão das estimativas dos primeiros $q$ autovalores e autovetores associados, obtidos na iteração $p$, pode ser melhorada utilizando-se, para cada subestrutura $s$, $q$ vetores admissíveis determinados a partir da seguinte equação,

$$\{\phi^{p+1}_{s,N_{sc}+i}\} = [A_s]\{u^i_{s,p}\} \quad i = 1, 2, \ldots, q \tag{B.20}$$

onde os vetores $\{u^i_{s,p}\}$ $(i = 1, 2, \ldots, q)$ são determinados na iteração precedente.

O processo de refinamento indicado por (B.20) pode ser executado em paralelo para todas as subestruturas. Observa-se que apenas os últimos $q$ vetores admissíveis são alterados a cada iteração, enquanto os primeiros $N_s - q$ são sempre tomados como os vetores de resposta estática da subestrutura.

Portanto, este procedimento pode ser resumido nos seguintes passos [27]:

- selecionar $q$ vetores admissíveis para a subestrutura $\{\phi^0_{si}\}$ $(i = N_{sc} + 1, \ldots, N_{sc} + q; s = 1, \ldots, m)$.
- usar os vetores estáticos $\{F_s\}_i$ $(i = 1, \ldots, N_{sc})$ e os vetores admissíveis $\phi^p_{si}$ $(i = N_{sc} + 1, \ldots, N_{sc} + q)$ na equação (B.19) para representar cada subestrutura $s$; calcular $[\Lambda^p_i]$ e $\{u^{(r)}_{s,p}\}$ $(r = 1, \ldots, n)$ pela resolução da equação (B.7).
- se os autovalores e autovetores associados convergirem, termina-se o processo iterativo. Caso contrário, aplica-se a equação (B.20) para produzir vetores admissíveis melhorados $\{\phi^{p+1}_s\}_i$ $(i = N_{sc} + 1, \ldots, N_{sc} + q)$ para cada subestrutura $s$ e retorna-se ao passo 2 com $p = p + 1$.

Assim, este processo iterativo converge para os primeiros $q$ autovalores não nulos da estrutura intermediária.

# Apêndice C

# Linguagens Orientadas por Objetos

## C.1 Introdução

O conceito de programação orientada por objetos possui várias definições. Assim, é natural que existam várias linguagens de programação denominadas orientadas por objetos, embora as características utilizadas e o modo de implementá-las não sejam as mesmas.

Segundo Pascoe [38], uma linguagem deve possuir os seguintes elementos para suportar a programação por objetos :

- ocultamento de informação
- abstração de dados
- ligação dinâmica
- mecanismo de herança.

Nas seções seguintes serão descritas, de maneira suscinta, as principais características das linguagens *Smalltalk-80* e *Objective-C*, procurando identificar os conceitos de programação por objetos utilizados e como são implementados.

## C.2 Smalltalk-80

*Smalltalk-80* é um sistema com uma linguagem e um ambiente de programação integrados. É o resultado de uma década de pesquisas visando permitir a criação de programas de alta funcionalidade e interativos [31].

Uma das partes do sistema *Smalltalk-80* é a sua *máquina virtual*, a qual provê o armazenamento orientado por objetos, processamento através de mensagens e a interação gráfica com o usuário.

O sistema *Smalltalk-80* introduz apenas cinco conceitos principais: *objeto*, *mensagem*, *classe*, *herança* e *métodos*. Isto se deve à uniformidade com que o modelo por objetos é tratado no sistema. Assim, todos os componentes do sistema são considerados objetos, possibilitando uma mesma consistência em todas as partes do sistema. Portanto, o usuário necessita assimilar apenas o conceito de objeto para explorar qualquer parte do sistema *Smalltalk-80*.

### C.2.1 A Linguagem de Programação Smalltalk

Devido aos poucos conceitos introduzidos no sistema, a linguagem *Smalltalk-80* define uma sintaxe apenas para :

- declarar nomes de objetos e atribuir valores aos mesmos.
- enviar mensagens.
- definir novas classes e métodos.

As mensagens são definidas por expressões, as quais declaram o receptor, seletor e argumentos. Quando uma expressão é executada, a mensagem descrita por esta expressão é enviada para o receptor. Três tipos de mensagens são possíveis [30]:

- unitárias.
- binárias.
- com palavras chaves.

Uma mensagem unitária consiste de apenas um identificador denominado seletor unitário. Por exemplo,

```
vetor norma
```

inidica que a mensagem com seletor *norma* é enviada ao objeto *vetor*.

Uma mensagem binária possui um argumento e um seletor. Este seletor é descrito por um dos seletores binários, tais como operadores aritméticos ($+$, $-$, $*$ e $/$) e relacionais ($==$, $\sim=$). Assim,

```
a + 1
```

é uma mensagem binária com seletor + e argumento 1, enviada ao objeto *a*.

Uma mensagem com palavra chave tem um ou mais argumentos e um seletor que é constituído de uma série de palavras chaves, uma para cada argumento. Uma palavra chave é um identificador seguido de dois pontos. Por exemplo:

```
janela movepara: novaposicao modo: reverso
```

é uma mensagem com palavra chave enviada ao objeto *janela* com seletor *move para: modo:* e argumentos *novaposicao* e *reverso*.

O valor de uma variável pode ser usado como receptor ou argumento de uma mensagem. Pode-se alterar o valor de uma variável através de uma expressão de atribuição, a qual consiste do nome da variável, o operador ← e a descrição do objeto. Portanto,

```
indice ← indice + 1
```

significa que o novo valor da variável *indice* é o resultado da mensagem *+ 1* enviada ao conteúdo de *indice*.

A estrutura básica de uma classe consiste do nome da classe, os nomes das variáveis de instância e um conjunto de métodos utilizados para responder às mensagens. Considere, por exemplo a classe *Ponto*, cujas instâncias representam pontos em um sistema de coordenadas bidimensional:

```
nome da classe                  Ponto
nome das variáveis de instância x y
métodos
    x: CoordenadaX y: CoordenadaY  ||
        x ← CoordenadaX
        y ← CoordenadaY

    x  ||
        ↑ x

    y  ||
        ↑ y

    +umPonto | somaX somaY |
        somaX ← x + PontoX.
        somaY ← y + PontoY.
        ↑Ponto newX: somaX Y: somaY
```

```
-umPonto | diferencaX diferencaY |
    diferencaX ← x - PontoX.
    diferencaY ← y - PontoY.
    ↑Ponto newX: diferencaX Y: diferencaY
```

Os métodos possuem três partes básicas [30] :

- um padrão de mensagem.
- variáveis temporárias.
- expressões.

As três partes de um método são separadas por barras verticais (|). O padrão de mensagem consiste de um seletor e argumentos. As expressões são separadas por pontos (.) e a última expressão pode ser precedida por uma seta (↑). Portanto, no método para o seletor + da classe *Ponto*, tem-se o padrão de mensagem *+umPonto*, as variáveis temporárias *somaX* e *somaY* e três expressões sendo a última precedida por uma seta ↑, indicando o valor a ser tomado pela mensagem que invocou o método.

Existem três passos executados por um objeto quando do recebimento de uma mensagem [30] :

1. encontrar o método cujo padrão de mensagem tem o mesmo seletor que a mensagem enviada e criar um conjunto de variáveis para os valores dos argumentos.
2. criar um conjunto de variáveis temporárias correspondentes aos nomes entre as barras verticais.
3. executar as expressões no método sequencialmente.

A linguagem *Smalltalk-80* permite utilizar o mecanismo de herança entre classes. Portanto, para completar a descrição de uma classe é necessário acessar a sua superclasse, a superclasse desta superclasse, e assim por diante até alcançar o topo desta hierarquia. Apenas a classe *Object* não contém nenhuma superclasse em todo o sistema. Logo, todas as classes herdam os métodos da classe *Object*, a qual não contém nenhuma variável de instância.

Como uma classe é também um objeto, existe uma classe que descreve uma classe, sendo esta denominada *metaclasse*. Assim, uma classe tem as suas próprias variáveis de instância, representando a descrição de suas instâncias, e um conjunto de métodos para a criação, inicialização e modificação da sua descrição [30].

Todas as estruturas de controle são baseadas em objetos denominados *blocos*. Um bloco, como um método, é uma sequência de expressões delimitadas por pontos. Estas expressões podem ser precedidas por um ou mais identificadores seguidos de dois pontos. Quando se encontram colchetes em um método, um bloco é criado. Outras características da linguagem e exemplos de aplicação podem ser encontrados em [31].

### C.2.2 O Ambiente de Programação Smalltalk

O ambiente de programação *Smalltalk-80* [31] corresponde à integração da linguagem de programação, ferramentas de suporte (editores, depuradores etc) e o sistema operacional, todos residindo no mesmo espaço de endereço virtual. Assim, cria-se um ambiente de programação interativo e gráfico, onde cada componente é apresentado de maneira significativa para observação e manipulação pelo usuário.

Este ambiente objetiva a criatividade pessoal do programador. Todas as características do sistema estão disponíveis, mesmo as mais básicas, tais como àquelas relacionadas ao sistema operacional ou a representação de ponto flutuante. O código fonte de todos os componentes do sistema é disponível e pode ser facilmente alterado.

O sistema físico recomendado para se trabalhar com Smalltalk-80 deve possuir um vídeo de alta resolução gráfica e dispositivos de entrada, tais como *light pen* e *mouse*. Através destes dispositivos, o usuário pode selecionar informações apresentadas no vídeo e invocar mensagens para interagir com esta informação [31].

Finalmente, deve-se ressaltar que o sistema *Smalltalk-80* contém várias classes e instâncias com várias funções. Um conjunto destas classes constituem o *núcleo* do sistema *Smalltalk-80*. O sistema contém, ainda, classes para suportar estruturas de dados básicas, coleções de objetos e interação gráfica [31].

## C.3 Objective-C

A Linguagem *Objective-C* [32] é um superconjunto da linguagem *C* [33]. As suas declarações são pré-processadas para as declarações convencionais de *C*, para posterior compilação. Em um arquivo de *Objective-C* podem ser incluídas declarações *C*, sendo que estas não são alteradas pelo pré-processador.

Uma das características de uma linguagem híbrida é a possibilidade de acessar, diretamente, as informações locais de um objeto. Portanto, as partes de um programa que requerem maior eficiência, podem ser desenvolvidas utilizando-se ligação estática e verificação de tipos. De modo geral, código de baixo nível é melhor desenvolvido utilizando-se a maneira convencional de programação, enquanto é conveniente implementar o código ao nível do usuário através de objetos [32].

Basicamente, *Objective-C* adiciona um novo tipo (objeto) e uma nova operação (mensagem) à linguagem *C*. A ligação entre tipos de dados e operadores pode ser feita em tempo de compilação (ligação estática) ou de execução (ligação dinâmica).

Os objetos são identificados pelo tipo *id* definido pela linguagem. A única operação legal sobre o identificador de um objeto é a expressão de mensagem. Os identificadores podem ser armazenados em variáveis e utilizados como argumentos de mensagens e funções. Assim, os identificadores podem ser utilizados como qualquer outro tipo definido em *C*. Para se definir um objeto utiliza-se uma declaração do tipo:

```
id umObjeto;
```

A sintaxe utilizada para as mensagens é análoga àquela empregada em *Smalltalk-80*, podendo-se utilizar mensagens unitárias e com palavras chaves. Logo, as expressões

```
[Circulo desenha];
[Circulo desenha: azul];
```

correspondem a uma mensagem unitária e outra com palavra chave, respectivamente.

Os objetos de fábrica (*Factory Objects*) correspondem às classes de *Smalltalk-80*, sendo construídos a partir de um arquivo de descrição de classe. Estes objetos são constituídos de uma parte local, contendo dados particulares a uma instância, e uma parte compartilhada, a qual é comum a todas as instâncias da classe. As instâncias são criadas, como em *Smalltalk-80*, através do método *new*.

Segundo Cox [32], o processo de desenvolvimento de programas deve ser realizado através da produção de módulos, denominados *Software-ICs*. Como as classes de *Smalltalk-80*, estes módulos contém métodos de instância e de classe.

A linguagem *Objective-C* possui um conjunto de classes (*Software-ICs*), de forma análoga à estrutura de classes de *Smalltalk-80*. No nível mais alto desta hierarquia tem-se a classe *Object*. Estes módulos possuem as mais variadas funções, podendo serem utilizados pelo programador através do mecanismo de herança.

Como exemplo de uma classe, considere a classe *Fruta*:

```
= Fruta : Object
{
    char *cor;                    // cor da fruta
    int diametro;                 // diametro da fruta
    }
```

```
// metodo de fabrica
+criar {                                    // cria uma nova instancia
    id novaInstancia;                       // variavel local
    novaInstancia = [self new];             // cria instancia de fruta
    [novaInstancia diametro: 1];            // inicializa diametro da fruta
    [novaInstancia cor: "vermelha"];        // inicializa cor da fruta
    }

// metodos de instancia
-cor: (char *) umaCor {                     // inicializa cor da fruta
    cor = umaCor;
    return self;
    }

-diametro: (int) tamanho {                  // inicializa diametro da fruta
    diametro = tamanho;
    return self;
    }
=:
```

A primeira declaração indica que a classe *Fruta* herda as características da classe *Object*. As variáveis de instância são *cor* e *diametro*. Os métodos de fábrica são introduzidos pelo sinal + e os de instância pelo sinal - .

# Apêndice D

# Modelo de Documentação para Análise de Requerimentos

Neste Apêndice, apresenta-se o modelo de documentação utilizado para a análise de requerimentos dos objetos do programa FAE [44]. Para isso, considera-se o documento confeccionado para o objeto *Matriz*.

# Matriz

## D.1 Introdução

Uma representação e um conjunto de operações devem ser implementados para matrizes cheias e esparsas. Estas matrizes são reais, simétricas ou não simétricas e de ordens arbitrárias. Operações básicas de criação, inicialização e acesso aos elementos, assim como operações algébricas de criação, subtração, multiplicação, multiplicação por escalar, transposição e transformação em identidade devem ser implementadas.

## D.2 Objetos, Atributos e Operações

A Tabela D.1 apresenta os objetos, seus atributos e operações identificados no processo de análise.

| Objetos | Atributos | Operações |
|---|---|---|
| Matriz Cheia | reais simétricas ou não simétricas ordens arbitrárias | criação |
| | | inicialização |
| | | acesso aos elementos |
| | | subtração |
| Matriz Esparsa | | adição |
| | | multiplicação |
| | | multiplicação por escalar |
| | | transposição |
| | | transformação em identidade |

Tabela D.1: Objetos, atributos e operações de *Matriz Cheia* e *Matriz Esparsa*

A Figura D.1 ilustra a estrutura do objeto *Matriz* e dos objetos identificados na Tabela D.1, juntamente com as suas operações.

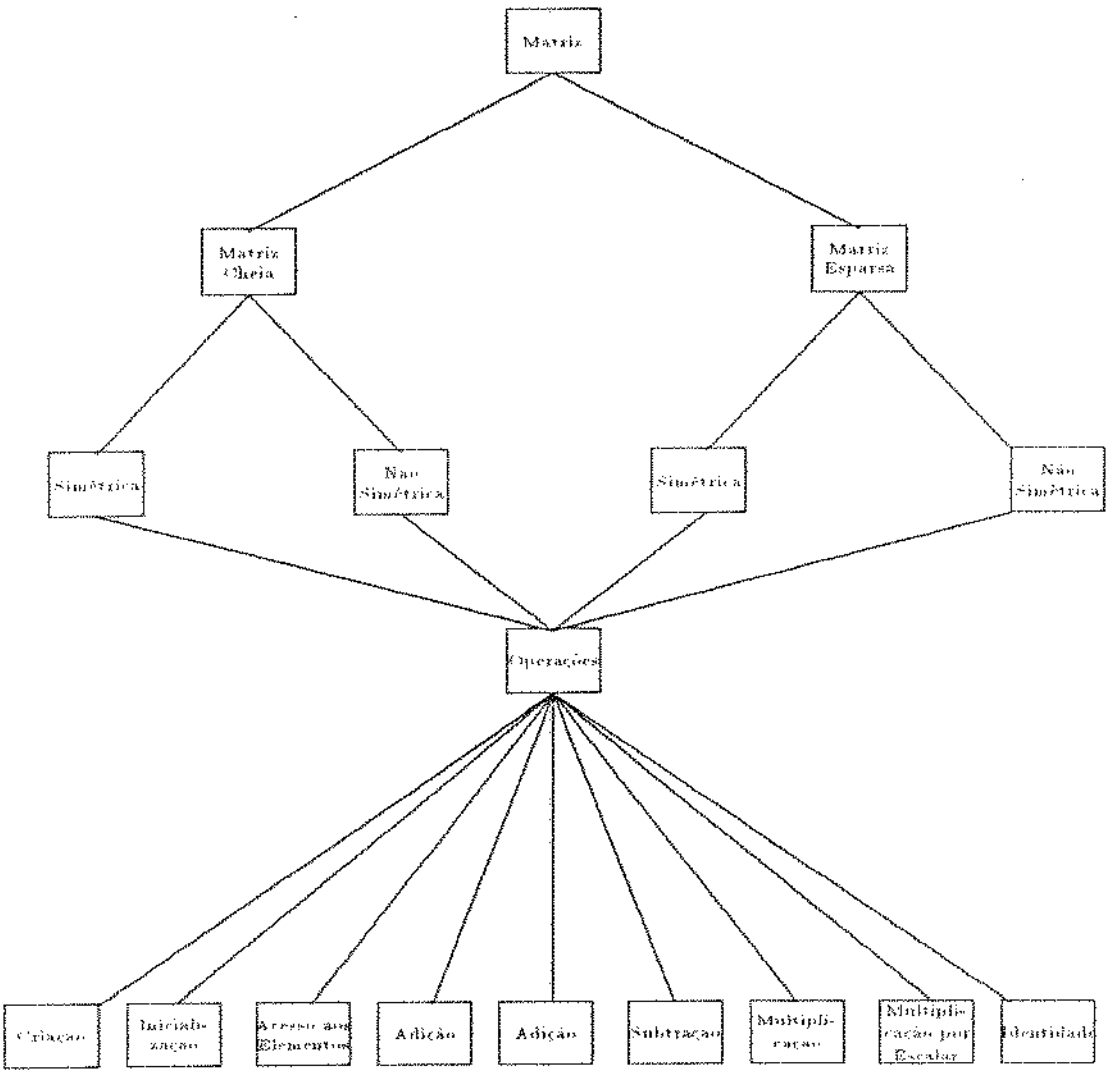


Figura D.1: Estrutura do objeto *Matriz*.

## D.3 Descrição Funcional

A seguir cada uma das operações definidas na Tabela D.1 são apresentadas. Como as operações são comuns às matrizes cheia e esparsa, faz-se apenas uma descrição enfatizando as diferenças das operações para ambas as matrizes.

### D.3.1 Criação

**Definição** cria uma matriz de ordem especificada e inicializa seus elementos com zero.

**Restrições** dois números inteiros, representando a ordem da matriz, são fornecidos. As matrizes simétricas devem ser quadradas. Retorna-se um identificador para a matriz.

**Performance** Para as matrizes cheias e simétricas, armazena-se apenas a sua parte superior, enquanto para matrizes esparsas apenas os elementos não nulos.

**Restrições de Projeto** a estrutura de dados, representando a matriz, deve ser alocada dinamicamente.

### D.3.2 Inicialização

**Definição** inicializa um elemento de uma matriz com um certo valor dado.

**Restrições** o índice do elemento e o valor a ser atribuído devem ser fornecidos.

**Restrições do Projeto** quando se tentar se inicializar um elemento não pertencente à matriz imprime-se uma mensagem de advertência e abandona-se a operação.

### D.3.3 Acesso aos Elementos

**Definição** acessa o elemento da linha e coluna dadas.

**Restrições** especificam-se a linha e a coluna do elemento. Retorna-se, então, o conteúdo do elemento da matriz.

**Restrições de Projeto** imprime-se uma mensagem de adverténcia e abandona-se a execução da operação quando se acessar um elemento não existente na matriz.

### D.3.4 Adição

**Definição** efetua a adição de duas matrizes.

**Restrições** as matrizes a serem somadas e a matriz contendo o resultado devem ser especificadas. As matrizes devem ter a mesma ordem.

**Performance** para as matrizes esparsas, os possíveis elementos a serem acrescentados na matriz resultado, devem ser alocados durante a execução da operação.

**Restrições de Projeto** Para as matrizes de ordens diferentes, imprime-se uma mensagem de advertência e não executando a operação.

### D.3.5 Subtração

**Definição** efetua a subtração de duas matrizes.

**Restrições** especificam-se as matrizes a serem somadas e a matriz resultado, todas de mesma ordem. Retorna-se o identificador da matriz resultado.

**Performance** para matrizes esparsas, os possíveis elementos a serem acrescentados à matriz, contendo o resultado, devem ser alocados durante a execução.

**Restrições de Projeto** para matrizes de ordens diferentes, imprime-se uma mensagem de advertência e abandona-se a operação.

### D.3.6 Multiplicação

**Definição** multiplica duas matrizes.

**Restrições** o número de colunas da primeira matriz deve ser igual ao número de linhas da segunda matriz. As matrizes a serem multiplicadas e àquela contendo o resultado são especificadas. Retona-se a matriz resultado.

**Performance** para matrizes esparsas, os elementos não nulos provenientes da operação devem ser alocados durante a execução da operação.

**Restrições de Projeto** para matrizes de ordem inconsistentes, imprime-se uma mensagem de advertência e abandona-se a execução da operação.

### D.3.7 Multiplicação por Escalar

**Definição** multiplica uma matriz por uma escalar.

**Restrições** especifica-se a matriz e um escalar real. Retorna-se a matriz multiplicada pelo escalar.

### D.3.8 Transposição

**Definição** transpõe uma matriz.

**Restrições** a matriz a ser transposta e a matriz contendo o resultado devem ser especificadas.

### D.3.9 Identidade

**Definição** transforma uma matriz em identidade.

**Restrições** como parâmetro de entrada, fornece-se uma matriz quadrada. Retorna-se, então, a matriz identidade.

**Performance** para matrizes esparsas, armazenam-se apenas os elementos da diagonal.

**Restrições de Projeto** imprime-se uma mensagem de advertência e abandona-se a operação quando a matriz de entrada não for quadrada.

## D.4 Critérios de Validação

**Performance** as matrizes devem ocupar apenas a memória necessária, de acordo com as suas caracteríticas. As operações devem ser o mais eficiente possível, evitando processamento desnecessário.

**Classe de Testes** testes funcionais para cada operação e testes estruturais devem ser aplicados. Para os testes estruturais, deve-se garantir que todas as instruções sejam executadas pelo menos uma vez. Devem ser testados alguns tipo de matrizes cheias e esparsas, tais como retangulares, quadradas, simétricas, não simétricas e unitária.

**Resposta Esperada** deseja-se que todas as operações sejam executadas corretamente e de maneira eficiente, de acordo com as suas especificações.