UNIVERSIDADE FEDERAL DE MINAS GERAIS
Escola de Engenharia
Departamento de Engenharia de Estruturas
Curso de Pós-Graduação em Engenharia de Estruturas

ELEMENTOS FINITOS PARAMÉTRICOS IMPLEMENTADOS EM JAVA

Marcelo Lucas de Almeida

Dissertação apresentada ao curso de Pós-Graduação em Engenharia de Estruturas da UNIVERSIDADE FEDERAL DE MINAS GERAIS, como parte dos requisitos para obtenção do título de MESTRE EM ENGENHARIA DE ESTRUTURAS.

Orientador: Roque Luiz da Silva Pitangueira

Belo Horizonte
Agosto de 2005

UNIVERSIDADE FEDERAL DE MINAS GERAIS
ESCOLA DE ENGENHARIA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA DE ESTRUTURAS

"ELEMENTOS FINITOS PARAMÉTRICOS IMPLEMENTADOS EM JAVA"

Marcelo Lucas de Almeida

Dissertação apresentada ao Programa de Pós-Graduação em Engenharia de Estruturas da Escola de Engenharia da Universidade Federal de Minas Gerais, como parte dos requisitos necessários à obtenção do título de "Mestre em Engenharia de Estruturas".

Comissão Examinadora:

_____
Prof. Dr. Roque Luiz da Silva Pitangueira
DEES - UFMG - (Orientador)

_____
Prof. Dr. Alcebíades de Vasconcellos Filho
DEES - UFMG

_____
Prof. Dr. Gabriel de Oliveira Ribeiro
DEES-UFMG

_____
Prof. Dr. Eduardo Nobre Lages
UFAL

Belo Horizonte, 31 de agosto de 2005

UNIVERSIDADE FEDERAL DE MINAS GERAIS
ESCOLA DE ENGENHARIA
DEPARTAMENTO DE ENGENHARIA DE ESTRUTURAS

Os componentes da banca examinadora aqui citados certifica que leram e recomendam ao departamento de Engenharia de Estruturas à aceitação da dissertação entitulada "**Elementos Finitos Paramétricos Implementados em Java**" defendida por **Marcelo Lucas de Almeida** como parte dos requisitos para obtenção do título de **Mestre em Engenharia de Estruturas**.

Datado: Agosto de 2005

Orientador: _____
Roque Luiz da Silva Pitangueira

Examinadores: _____

_____

Dedico este trabalho às pessoas que me fazem querer ser um homem melhor, àquelas que dão sentido a todo meu esforço. Enfim, às pessoas que sempre estarão ao meu lado:

minha mãe, Marilda; meu pai, Luiz; minhas irmãs, Lidia e Fabiana; e à mulher da minha vida, Adriana.

Índice

Lista de Tabelas

Lista de Figuras

Lista de Abreviaturas, Siglas e Símbolos

API	Aplication program interface
$[B]$	Matriz das derivadas das funções de forma
$\{b\}$	Vetor das forças de volume
$\{d\}$	Vetor de deslocamentos nodais
DOF	Degrees of freedom
$[E]$	Matriz das propriedades constitutivas de um material
$\{f\}$	Vetor do carregamento nodal equivalente
INSANE	Interactive structural analysis environment
$[J]$	Matriz da transformação jacobiana
JVM	Java virtual machine
$[k]^e$	Matriz de rigidez de um elemento
L_e	Comprimento de um elemento
$[N]$	Matriz das funções de forma
MEC	Método dos elementos de contorno
MEF	Método dos elementos finitos
MVC	Model-view-controller
POO	Programação orientada a objetos
$\{q\}$	Vetor das forças de superfície
S_e	Área da face de um elemento
$\{t\}$	Vetor das forças de linha
uc	Unidade de comprimento
uf	Unidade de força
UML	Unified modelling language
V_e	Volume de um elemento
XML	Extensible markup language
WEB	Rede internacional de computadores

ϵ	Deformação
ξ	Coordenada adimensional
ζ	Coordenada adimensional
η	Coordenada adimensional
σ	Tensão normal
τ	Tensão cisalhante

Resumo

Esta dissertação de mestrado refere-se à implementação computacional da formulação paramétrica do método dos elementos finitos (MEF) utilizando a linguagem java. Todo o trabalho foi implementado no núcleo numérico do INSANE *(INteractive Structural ANalysis Environment)*, um sistema computacional que visa a apropriação dos modernos recursos para desenvolvimento de software em favor da pesquisa na área de métodos numéricos e computacionais aplicados à engenharia.

Apresenta-se um estudo da formulação paramétrica do MEF, identificando suas generalidades e correlações com a metodologia de programação orientada a objetos (POO) e verificando-se a perfeita adequação desta metodologia para a referida formulação. Após uma breve revisão dos principais conceitos da metodologia de POO, discutem-se as principais vantagens da utilização da linguagem Java. Faz-se uma análise orientada a objetos buscando-se identificar as principais classes necessárias à representação do problema.

O projeto orientado a objetos da implementação é, então, apresentado com o auxílio de diagramas UML *(Unified Modelling Language)* apropriados.

Os recursos do MEF disponibilizados consistem de vários tipos de elementos paramétricos, incluindo os elementos unidimensionais de dois, três e quatro nós; os elementos bidimensionais quadrilaterais e quadrilaterais axissimétricos de quatro, oito e nove nós; os elementos bidimensionais triangulares e triangulares axissimétricos de três, seis e dez nós; e os elementos tridimensionais hexaédricos de oito e vinte nós. Os modelos de análise implementados são: unidimensional; bidimensional de estado plano de tensão, estado plano de deformação e axissimétrico; e tridimensional. Para o cálculo das integrais relacionadas à formulação paramétrica, implementa-se a integração numérica de Gauss com

várias ordens possíveis. Os carregamentos implementados abrangem cargas distribuídas em uma linha, área ou em um volume. Implementa-se também material elástico linear isotrópico e solução por equilíbrio para problemas de análise de tensões.

O correto funcionamento dos vários recursos é comprovado através de diversos exemplos.

Abstract

This master's thesis refers to the implementation of parametric formulation of finite element method (FEM) in Java language. All this work was implemented in numeric core of INSANE (INteractive Structural ANalysis Environment), a computational system which aims the appropriation of modern recourses for software development to help research in computational and numeric methods applied to engineering.

The parametric formulation of FEM is studied, enumerating its generalities and correlations with object oriented programming (OOP). It is verified that the OOP are quite appropriated for the implementation of FEM parametric formulation.

An object oriented analysis to identify the main necessary classes of the problem representation is done.

The implementation's object oriented project is shown with unified modelling language (UML).

The implemented FEM recourses in this work are several types of parametric elements including one-dimensional elements with two, three an four nodes; two-dimensional quadrilateral and axisymmetric quadrilateral elements with four, eight and nine nodes; two-dimensional triangular and axisymmetric triangular elements with three, six and ten nodes; and three-dimensional hexahedral elements with eight and twenty nodes. The analysis models implemented are: one-dimensional; two dimensional plane stress and plane strain and axisymmetric; and three-dimensional. For the integral calculus related

to parametric formulation a Gauss numeric integration was implemented. The implemented distributed loads are in lines, areas and in volumes. It was implemented linear elastic isotropic material and solution by equilibrium for problems of stress analysis.

The recourses correct functioning are validated by several examples shown.

Agradecimentos

É um grande prazer retribuir a todos que ajudaram nesta difícil tarefa de me tornar um mestre em engenharia de estruturas. Tentarei expressar aqui minha gratidão.

A Deus por ter me permitido nascer em uma família cheia de entusiasmo e energia, tão necessários para enfrentar os desafios diários. A Deus também agradeço por me colocar à frente destes desafios para que assim eu pudesse amadurecer como ser humano.

A minha mãe que sempre esteve pronta para ouvir meus problemas, compreendê-los e me ajudar a superá-los.

Ao meu pai pelo incentivo no desenvolvimento do mestrado e por seu exemplo de luta, mostrando que podemos alcançar nossos objetivos.

A minha irmã Lidia, uma referência de companheirismo e alegria, pelo exemplo de força, e por me mostrar alguns dos meus erros.

A minha irmã Fabiana, por abaixar o volume do som quando eu estava estudando ou dormindo.

À Adriana, fonte inesgotável de inspiração para mim, me ajudando com ótimos conselhos e idéias. Por me acolher nos momentos em que precisei. E também pela paciência comigo nos momentos em que não pude estar presente.

Aos meus amigos e amigas do mestrado, companheiros nas exaustivas horas de estudo.

Ao Professor Roque, sempre disposto a me ajudar e dividir comigo seus conhecimentos e experiências de vida. Pela dedicação como orientador e pelo empenho em me ensinar. Por me mostrar que um trabalho bem feito necessita de muita paciência e concentração. E ainda, por me encorajar e acreditar que eu conseguiria realizar este trabalho.

Aos professores do departamento de engenharia de estruturas pelo exemplo de ética profissional e competência.

Aos funcionários do departamento de engenharia de estruturas pela prestação de bons serviços.

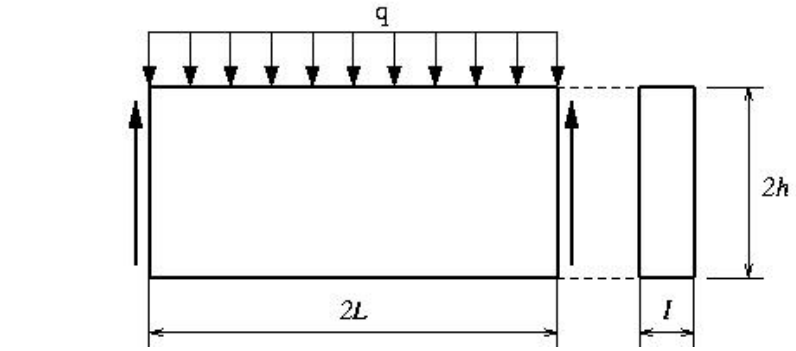
À CAPES por viabilizar financeiramente a concretização deste trabalho.

Capítulo 1

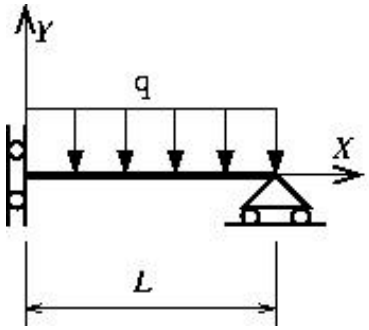
Introdução

Costuma-se dividir o processo de análise estrutural em três etapas conforme a figura 1.1. Orientando-se por experiência de projeto, o problema de meio contínuo da estrutura real (figura 1.1(a)) é substituído por um modelo matemático utilizando-se hipóteses simplificadoras (figura 1.1(b)). Tal modelo matemático é expresso por equações diferenciais (ordinárias ou parciais) cujas soluções, ditas soluções analíticas, são conhecidas apenas em alguns poucos casos simples. Para superar as limitações de tais soluções, adota-se um modelo numérico aproximado, dito modelo discreto (figura 1.1(c)). Nos modelos discretos, as equações são algébricas e as grandezas são determinadas em um número finito de pontos, diferentemente das soluções analíticas cujas equações diferenciais, quando resolvidas, permitem avaliar as grandezas em um número infinito de pontos. Dentre os métodos discretos mais utilizados destacam-se o método dos elementos finitos (MEF) e o método dos elementos de contorno (MEC).

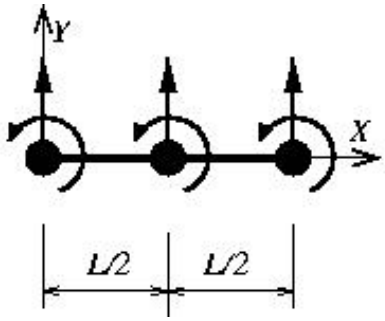
A pesquisa na área de métodos numéricos e computacionais para os referidos modelos discretos procura um aprimoramento das hipóteses simplificadoras dos mesmos. Isto é feito de forma a ampliar complexidades a partir dos conceitos já consolidados. Entretanto, observa-se sempre um recomeço do processo ao se recriarem as ferramentas relativas às tecnologias dominadas. Um exemplo ilustrativo deste fato é a (re)implementação computacional de algoritmos de solução de sistemas de equações algébricas lineares, toda vez que os mesmos são usados como parte do processo de aprimoramento de determinado modelo discreto.



(a) Problema de Meio Contínuo



(b) Modelo Matemático



(c) Modelo Discreto

Figura 1.1: Modelagem de uma Viga

Ao longo do tempo, algumas iniciativas de desenvolvimento de software pela comunidade acadêmica resultaram em produtos dependentes de sistema operacional, pouco amigáveis, escritos em linguagens de programação não apropriadas, de expansão, manutenção e distribuição difíceis, desenvolvidos por equipes fechadas, com documentação deficiente, entre outras limitações. Tais fracassos podem ser creditados à falta de disposição da comunidade em se apropriar das tecnologias emergentes ou mesmo à inexistência das mesmas.

Esta constatação confronta-se com o surgimento e aprimoramento de soluções tecnológicas para desenvolvimento de software, como programação orientada a objetos, linguagem Java, XML (eXtensible Markup Language), padrões de projeto de software (Gamma, Helm, Johnson & Vlissides 1995), entre outras. Estas soluções permitem o desenvolvimento de sistemas computacionais segmentados, amigáveis a mudanças e escaláveis em complexidade (Alvim 2003).

Portanto, o desafio de desenvolver sistemas utilizando estes recursos é condição obrigatória para aprimoramento da agilidade e criatividade da pesquisa na área.

1.1 O Projeto INSANE

A utilização de modelos discretos de análise estrutural compreende três etapas principais inter-relacionadas: (1) criação do modelo, (2) montagem e resolução do modelo e (3) avaliação de resultados. Na criação do modelo, o analista informa as hipóteses simplificadoras relativas à geometria, material, carregamento e condições de contorno e estas são representadas com entidades matemáticas apropriadas, gerando assim o que se denomina malha e os atributos do modelo. Na etapa de montagem e resolução do modelo, combinam-se as informações matematicamente representadas, de modo a produzir equações algébricas que, quando solucionadas, permitem obter as diversas grandezas. Na avaliação de resultados, o analista faz crítica e verifica a adequação dos mesmos ao problema em estudo.

Para disponibilização deste processo em computadores, normalmente a referida divisão é adotada através de programas de pré-processamento (para a criação dos modelos com recursos gráficos interativos), processamento (para a montagem e resolução numérica do modelo) e pós-processamento (para visualização gráfica de resultados).

As possibilidades que os recursos tecnológicos para desenvolvimento de software oferecem para cada uma das três etapas constituem amplo campo de pesquisa na área de métodos numéricos e computacionais aplicados à engenharia.

O domínio destes recursos e a aplicação dos mesmos no aprimoramento progressivo dos modelos, sem ter que recomeçar o processo a cada novo aperfeiçoamento, requer um ambiente computacional segmentado, amigável a mudanças e escalável em complexidade.

O projeto INSANE (Interactive Structural Analysis Environment) objetiva o desenvolvimento de um ambiente computacional com as características acima citadas. Informações adicionais sobre o projeto podem ser encontradas na internet (*www.dees.ufmg.br/insane*). A figura 1.2 mostra uma visualização da atual interface gráfica com o usuário do ambiente. Como pode ser visto na figura, o ambiente é constituído de três segmentos: pré-processador, processador e pós-processador. Os pré e pós-processadores são aplicações gráficas interativas, implementadas na linguagem Java que disponibilizará, respectivamente, ferramentas de pré e pós-processamento de diferentes modelos discretos. O

processador é uma aplicação, também implementada em Java, que representa o núcleo numérico do sistema. Este núcleo é responsável pela obtenção dos resultados de diferentes modelos discretos de análise estrutural. A persistência dos dados compartilhados pelas três aplicações é alcançada através de uma interface baseada em arquivos XML e/ou objetos Java.

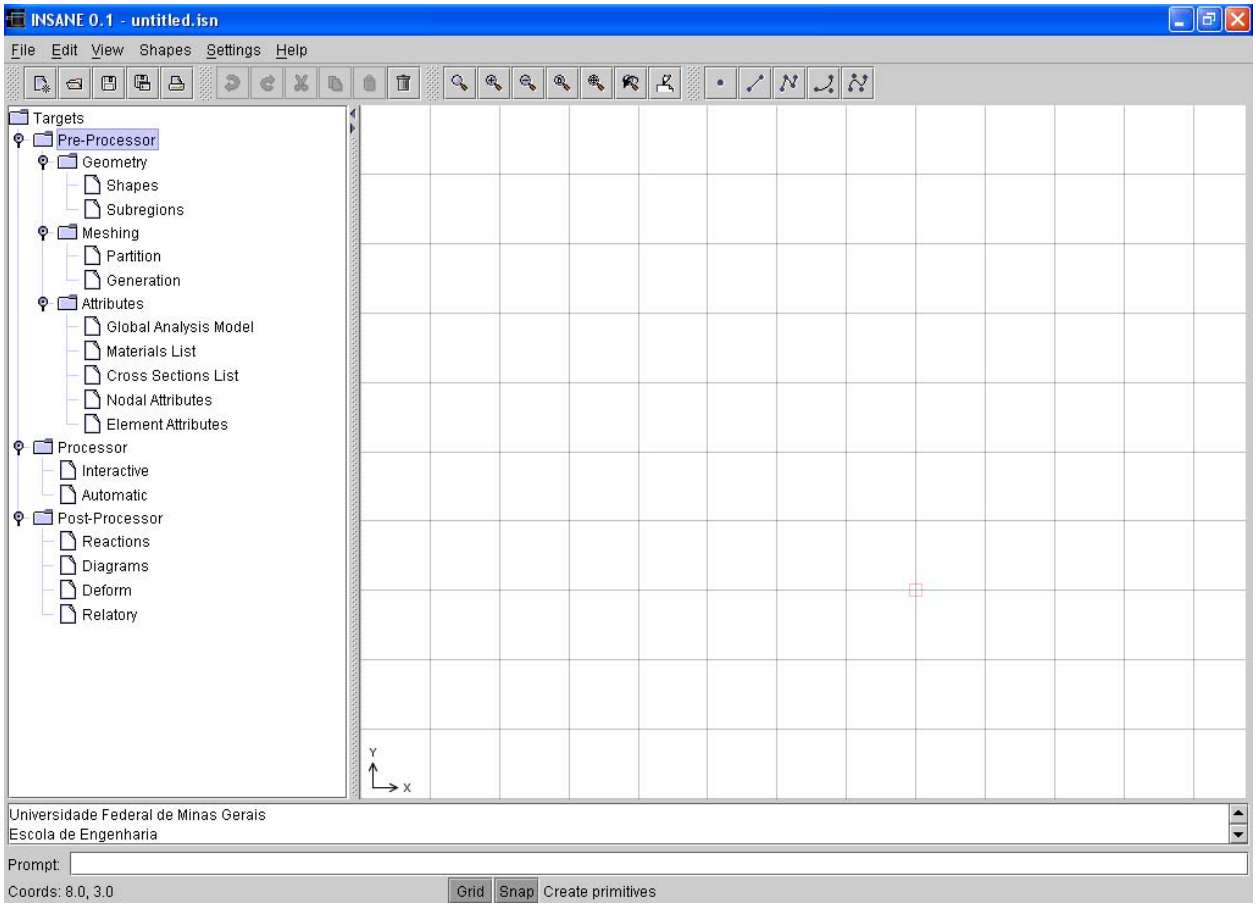


Figura 1.2: Projeto Preliminar do Ambiente

Cada uma destas aplicações é implementada segundo o paradigma de programação orientada a objetos (POO).

O processador utiliza os diversos conceitos do paradigma de POO (classes, herança, polimorfismo etc) de modo a possuir a segmentação necessária ao aprimoramento progressivo do sistema. As aplicações gráficas interativas de pré e pós-processamento, também implementadas segundo POO, utilizam o padrão de projeto de software (Gamma

et al. 1995) denominado *Model-View-Controller* (MVC). Este padrão é bastante apropriado uma vez que preconiza a separação do processamento da informação de sua representação gráfica (Pietro 2001), facilitando assim os trabalhos de expansão e manutenção destas aplicações.

A linguagem (XML) está sendo adotada como padrão para troca de documentos através da internet. Com a tecnologia dos "WEB Services", praticamente qualquer software ou componente de software (orientado a objetos) pode ser utilizado remotamente, sendo necessário somente que as partes "conversem" em XML. A tecnologia SOAP ("Simple Object Acces Protocol"), por exemplo, opera sobre a WEB de maneira que suas mensagens (requisições e respostas) sejam simplesmente documentos XML (Braz 2003). Assim, a opção de fazer a persistência dos dados em arquivos XML e a segmentação propiciada pela utilização de POO permitirá que o sistema ou partes deste possa, futuramente, ser utilizado através da internet.

1.2 Objetivo Específico

Dentre os modelos discretos para análise de problemas de engenharia, os baseados no MEF são os mais difundidos.

Pode-se creditar à formulação paramétrica o grande desenvolvimento e aceitação do MEF como ferramenta de engenharia. Como será detalhado adiante, esta formulação permite que as entidades matemáticas do modelo discreto possam ser calculadas de uma única maneira, quaisquer que sejam as hipóteses relativas à cinemática, geometria, material e carregamento do modelo. Este potencial de generalização da formulação paramétrica do MEF a torna propícia ao paradigma de programação orientada a objetos (POO).

O objetivo específico da dissertação de mestrado que aqui se apresenta é a implementação computacional da formulação paramétrica do MEF, no núcleo numérico do sistema acima discutido.

Capítulo 2

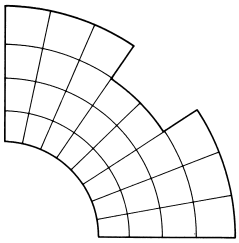
Formulação Paramétrica do MEF

No modelo de deslocamentos do MEF o domínio do problema é dividido em subdomínios de dimensões finitas denominados elementos finitos, onde o campo de deslocamentos é arbitrado. A figura 2.1 apresenta exemplos de possíveis discretizações para análise de estruturas utilizando o MEF (figuras retiradas de (Weaver & Johnston 1984)). Escrevendo-se aproximações para o campo de deslocamentos de cada elemento em função dos deslocamentos nodais, obtém-se um sistema de equações algébricas que, quando resolvido, permite solucionar o problema (Pitangueira 2000).

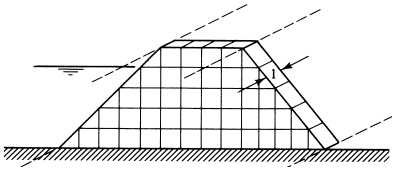
A formulação paramétrica do MEF generaliza este conceito, descrevendo as referidas aproximações em função de parâmetros adimensionais de um sistema de coordenadas congruente com a geometria do elemento.

Esta estratégia, além de facilitar o cálculo de derivadas e integrais inerentes ao método, confere ao mesmo grande generalidade. Isto significa que as diversas grandezas do modelo podem ser calculadas através de procedimentos unificados, que contemplam a maioria das hipóteses relativas à geometria, cinemática do contínuo, comportamento do material e carregamento.

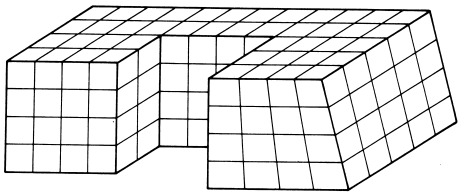
Um elemento finito é considerado isoparamétrico se as mesmas funções definirem as aproximações para geometria e deslocamentos. Se as funções de interpolação da geometria são de ordem menor do que as funções de interpolação para deslocamentos, o elemento é considerado subparamétrico. Para o elemento superparamétrico as funções de interpolação da geometria são de ordem maior do que as funções de interpolação para deslocamentos.



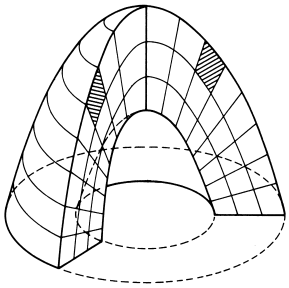
(a) Estado plano de tensão



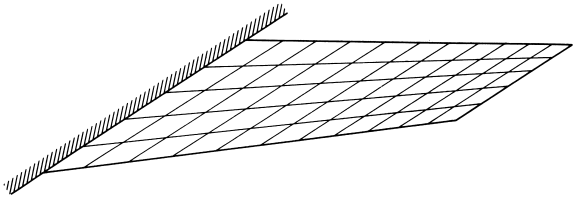
(b) Estado plano de deformação



(c) Sólido



(d) Sólido Axissimétrico



(e) Placa submetida à flexão

Figura 2.1: Exemplos de possíveis discretizações para aplicação do MEF

2.1 Funções de Aproximação

As características geométricas de determinados elementos finitos levam ao uso de sistemas de coordenadas naturais em lugar do sistema de coordenadas cartesiano. Formulações para triângulos, quadriláteros e suas variações tridimensionais representam bem a utilidade do sistema de coordenadas naturais. O propósito da utilização do sistema natural é que as derivadas e integrais necessárias no cálculo de diversas grandezas são simplificadas com o uso de um sistema de coordenadas congruente com a geometria do elemento.

A figura 2.2 (Zienkiewicz 1979) mostra os sistemas de coordenadas naturais normalmente utilizados para parametrizar aproximações em domínios unidimensionais (2.2(a)), bidimensionais (2.2(a)) e tridimensionais (2.2(b)).

A figura 2.2 também mostra que a parametrização das aproximações envolve o mapeamento do elemento, definido no sistema de coordenadas naturais e normalmente denominado elemento padrão, para sua forma distorcida no sistema de coordenadas cartesiano. Este processo é completamente geral, requerendo somente que se estabeleça uma correspondência biunívoca entre os sistemas de coordenadas, na forma

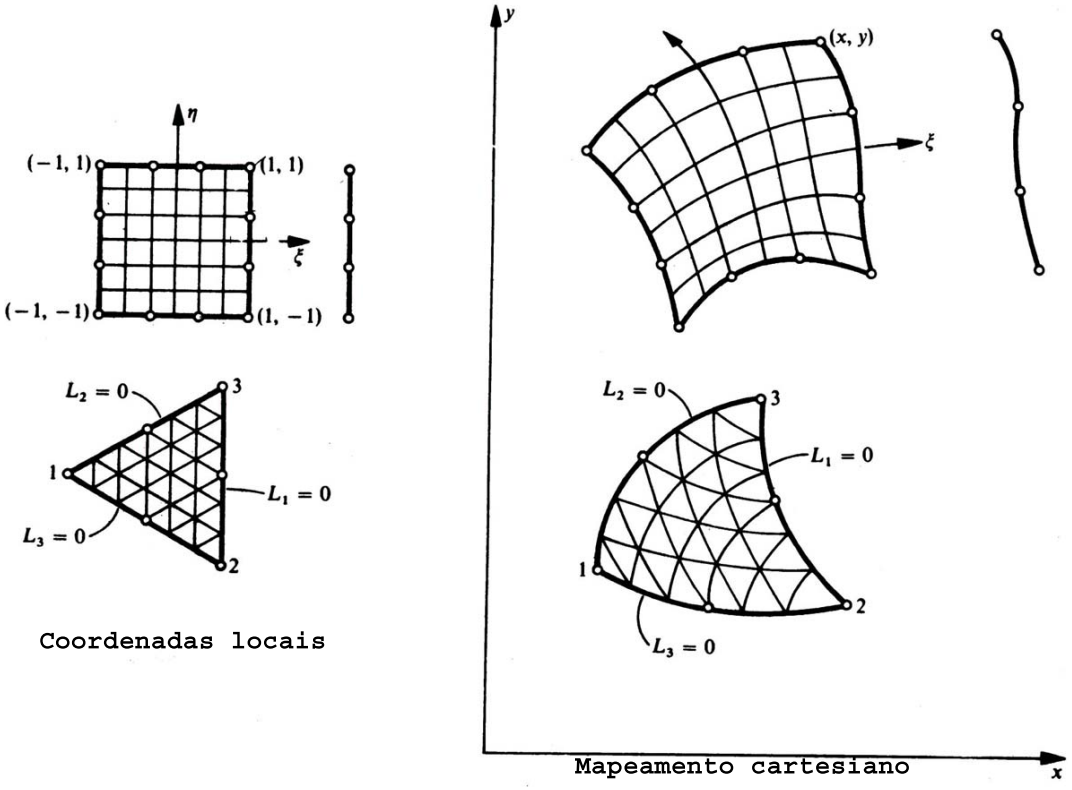
$$\left\{ \begin{array}{c} X \\ Y \\ Z \end{array} \right\} = \left\{ \begin{array}{c} X(\xi,\eta,\zeta) \\ Y(\xi,\eta,\zeta) \\ Z(\xi,\eta,\zeta) \end{array} \right\} \tag{2.1.1}$$

Uma vez que tal correspondência é estabelecida, as funções de aproximação podem ser obtidas no sistema natural de coordenadas e, através de transformações apropriadas, as propriedades do elemento finito podem ser determinadas.

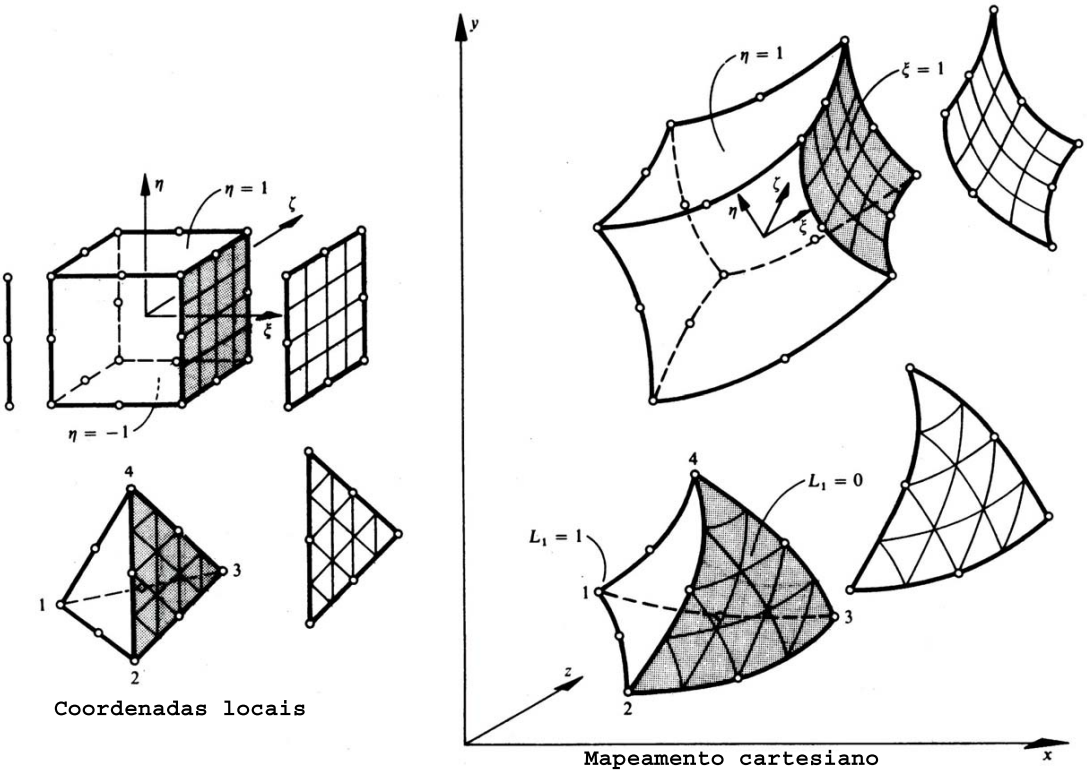
Assim, a aproximação de uma grandeza qualquer, por exemplo, a coordenada x de um ponto, válida no domínio do elemento finito, é normalmente escrita na forma

$$x = [\ N\ ]\ \{x\}^e \tag{2.1.2}$$

onde $[\ N\ ]$ é a chamada matriz das funções de aproximação ou funções de forma, escritas no sistema natural de coordenadas e $\{x\}^e$ é o vetor dos valores nodais da grandeza que se deseja aproximar (neste caso, a coordenada x de um ponto).



(a) Domínios unidimensionais e bidimensionais



(b) Domínios tridimensionais

Figura 2.2: Sistemas de coordenadas naturais

Tabela 2.1: Funções de forma e suas derivadas para o elemento plano de 9 nós

Nó	N	$\partial N/\partial\xi$	$\partial N/\partial\eta$
1	$\frac{1}{4}(1-\xi)(1-\eta)\xi\,\eta$	$\frac{1}{4}(1-2\xi)(1-\eta)\eta$	$\frac{1}{4}(1-\xi)(1-2\eta)\xi$
2	$-\frac{1}{2}(1-\xi^2)(1-\eta)\eta$	$(1-\eta)\xi\eta$	$-\frac{1}{2}(1-\xi^2)(1-2\eta)$
3	$-\frac{1}{4}(1+\xi)(1-\eta)\xi\eta$	$-\frac{1}{4}(1+2\xi)(1-\eta)\eta$	$-\frac{1}{4}(1+\xi)(1-2\eta)\xi$
4	$\frac{1}{2}(1+\xi)(1-\eta^2)\xi$	$\frac{1}{2}(1+2\xi)(1-\eta^2)$	$-(1+\xi)\xi\eta$
5	$\frac{1}{4}(1+\xi)(1+\eta)\xi\eta$	$\frac{1}{4}(1+2\xi)(1+\eta)\eta$	$\frac{1}{4}(1+\xi)(1+2\eta)\xi$
6	$\frac{1}{2}(1-\xi^2)(1+\eta)\eta$	$-(1+\eta)\xi\eta$	$\frac{1}{2}(1-\xi^2)(1+2\eta)$
7	$-\frac{1}{4}(1-\xi)(1+\eta)\xi\eta$	$-\frac{1}{4}(1-2\xi)(1+\eta)\eta$	$-\frac{1}{4}(1-\xi)(1+2\eta)\xi$
8	$-\frac{1}{2}(1-\xi)(1-\eta^2)\xi$	$-\frac{1}{2}(1-2\xi)(1-\eta^2)$	$(1-\xi)\xi\eta$
9	$(1-\xi^2)(1-\eta^2)$	$-2(1-\eta^2)\xi$	$-2(1-\xi^2)\eta$

A figura 2.3 mostra exemplos de mapeamento do sistema de coordenadas natural para as coordenadas cartesianas e a tabela 2.1 mostra as funções de forma e suas derivadas para um elemento finito plano de 9 nós. Como será discutido a seguir, as derivadas das funções de forma em relação às coordenadas naturais são necessárias para a obtenção das diversas propriedades do elemento finito.

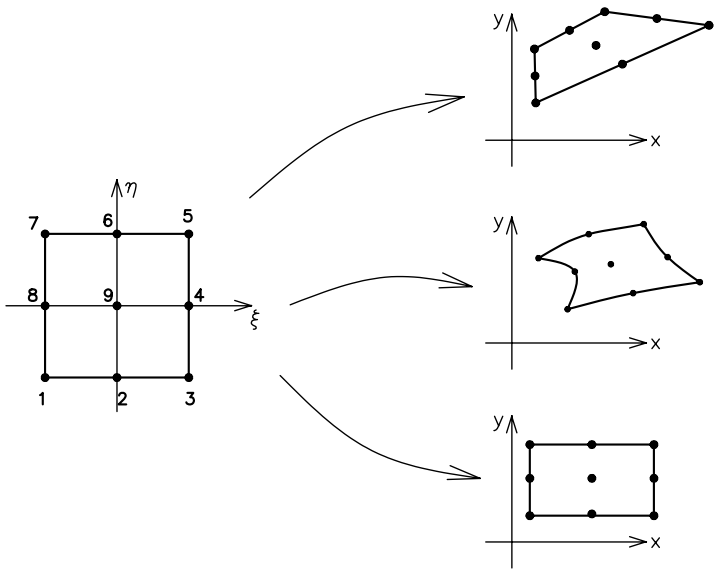


Figura 2.3: Elemento plano de 9 nós

2.2 Obtenção das Propriedades do Elemento

A análise estrutural através do MEF envolve a obtenção de matrizes que definem diversas propriedades do elemento, como rigidez, massa, amortecimento, carregamento nodal equivalente, dentre outras. Genericamente, pode-se expressar esta operação na forma da seguinte integral, definida no volume do elemento finito (Zienkiewicz 1979):

$$[P]^e = \int_{Ve} [G]\, dVe \tag{2.2.1}$$

onde $[P]^e$ é a matriz relativa à determinada propriedade e $[G]$ depende das funções de forma do elemento (matriz $[N]$ da expressão (2.1.2)) e de suas derivadas em relação ao sistema de coordenadas cartesiano global (ver figura 2.2).

2.2.1 Matriz de Rigidez

No caso da matriz de rigidez de um elemento ($[k]^e$), a forma dada em (2.2.1) fica

$$[k]^e = \int_{Ve} [B]^T [E][B]\, dVe \tag{2.2.2}$$

onde $[B]$ contém derivadas das funções de forma em relação às coordenadas cartesianas (x,y,z), $[E]$ contém propriedades do material e a integral é calculada no volume do elemento V_e.

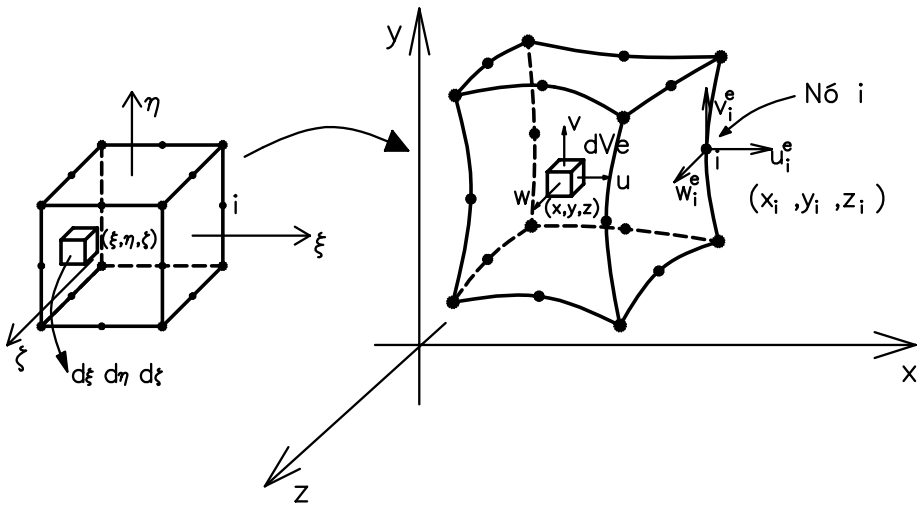


Figura 2.4: Formulação paramétrica tridimensional

Para demonstrar a generalidade dos elementos finitos paramétricos é apresentada a formulação para o caso tridimensional (ver figura 2.4). Como as componentes de deformação são definidas por derivações dos deslocamentos em relação às coordenadas cartesianas e as funções de interpolação são expressas em termos das coordenadas naturais, é necessário obter aquelas derivadas a partir de derivadas destas funções em relação às coordenadas naturais (Soriano & Lima 1999). Pela regra da cadeia tem-se:

$$\begin{aligned} N_{i,\xi} &= N_{i,x}\ x,_{\xi} + N_{i,y}\ y,_{\xi} + N_{i,z}\ z,_{\xi} \\ N_{i,\eta} &= N_{i,x}\ x,_{\eta} + N_{i,y}\ y,_{\eta} + N_{i,z}\ z,_{\eta} \\ N_{i,\zeta} &= N_{i,x}\ x,_{\zeta} + N_{i,y}\ y,_{\zeta} + N_{i,z}\ z,_{\zeta} \end{aligned} \tag{2.2.3}$$

ou em forma matricial

$$\left\{ \begin{array}{c} N_{i,\xi} \\ N_{i,\eta} \\ N_{i,\zeta} \end{array} \right\} = [J] \left\{ \begin{array}{c} N_{i,x} \\ N_{i,y} \\ N_{i,z} \end{array} \right\} \tag{2.2.4}$$

sendo $[J]$ a matriz da transformação jacobiana

$$\left[\ J\ \right] = \left[\begin{array}{ccc} x,_{\xi} & y,_{\xi} & z,_{\xi} \\ x,_{\eta} & y,_{\eta} & z,_{\eta} \\ x,_{\zeta} & y,_{\zeta} & z,_{\zeta} \end{array} \right] \tag{2.2.5}$$

Substituindo a definição de geometria (2.1.2) na equação (2.2.5), tem-se

$$\left[\ J\ \right] = \left[\begin{array}{ccc} [N],_{\xi} & [N],_{\xi} & [N],_{\xi} \\ [N],_{\eta} & [N],_{\eta} & [N],_{\eta} \\ [N],_{\zeta} & [N],_{\zeta} & [N],_{\zeta} \end{array} \right] \left\{ \begin{array}{c} \{x\}^e \\ \{y\}^e \\ \{z\}^e \end{array} \right\} = \sum_{i=1}^{p} \left[\begin{array}{ccc} N_{i,\xi}\ x_i & N_{i,\xi}\ y_i & N_{i,\xi}\ z_i \\ N_{i,\eta}\ x_i & N_{i,\eta}\ y_i & N_{i,\eta}\ z_i \\ N_{i,\zeta}\ x_i & N_{i,\zeta}\ y_i & N_{i,\zeta}\ z_i \end{array} \right] \tag{2.2.6}$$

onde p é o número de pontos nodais do elemento. A equação (2.2.6) permite a obtenção da matriz Jacobiana a partir das coordenadas nodais. No caso de correspondência biunívoca entre ξ, η, ζ e x, y, z, esta matriz $[J]$ é não-singular, permitindo escrever, a partir de (2.2.4) e para um ponto qualquer do elemento

$$\left\{ \begin{array}{c} N_{i,x} \\ N_{i,y} \\ N_{i,z} \end{array} \right\} = [J]^{-1} \left\{ \begin{array}{c} N_{i,\xi} \\ N_{i,\eta} \\ N_{i,\zeta} \end{array} \right\} \tag{2.2.7}$$

Para problemas em regime de pequenos deslocamentos e deformações, as componentes de deformação podem ser obtidas por

$$\left\{ \varepsilon \right\} = \left\{ \begin{array}{c} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{zz} \\ \gamma_{xy} \\ \gamma_{xz} \\ \gamma_{yz} \end{array} \right\} = \left\{ \begin{array}{c} u_{,x} \\ v_{,y} \\ w_{,z} \\ u_{,y} + v_{,x} \\ u_{,z} + w_{,x} \\ v_{,z} + w_{,y} \end{array} \right\} = [H] \left\{ \begin{array}{c} u_{,x} \\ u_{,y} \\ u_{,z} \\ v_{,x} \\ v_{,y} \\ v_{,z} \\ w_{,x} \\ w_{,y} \\ w_{,z} \end{array} \right\} \tag{2.2.8}$$

onde u, v, e w são deslocamentos nas direções cartesianas X, Y e Z, respectivamente, e

$$\left[H \right] = \left[\begin{array}{ccccccccc} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \end{array} \right] \tag{2.2.9}$$

A partir da interpolação de deslocamentos $u = [N] \{u\}^e$ tem-se

$$\left\{ \begin{array}{c} u_{,x} \\ u_{,y} \\ u_{,z} \end{array} \right\} = \left[\begin{array}{c} [N]_{,x} \\ [N]_{,y} \\ [N]_{,z} \end{array} \right] \left\{ u \right\}^e \tag{2.2.10}$$

que utilizando (2.2.7) pode-se escrever

$$\left\{ \begin{array}{c} u_{,x} \\ u_{,y} \\ u_{,z} \end{array} \right\} = [J]^{-1} \left[\begin{array}{c} [N]_{,\xi} \\ [N]_{,\eta} \\ [N]_{,\zeta} \end{array} \right] \left\{ u \right\}^e \tag{2.2.11}$$

Equações análogas podem ser escritas para v e w. Substituindo essas equações em (2.2.8), escrevem-se as componentes de deformação em termos dos deslocamentos nodais e de derivadas das funções de interpolação em relação às coordenadas naturais ξ, η, e ζ, na

forma

$$\left\{ \varepsilon \right\} = [H] \begin{bmatrix} [J]^{-1} & \cdot & \cdot \\ \cdot & [J]^{-1} & \cdot \\ \cdot & \cdot & [J]^{-1} \end{bmatrix} \begin{bmatrix} [N]_{,\xi} & \cdot & \cdot \\ [N]_{,\eta} & \cdot & \cdot \\ [N]_{,\zeta} & \cdot & \cdot \\ \cdot & [N]_{,\xi} & \cdot \\ \cdot & [N]_{,\eta} & \cdot \\ \cdot & [N]_{,\zeta} & \cdot \\ \cdot & \cdot & [N]_{,\xi} \\ \cdot & \cdot & [N]_{,\eta} \\ \cdot & \cdot & [N]_{,\zeta} \end{bmatrix} \begin{Bmatrix} \{u\}^e \\ \{v\}^e \\ \{w\}^e \end{Bmatrix} = [B] \begin{Bmatrix} \{u\}^e \\ \{v\}^e \\ \{w\}^e \end{Bmatrix} = [B] \{d\} \tag{2.2.12}$$

Desta forma, pode-se obter a matriz de rigidez do elemento utilizando a equação (2.2.2) e a equação (2.2.12), lembrando que $dV_e = \mid J \mid d\xi d\eta d\zeta$,

$$[k]^e = \int_{-1}^{+1} \int_{-1}^{+1} \int_{-1}^{+1} [B]^T [E][B] \mid J \mid d\xi d\eta d\zeta \tag{2.2.13}$$

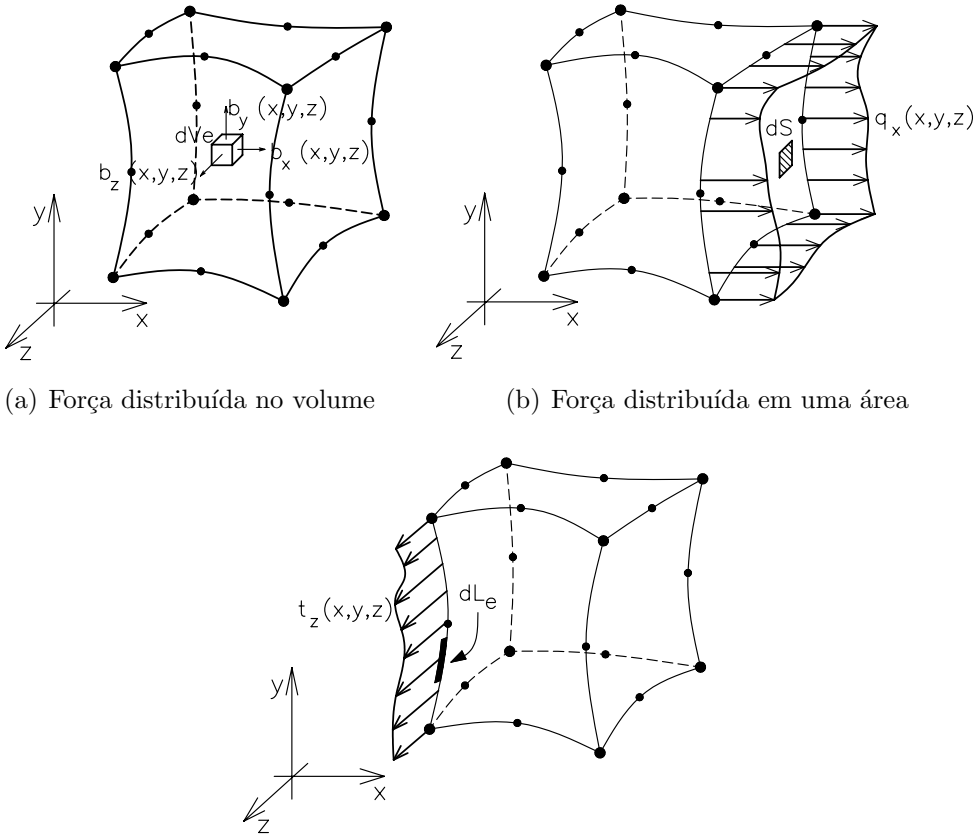
2.2.2 Carregamento Nodal Equivalente

De forma análoga, o carregamento nodal equivalente devido a forças de volume $\{b\}$ pode ser escrito da seguinte forma (ver figura 2.5 (a)):

$$\left\{ f \right\}_b^e = \int_{-1}^{+1} \int_{-1}^{+1} \int_{-1}^{+1} \begin{Bmatrix} [N]^T & \cdot & \cdot \\ \cdot & [N]^T & \cdot \\ \cdot & \cdot & [N]^T \end{Bmatrix} \left\{ b \right\} \mid J \mid d\xi d\eta d\zeta \tag{2.2.14}$$

sendo que

$$\left\{ b \right\} = \begin{Bmatrix} b_x(\xi,\eta,\zeta) \\ b_y(\xi,\eta,\zeta) \\ b_z(\xi,\eta,\zeta) \end{Bmatrix} = \begin{Bmatrix} [N] & \cdot & \cdot \\ \cdot & [N] & \cdot \\ \cdot & \cdot & [N] \end{Bmatrix} \begin{Bmatrix} \{b\}_x^e \\ \{b\}_y^e \\ \{b\}_z^e \end{Bmatrix} \tag{2.2.15}$$



(a) Força distribuída no volume

(b) Força distribuída em uma área

(c) Força distribuída em uma linha

Figura 2.5: Generalização das cargas atuantes em um elemento finito paramétrico

Para o caso de forças de superfície $\{q\}$, em face de ξ constante (+1 ou -1), por exemplo (ver figura 2.5 (b)), tem-se:

$$\left\{ f \right\}_q^e = \int_{-1}^{+1} \int_{-1}^{+1} \left[\begin{array}{ccc} [N]^T & \cdot & \cdot \\ \cdot & [N]^T & \cdot \\ \cdot & \cdot & [N]^T \end{array} \right]_{\xi = const.} \left\{ q \right\} dS_e \tag{2.2.16}$$

sendo que

$$dS_e \; = \; \mid J \mid_{\xi = const.} d\eta d\zeta \tag{2.2.17}$$

com $\mid J \mid$ representando o determinante da transformação jacobiana do subdomínio em que o carregamento está aplicado.

$$\left\{ q \right\} = \left\{ \begin{array}{c} q_x(\xi,\eta,\zeta) \\ q_y(\xi,\eta,\zeta) \\ q_z(\xi,\eta,\zeta) \end{array} \right\} = \left[\begin{array}{ccc} [N] & \cdot & \cdot \\ \cdot & [N] & \cdot \\ \cdot & \cdot & [N] \end{array} \right]_{\xi = const.} \left\{ \begin{array}{c} \{q\}_x^e \\ \{q\}_y^e \\ \{q\}_z^e \end{array} \right\} \tag{2.2.18}$$

Para o caso de forças de linha $\{t\}$, em aresta com ξ e ζ constantes, por exemplo (ver figura 2.5 (c)), tem-se

$$\left\{ f \right\}_t^e = \int_{-1}^{+1} \left[\begin{array}{ccc} [N]^T & \cdot & \cdot \\ \cdot & [N]^T & \cdot \\ \cdot & \cdot & [N]^T \end{array} \right]_{\xi = const.,\ \zeta = const.} \left\{ t \right\} dL_e \qquad (2.2.19)$$

sendo que

$$dL_e = [(\frac{\partial x}{\partial \eta})^2 + (\frac{\partial y}{\partial \eta})^2]^{\frac{1}{2}} d\eta \qquad (2.2.20)$$

$$\left\{ t \right\} = \left\{ \begin{array}{c} t_x(\xi,\eta,\zeta) \\ t_y(\xi,\eta,\zeta) \\ t_z(\xi,\eta,\zeta) \end{array} \right\} = \left[\begin{array}{ccc} [N] & \cdot & \cdot \\ \cdot & [N] & \cdot \\ \cdot & \cdot & [N] \end{array} \right]_{\xi = const.,\ \zeta = const.} \left\{ \begin{array}{c} \{t\}_x^e \\ \{t\}_y^e \\ \{t\}_z^e \end{array} \right\} \qquad (2.2.21)$$

Pode-se mostrar que as integrais em 2.2.16 e 2.2.19 podem ser calculadas utilizando-se apenas as funções de forma relativas ao subdomínio em que o carregamento está aplicado. Assim, para o caso da figura 2.5 (b), basta usar as funções de forma do elemento quadrilateral de oito nós para obter o carregamento nodal equivalente à $q_x(\xi,\eta,\zeta)$. Da mesma forma basta usar as funções de forma do elemento unidimensional de três nós para obter o carregamento nodal equivalente à $t_z(\xi,\eta,\zeta)$ da figura 2.5 (c).

2.3 Integração Numérica

Quando se define a geometria do elemento através de interpolação das coordenadas nodais e/ou em elementos de ordem elevada, as integrações necessárias ao cálculo da matriz de rigidez e dos vetores de forças nodais equivalentes são muito elaboradas e laboriosas de serem levadas a efeito analiticamente, na grande maioria dos casos (Soriano & Lima 1999). Utiliza-se, então, a integração numérica.

A figura 2.6 ilustra o método mais simples para integração numérica da função f na

variável ξ, que é por retângulos. Adotando-se p pontos igualmente espaçados, escreve-se:

$$I = \int_{-1}^{+1} f(\xi) d\xi \;\cong\; \Delta\xi \sum_{i=1}^{p} \; f(\xi_i) \tag{2.3.1}$$

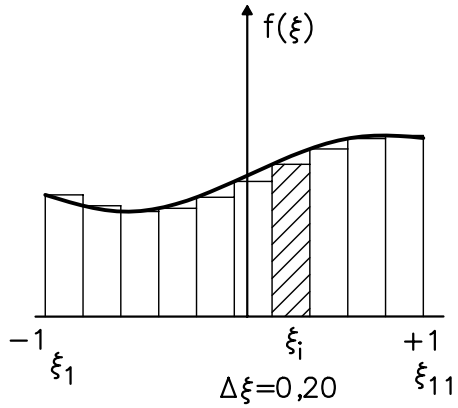


Figura 2.6: Integração numérica

Dentre os métodos de integração numérica unidimensional o mais eficiente para uma dada precisão, e largamente utilizado no método dos elementos finitos, é o método de Gauss-Legendre. Fixado um número p de pontos de cálculo do integrando no método de integração de Gauss na variável ξ, por exemplo, as posições destes pontos e os multiplicadores dos correspondentes valores do integrando, fatores-peso ω_i, encontram-se definidos para polinômios de várias ordens, visando a melhor precisão para a aproximação.

$$I = \int_{-1}^{+1} f(\xi) d\xi \;\cong\; \sum_{i=1}^{p_i} \; \omega_i \, f(\xi_i) \tag{2.3.2}$$

Esta equação expressa que a integração com p pontos é uma soma ponderada que requer a determinação de 2p incógnitas ω_i e ξ_i, que uma vez determinadas definem a integração exata de um polinômio de grau (2p-1).

Para integração em duas e em três variáveis independentes, em princípio poder-se-ia também determinar as posições dos pontos e correspondentes fatores-peso que conduzam ao melhor resultado para a integração numérica. Contudo, por se verificar que se obtém bons resultados, o usual é adotar sucessivamente a integração unidimensional de Gauss para cada uma das variáveis independentes desconsiderando-se a influência das demais.

Assim, faz-se para o caso bidimensional

$$I = \int_{-1}^{+1}\int_{-1}^{+1} f(\xi,\eta)d\xi d\eta \cong \int_{-1}^{+1}\left\{\sum_{i=1}^{p_i}\omega_i f(\xi_i,\eta)\right\}d\eta \;\cong \sum_{j=1}^{p_j}\omega_j\left\{\sum_{i=1}^{p_i}\omega_i f(\xi_i,\eta_j)\right\} = \\ = \sum_{i=1}^{p_i}\sum_{j=1}^{p_j}\omega_i\omega_j f(\xi_i,\eta_j) \quad (2.3.3)$$

sendo p_i e p_j os números de pontos de integração de Gauss nas direções ξ e η, respectivamente. A figura 2.7 mostra a localização de 2 e 3 pontos de integração em cada direção do caso plano.

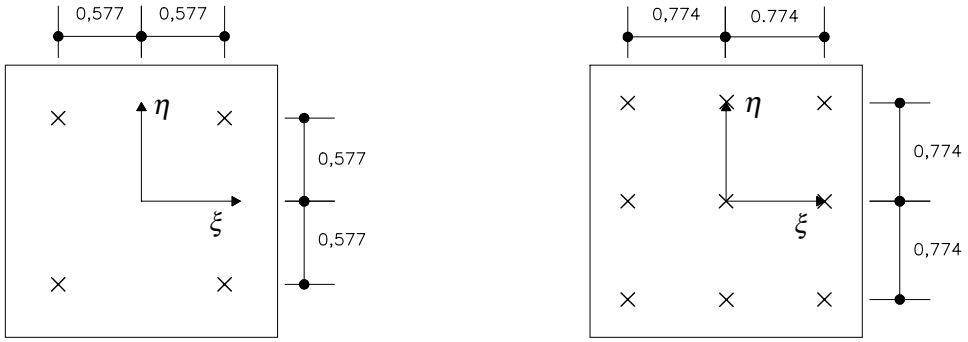


Figura 2.7: Localização de pontos de integração

De forma semelhante, escreve-se para o caso tridimensional

$$I = \int_{-1}^{+1}\int_{-1}^{+1}\int_{-1}^{+1} f(\xi,\eta,\zeta)d\xi d\eta d\zeta \;\cong\; \sum_{i=1}^{p_i}\sum_{j=1}^{p_j}\sum_{k=1}^{p_k}\omega_i\,\omega_j\,\omega_k\,f(\xi_i,\eta_j,\zeta_k) \qquad (2.3.4)$$

Para os elementos triangulares são desenvolvidos procedimentos numéricos de integração nas coordenadas triangulares. A figura 2.8 especifica a localização de pontos de integração para as integrações linear, quadrática e cúbica. Assim, escreve-se

$$I = \int_0^1\int_0^{1-\xi_1} f(\xi_1,\xi_2,\xi_3)\;d\xi_2\,d\xi_1 \;\cong\; \sum_{i=1}^{p}\;\omega_i\,f(\xi_1,\xi_2,\xi_3)_i \qquad (2.3.5)$$

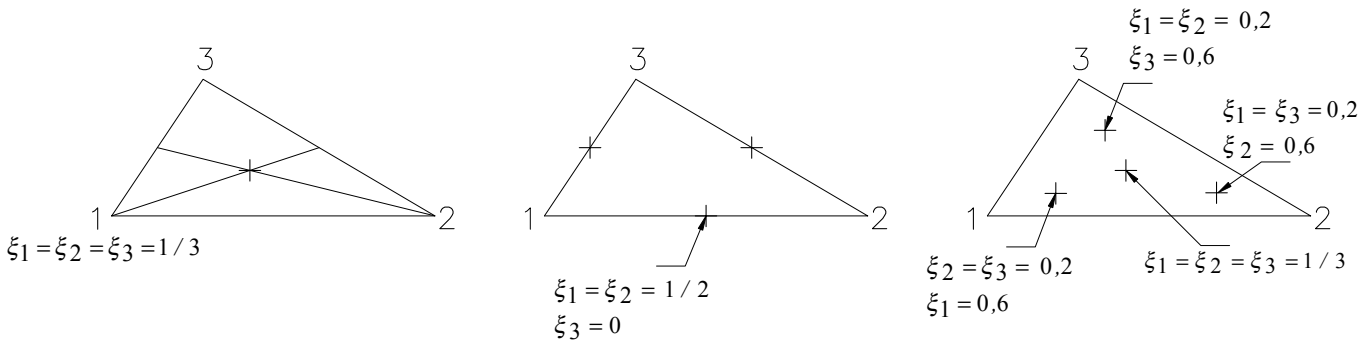


Figura 2.8: Localização dos pontos de integração em coordenadas triangulares

Semelhantemente ao caso anterior, para as coordenadas tetraédricas escreve-se

$$I = \int_0^1 \int_0^{1-\xi_1} \int_0^{1-\xi_1-\xi_2} f(\xi_1, \xi_2, \xi_3, \xi_4)\, d\xi_3 d\xi_2\, d\xi_1 \;\cong\; \sum_{j=1}^{p} \omega_j\, f(\xi_1, \xi_2, \xi_3, \xi_4)_j \qquad (2.3.6)$$

Capítulo 3

Recursos Utilizados no Desenvolvimento da Aplicação

3.1 Paradigma de Programação Orientada a Objetos

3.1.1 Coleções de Objetos

Muitas aplicações para as quais desejamos desenvolver um programa consistem em sistemas bastante complexos. Uma maneira natural de lidar com a complexidade de um sistema é dividi-lo em subsistemas mais simples, de maneira que o comportamento do sistema, como um todo, possa ser expresso em termos dos comportamentos de seus subsistemas e das interações entre eles.

Para que essa abordagem possa ser espelhada diretamente em um programa para simulação ou controle de um sistema, linguagens de programação mais modernas oferecem recursos para construção de um programa como uma coleção de componentes de programa, com interfaces bem definidas, que especificam as interações entre esses componentes.

No paradigma de programação orientada por objetos, a construção de um programa para implementação de um determinado sistema baseia-se em uma correspondência natural e intuitiva entre esse sistema e a simulação do comportamento do mesmo: a cada

entidade do sistema corresponde, durante a execução do programa, *um objeto*, com atributos e comportamento descritos por um componente desse programa.

O desenvolvimento de software para implementação de um sistema envolve fases de análise, projeto e implementação desse sistema. O princípio em que se baseia o paradigma de orientação por objetos, o de que existe uma correspondência entre componentes do sistema e objetos, torna mais simples esse processo. Objetos constituem limites naturais para construções de *abstrações de dados*: todas as informações referentes a uma dada entidade são confinadas em um determinado objeto, que se relaciona com outros objetos mediante uma interface bem definida.

A maioria das linguagens de programação orientadas por objetos usa o conceito de *classe*, para descrição de grupos de objetos semelhantes. Um programa nessas linguagens consiste em uma coleção de definições de classes, que descrevem os objetos que implementam entidades de um sistema (Camarão & Figueiredo 2003).

3.1.2 Classes e Objetos

Uma classe é um componente de programa que descreve a ”estrutura” e o ”comportamento” de um grupo de objetos semelhantes - isto é, as informações que caracterizam o estado desses objetos e as ações (ou operações) que eles podem realizar. Os objetos de uma classe - também chamados de *instâncias* da classe - são criados durante a execução de programas.

Uma classe é formada, essencialmente, por *construtores de objetos* dessa classe, variáveis e métodos. A criação de um objeto dessa classe consiste na criação de cada uma das variáveis do objeto, especificadas na classe. Os valores armazenados nessas variáveis determinam o *estado* do objeto. Uma variável de um objeto é também chamada de ”atributo” desse objeto.

Objetos podem ”receber mensagens”, sendo uma mensagem basicamente uma chamada a um método específico de um objeto, que realiza uma determinada operação, em geral dependente do estado desse objeto. A execução de uma chamada a um método de um objeto pode modificar o estado desse objeto, isto é, modificar os valores dos seus atributos,

e pode retornar um resultado (Camarão & Figueiredo 2003).

3.1.3 Abstração

Abstrair significa decompor um sistema complicado em suas partes fundamentais e descrevê-las em uma linguagem simples e precisa. A descrição das partes de um sistema implica atribuir-lhes um nome e descrever suas funcionalidades. Por exemplo, a interface gráfica com o usuário de um editor de textos compreende a abstração de um menu "editar" que oferece várias opções de edição de texto incluindo recortar e colar porções de texto ou outros objetos gráficos. Sem entrar em detalhes sobre como uma interface gráfica com o usuário representa e exibe textos ou objetos gráficos, os conceitos de "recortar" e "colar" são simples e precisos. Uma operação de recorte apaga o texto ou gráfico selecionado e o coloca em uma área de armazenamento externa. A operação de colagem insere o conteúdo externamente armazenado em uma localização específica do texto. Dessa forma, a funcionalidade abstrata do menu "editar" e suas operações de recortar e colar são definidas em uma linguagem precisa o suficiente para ser clara e simples o bastante para "abstrair" os detalhes desnecessários. Essa combinação de clareza e simplicidade traz benefícios à robustez, uma vez que leva a implementações corretas e compreensíveis (Goodrich & Tamassia 2002).

3.1.4 Encapsulamento

Outro princípio importante em projeto orientado a objetos é o conceito de *encapsulamento*, que estabelece que os diferentes componentes de um sistema de software não devem revelar detalhes internos de suas respectivas implementações. Analisemos novamente o exemplo do menu "editar" da interface gráfica com o usuário de um editor de textos, com suas opções "recortar" e "colar". Uma das razões pelas quais o menu "editar" é tão útil é porque compreendemos perfeitamente como usá-lo sem entender como é implementado. Não precisamos saber como o menu é desenhado, como o texto selecionado para ser recortado ou colado é representado, como essas porções de um texto são armazenadas na área externa ou como os diferentes objetos tais como gráficos, imagens ou desenhos são

identificados, armazenados e transferidos para a e da área externa. Desta forma, o código associado com o menu ”editar” não depende necessariamente de todos esses detalhes para funcionar corretamente. Em vez disso, o menu ”editar” deveria oferecer uma interface suficientemente específica para que outros componentes de software usassem seus métodos de forma efetiva, pedindo, ao mesmo tempo, interfaces bem definidas dos outros componentes de software que necessita. Genericamente, o princípio do encapsulamento propõe que todos os componentes de um grande sistema de software operem dentro de uma filosofia de conhecer o mínimo necessário sobre os demais.

Uma das maiores vantagens do encapsulamento é que ele oferece ao programador liberdade na implementação dos detalhes do sistema. A única restrição ao programador é manter a interface abstrata que é percebida pelos de fora. Por exemplo, o programador do código do menu ”editar” da interface gráfica com o usuário de um editor de textos pode, em um primeiro momento, implementar as operações de copiar e colar copiando e restaurando telas para a área externa de armazenamento. Mais tarde, pode ficar insatisfeito com essa implementação, uma vez que não permite um armazenamento compacto da seleção e não distingue objetos gráficos de textos. Se o programador tiver projetado a interface das operações de copiar e colar tendo em mente o encapsulamento, trocar a implementação por uma que armazene o texto como texto e os objetos gráficos em uma forma compacta apropriada não irá causar nenhum problema aos métodos que necessitam interagir com esta interface gráfica com usuário. Dessa forma, encapsulamento permite a adaptação porque autoriza a alteração de detalhes de partes de um programa sem afetar de forma negativa outros componentes (Goodrich & Tamassia 2002).

3.1.5 Modularidade

Além de abstração e do encapsulamento, outro princípio fundamental de projeto orientado a objetos é a *modularidade*. Sistemas modernos de software normalmente estão compostos por vários componentes diferentes que devem interagir corretamente, fazendo com que o sistema como um todo funcione de forma adequada. Para se manter essas interações corretas é necessário que os diversos componentes estejam bem organizados.

Na abordagem orientada a objetos, essa organização se centra no conceito de *modularidade*. A modularidade se refere a uma estrutura de organização na qual os diferentes componentes de um sistema de software são divididos em unidades funcionais separadas. Por exemplo, uma casa ou um apartamento podem ser vistos como sendo compostos por várias unidades funcionais: sistema elétrico, aquecimento e refrigeração, encanamentos e estruturas. Ao invés de ver esses sistemas como uma mixórdia de fios, respiradouros, tubos e quadros, o arquiteto que projetar uma casa ou apartamento de forma organizada os verá como módulos separados que interagem de uma forma bem definida. Ao fazer isso, está usando a modularidade para obter uma clareza de idéias que forneçam uma forma natural de organizar funções em unidades gerenciáveis distintas. Assim, o uso de modularidade em sistemas de software também pode oferecer uma ferramenta poderosa de organização que traz clareza para uma implementação.

A estrutura imposta pela modularidade auxilia a tornar o software reutilizável. Se os módulos do software forem escritos de uma forma abstrata para resolver problemas genéricos, então os módulos podem ser reutilizados quando instâncias do mesmo problema geral surgirem em outros contextos. Por exemplo, a estrutura de definição de uma parede é a mesma de casa para casa, sendo normalmente definida em termos de tipo de isolamento desejado, tipo de acabamento etc. O arquiteto organizado pode, assim, reutilizar suas definições de parede de uma casa para outra. Ao reutilizar tais definições, algumas partes podem exigir adaptações, por exemplo, uma parede em um edifício comercial pode ser similar à de uma casa, mas o sistema elétrico pode ser diferente. Sendo assim, nosso arquiteto pode querer organizar os vários componentes, tais como os componentes elétricos e as estruturas, de uma forma *hierárquica*, que agrupem definições abstratas similares em níveis, partindo da mais específica para a mais geral, na medida em que se percorre a hierarquia. Esse tipo de hierarquia também é útil no projeto de software, quando agrupa funcionalidades comuns no nível mais geral e vê comportamentos especializados como uma extensão do comportamento geral (Goodrich & Tamassia 2002).

3.1.6 Herança

Para evitar código redundante, o paradigma de orientação a objetos oferece uma estrutura hierárquica e modular para reutilização de código através de uma técnica conhecida como *herança*. Esta técnica permite projetar classes genéricas que podem ser especializadas em classes mais particulares, onde as classes especializadas reutilizam o código das mais genéricas. A classe genérica, também conhecida por *classe base* ou *superclasse*, define variáveis de instância "genéricas" e métodos que se aplicam em uma variada gama de situações. A classe que *especializa*, ou *estende* ou *herda* de uma superclasse não necessita fornecer uma nova implementação para os métodos genéricos, uma vez que os herda. Deve apenas definir aqueles métodos que são especializados para esta *subclasse* em particular (também conhecida com classe *derivada*) (Goodrich & Tamassia 2002).

3.1.7 Polimorfismo

Literalmente, "polimorfismo" significa "muitas formas". No contexto de projeto orientado a objetos, entretanto, refere-se à habilidade de uma variável de objeto de assumir formas diferentes. Linguagens orientadas a objetos referenciam objetos usando variáveis referência. Uma variável referência o deve especificar que tipo de objeto ela é capaz de referenciar em termos de uma classe S. Isso implica, entretanto, que o também pode se referir a qualquer objeto pertencente à classe T derivada de S. Analise agora o que acontece se S define um método a() e T também define um método a(). A seqüência de ativação de métodos sempre é iniciada com a busca pela classe mais restritiva à qual se aplica. Ou seja, quando o se refere a um objeto da classe T e o.a() é invocado, então será ativada a versão de T do método a(), em lugar da versão de S. Neste caso, diz-se que T ***sobrescreve*** o método a() de S. Por outro lado, se o se refere a um objeto da classe S (que, ao contrário, não é um objeto da classe T), quando o.a() for ativado, será executada a versão de S de a(). Um polimorfismo como esse é útil porque aquele que chama o.a() não precisa saber quando o se refere a uma instância de T ou S para poder executar a versão correta de a(). Dessa forma, a variável de objeto o pode ser *polimórfica*, ou assumir muitas formas, dependendo da classe específica dos objetos aos quais está se

referindo. Esse tipo de funcionalidade permite a uma classe especializada T estender uma classe S, herdar os métodos genéricos de S e redefinir outros métodos de S, de maneira que sejam incluídos como propriedades específicas dos objetos T.

Algumas linguagens orientadas a objetos também oferecem um tipo de polimorfismo "em cascata", que é mais precisamente conhecido como ***sobrecarga*** de métodos. A sobrecarga ocorre quando uma única classe T tem vários métodos como o mesmo nome, desde que cada um tenha uma *assinatura* diferente. A assinatura de um método é uma combinação entre seu nome e o tipo e a quantidade de argumentos que são passados para o mesmo. Dessa forma, mesmo que vários métodos de uma classe tenham o mesmo nome, eles são distinguíveis pelo compilador pelo fato de terem diferentes assinaturas, ou seja, na verdade são desiguais. Em linguagens que possibilitam a sobrecarga de métodos, o ambiente de execução determina qual método ativar para uma determinada chamada de método que percorre a hierarquia de classes em busca do primeiro método cuja assinatura combine com a do método que está sendo invocado. Por exemplo, imagine uma classe T que define o método a(), derivada da classe U que define o método a(x,y). Se um objeto o da classe T recebe a mensagem "o.a(x,y)", então a versão de U do método a() é ativada (com os dois parâmetros x e y). Assim, o verdadeiro polimorfismo aplica-se apenas a métodos que têm a mesma assinatura mas estão definidos em classes diferentes.

A herança, o polimorfismo e a sobrecarga de métodos suportam o desenvolvimento de software reutilizável. Podemos estabelecer classes que herdam as variáveis e os métodos de instância genéricos e que podem, a seguir, definir novas variáveis e métodos de instância mais específicos que lidam com os aspectos particulares dos objetos da nova classe (Goodrich & Tamassia 2002).

3.2 Linguagem Java

Dentre as linguagens que suportam o paradigma de programação orientada a objetos, as mais utilizadas são C++ e Java. Alguns estudos indicam que Java é mais apropriada devido a vários aspectos que não são encontrados em C++. Entre estes aspectos, quatro são particularmente relevantes e foram analisados durante o processo de escolha da

linguagem Java para implementação deste trabalho de dissertação. Estes aspectos são: independência de sistema operacional; performance numérica; capacidade de reutilização do software e suporte à persistência dos dados.

3.2.1 Portabilidade

Java independe do sistema operacional, pois utiliza um processo diferente da compilação ou interpretação tradicionalmente conhecidos.

Um interpretador é, como o nome indica, um programa que interpreta diretamente as frases do programa fonte, isto é, simula a execução dos comandos desse programa sobre um conjunto de dados, também fornecidos como entrada para o interpretador. A interpretação de programas escritos em uma determinada linguagem define uma "máquina virtual", na qual é realizada a execução de instruções dessa linguagem.

A interpretação de um programa em linguagem de alto nível pode ser centenas de vezes mais lenta do que a execução do código objeto gerado para esse programa pelo compilador. A razão disso é que o processo de interpretação envolve simultaneamente a análise e simulação da execução de cada instrução do programa, ao passo que essa análise é feita previamente, durante a compilação, no segundo caso. Apesar de ser menos eficiente, o uso de interpretadores muitas vezes é útil principalmente devido ao fato de que, em geral, é mais fácil desenvolver um interpretador do que um compilador para uma determinada linguagem.

Esse aspecto foi explorado pelos projetistas da linguagem Java, no desenvolvimento de sistemas (ou ambientes) para programação e execução de programas nessa linguagem: esses ambientes são baseados em uma combinação dos processos de compilação e interpretação. Um ambiente de programação Java é constituído de um compilador Java, que gera um código de mais baixo nível, chamado de *bytecodes*, que é então interpretado. Um interpretador de *bytecodes* interpreta instruções da chamada "Máquina Virtual Java", nome abreviado como *JVM*. Esse esquema usado no ambiente de programação Java não apenas contribuiu para facilitar a implementação da linguagem em um grande

número de computadores diferentes, mas constitui uma característica essencial no desenvolvimento de aplicações voltadas para a internet, pois possibilita que um programa compilado em determinado computador possa ser transferido através da rede e executado em qualquer outro computador que disponha de um interpretador de *bytecodes* (Camarão & Figueiredo 2003).

3.2.2 Comparação de Performance entre Java e C++

No artigo (Nikishkov, Nikishkov & Savchenko 2003) foi comparada a performance do código para elementos finitos desenvolvido em Java e do código análogo em C++, para solução de problemas de elasticidade tridimensional. Para executar o código Java foram feitos testes empregando as versões 1.1, 1.2, 1.3 e 1.4 da *JVM*, mostrando que o uso de diferentes *Máquinas Virtuais Java* pode levar a uma diferença considerável de performance.

Para o experimento foi resolvido o problema de um cubo elástico tridimensional submetido à tração simples. Para a discretização do problema foi utilizado o elemento de *tijolo* de 20 nós. O número de graus de liberdade (DOF) da discretização foi variado de 1275 a 24843. O computador utilizado no teste foi um *Desktop* com processador *Intel Pentium 4* com capacidade 1.8 GHz e sistema operacional *Windows XP Professional*. O código C++ foi compilado usando *Microsoft Visual C++ 6.0* com máxima velocidade de otimização. O código Java foi compilado usando o compilador *javac* desenvolvido pela *Sun Microsystems* e rodado usando *JVM´s* 1.1.8, 1.2.2-011 com Symantec JIT compiler, Java HotSpot Client VM $1.3.1 - 02 - b02$ e Java HotSpot Client VM 1.4.0-b92.

A figura 3.1 mostra os resultados obtidos para o cálculo da matriz de rigidez. Os valores do gráfico são referentes ao coeficiente tempo C++/Java. Observando o gráfico, percebe-se que a melhor JVM para resolver o problema é a 1.2 e que essa é ainda mais eficiente do que C++.

A figura 3.2 mostra os resultados obtidos para montagem da matriz de rigidez esparsa. Novamente JVM 1.2 mostra-se mais eficiente do que o compilador C++.

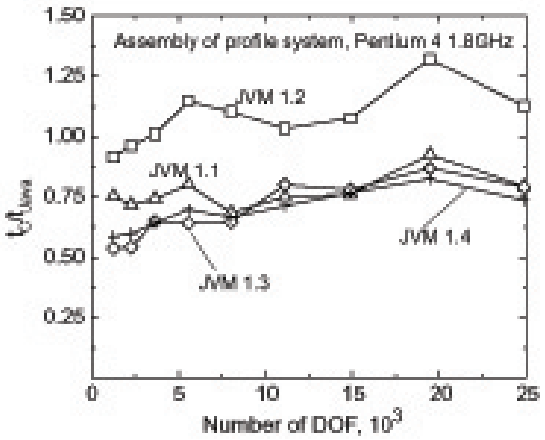


Figura 3.1: Eficiência para cálculo da matriz de rigidez

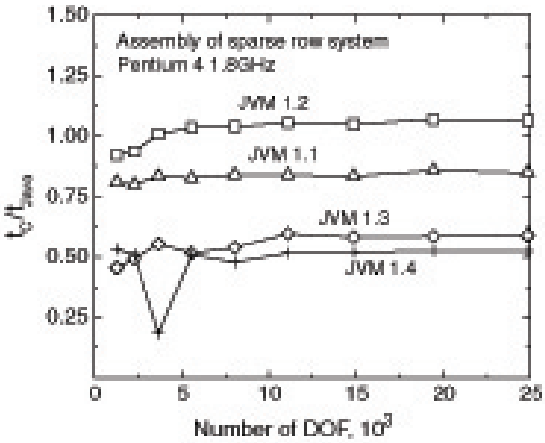


Figura 3.2: Eficiência para montagem da matriz de rigidez esparsa

3.2.3 Capacidade de Reutilização de Software em Java

Programadores Java concentram-se na elaboração de novas classes e reutilização de classes existentes. Existem muitas bibliotecas de classe e outras estão sendo desenvolvidas em todo o mundo. O software é, então, construído a partir de componentes amplamente disponíveis, portáveis, bem-documentados, cuidadosamente testados e bem definidos. Esse tipo de capacidade de reutilização de software acelera o desenvolvimento de programas poderosos e de alta qualidade (Deitel & Deitel 2001).

Para perceber o potencial completo da capacidade de reutilização de software, precisa-se aprimorar os esquemas de catalogação, os esquemas de licença, os mecanismos de proteção que assegurem que as cópias-mestras das classes não sejam corrompidas, os esquemas de descrição que projetistas de sistema utilizam para determinar se objetos existentes

atendem às necessidades, os mecanismos de navegação que determinam as classes que estão disponíveis e o grau em que elas atendem aos requisitos de desenvolvimento de software, e assim por diante. Muitos problemas interessantes de pesquisa e desenvolvimento foram solucionados e muitos outros necessitam ser resolvidos. Esses problemas acabarão sendo resolvidos de uma forma ou de outra, uma vez que o valor potencial da reutilização de software é enorme (Deitel & Deitel 2001).

3.3 Persistência de Dados com XML

O armazenamento de dados em variáveis e arranjos (vetores e matrizes) é temporário - os dados são perdidos quando uma variável local "sai do escopo" ou quando o programa termina. Arquivos são utilizados para retenção a longo prazo de grandes quantidades de dados, mesmo depois de terminar a execução do programa que criou os dados. Os dados mantidos em arquivos são freqüentemente chamados de dados persistentes (Deitel & Deitel 2001).

A adoção da web como veículo de acesso a sistemas de informação trouxe novamente a preocupação com a estrutura dos documentos. Primeiro, para fornecer o mesmo conteúdo em formatos alternativos, personalizados para computadores *Desktop*, celulares, auto-atendimento telefônico ou para impressão em papel; segundo, para possibilitar o acesso às informações por outras aplicações, em vez de apenas por usuários humanos (Lozano 2003).

Estudando as formas disponíveis atualmente para armazenar dados persistentes, a mais indicada para implementação deste trabalho é o padrão XML (eXtensible Markup Language). O XML é um formato padronizado de arquivo texto, projetado para escrever e estruturar dados.

Se o XML fosse apenas "mais uma forma" de gerar sites web não teria feito tanto sucesso. O grande diferencial está na possibilidade de se processar a informação contida no documento original, ignorando a formatação fornecida pelas folhas de estilo CSS ou XSLT, tornando o XML um formato universal para importação e exportação de dados (Lozano 2003).

O XML gerou um conjunto de tecnologias rico e útil para uma vasta gama de aplicações. Não há revolução alguma no XML, mas apenas novas maneiras de realizar tarefas que já eram possíveis antes, com outras tecnologias. O diferencial é que as novas maneiras são portáveis, independentemente de linguagem de programação ou sistema operacional e baseadas em padrões abertos (Lozano 2003).

A plataforma de desenvolvimento Java oferece todas as API´s (Aplication Program Interfaces) necessárias à escrita de programas capazes de ler, criar e editar documentos XML. Tais API´s permitem a leitura e a escrita dos documentos em arquivos, conexões TCP/IP, Strings e outros meios de Entrada/Saída (Liesenfeld 2002).

3.4 Representação Gráfica na POO - A UML

A apresentação gráfica de um programa orientado a objetos é um artifício muito utilizado para facilitar a visualização das entidades e suas relações. Dentre as diversas linguagens gráficas disponíveis, a mais sistematicamente elaborada, sendo, também, a mais aceita, é a *Unified Modelling Language* (UML). A simbologia de UML adotada neste trabalho é brevemente explicada.

A figura 3.3 mostra um diagrama de classe UML. O diagrama é dividido em três campos. O campo superior contém o nome da classe; no campo abaixo se declaram as variáveis daquela classe, enquanto que no último campo se declaram os operadores (métodos) dessa classe.

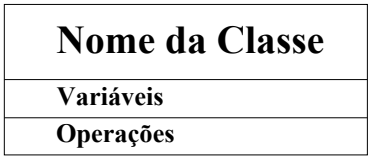


Figura 3.3: Diagrama de classe na UML

A figura 3.4 mostra um diagrama de herança, no qual pode-se visualizar duas subclasses derivando da superclasse. Neste trabalho, adota-se o critério de representar as classes que deverão ser criadas ou modificadas em destaque como o exemplo da subclasse 2.

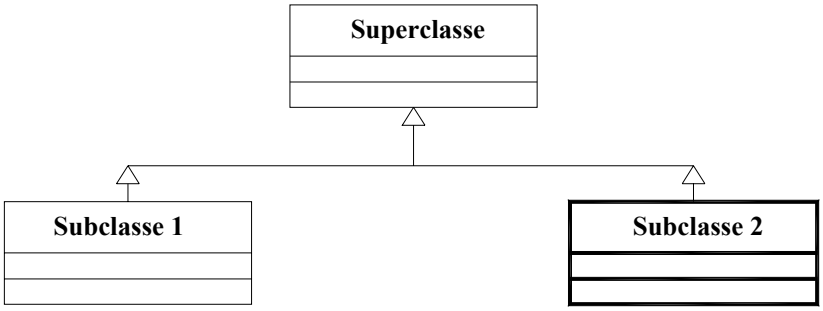


Figura 3.4: Diagrama de herança UML

A figura 3.5 mostra um diagrama de instâncias de uma dada classe. Ao lado das linhas anota-se o número de relações entre as classes. As relações são as seguintes:

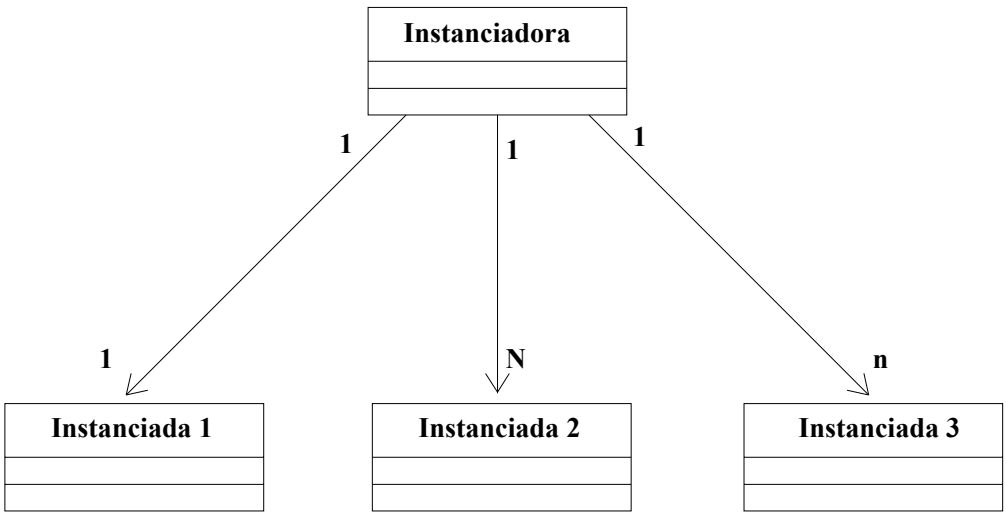


Figura 3.5: Diagrama de instâncias na UML

i. Classe 1: a relação é de um para um, o que significa que um objeto da classe instanciadora se relaciona com um objeto da classe instanciada;

ii. Classe 2: a relação é de um para ´N´, onde ´N´ é um número conhecido. Isso significa que um objeto da classe instanciadora se relaciona com um número definido ´N´ de objetos da classe instanciada;

iii. Classe 3: a relação é de um para ´n´, onde ´n´ é um número indefinido. Isso significa que um objeto da classe instanciadora se relaciona com um número indefinido de objetos da classe instanciada.

Capítulo 4

Análise Orientada a Objetos para a Formulação Paramétrica do MEF

O processo de desenvolvimento de software orientado a objetos compreende três fases principais. Inicialmente, na fase de análise orientada a objetos, procura-se identificar as classes com seus possíveis atributos e operações, que satisfaçam os requisitos e especificações do sistema. A segunda fase, denominada projeto orientado a objetos, prepara a implementação definindo os módulos de software e identificando as interações entre os mesmos. A fase de programação orientada a objetos é a terceira fase e refere-se à implementação do projeto do software em uma linguagem de programação que suporte o paradigma.

Este capítulo procede a análise orientada a objetos para a formulação paramétrica do método dos elementos finitos. A partir de diversos trabalhos disponíveis na literatura (Lichao & Ashok 2001), (Martha, Menezes, Lages, Parente & Pitangueira 1996); entre outros e de reflexões mais atuais, procurar-se-á identificar as principais classes concernentes com a referida formulação. Dentre estes trabalhos destaca-se o FEMOOP ("Finite Element Method Object Oriented Program") como principal fonte inspiradora da análise orientada a objetos apresentada a seguir. O FEMOOP é um programa de elementos finitos, escrito em C++, que teve desenvolvimento inicial no Departamento de Engenharia Civil da Puc-Rio e que vem sendo utilizado em diferentes pesquisas em diversas universidades brasileiras ((Guimarães 1992), (Neto 1994), (Barros 1994), (Sybine 1997), (Lages 1997), (Pitangueira 1998), (Noronha 1998), (Júnior 2000), (Holanda 2000), (Silva 2001), (Simão 2003), (Fuina 2004)).

O conceito mais significativo do Método dos Elementos Finitos é, como o próprio nome sugere, o conceito de elemento. Assim é natural a existência da *classe elemento*. Lembrando-se que um elemento possui pontos nodais, um material e funções de forma, pode-se evoluir a análise, criando-se outras classes correlatas. Entretanto, para maior clareza e enriquecimento da análise, é relevante referir-se à equação(2.2.13), abaixo repetida, que permite calcular a matriz de rigidez de um elemento finito:

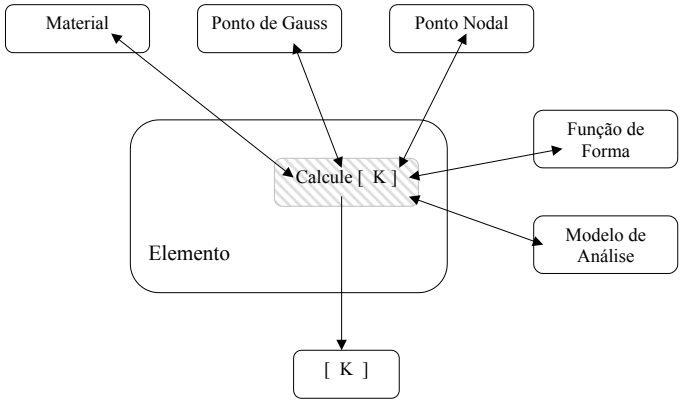
$$[k]^e = \int_{-1}^{+1}\int_{-1}^{+1}\int_{-1}^{+1} [B]^T[E][B] \mid J \mid d\xi d\eta d\zeta \tag{4.0.1}$$

A equação (4.0.1) revela que a obtenção da matriz de rigidez de um elemento depende de propriedades do material (para montagem da matriz $[E]$) e de derivadas das funções de forma (para montagem da matriz $[B]$) que, por sua vez, dependem dos pontos nodais, justificando assim a criação das *classes material*, *função de forma* e *ponto nodal*.

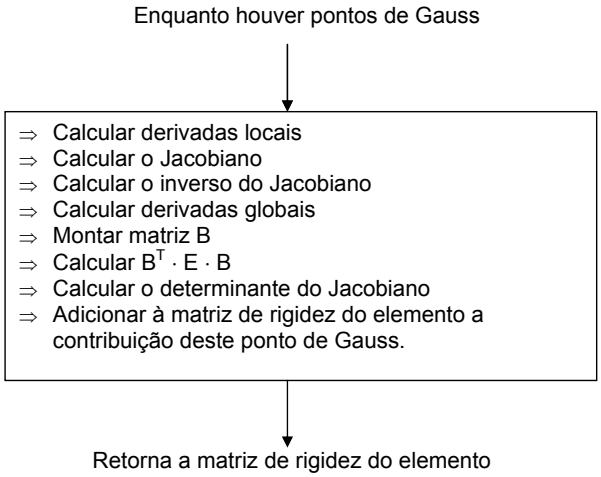
A equação (4.0.1) também mostra a necessidade de integração nas coordenadas adimensionais ξ, η e ζ. Lembrando que a quadratura de Gauss é a técnica mais empregada para proceder a referida integração e que a mesma se baseia na existência de pontos de integração internos ao elemento e pesos a estes associados, é razoável criar a *classe ponto de integração*.

Outro aspecto relevante, não tão explícito na equação (4.0.1), é o processo de montagem das matrizes $[E]$ e $[B]$. Os tamanhos destas matrizes, bem como o arranjo dos parâmetros do material e das derivadas das funções de forma para a formação das mesmas dependem do modelo de análise do elemento finito. Diferentes arranjos das propriedades do material originam diferentes matrizes $[E]$, se o modelo de análise é de estado plano de tensão ou estado plano de deformação. Diferentes arranjos das derivadas das funções de forma originam diferentes matrizes $[B]$, se o modelo de análise é axissimétrico ou sólido. Assim, como sugerido por Martha et al.(1996), é fundamental a criação da *classe modelo de análise*. A figura 4.1 ilustra a interação entre as classes concebidas, bem como o algoritmo para montagem da matriz de rigidez de um elemento paramétrico.

Uma análise semelhante à feita acima pode ser adotada para o cálculo do carregamento



(a) Interação entre as classes



(b) Algoritmo

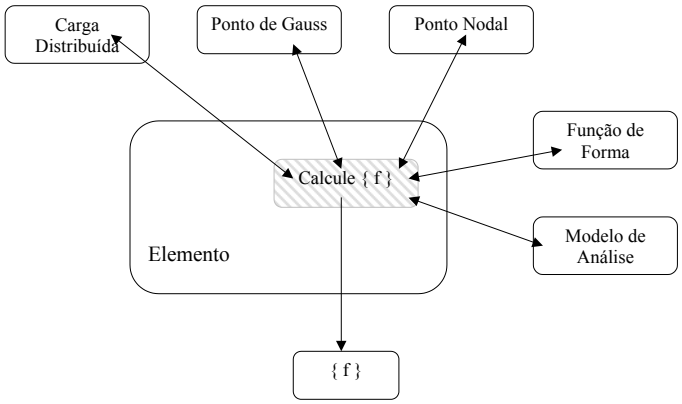
Figura 4.1: Montagem da matriz de rigidez de um elemento paramétrico

nodal equivalente, partindo-se de:

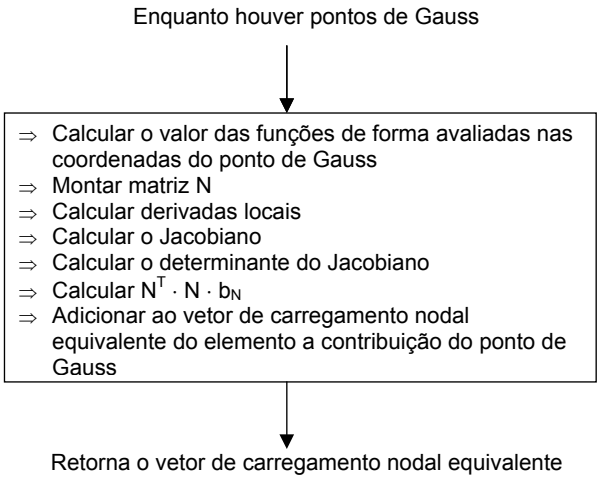
$$\{f\}^e = \int_s [N]^T [N] \{b\}_N \, ds \tag{4.0.2}$$

Esta equação mostra uma dependência com a matriz das funções de forma $[N]$ e com valores das cargas distribuídas prescritas nos nós $\{b\}_N$, além da necessidade de integração numérica. Assim, a equação (4.0.2), além de corroborar a criação das classes já mencionadas, aponta para a necessidade de uma classe que represente os valores das cargas distribuídas, prescritos nos nós do elemento. A figura 4.2 ilustra a interação entre as classes concebidas, bem como o algoritmo para montagem do carregamento nodal equivalente de um elemento paramétrico. É importante ressaltar que, para determinado

elemento finito, os pontos de integração, as funções de forma e o modelo de análise mostrados na figura 4.2 (a) não são os mesmos da figura 4.1 (a). Entretanto, cada elemento deve caracterizar estas três grandezas para tratamento de suas cargas de linha, de superfície e de volume. Esta análise indica a necessidade de uma classe (*Integração Paramétrica*) para proceder a integração do carregamento nodal equivalente que, através do mecanismo de delegação (Santos 2003), possa ser utilizada por cada elemento finito.



(a) Interação entre as classes



(b) Algoritmo

Figura 4.2: Montagem do carregamento nodal equivalente de um elemento paramétrico

Além das classes necessárias à caracterização de um elemento finito, há que se preocupar com aquelas relativas ao modelo discreto como um todo. Assim, uma classe responsável pelas diversas coleções de objetos de um modelo discreto (elementos, nós, materiais etc) é necessária. Esta classe denominada *modelo* cria os diversos objetos da discretização através da interação com os dados persistidos em arquivos XML. Com o

modelo discreto gerenciado pela *classe modelo*, a solução do mesmo precisa ser adequadamente tratada, justificando assim a criação de uma classe específica para este fim (*classe solução*). Finalmente, resta criar uma classe responsável pela definição do problema a ser resolvido. Esta classe faz requisições apropriadas às classes *modelo* e *solução* de maneira a produzir os resultados desejados para determinado problema (mecânica estrutural, transferência de calor, entre outros). A classe com estas características é denominada *classe controladora do problema*.

Além dos algoritmos mostrados nas figuras 4.1 (b) e 4.2 (b), vários outros algoritmos genéricos, independentes do tipo de elemento, aparecem em qualquer modelo do MEF. São os algoritmos de montagem das matrizes globais a partir das matrizes dos elementos, baseados na técnica da rigidez direta. Tais algoritmos, na análise orientada a objetos que aqui se discute, são de responsabilidade da *classe modelo*.

A tabela 4.1 apresenta as classes concebidas, resumindo a finalidade de cada uma.

Tabela 4.1: Classes criadas na análise orientada a objetos do sistema

Classe	**Finalidade**
Elemento	Armazenar e gerenciar outras classes que serão seus atributos: nós, conectividade, pontos de Gauss etc.
Material	Definir as propriedades físicas do material
Função de Forma	Definir as funções de forma com suas derivadas para os elementos finitos paramétricos
Ponto Nodal	Definir os atributos pertencentes a um determinado nó: coordenadas cartesianas, restrições etc
Ponto de Gauss	Definir os atributos de um ponto de Gauss: coordenadas adimensionais e peso
Modelo de Análise	Definir o tamanho e arranjo das matrizes $[E]$ e $[B]$ para os diferentes tipos de análise
Modelo	Definir as diversas coleções de objetos que representarão a discretização (elementos, nós, materiais etc)
Solução	Definir os diferentes tipos de soluções
Controladora do Problema	Gerenciar as classes *Modelo* e *Solução* para obtenção dos resultados desejados de acordo com determinado problema
Carga Distribuída	Definir os atributos comuns às cargas distribuídas por unidade de comprimento, área e volume
Integração Paramétrica	Obter as cargas nodais equivalentes para as forças por unidade de comprimento, área e volume

Capítulo 5

Projeto Orientado a Objetos

As classes concebidas na análise orientada a objetos anterior (ver tabela 4.1) são denominadas no programa conforme a tabela 5.1.

Tabela 5.1: Denominações adotadas para as classes

Classe	**Denominação no Programa**
Elemento	Element (∗)
Material	Material (∗)
Função de Forma	Shape (∗)
Ponto Nodal	Node
Ponto de Gauss	IntegrationPoint (∗)
Modelo de Análise	AnalysisModel (∗)
Modelo	FemModel
Solução	Solution (∗)
Controladora do Problema	Driver (∗)
Integração Paramétrica	ParametricIntegration (∗)
Carga Distribuída	ElementForce
Carga Concentrada	PointForce

(∗) classes que serão estendidas através do mecanismo de herança.

5.1 Hierarquia de Classes

As classes marcadas com (∗) na tabela 5.1 são classes genéricas e portanto, precisam ser estendidas para caracterização dos diferentes casos particulares. Mecanismos de herança são então utilizados para este fim e diagramas UML (Unified Modelling Language) apropriados (Fowler & Scott 2000) são adotados para representá-los.

As figuras 5.1, 5.2, 5.3, 5.4, 5.5, 5.6, 5.7 e 5.8 mostram as hierarquias projetadas para as classes marcadas com (∗) na tabela 5.1. Nestas figuras, as subclasses destacadas

indicam que somente elas são implementadas na dissertação de mestrado que aqui se apresenta. A figura 5.1 mostra a hierarquia da classe *Driver* com subclasses apropriadas para os diversos tipos de problemas que podem ser modelados através do MEF, tais como problema de análise de tensões (contemplado com a subclasse *StructuralMech*, única implementada neste trabalho), de transferência de calor (subclasse *HeatTransfer*) e de mecânica dos fluidos (subclasse *FluidFlow*).

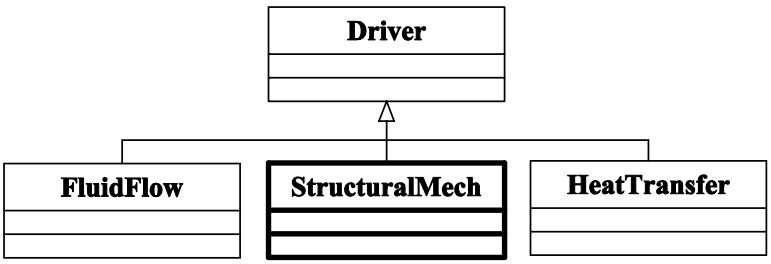


Figura 5.1: Hierarquia da classe *Driver*

A figura 5.2 mostra a hierarquia da classe *Solution* cuja finalidade é realizar as operações matemáticas necessárias à obtenção da solução requerida para o modelo como, por exemplo, a solução de equilíbrio (subclasse *Equilibrium*). Neste trabalho somente é implementada a classe *OnePointEq*, que obtém a solução por equilíbrio em um ponto.

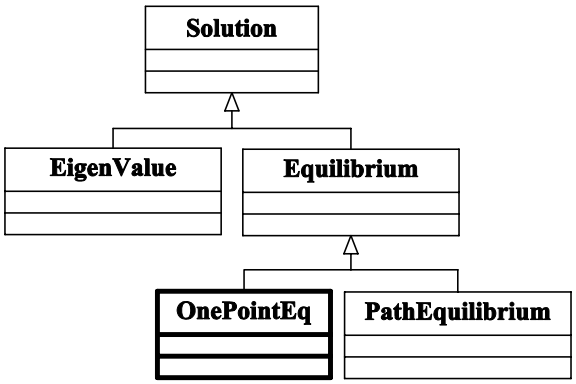


Figura 5.2: Hierarquia da classe *Solution*

A figura 5.3 mostra a hierarquia da classe *Element* e a subclasse criada para implementação da formulação paramétrica (subclasse *ParametricElement*) juntamente com as classes concebidas para representar os elementos de barra (subclasse *Line*), quadriláteros (subclasse *Quadrilateral*), triangulares (subclasse *Triangular*) e hexaédricos (subclasse

Hexahedral). Esta terceira camada da herança foi necessária para a criação adequada dos pontos de integração e montagem da matriz de coordenadas nodais de cada subclasse. A última camada da hierarquia da figura 5.3 mostra os diversos elementos paramétricos implementados: *ElmL2*, *ElmL3* e *ElmL4* são elementos de linha com 4, 8 e 9 nós, respectivamente; *ElmQ4*, *ElmQ8* e *ElmQ9* são elementos quadrilaterais planos com 4, 8 e 9 nós, respectivamente; *ElmAxiQ4*, *ElmAxiQ8* e *ElmAxiQ9* são elementos quadrilaterais axissimétricos com 4, 8 e 9 nós, respectivamente; *ElmT3*, *ElmT6* e *ElmT10* são elementos triangulares planos com 3, 6 e 10 nós, respectivamente; *ElmAxiT3*, *ElmAxiT6* e *ElmAxiT10* são elementos triangulares axissimétricos com 3, 6 e 10 nós, respectivamente; *ElmH8* e *ElmH20* são elementos hexaédricos sólidos com 8 e 20 nós. Esta última camada da herança foi necessária para atribuir a cada elemento sua função de forma e definir os objetos do tipo *ParametricIntegration*, usados no cálculo do carregamento nodal equivalente, para cada elemento finito.

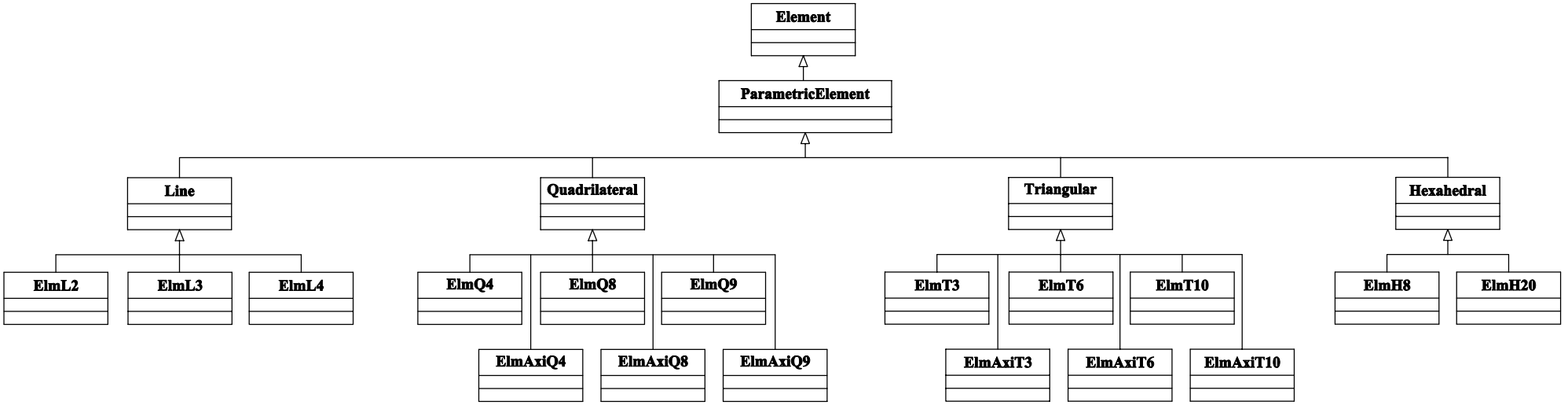


Figura 5.3: Hierarquia da classe *Element*

A figura 5.4 mostra a hierarquia da classe *IntegrationPoint* que tem por finalidade representar os pontos de integração com suas coordenadas e respectivos pesos. A sub-classe *GaussPoint* representa um ponto de Gauss em coordenadas naturais, a sub-classe *AreaGaussPoint* representa um ponto de Gauss em coordenadas de área e a sub-classe *VolumeGaussPoint* representa um ponto de Gauss em coordenadas de volume.

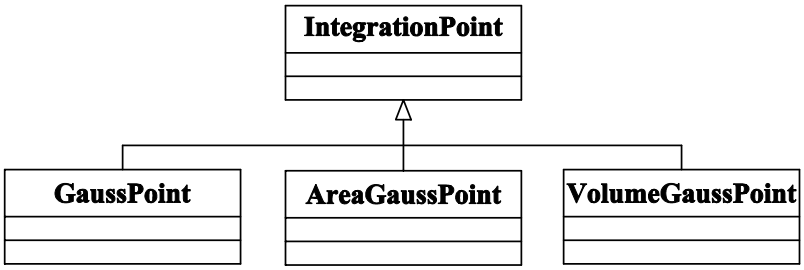


Figura 5.4: Hierarquia da classe *IntegrationPoint*

A figura 5.5 mostra a hierarquia da classe *AnalysisModel* que tem por finalidade agrupar os tipos de análise a serem inicialmente disponibilizados pelo programa: análise unidimensional (subclasse *LineAnalysisM*), tridimensional (subclasse *SolidAnalysisM*), axissimétrica (subclasse *AxisymetricAnalysisM*), de estado plano de tensões (subclasse *PlaneStressAnalysisM*) e de estado plano de deformação (subclasse *PlaneStrainAnalysisM*).

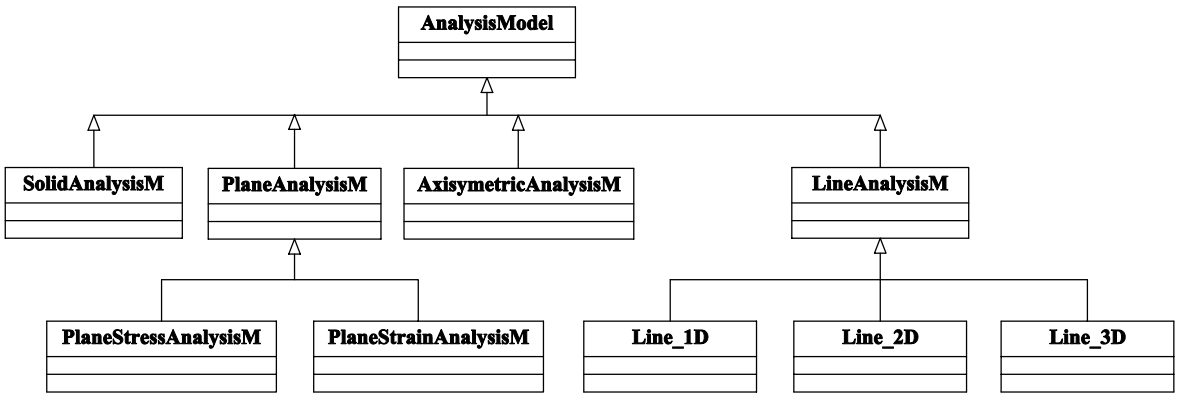


Figura 5.5: Hierarquia da classe *AnalysisModel*

A figura 5.6 mostra a hierarquia da classe *Shape* que tem por finalidade agrupar os diferentes tipos de funções de forma (e suas derivadas) para os diferentes tipos de elementos unidimensionais, bidimensionais e tridimensionais. Na terceira camada da

hierarquia da figura 5.6 mostram-se as várias funções de forma disponibilizadas: *L2*, *L3* e *L4* para os elementos unidimensionais com 2, 3 e 4 nós; *Q4*, *Q8* e *Q9* para os elementos quadrilaterais com 4, 8 e 9 nós; *T3*, *T6* e *T10* para os elementos triangulares com 3, 6 e 10 nós; *H8* e *H20* para os elementos hexaédricos sólidos com 8 e 20 nós.

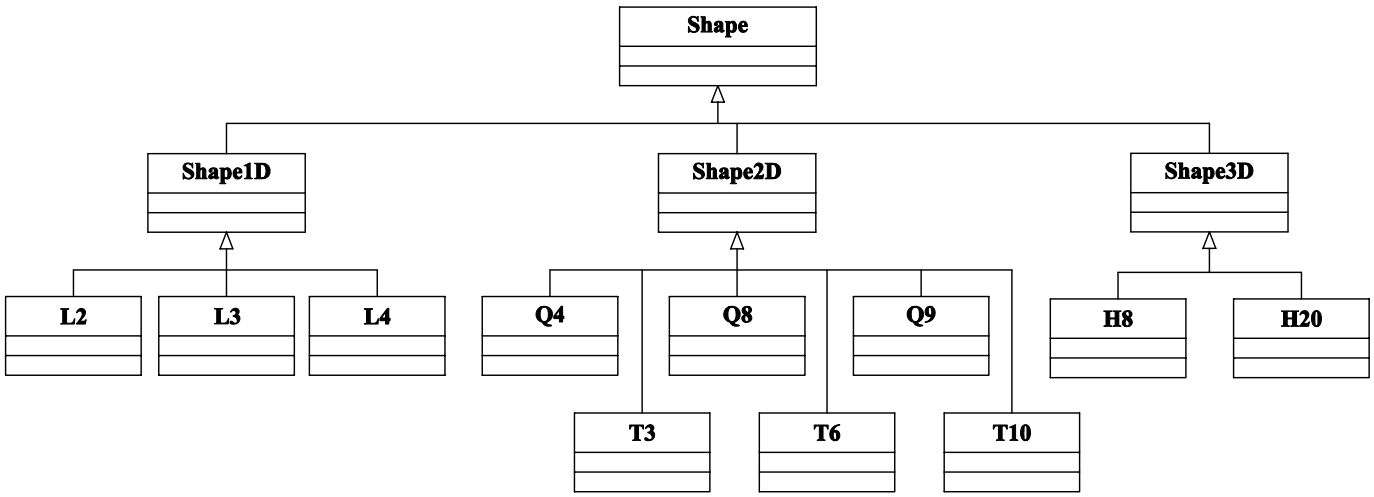


Figura 5.6: Hierarquia da classe *Shape*

A figura 5.7 mostra a hierarquia da classe *Material* que tem como finalidade armazenar os métodos e atributos comuns aos diferentes tipos de materiais tais como ortotrópicos, isotrópicos (implementados neste trabalho) e não-lineares.

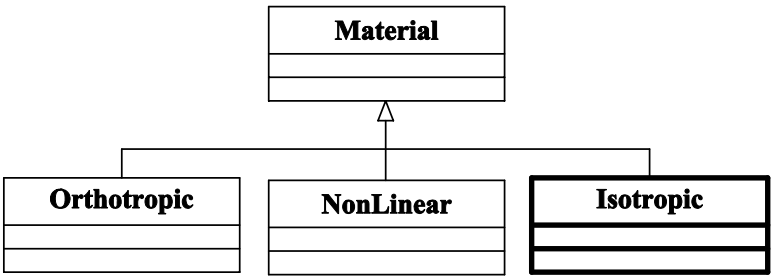


Figura 5.7: Hierarquia da classe *Material*

A figura 5.8 mostra a hierarquia da classe *ParametricIntegration* que tem como finalidade armazenar os métodos e atributos necessários ao cálculo dos diferentes tipos de integrações paramétricas, tais como, integrais ao longo de linhas, áreas ou volumes.

A figura 5.9 mostra os diagramas das classes *FemModel*, *Node*, *CrossSection*, *IntegrationOrder*, *ElementForce* e *PointForce* que não possuem subclasses.

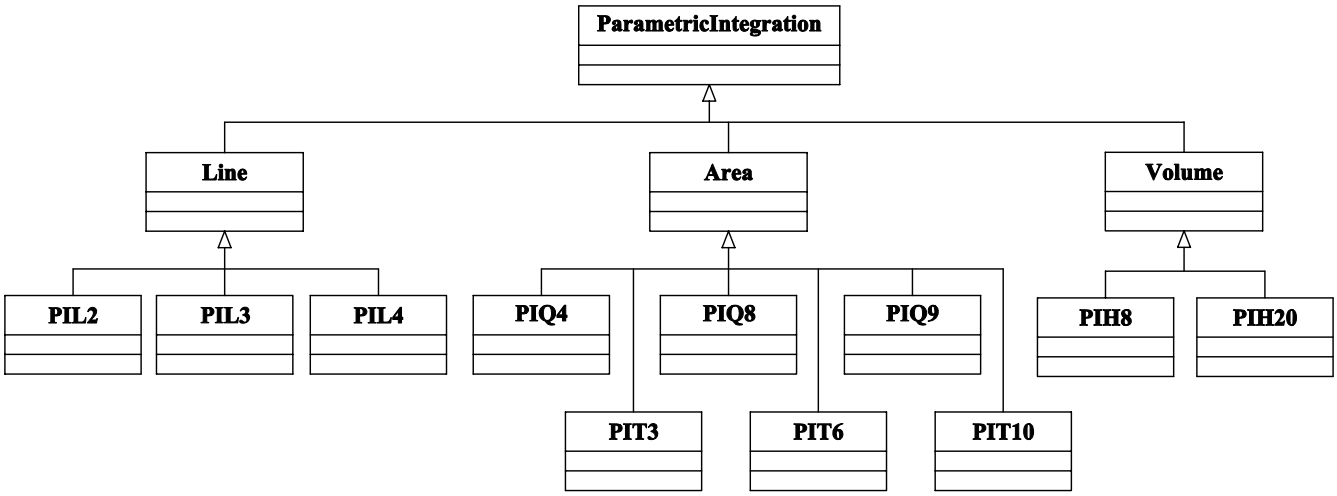


Figura 5.8: Hierarquia da classe *ParametricIntegration*

A classe *FemModel* tem a finalidade de gerenciar e armazenar os objetos inerentes a um modelo de elementos finitos. A classe *Node* representa um objeto do tipo nó e armazena os atributos de um nó qualquer como coordenadas, intensidade e direção das forças aplicadas, restrições etc; e possui métodos responsáveis pelo acesso a estes atributos.

A classe *CrossSection* tem a finalidade de armazenar as propriedades geométricas pertencentes a uma seção de um elemento finito, por exemplo, área, inércia, espessura etc.

A classe *IntegrationOrder* guarda informações referentes às ordens de integração para elementos finitos em coordenadas naturais.

A classe *ElementForce* representa as cargas distribuídas nos elementos finitos na forma de valores prescritos das forças sobre os nós. Tais valores prescritos juntamente com as coordenadas dos nós onde as forças são aplicadas são guardados em uma classe denominada *PointForce*. A mesma classe *ElementForce* é usada para representar uma carga distribuída sobre uma linha, área ou volume sendo denominada *LineElementForce*, *SurfaceElementForce* ou *VolumeElementForce*.

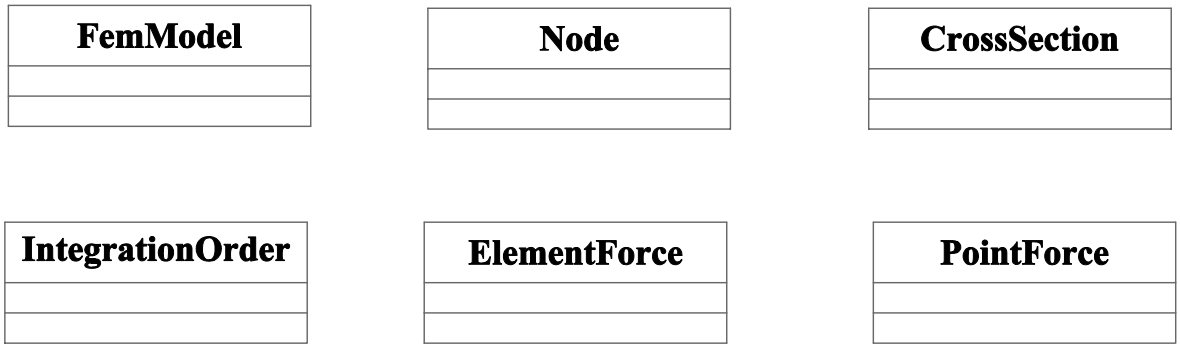


Figura 5.9: Diagramas das classes *FemModel*, *Node*, *CrossSection*, *IntegrationOrder*, *ElementForce* e *PointForce*

5.2 Interação entre as classes

As interações entre os objetos do sistema podem ser representadas utilizando os diagramas de instâncias. Estes diagramas informam graficamente os tipos e quantidades de objetos criados por cada classe do programa.

A figura 5.10 mostra o diagrama de instâncias da classe *Driver*. O *Driver* possui um objeto *Solution* representando o tipo de solução escolhida e um objeto *FemModel* representando o tipo de modelo escolhido.

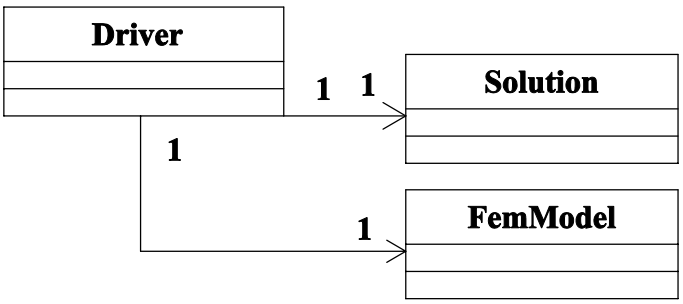


Figura 5.10: Objetos instanciados pela classe *Driver*

A figura 5.11 mostra o diagrama de instâncias da classe *FemModel*. A classe *FemModel* possui objetos do tipo *Node*, objetos do tipo *Element*, objetos do tipo *Material*, objetos do tipo *CrossSection*, objetos do tipo *AnalysisModel*, objetos do tipo *Shape* e objetos do tipo *IntegrationOrder*.

A figura 5.12 mostra o diagrama de instâncias da classe *ParametricElement*. Cada

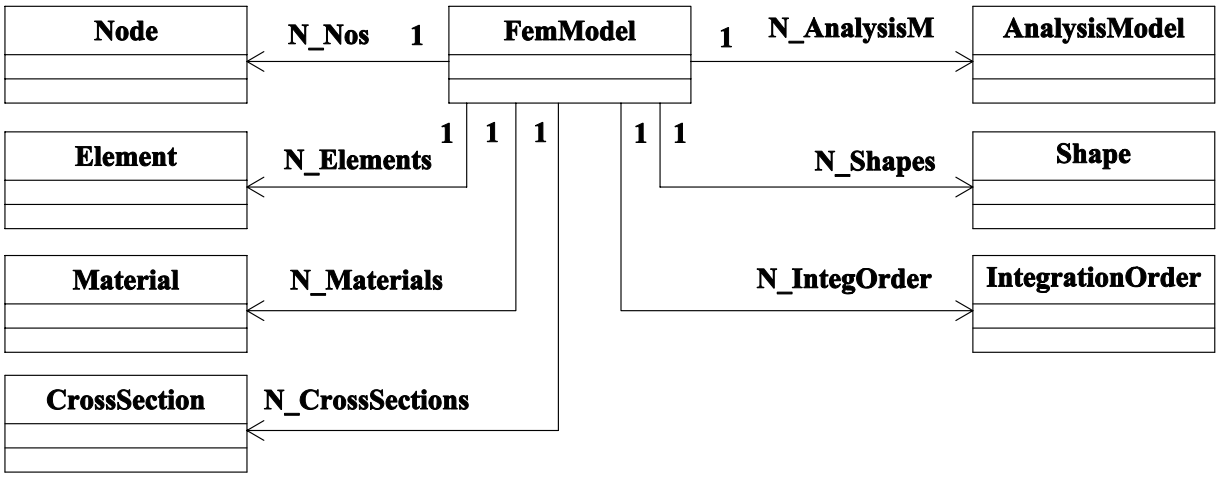


Figura 5.11: Objetos instanciados pela classe *FemModel*

objeto do tipo *ParametricElement* referencia um objeto do tipo *AnalysisModel*, representando o tipo de análise, um objeto do tipo *Shape*, representando as funções de forma do elemento, um objeto do tipo *Material*, representando o material do elemento e objetos do tipo *Node*, representando os nós do elemento e possui objetos do tipo *ElementForce*, representando as forças por unidade de comprimento, área ou volume, objetos do tipo *PointForce* representando as forças concentradas e os valores nodais prescritos das cargas distribuídas, e objetos do tipo *IntegrationPoint* representando os pontos de Gauss do elemento.

A figura 5.13 mostra o diagrama de instâncias da classe *ElementForce*, indicando que as forças de corpo, superfície ou de linha são descritas através de uma lista de objetos do tipo *PointForce*, representando o valor prescrito da força no nó. Portanto cada *ElementForce* faz referência a objetos do tipo *PointForce*.

A figura 5.14 mostra o diagrama de instâncias da classe *Node*. Cada objeto do tipo *Node* possui um objeto do tipo *Coord*, representando suas coordenadas cartesianas, um objeto do tipo *Force*, representando os valores das forças nodais, um objeto do tipo *Spring*, representando efeitos de mola no nó, um objeto do tipo *Reactions*, representando as reações, um objeto do tipo *PreDisplacement*, representando deslocamentos prescritos, um objeto do tipo *Restraints*, representando as restrições, um objeto do tipo *Displacement*, representando os deslocamentos do nó, um objeto do tipo *Equations*, representando as equações do nó e um objeto do tipo *Angle*, representando apoios inclinados.

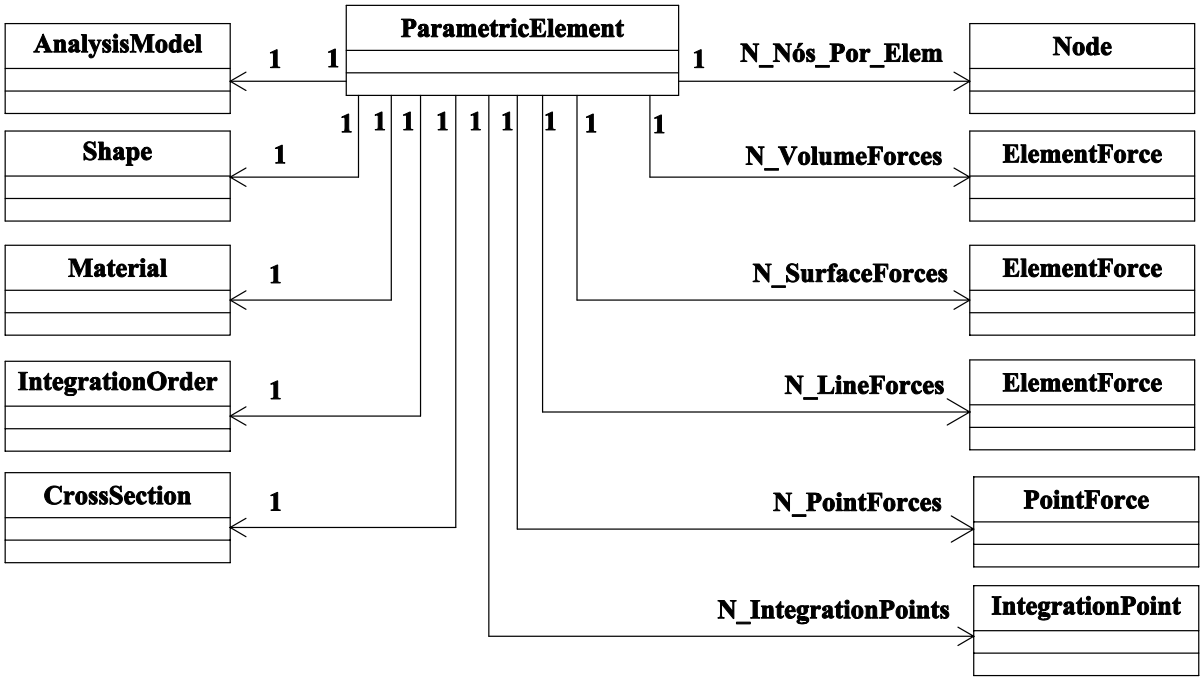


Figura 5.12: Objetos instanciados pela classe *ParametricElement*

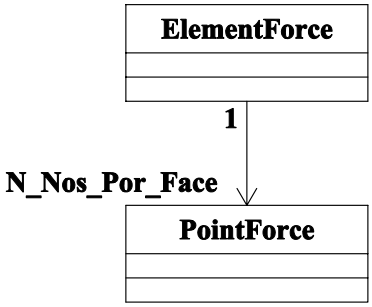


Figura 5.13: Objetos instanciados pela classe *ElementForce*

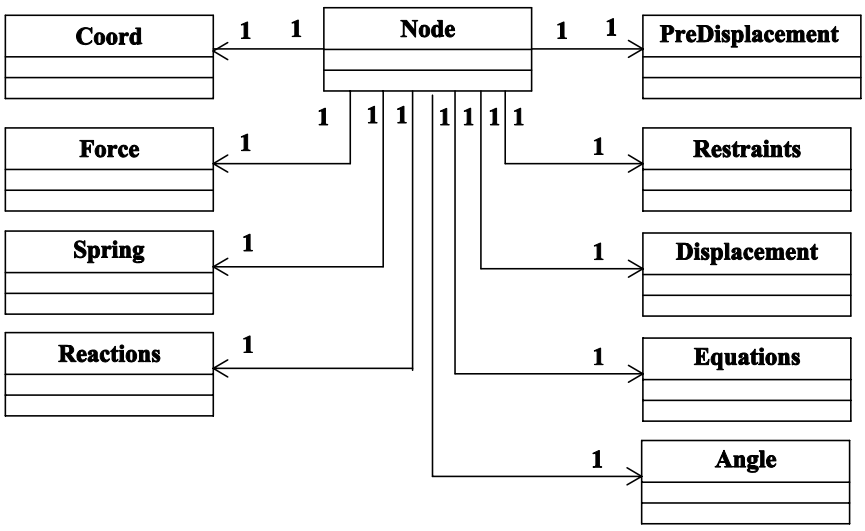


Figura 5.14: Objetos instanciados pela classe *Node*

5.3 Seqüências de atividades

Outro tipo de diagrama muito utilizado para representar interações entre objetos do sistema é o diagrama de seqüência. Este diagrama mostra a linha de vida de cada objeto do sistema durante a execução de determinada tarefa. As figuras 5.15, 5.16, 5.17, 5.18, 5.19, 5.20, 5.21, 5.22, 5.23, 5.24, 5.25 e 5.26 mostram o diagrama de seqüência do sistema para obtenção da solução de equilíbrio de um problema de análise de tensões através do MEF. A figura 5.15 refere-se à etapa de caracterização do problema, as figuras 5.16, 5.17, 5.18, 5.19, 5.20, 5.21, 5.22 e 5.23 à etapa de preenchimento do modelo, as figuras 5.24 e 5.25 à etapa de montagem das matrizes e vetores do modelo e a figura 5.26 à etapa de obtenção e persistência da solução.

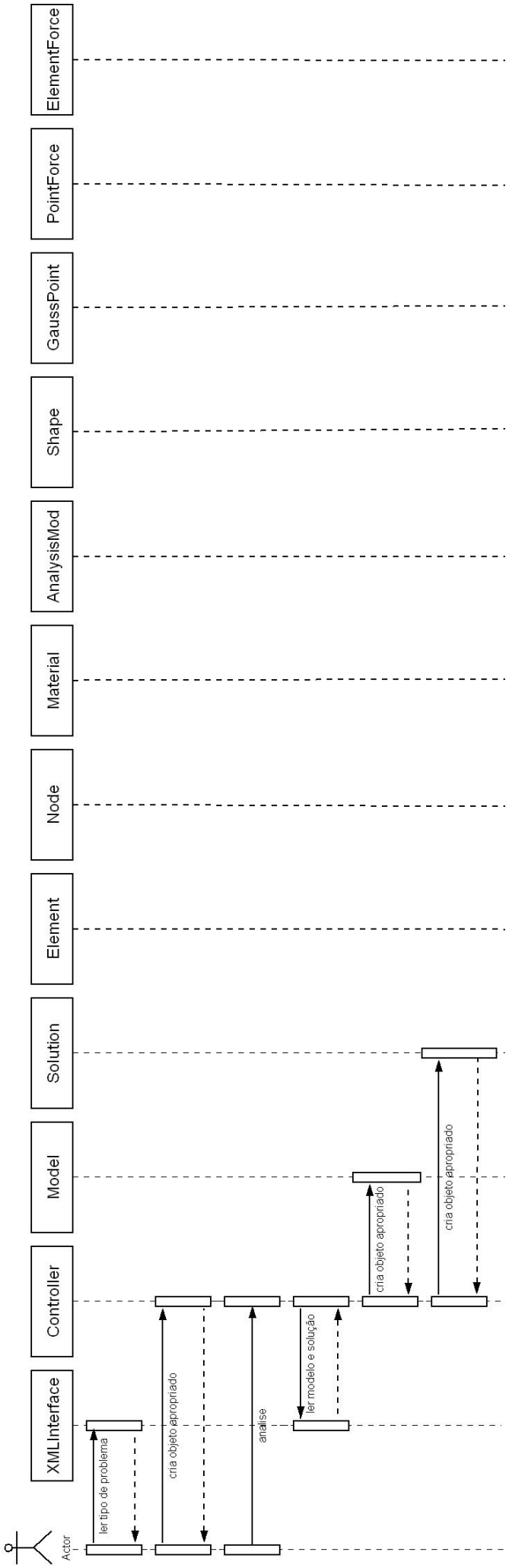


Figura 5.15: Diagrama de seqüência para o sistema - caracterização do problema e da solução

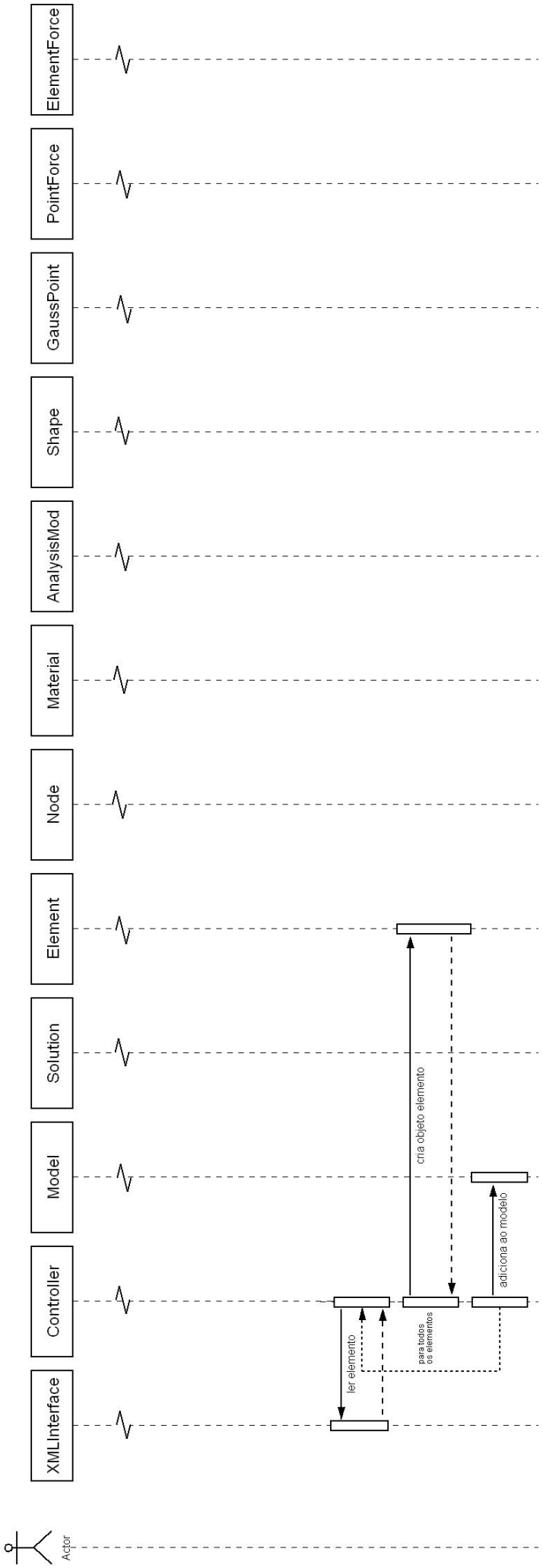


Figura 5.16: Diagrama de seqüência para o sistema - preenchimento do modelo com os objetos *Element*

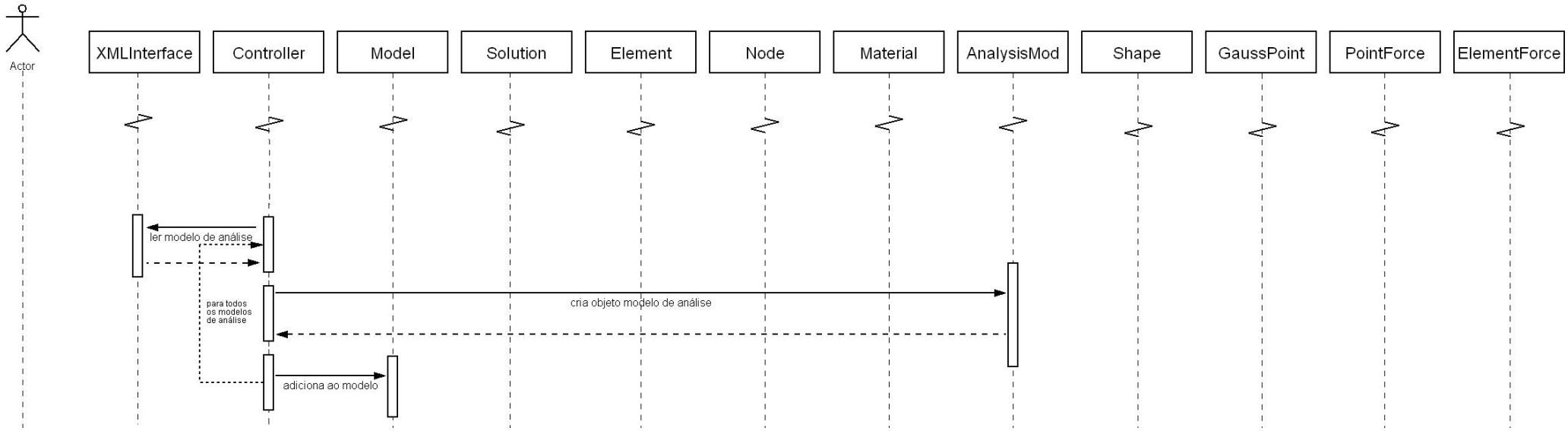


Figura 5.17: Diagrama de seqüência para o sistema - preenchimento do modelo com os objetos *AnalysisModel*

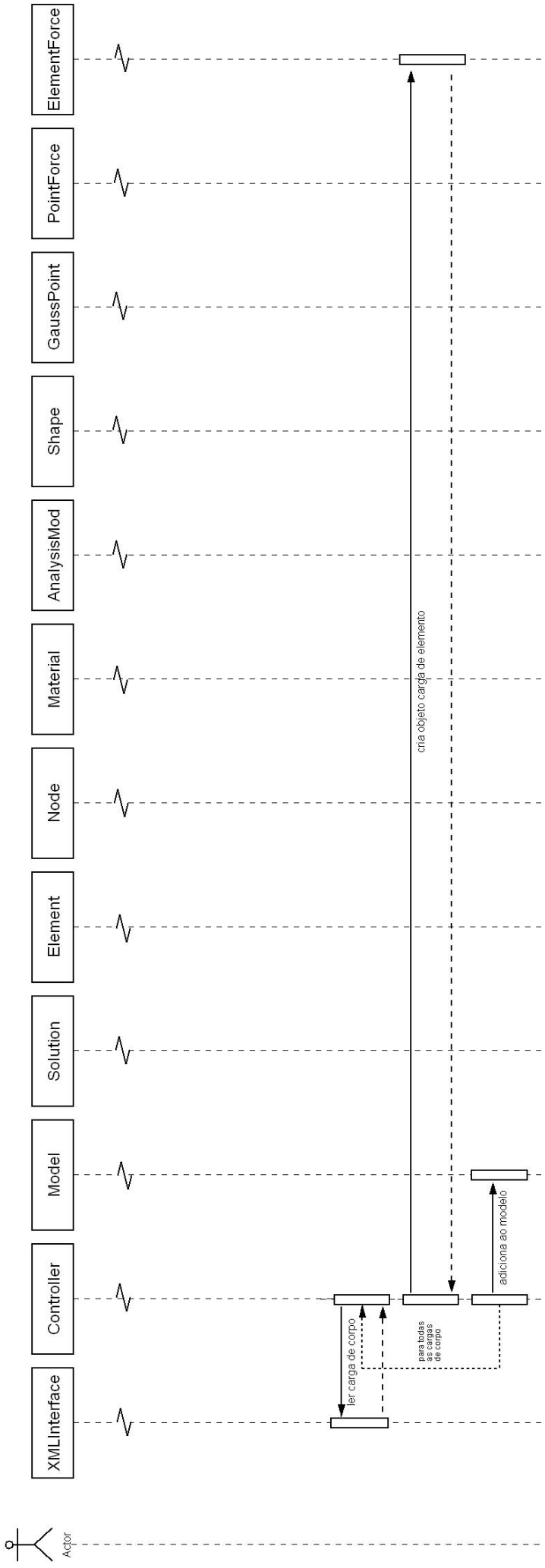


Figura 5.18: Diagrama de seqüência para o sistema - preenchimento do modelo com os objetos *ElementForce* representando as forças de corpo

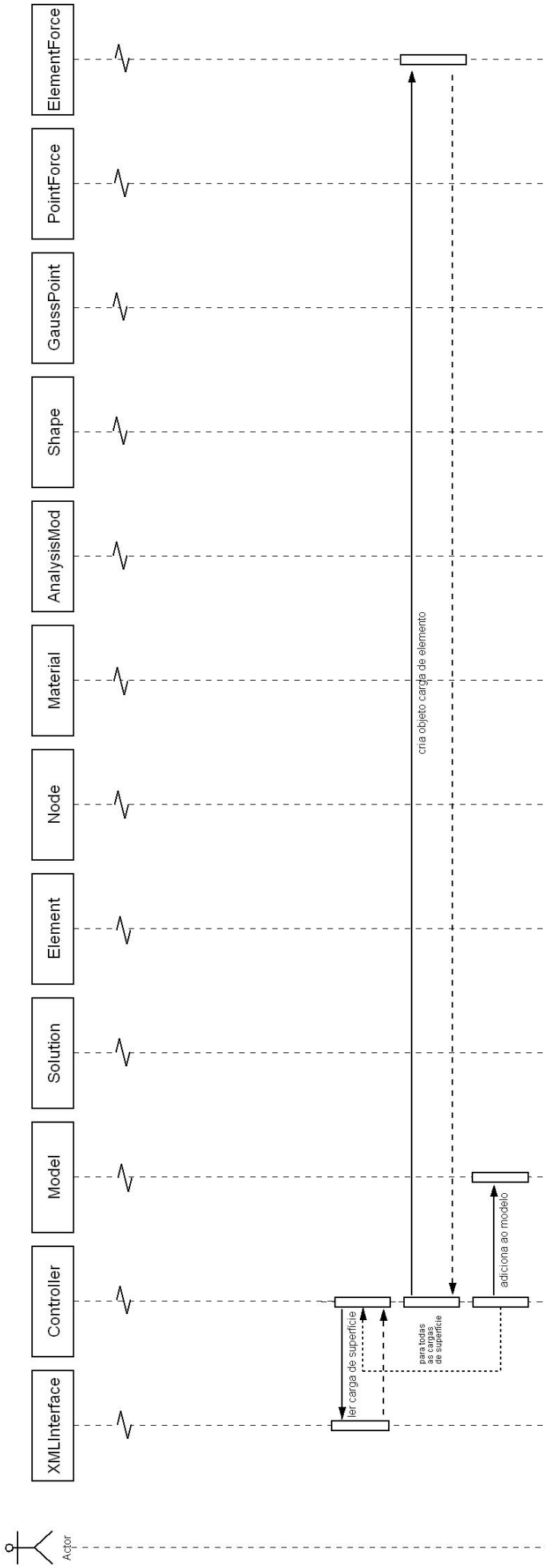


Figura 5.19: Diagrama de seqüência para o sistema - preenchimento do modelo com os objetos *ElementForce* representando as forças de superfície

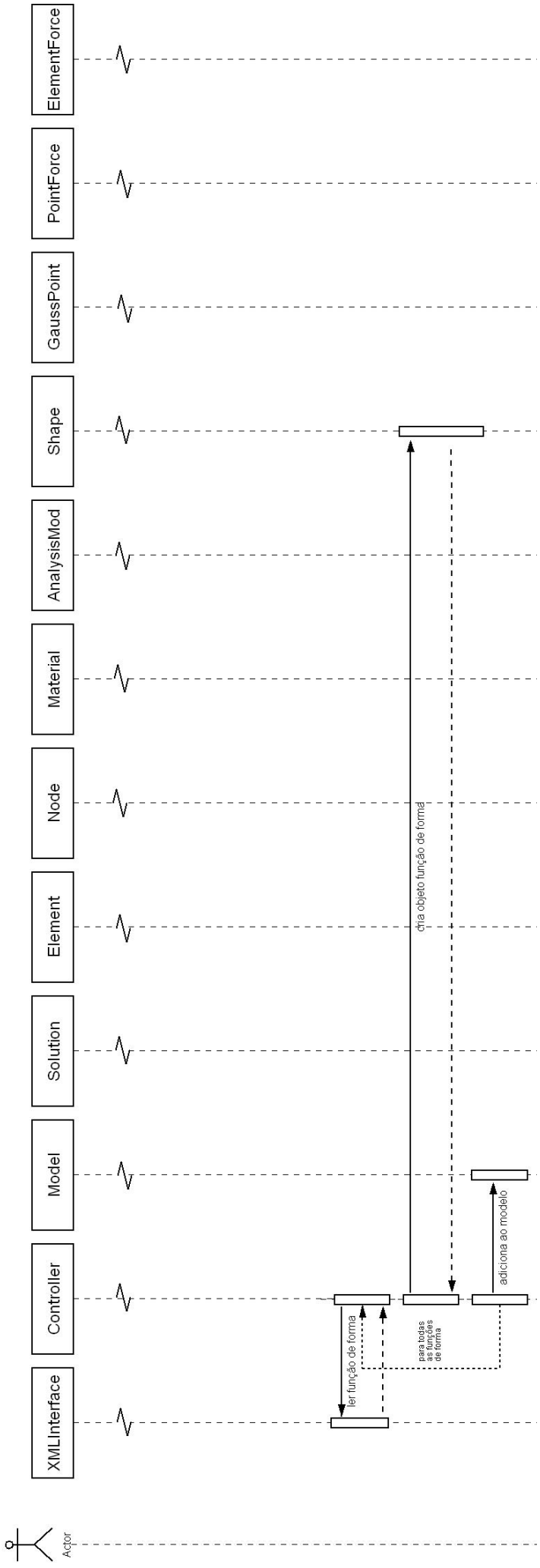


Figura 5.20: Diagrama de seqüência para o sistema - preenchimento do modelo com os objetos *Shape*

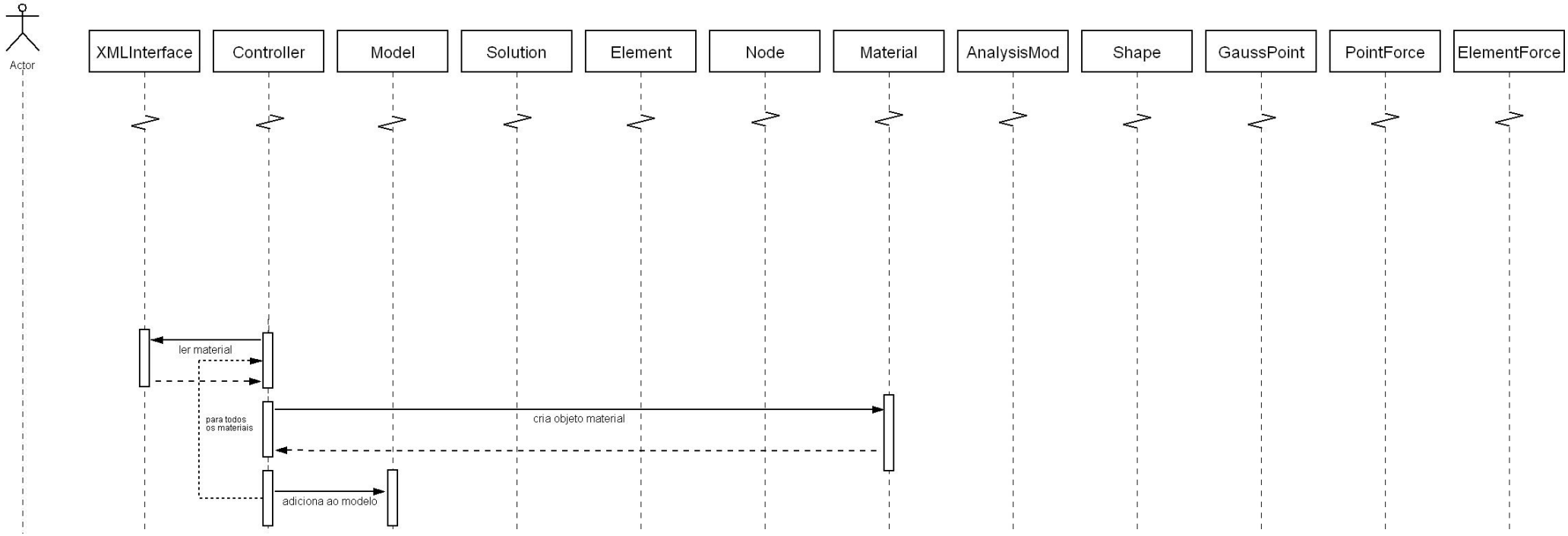


Figura 5.21: Diagrama de seqüência para o sistema - preenchimento do modelo com os objetos *Material*

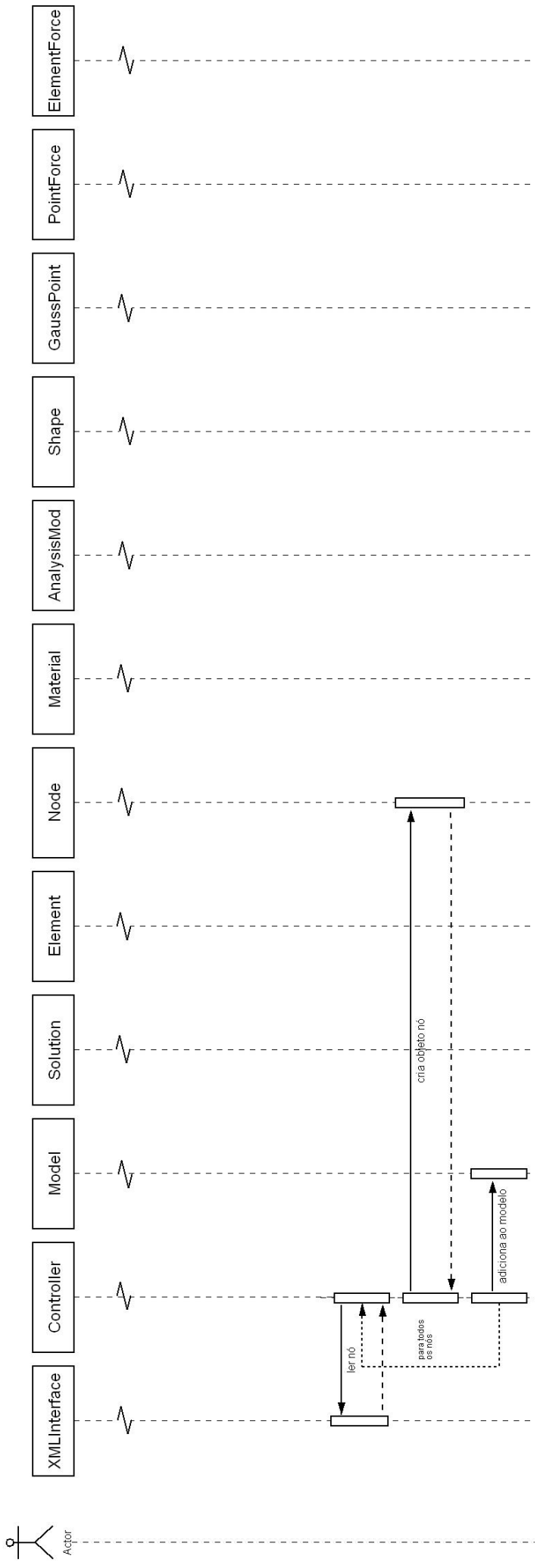


Figura 5.22: Diagrama de seqüência para o sistema - preenchimento do modelo com os objetos *Node*

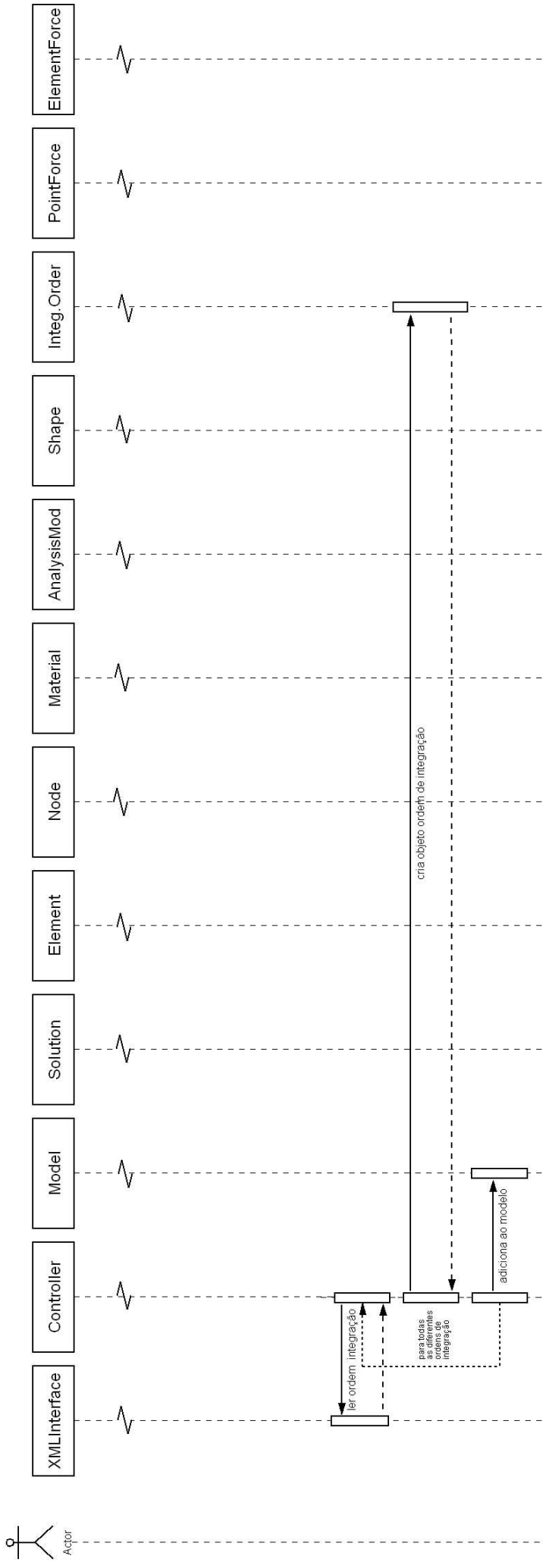


Figura 5.23: Diagrama de seqüência para o sistema - preenchimento do modelo com os objetos *IntegrationOrder*

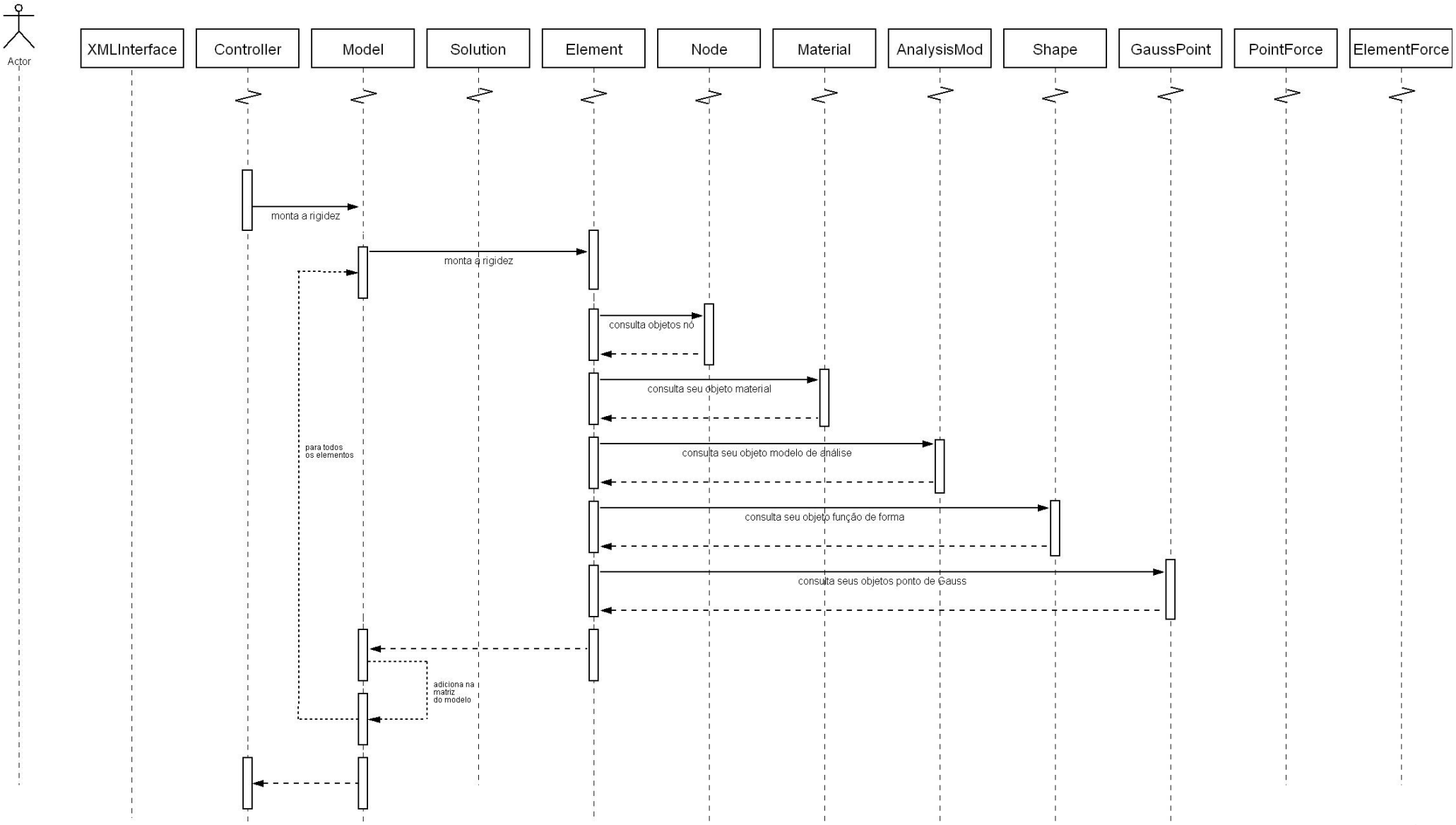


Figura 5.24: Diagrama de seqüência para o sistema - montagem da matriz de rigidez do modelo

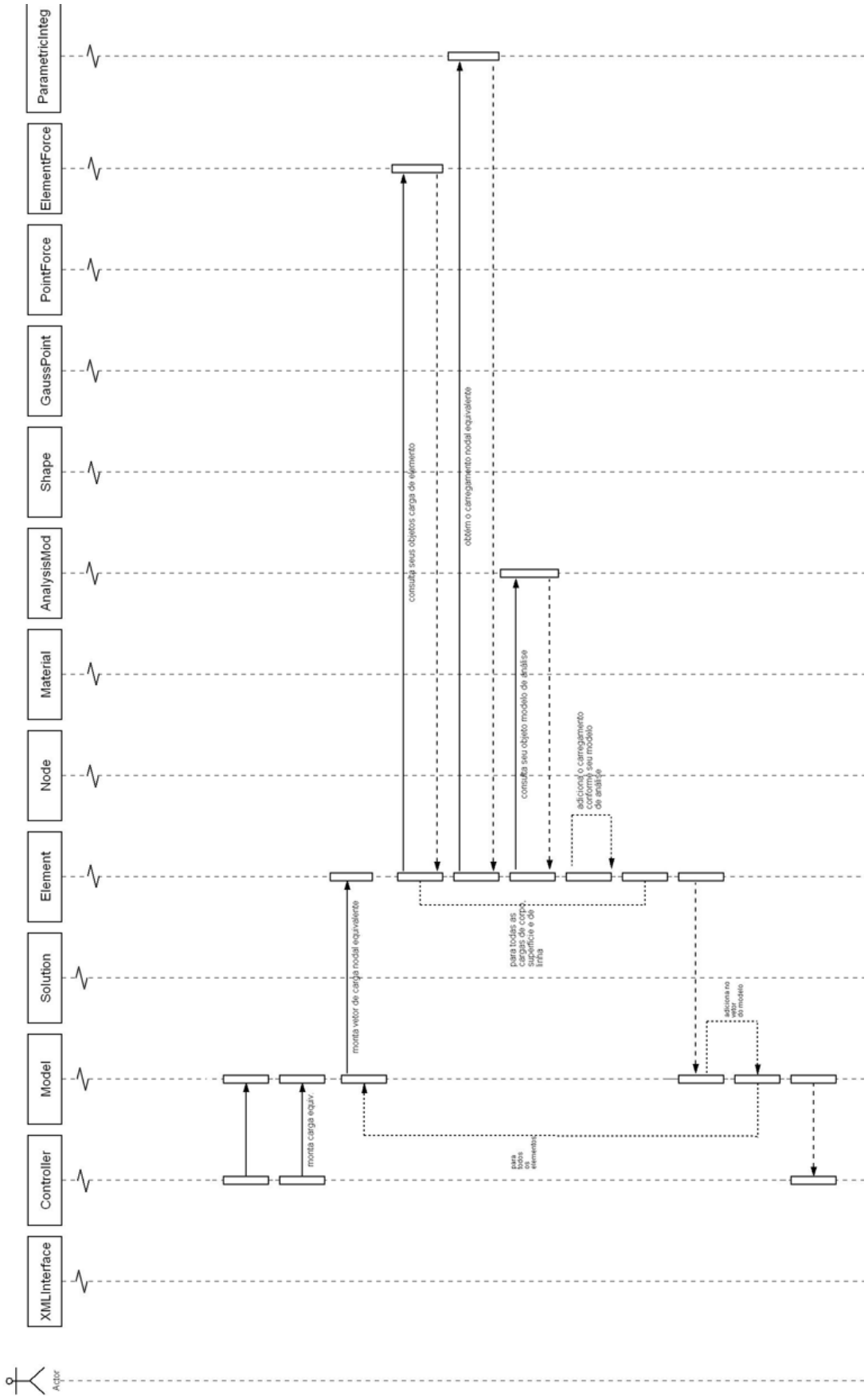


Figura 5.25: Diagrama de seqüência para o sistema - montagem do vetor de carregamento nodal equivalente do modelo

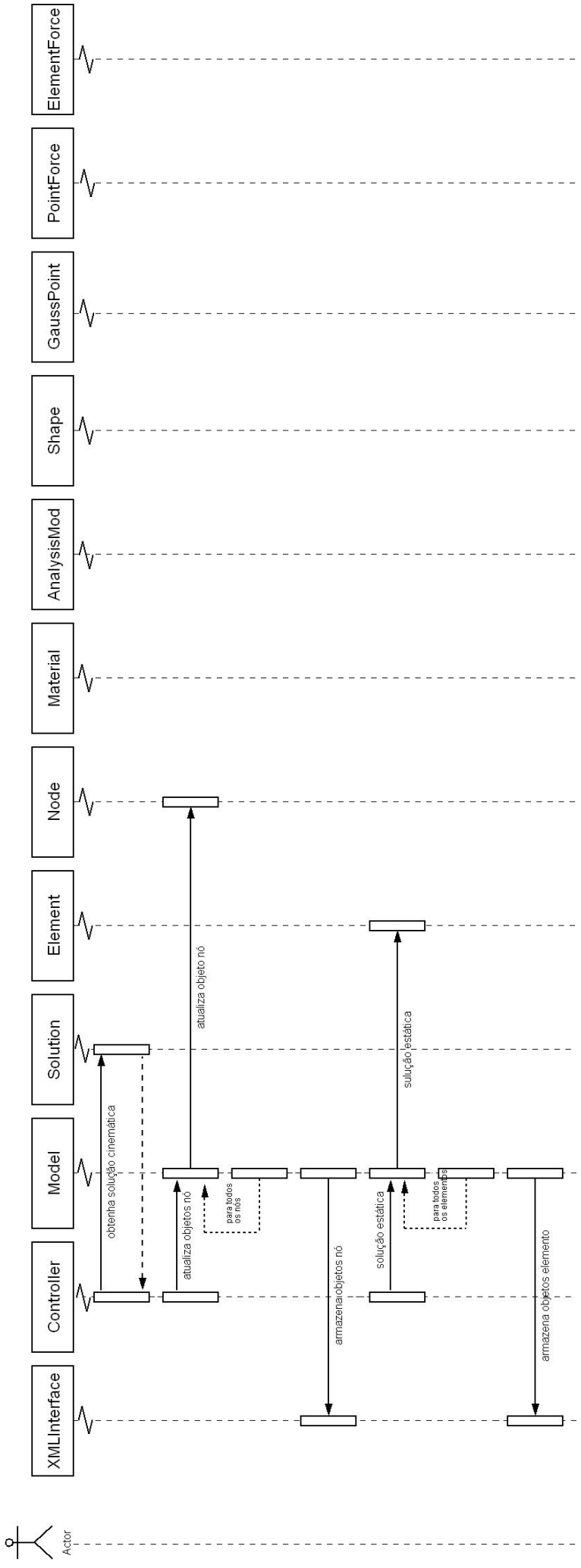


Figura 5.26: Diagrama de seqüência para o sistema - obtenção e persistência da solução

Capítulo 6

Exemplos de Verificação

6.1 Introdução

Neste capítulo são apresentados vários modelos de elementos finitos que utilizam os diversos recursos disponibilizados no sistema, resumidos na tabela 6.1.

Em todos os testes realizados obtém-se a solução por equilíbrio em um ponto, para problemas de análise de tensões, considerando-se material elástico linear isotrópico com módulo de elasticidade longitudinal E e coeficiente de Poisson ν (ver tabela 6.1).

Para os problemas bidimensionais, as malhas de elementos finitos são geradas com o auxílio do pré-processador (Gonçalves 2004), já discutido na seção 1.1, que persiste em disco um arquivo XML correspondente à malha, no formato mostrado no apêndice A.

Os exemplos a seguir são apresentados em seis grupos conforme indica a tabela 6.2. O grupo 1 mostra diversos testes de malha (*Patch Test*) envolvendo os diversos recursos do programa. Os grupos 2 a 6 são concebidos conforme o modelo de análise empregado (ver tabela 6.1). Assim, vários problemas unidimensionais (grupo 2), de estado plano de tensões (grupo 3), estado plano de deformações (grupo 4), axissimétrico (grupo 5) e tridimensionais (grupo 6) são modelados e, sempre que possível, as soluções obtidas pelo programa são comparadas com as soluções analíticas correspondentes.

Tabela 6.1: Recursos disponibilizados no sistema

Recurso	**Tipos disponíveis**		
Elemento	Linha	2 nós	*L2*
		3 nós	*L3*
		4 nós	*L4*
	Quadrilátero	4 nós	*Q4*
		8 nós	*Q8*
		9 nós	*Q9*
	Quadrilátero Axissimétrico	4 nós	*AxiQ4*
		8 nós	*AxiQ8*
		9 nós	*AxiQ9*
	Triângulo	3 nós	*T3*
		6 nós	*T6*
		10 nós	*T10*
	Triângulo Axissimétrico	3 nós	*AxiT3*
		6 nós	*AxiT6*
		10 nós	*AxiT10*
	Hexaedro	8 nós	*H8*
		20 nós	*H20*
Modelo de Análise	Unidimensional		*LineAnalysisM*
	Bidimensional	Estado plano de tensões	*PlaneStressAnalysisM*
		Estado plano de deform.	*PlaneStrainAnalysisM*
		Axissimétrico	*AxiSymetricAnalysisM*
	Tridimensional		*SolidAnalysisM*
Ordem de Integração	Coordenadas naturais cartesianas		1 a 8 pontos por direção
	Coordenadas de área		1, 3, 4 e 6 pontos
Carregamento	Linha		*LineElementForce*
	Área		*SurfaceElementForce*
	Volume		*VolumeElementForce*
Material	Isotrópico		*Isotropic*
Solução	Equilíbrio em um ponto		*OnePointEq*
Tipo de Problema	Mecânica Estrutural (Análise de Tensões)		*StructuralMech*

Tabela 6.2: Agrupamentos dos Exemplos

Grupo de Exemplos	Modelo de Análise	Seções no Capítulo
1	Vários	6.2 - *Patch Tests*
2	Unidimensional	6.3 - Tração Axial
3	Estado Plano de Tensões	6.4 - Viga Parede
		6.5 - Viga Armada
		6.6 - Chapa com Furo Circular
4	Estado Plano de Deformação	6.7 - Cunha
		6.8 - Barragem
		6.9 - Fundação
5	Axissimétrico	6.10 - Disco
		6.11 - Tubo
		6.12 - Problema de Boussinesq
6	Tridimensional	6.13 - Barra Prismática
		6.14 - Viga Biapoiada
		6.15 - Barra Curva
		6.16 - Dente de Engrenagem

6.2 *Patch Tests*

O *Patch Test* originalmente concebido por Bruce Irons verificava simplesmente se uma discretização com elementos de tamanhos aleatórios reproduzia exatamente o comportamento de um material elástico quando submetido a deslocamentos compatíveis com deformação constante. Esta motivação física levou-o a desenvolver um teste mais formal que se tornou um procedimento largamente usado para verificação de elementos finitos e os programas relacionados (Zienkiewicz, Chan, Taylor & Simo 1986).

Utiliza-se uma discretização com elementos de tamanhos aleatórios de tal maneira que ao menos um nó fique completamente cercado por elementos. Deve-se aplicar restrições nodais ao contorno juntamente com deslocamentos ou cargas compatíveis com um estado de deformação constante. Os nós que não estiverem no contorno não são carregados nem restritos (ver figura 6.1). Calculando as deformações (ou tensões) do modelo, o *Patch Test* estará atendido se em todos os pontos, em todos os elementos, as deformações calculadas forem iguais à solução exata de acordo com a precisão computacional admitida (Cook, Malkus & Plesha 1989).

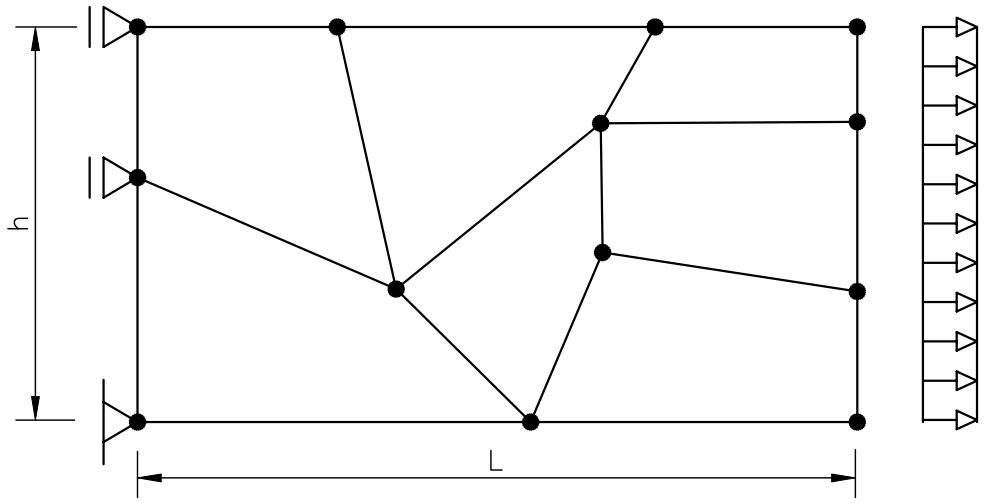


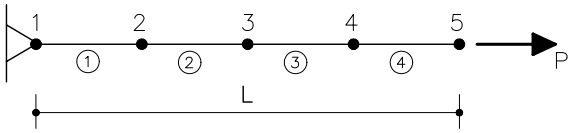
Figura 6.1: Possível malha de elementos quadrilaterais para um *Patch Test*

Portanto, é necessário verificar se os elementos paramétricos implementados neste trabalho atendem aos requisitos do *Patch Test*. Assim, são mostrados a seguir vários testes envolvendo diferentes malhas de elementos finitos.

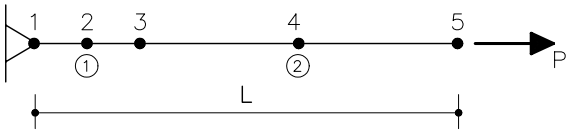
6.2.1 Elementos Unidimensionais

O objetivo deste exemplo é modelar uma barra submetida a um estado constante de deformação, utilizando os elementos unidimensionais de 2, 3 e 4 nós (figura 6.2) e modelo de análise uni-dimensional do tipo *LineAnalysisM*. Utiliza-se um módulo de elasticidade longitudinal E = 1,0 uf/uc^2, área da seção transversal da barra A = 0,5 uc^2, carga P = 10 uf e comprimento da barra L = 8 uc. (uf = unidade de força, uc = unidade de comprimento).

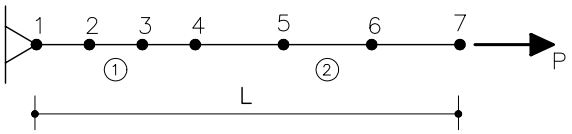
Todos os elementos destas discretizações apresentam deformação longitudinal constante e igual a 20. Apresentam também deslocamentos compatíveis com o estado constante de deformação. Assim, o *Patch Test* é atendido para estes elementos.



(a) 4 Elementos L2



(b) 2 Elementos L3

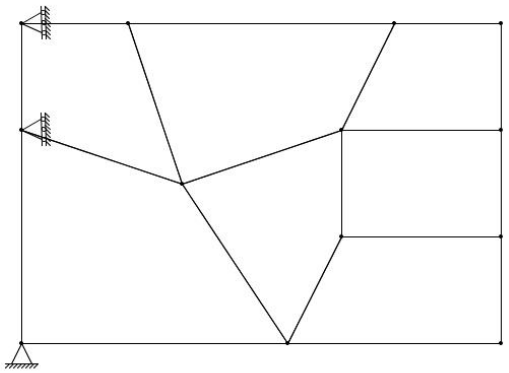


(c) 2 Elementos L4

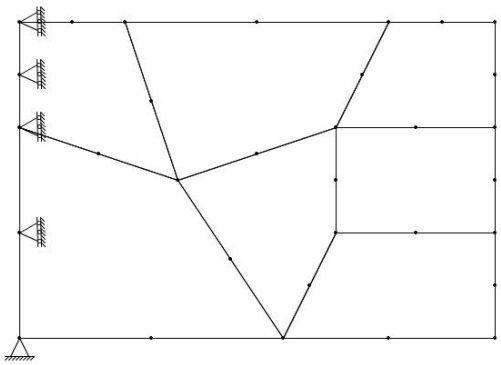
Figura 6.2: *Patch Test* para elementos unidimensionais submetidos à tração constante

6.2.2 Elementos Planos - Tração Constante

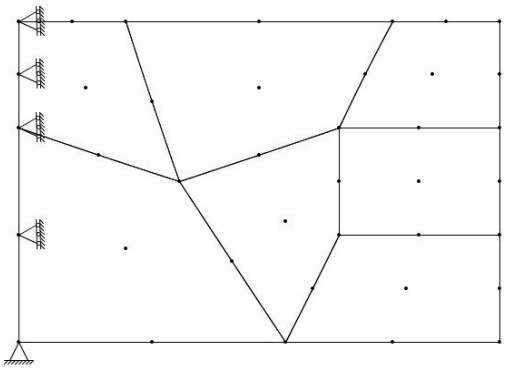
São apresentados neste sub-item vários exemplos de verificação do *Patch Test* para um mesmo problema. Trata-se de uma chapa submetida à tração constante de módulo unitário em uma das faces, ou seja, condição de carregamento compatível com deformação constante conforme a figura 6.1. A chapa tem espessura unitária, comprimento $L = 9$ uc, altura $h = 6$ uc e estado plano de tensões com $E = 1{,}0$ uf/uc^2 e $\nu = 0{,}3$. Para modelamento são utilizados elementos quadrilaterais *Q4 Q8* e *Q9* e triangulares *T3*, *T6* e *T10*, conforme mostra a figura 6.3.



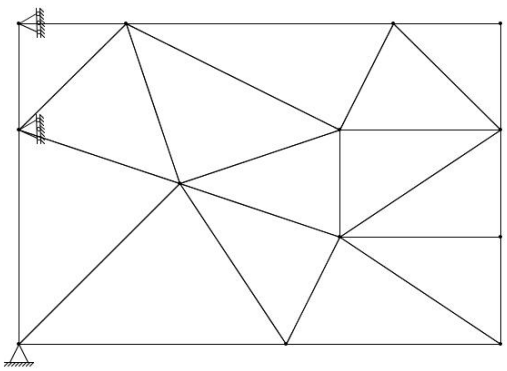
(a) Elementos Q4



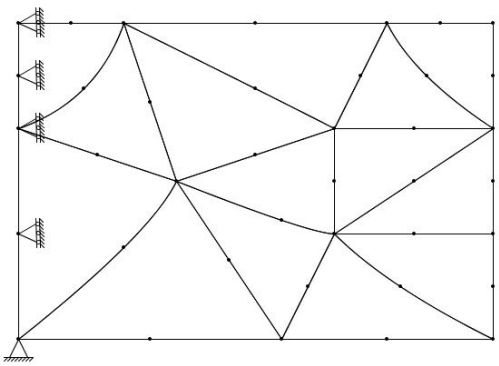
(b) Elementos Q8



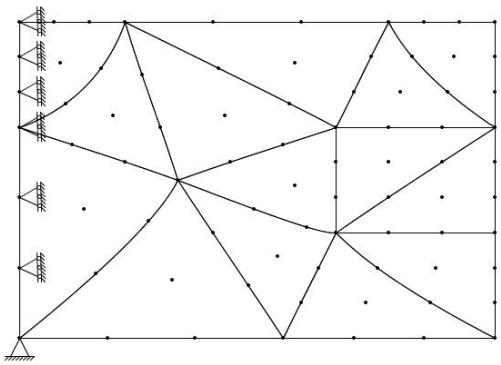
(c) Elementos Q9



(d) Elementos T3



(e) Elementos T6



(f) Elementos T10

Figura 6.3: *Patch Test* para elementos planos submetidos à tração constante

Os deslocamentos nodais de todas as discretizações acima coincidem com o campo de deslocamentos da solução exata. Para todas as malhas as tensões nos pontos de integração de Gauss e nos pontos nodais são: $\sigma_{xx} = 1,0\ uf/uc^2$ e $\sigma_{yy} = \tau_{xy} \approx 0$. Escreve-se aqui aproximadamente igual a zero para ressaltar que existe um erro computacional de no máximo 1×10^{-10}.

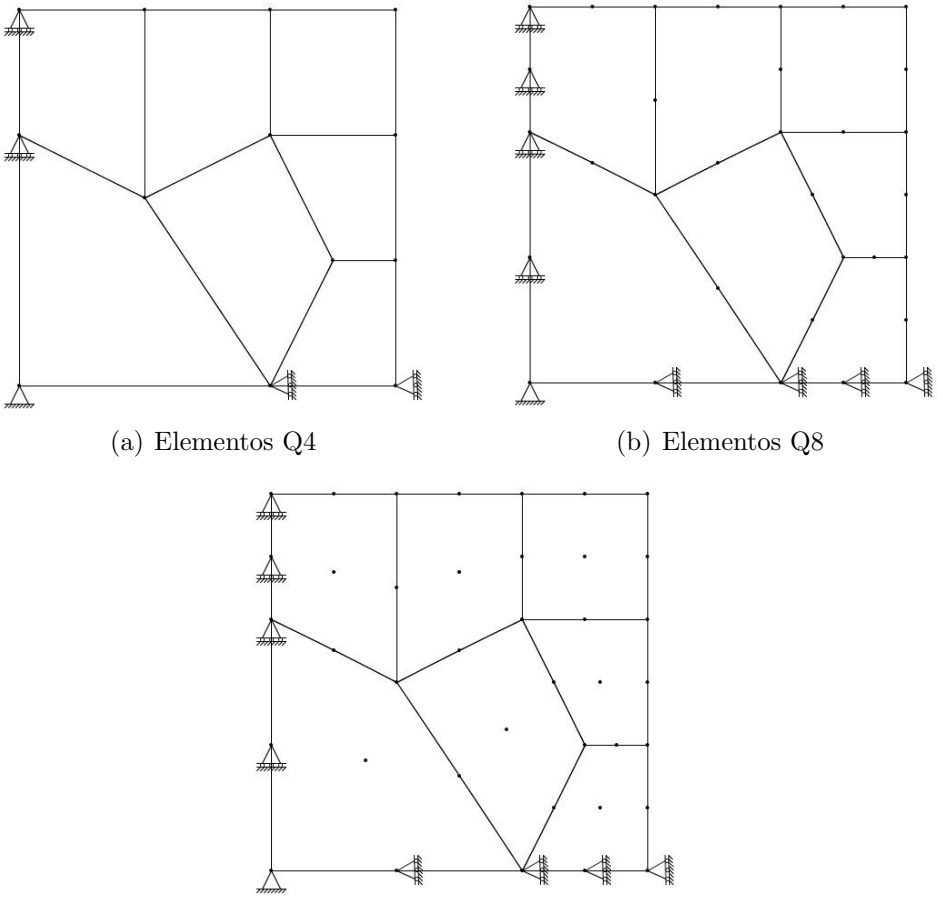
Portanto, as discretizações acima apresentam um estado constante de deformação e conclui-se que tais malhas passam neste *Patch Test*.

6.2.3 Elementos Planos - Cisalhamento Constante

Neste *Patch Test* é utilizada a mesma chapa do exemplo anterior, sendo alterado somente o comprimento de $L = 9\ uc$ para $L = 6\ uc$, considerando agora a atuação de uma carga de cisalhamento constante de módulo unitário atuando ao longo de todo o contorno da estrutura. Novamente deseja-se aplicar uma carga compatível com estado de deformação constante. Para modelamento são utilizados elementos quadrilaterais *Q4*, *Q8* e *Q9*, conforme mostra a figura 6.4.

Os deslocamentos nodais das discretizações acima também coincidem com o campo de deslocamentos da solução exata. Para todas as malhas as tensões nos pontos de integração de Gauss e nos pontos nodais são: $\tau_{xy} = 1,0\ uf/uc^2$ e $\sigma_x = \sigma_y = 0$.

Portanto, as discretizações acima apresentam um estado constante de deformação e conclui-se que tais malhas passam neste *Patch Test*.



(a) Elementos Q4

(b) Elementos Q8

(c) Elementos Q9

Figura 6.4: *Patch Test* para elementos planos submetidos à cisalhamento constante

6.2.4 Outros Exemplos de *Patch Test*

Os três testes abaixo, conforme sugeridos em (Zienkiewicz et al. 1986), referem-se ao problema de estado plano de tensões com variação de deslocamentos dada pelas equações $u(x, y) = 0,002 \times x$ e $v(x, y) = -0,0006 \times y$, módulo de elasticidade $E = 1000\ uf/uc^2$ e coeficiente de Poisson $\nu = 0{,}3$, cujos valores de tensões resultam em $\sigma_{yy} = \tau_{xy} = 0$ e $\sigma_{xx} = 2\,uf/uc^2$. Em todos os casos considera-se a malha da figura 6.5, detalhada na tabela 6.3, com quadratura de Gauss de 2×2 pontos de integração e espessura unitária.

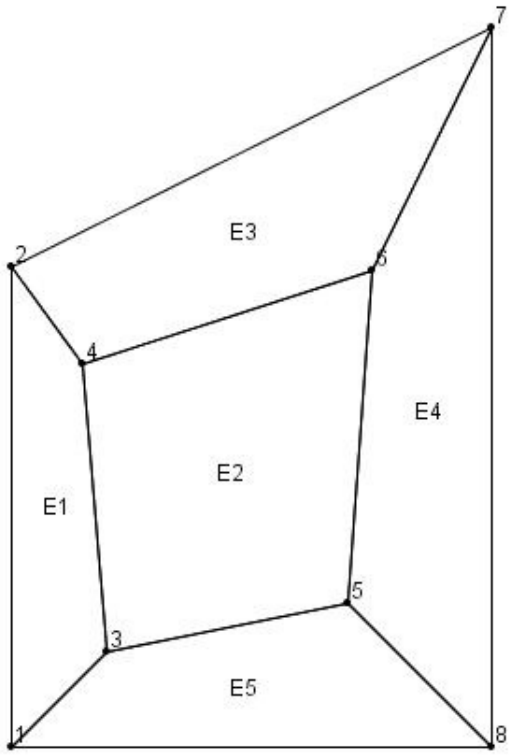


Figura 6.5: Malha para os testes A, B e C

Tabela 6.3: Informações para o *patch test* da figura 6.5

Nó	**Coordenadas (*uc*)**		**Deslocamentos (*uc*)**		**Forças (*uf*)**	
i	x_i	y_i	u_i	v_i	F_{xi}	F_{yi}
1	0,0	0,0	0,0	0,0	-2,0	0,0
2	0,0	2,0	0,0	-0,00120	-3,0	0,0
3	0,4	0,4	0,0008	-0,00024	0,0	0,0
4	0,3	1,6	0,0006	-0,00096	0,0	0,0
5	1,4	0,6	0,0028	-0,00036	0,0	0,0
6	1,5	2,0	0,0030	-0,00120	0,0	0,0
7	2,0	3,0	0,0040	-0,00180	2,0	0,0
8	2,0	0,0	0,0040	0,0	3,0	0,0

a) *Teste A*

Neste teste todos os nós da malha são restritos e deslocamentos de apoio são prescritos conforme a tabela 6.3, ou seja, são inseridos os valores exatos dos deslocamentos nodais (d_j) na i-ésima equação de equilíbrio, dada por:

$$K_{ij}\, d_j - f_i \equiv 0 \tag{6.2.1}$$

O teste é satisfeito se a igualdade acima for verificada. Após o processamento deste modelo, obteve-se os valores de forças nodais mostrados na tabela 6.3 e tensões nos pontos de integração de Gauss conforme a solução exata. Portanto este teste é atendido.

b) *Teste B*

Como no teste A não há uso explícito da inversão da matriz de rigidez, propõe-se o teste B, onde somente os nós externos da malha (1, 2, 7 e 8) são restritos e os deslocamentos de apoio para estes nós são prescritos conforme a tabela 6.3. Assim, a precisão da matriz de rigidez é testada e os deslocamentos calculados pelo programa para os nós internos da malha (3, 4, 5 e 6) coincidem com os valores dados na tabela 6.3. As tensões também coincidem com a solução exata e o maior erro ocorrido é de 10^{-15}.

c) *Teste C*

Neste teste o nó 1 está totalmente restrito e o nó 2 restrito somente na direção x. Forças nodais são aplicadas nos nós 7 e 8 para simular o efeito da tração uniforme $\sigma_{xx} = 2\,uf/uc^2$. Os valores das forças nos nós 7 e 8 também estão na tabela 6.3. Os resultados deste teste após processamento no programa coincidem com a solução exata, sendo os deslocamentos nodais mostrados na tabela 6.3 e as tensões nos pontos de integração de Gauss iguais a $\sigma_{xx} = 2\,uf/uc^2$.

6.3 Tração Axial

Este grupo de exemplos tem como objetivo testar os elementos finitos isoparamétricos de linha implementados, obtendo a solução discreta de uma barra submetida à carregamento distribuído linear na forma de tração. A barra em questão (figura 6.6) é horizontal, tem comprimento L e seção transversal constante A, sendo o material isotrópico com módulo de elasticidade longitudinal E. Para este exemplo é utilizado o modelo de análise *LineAnalysisM*, os elementos do tipo *L2, L3 e L4* e o carregamento *LineElementForce* (ver tabela 6.1).

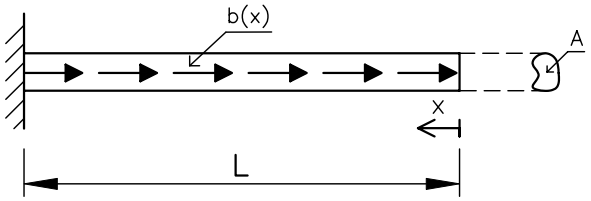


Figura 6.6: Barra em estudo

A figura 6.7 mostra a barra em estudo, submetida ao carregamento distribuído linear de tração. Para processamento, adota-se $L = 60{,}0\ uc$, $A = 2{,}0\ uc^2$ e $E = 30,0 \times 10^6\ uf/uc^2$. A solução exata para os deslocamentos *u(x)* e tensões $\sigma(x)$ deste problema é dada, respectivamente, por

$$\frac{3EA}{5L^3}\, u(x) = (\frac{x}{L})^3 - 1 \tag{6.3.1}$$

$$\frac{A}{5L^2}\, \sigma(x) = (\frac{x}{L})^2 \tag{6.3.2}$$

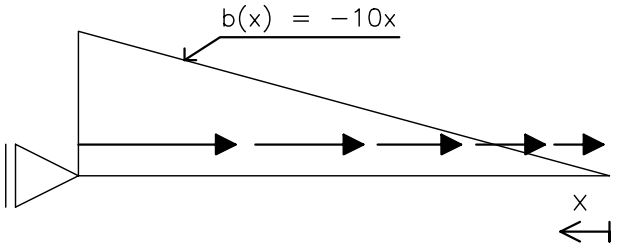


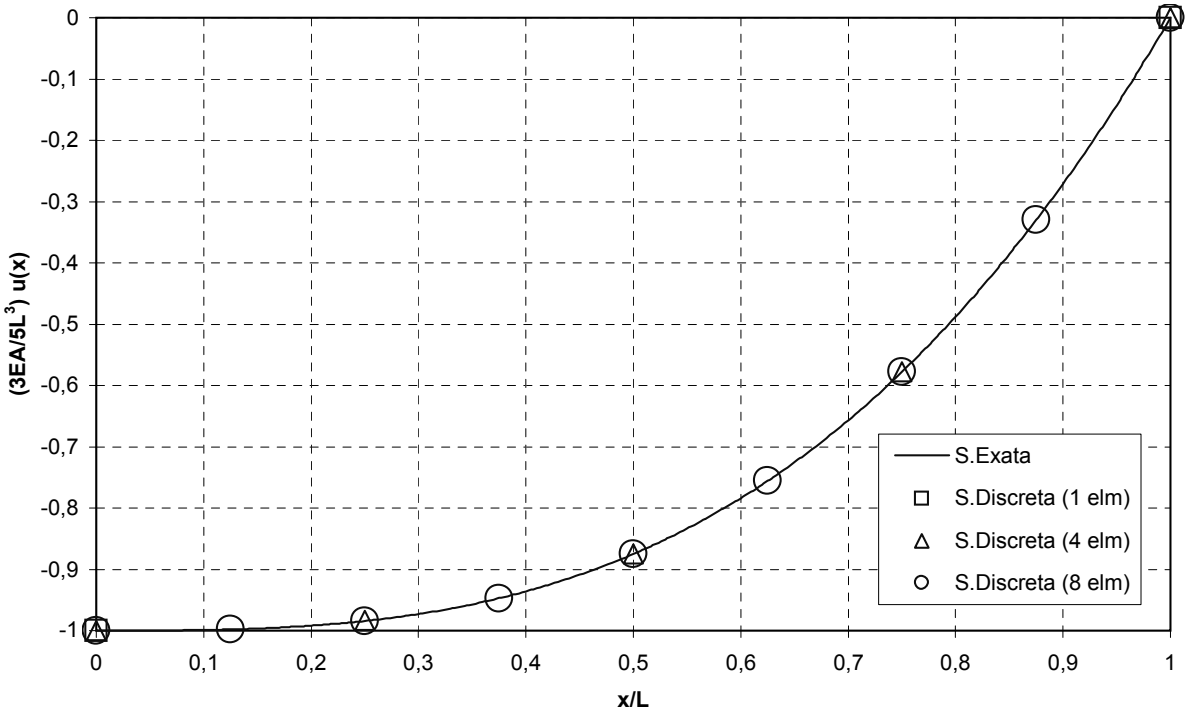
Figura 6.7: Barra submetida ao carregamento distribuído linear

Utiliza-se três malhas com elementos finitos do tipo *L2*: com 1, 4 e 8 elementos. A figura 6.8 (a) mostra os resultados, em unidades adimensionais, para os deslocamentos nodais calculados pelo programa para as três malhas, juntamente com a solução exata. Os valores das tensões nos pontos de integração são mostrados no gráfico da figura 6.8 (b). Neste exemplo é utilizado um ponto de integração de Gauss para o cálculo da matriz de rigidez de cada elemento finito, onde as tensões são obtidas.

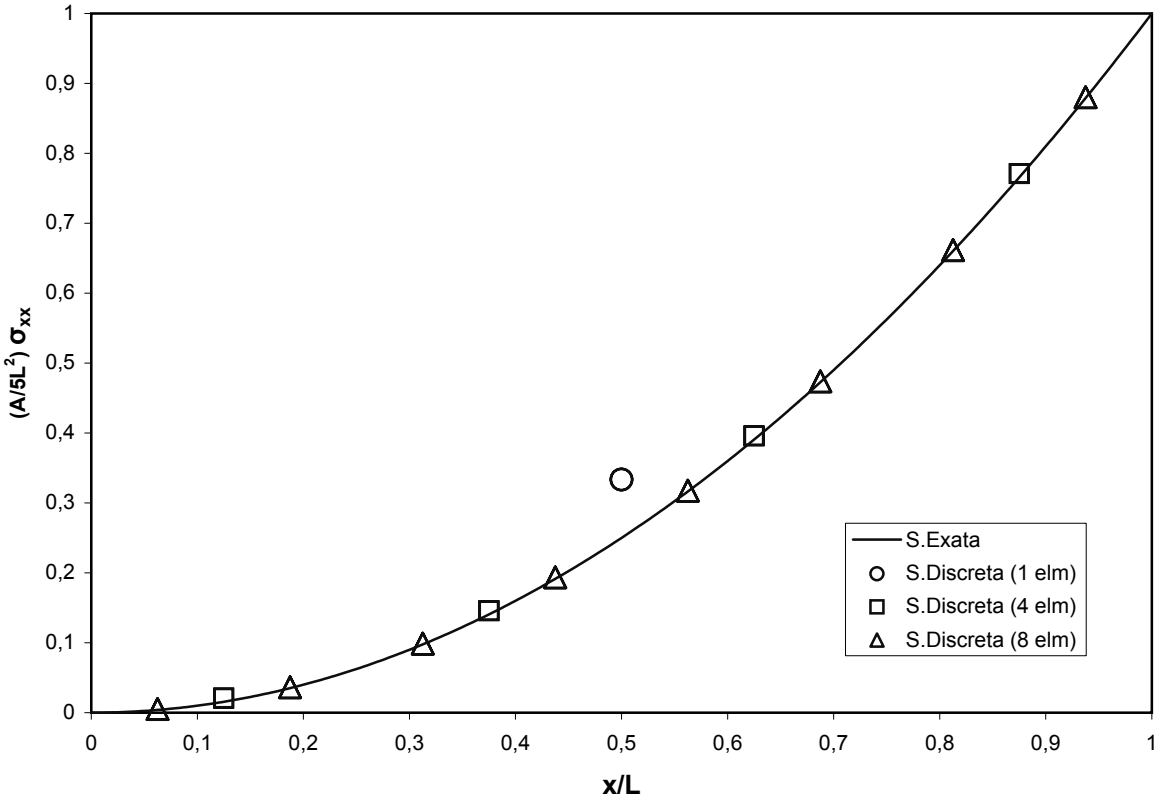
A discretização da barra em estudo com elementos *L3* é feita com 2 *L3* e 4 *L3* e os resultados, em unidades adimensionais, para deslocamentos nodais calculados pelo programa aparecem na figura 6.9 (a) juntamente com a solução exata. Os valores das tensões nos pontos de integração são mostrados no gráfico da figura 6.9 (b). Neste exemplo são utilizados dois pontos de integração de Gauss para o cálculo da matriz de rigidez de cada elemento finito, onde as tensões são obtidas.

Utilizando o elemento *L4* para modelar a barra em estudo são adotadas as discretizações com 2 *L4* e 4 *L4*, sendo os resultados para os deslocamentos nodais calculados pelo programa apresentados na figura 6.10 (a), juntamente com a solução exata. Os valores das tensões nos pontos de integração são mostrados no gráfico da figura 6.10 (b). Neste exemplo são utilizados três pontos de integração de Gauss para o cálculo da matriz de rigidez de cada elemento finito, onde as tensões são obtidas.

Observando-se os gráficos de deslocamentos nodais e tensões normais, para os elementos de linha de dois, três e quatro nós submetidos à tração axial, conclui-se que todos convergem para a solução exata ao refinar a malha.

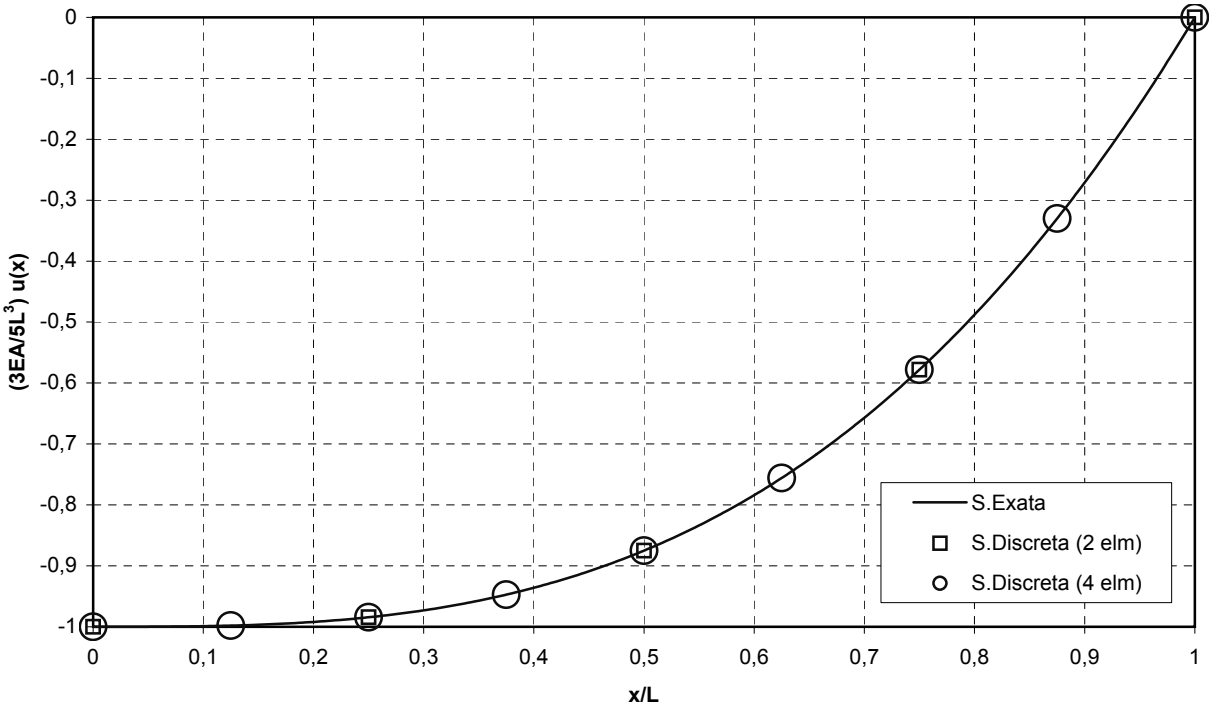


(a) Valores de deslocamentos

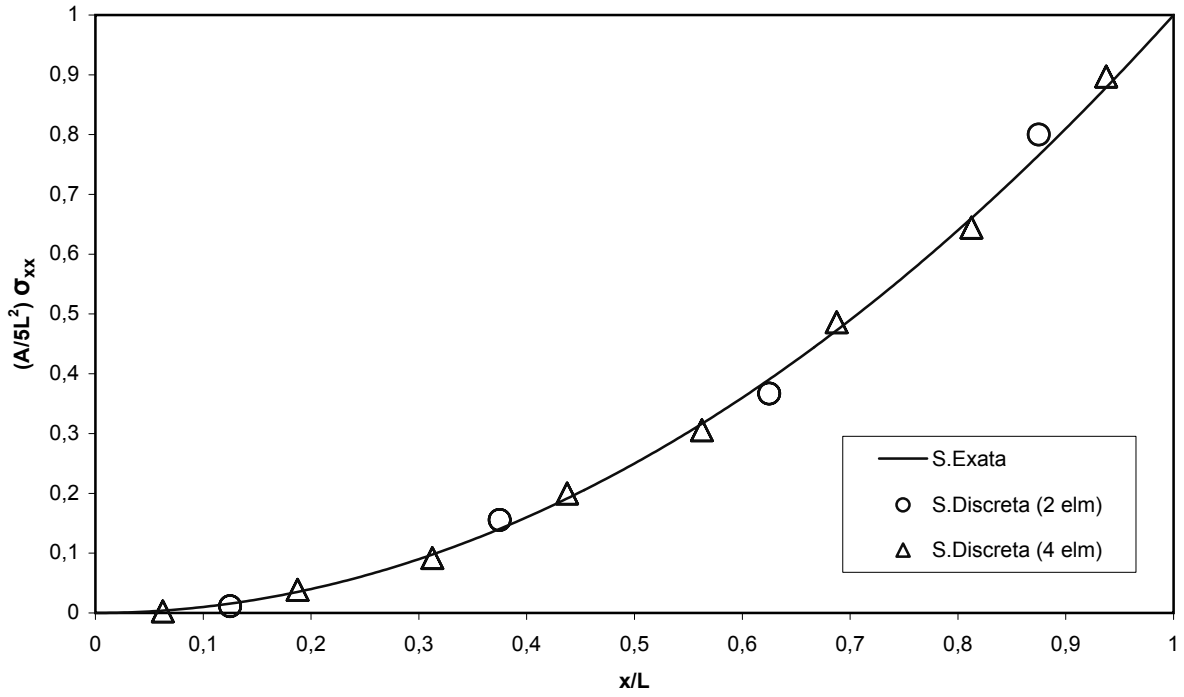


(b) Valores das tensões nos pontos de Gauss

Figura 6.8: Resultados obtidos com elementos L2

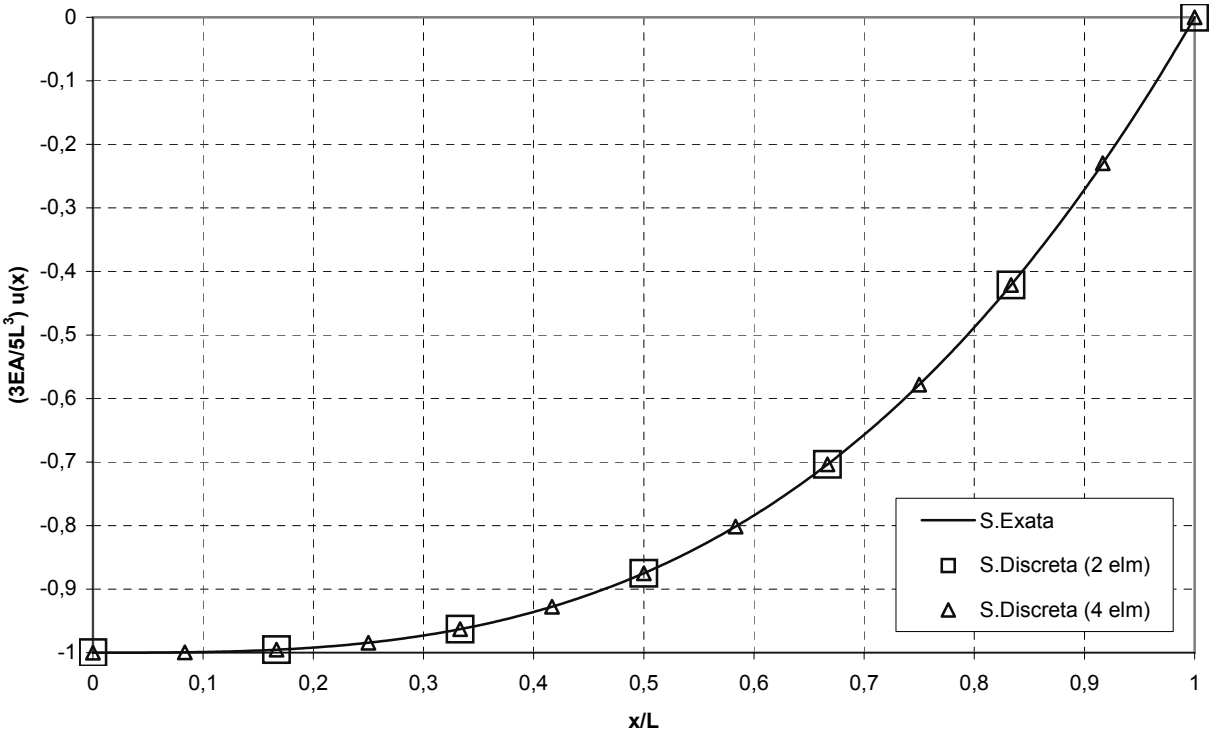


(a) Valores de deslocamentos

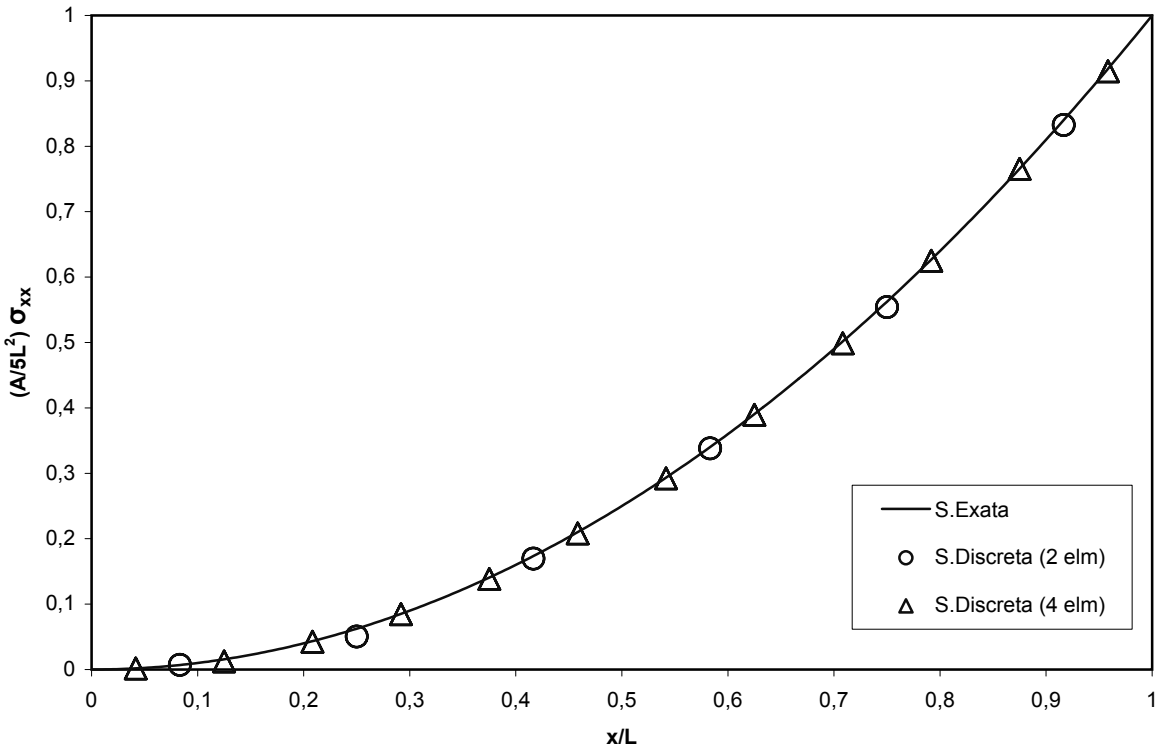


(b) Valores das tensões nos pontos de Gauss

Figura 6.9: Resultados obtidos com elementos L3



(a) Valores de deslocamentos



(b) Valores das tensões nos pontos de Gauss

Figura 6.10: Resultados obtidos com elementos L4

6.4 Viga Parede

Este exemplo tem o objetivo de verificar os elementos triangulares e quadrilaterais implementados, modelando uma viga parede submetida a um carregamento distribuído constante.

As configurações geométrica e de cargas da viga estão mostradas na figura 6.11. As condições de apoio devem ser tais que os deslocamentos verticais das faces laterais da viga sejam nulos. Considera-se material isotrópico com $E = 2 \times 10^5\ MPa$ e $\nu = 0,3$. Assim, são utilizados o modelo de análise *PlaneStressAnalysisM*, carregamento do tipo *LineElementForce* e os elementos *T3*, *T6*, *T10*, *Q4*, *Q8* e *Q9* para discretizar este modelo (ver tabela 6.1).

A solução exata deste problema pode ser encontrada em (Timoshenko & Goodier 1980). O valor da flecha no centro da viga (x = y = 0) é $\delta = 0,7931\ mm$ e a variação de tensões σ_{xx} para a seção em x = 0 é dada na equação 6.4.1.

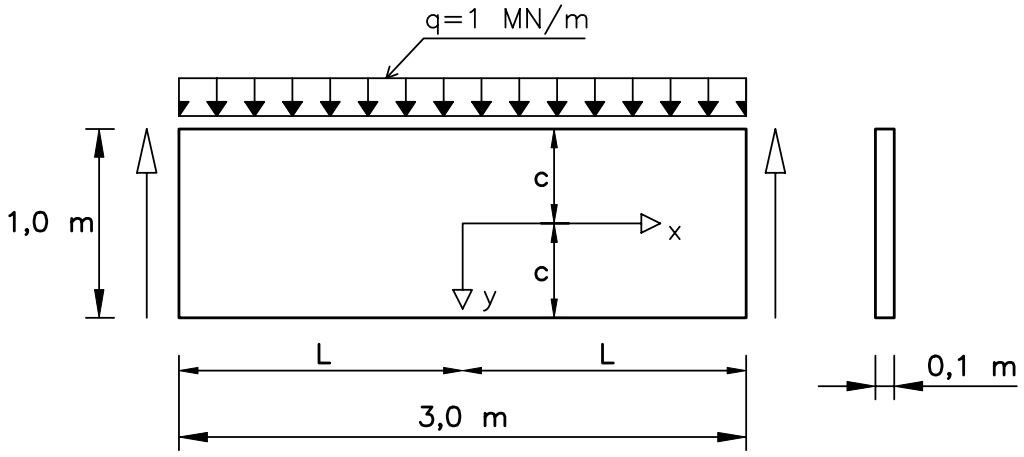


Figura 6.11: Viga parede proposta

$$\sigma_{xx} = \frac{q\,L^2}{2I}\,y + \frac{q}{2I}\,(\frac{2}{3}y^3 - \frac{2}{5}c^2 y) \;=\; 40\,y^3 + 129\,y \tag{6.4.1}$$

6.4.1 Malhas com Elementos Triangulares de Três Nós

Para discretizar o modelo proposto da viga parede com elementos *T3* são utilizadas as malhas mostradas nas figuras 6.12, 6.13 e 6.14, com 16, 96 e 192 elementos, respectivamente. Os resultados para o deslocamento δ no meio da viga (x = y = 0) aparecem na tabela 6.4. A variação das tensões σ_{xx} ao longo da altura da viga é mostrada no gráfico da figura 6.15 e refere-se aos valores de σ_{xx} nos pontos de Gauss localizados em uma reta vertical nos elementos à esquerda do eixo de simetria da viga parede. Para estas três malhas é utilizado um ponto de integração de Gauss em cada elemento finito.

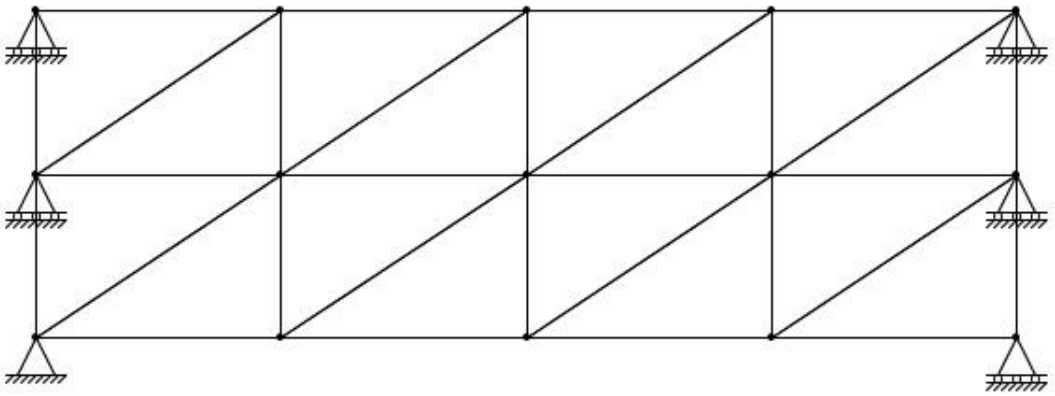


Figura 6.12: Malha com 16 elementos *T3*

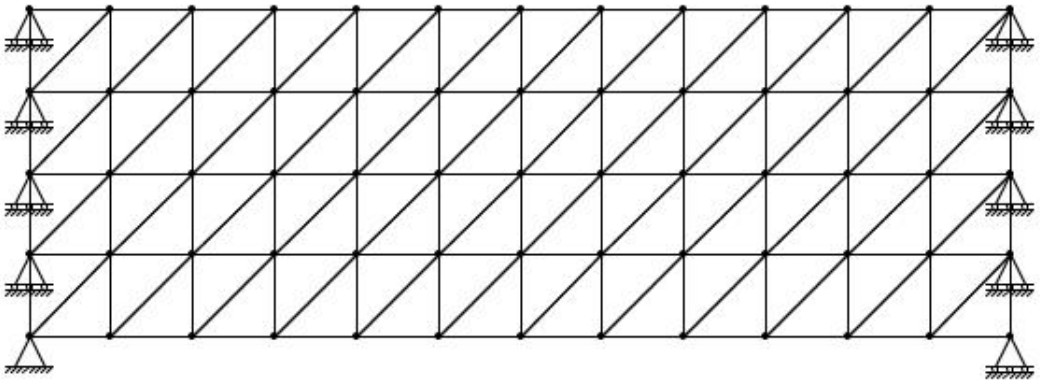


Figura 6.13: Malha com 96 elementos *T3*

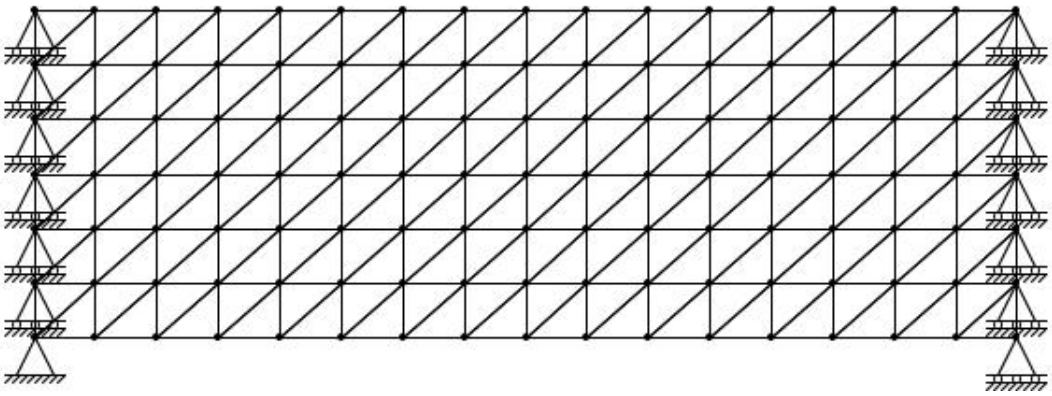


Figura 6.14: Malha com 192 elementos *T3*

Tabela 6.4: Deslocamentos para as malhas de elementos *T3*

Malha	$\delta\ (mm)$	Erro percentual relativo
16 *T3*	0,4019	49,32
96 *T3*	0,6532	17,64
192 *T3*	0,7155	9,78

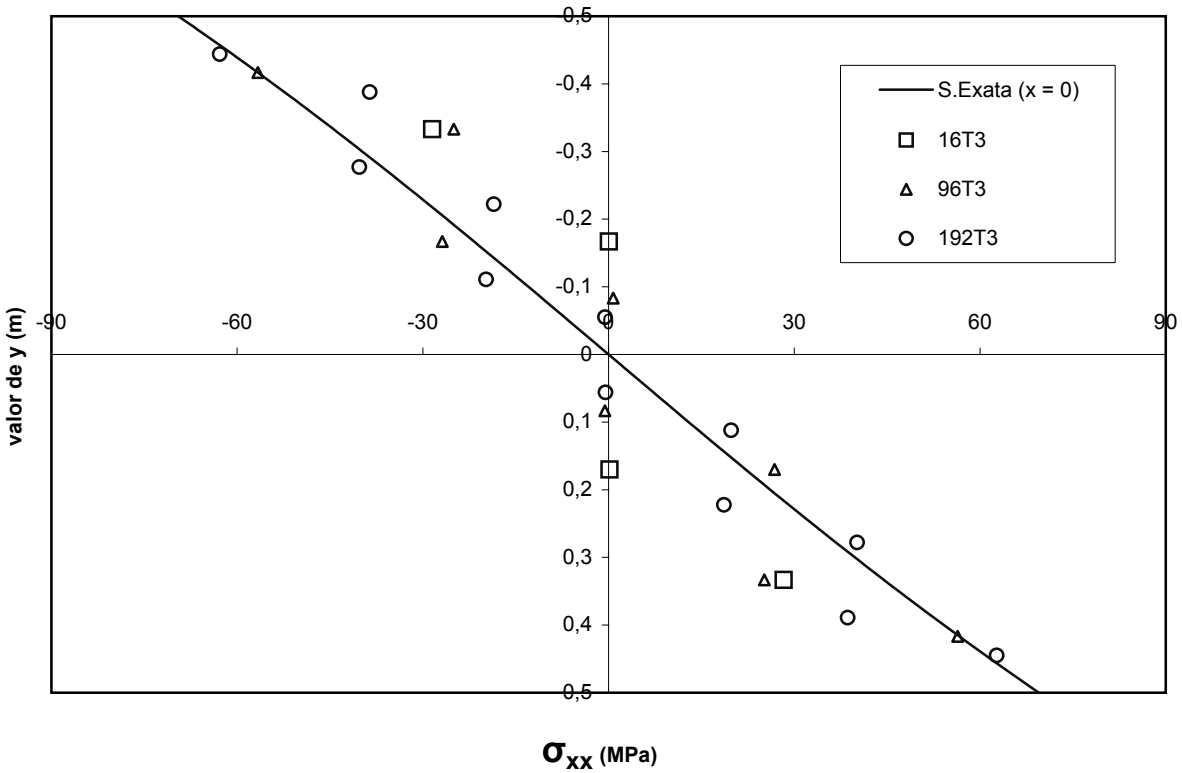


Figura 6.15: Variação das tensões σ_{xx} para as malhas de elementos *T3*

6.4.2 Malhas com Elementos Triangulares de Seis Nós

Nas figuras 6.16, 6.17 e 6.18 estão apresentadas as malhas de elementos *T6* adotados para discretizar a viga parede em estudo. São utilizados 4, 24 e 48 elementos finitos. Os resultados para o deslocamento δ no meio da viga (x = y = 0) aparecem na tabela 6.5. A variação das tensões σ_{xx} ao longo da altura da viga é mostrada no gráfico da figura 6.19 e refere-se aos valores de σ_{xx} nos pontos de Gauss localizados em uma reta vertical nos elementos à esquerda do eixo de simetria da viga parede. Para estas três malhas são utilizados três pontos de integração de Gauss em cada elemento finito.

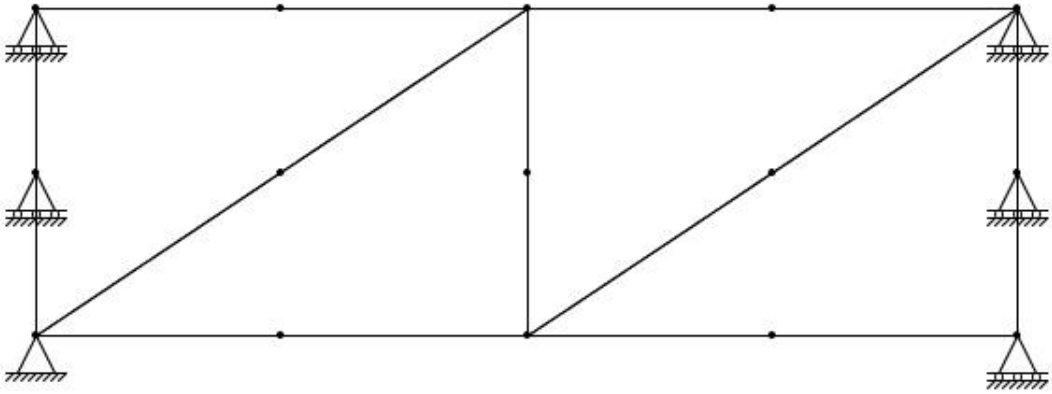


Figura 6.16: Malha com 4 elementos *T6*

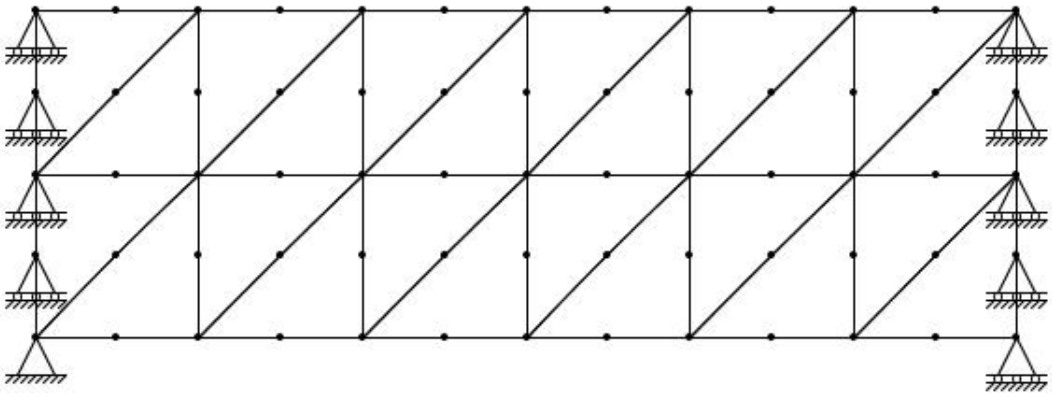


Figura 6.17: Malha com 24 elementos *T6*

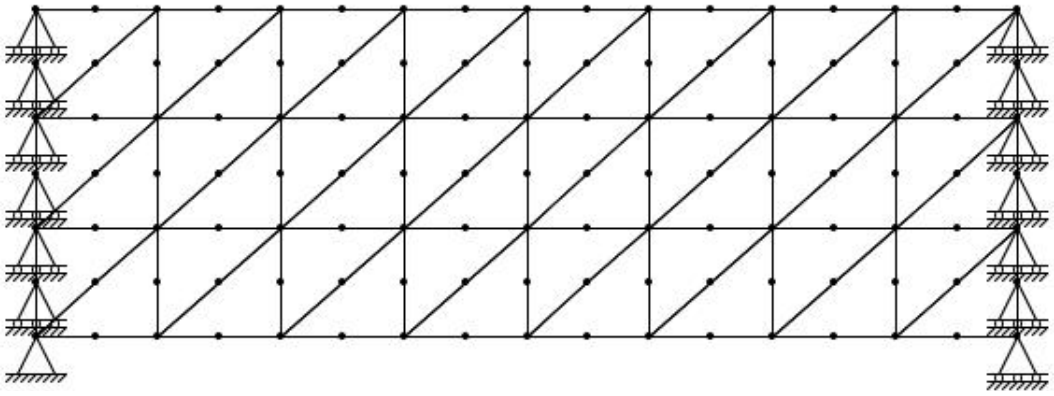


Figura 6.18: Malha com 48 elementos *T6*

Tabela 6.5: Deslocamentos para as malhas de elementos *T6*

Malha	$\delta\ (mm)$	Erro percentual relativo
4 *T6*	0,7018	11,51
24 *T6*	0,7863	0,86
48 *T6*	0,7898	0,42

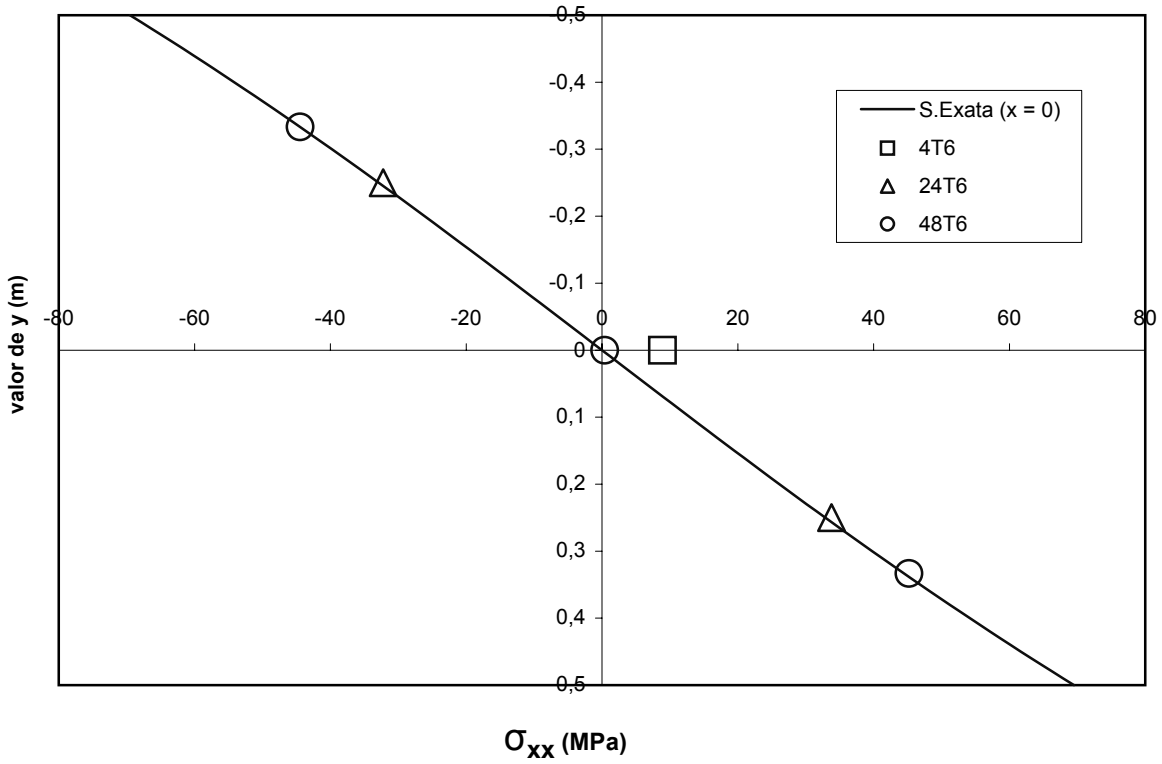


Figura 6.19: Variação das tensões σ_{xx} para as malhas de elementos *T6*

6.4.3 Malhas com Elementos Triangulares de Dez Nós

Nas figuras 6.20 e 6.21 estão apresentadas as malhas de elementos *T10* consideradas para discretizar a viga parede em estudo. São utilizados 8 e 10 elementos finitos. Os resultados para o deslocamento δ no meio da viga aparecem na tabela 6.6. A variação das tensões σ_{xx} ao longo da altura da viga é mostrada no gráfico da figura 6.22 e refere-se aos valores de σ_{xx} nos pontos de Gauss localizados em uma reta vertical nos elementos à esquerda do eixo de simetria da viga parede. Para estas três malhas são utilizados seis pontos de integração de Gauss em cada elemento finito.

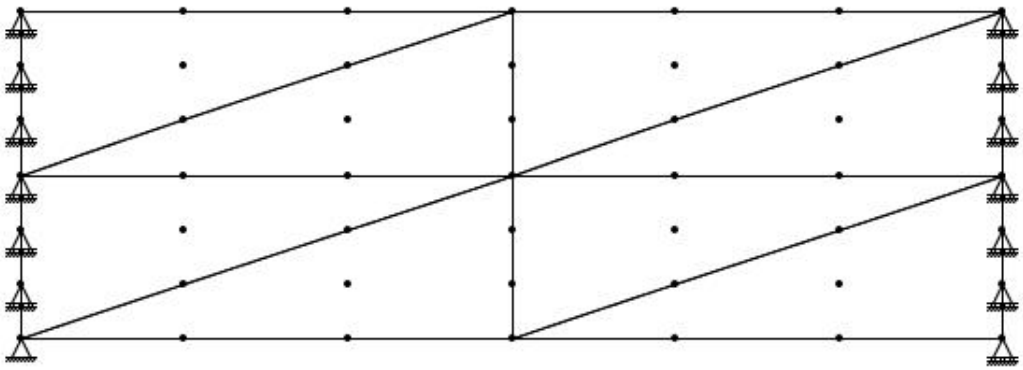


Figura 6.20: Malha com 8 elementos *T10*

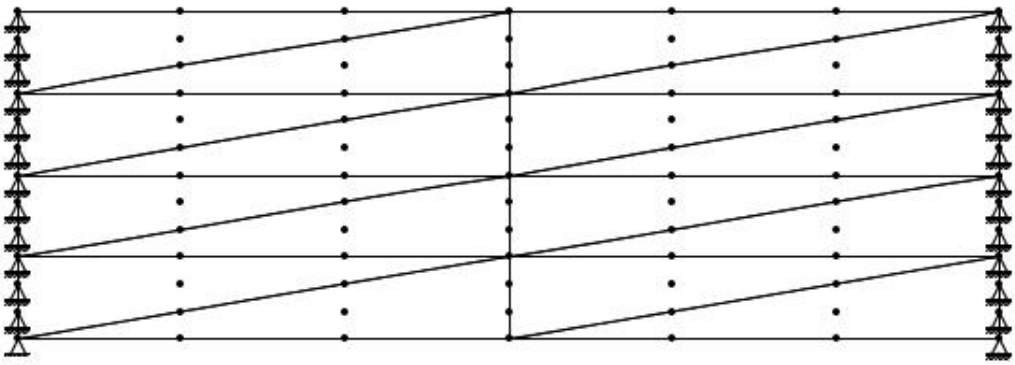


Figura 6.21: Malha com 16 elementos *T10*

Tabela 6.6: Deslocamentos para as malhas de elementos *T10*

Malha	δ (mm)	Erro percentual relativo
8 *T10*	0,7710	2,79
16 *T10*	0,7714	2,74

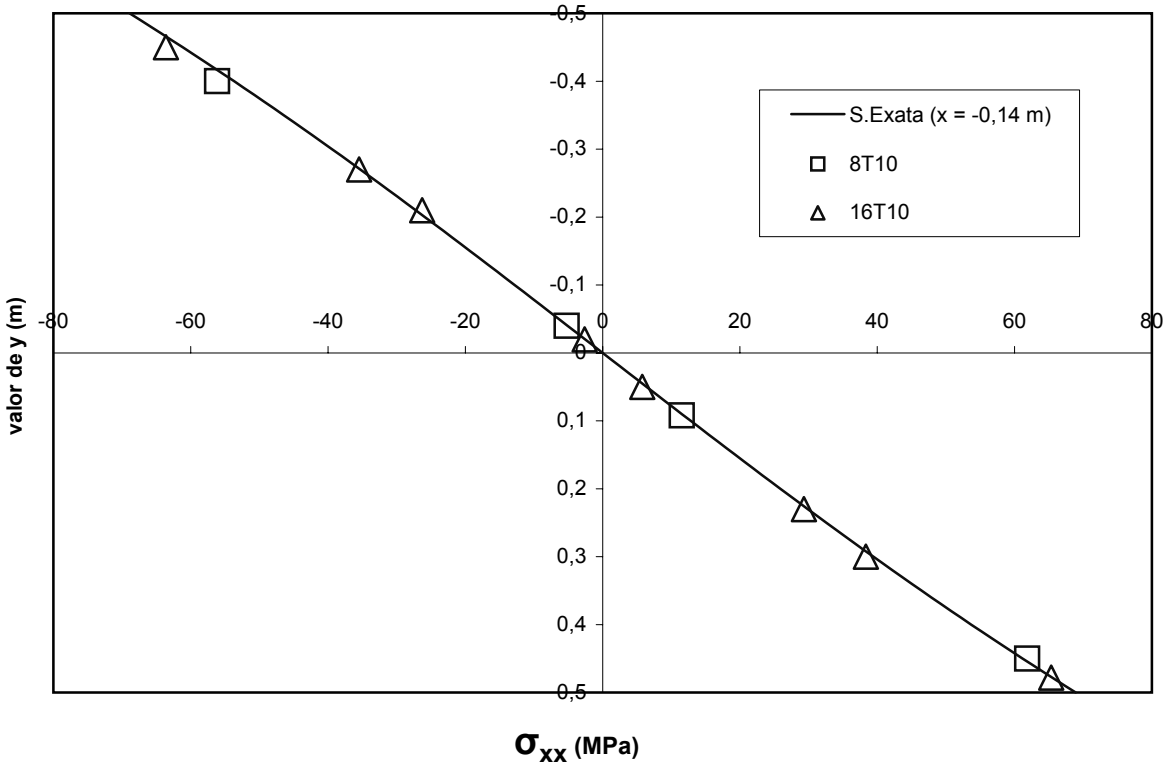


Figura 6.22: Variação das tensões σ_{xx} para as malhas de elementos *T10*

6.4.4 Malhas com Elementos Quadrilaterais de Quatro Nós

Para discretizar o modelo proposto da viga parede com elementos *Q4* são utilizadas as malhas mostradas nas figuras 6.23, 6.24 e 6.25 com 3, 12 e 48 elementos, respectivamente. Os resultados para o deslocamento δ no meio da viga (x = y = 0) aparecem na tabela 6.7. A variação das tensões σ_{xx} ao longo da altura da viga é mostrada no gráfico da figura 6.26 e refere-se aos valores de σ_{xx} nos pontos de Gauss localizados em uma reta vertical nos elementos à esquerda do eixo de simetria das malhas. Para estas três malhas são utilizados 2×2 pontos de integração de Gauss em cada elemento finito.

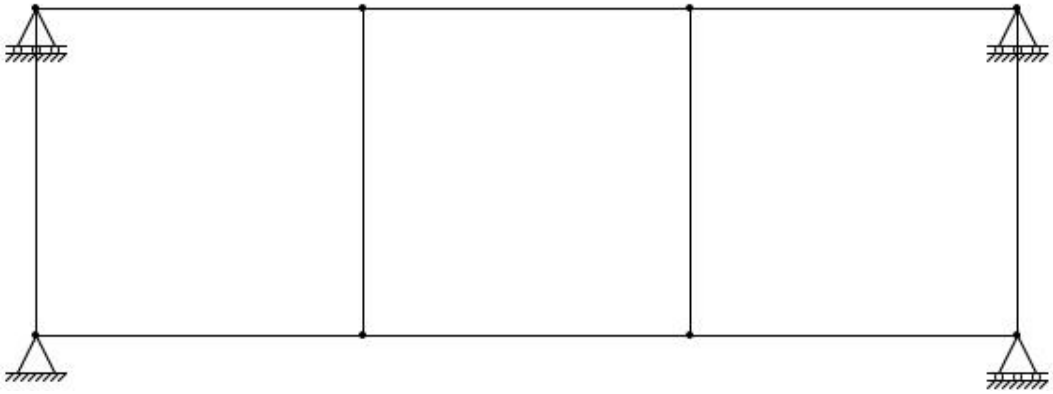


Figura 6.23: Malha com 3 elementos *Q4*

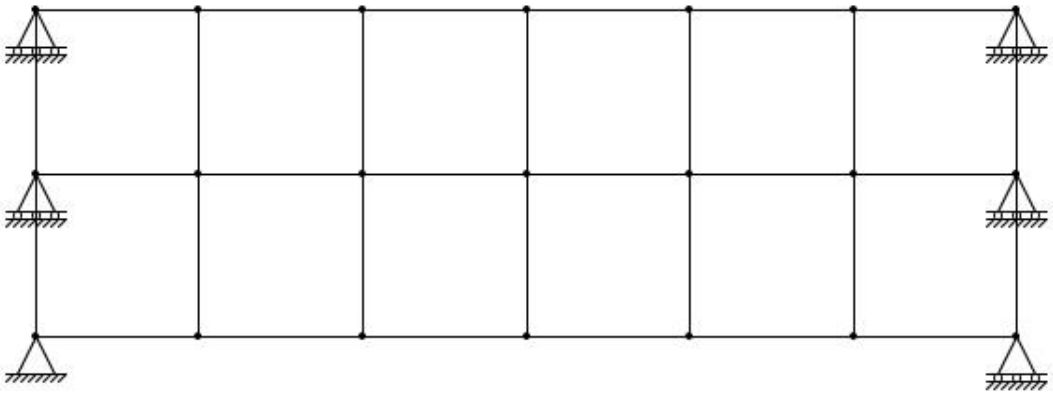


Figura 6.24: Malha com 12 elementos *Q4*

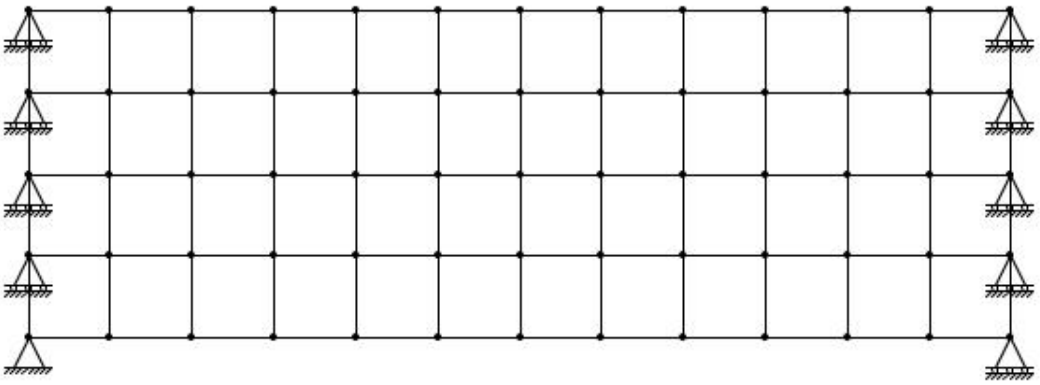


Figura 6.25: Malha com 48 elementos *Q4*

Tabela 6.7: Deslocamentos para as malhas de elementos *Q4*

Malha	δ (mm)	Erro percentual relativo
3 *Q4*	0,4333	45,37
12 *Q4*	0,6816	14,06
48 *Q4*	0,7594	4,25

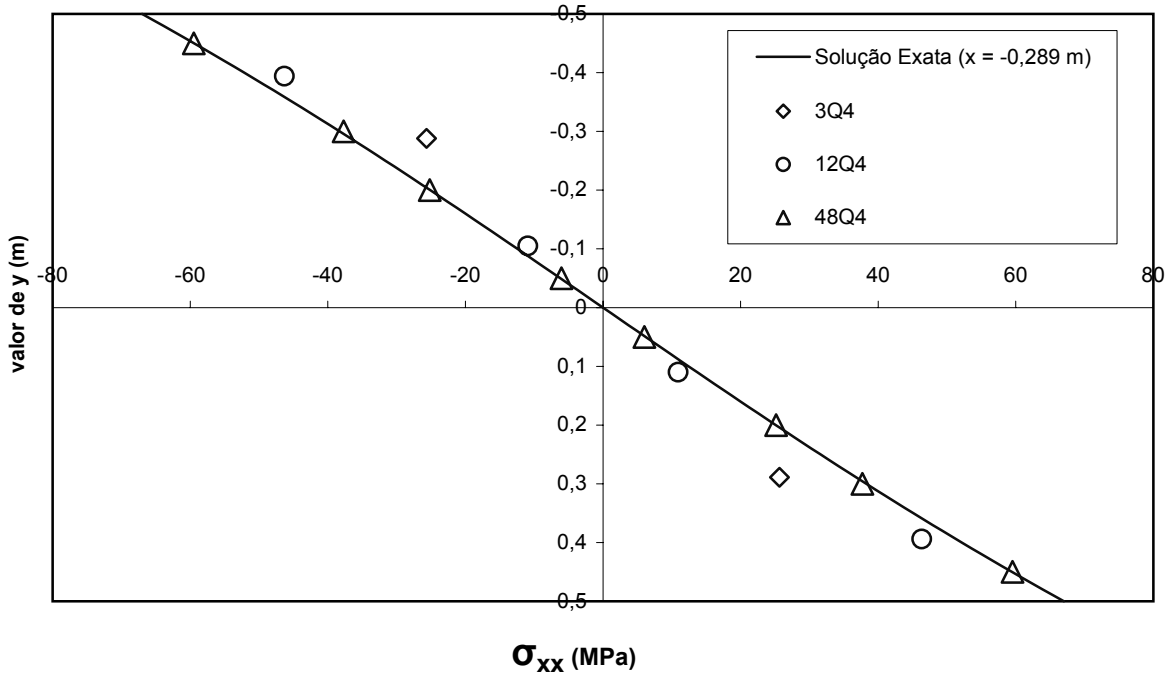


Figura 6.26: Variação das tensões σ_{xx} para as malhas de elementos *Q4*

6.4.5 Malhas com Elementos Quadrilaterais de Oito Nós

Para discretizar o modelo proposto da viga parede com elementos *Q8* são utilizadas as malhas mostradas nas figuras 6.27, 6.28 e 6.29 com 3, 12 e 48 elementos, respectivamente. Os resultados para o deslocamento δ no meio da viga aparecem na tabela 6.8. A variação das tensões σ_{xx} ao longo da altura da viga é mostrada no gráfico da figura 6.30 e refere-se aos valores de σ_{xx} nos pontos de Gauss localizados em uma reta vertical nos elementos à esquerda do eixo de simetria das malhas. Para estas três malhas são utilizados 3×3 pontos de integração de Gauss em cada elemento finito.

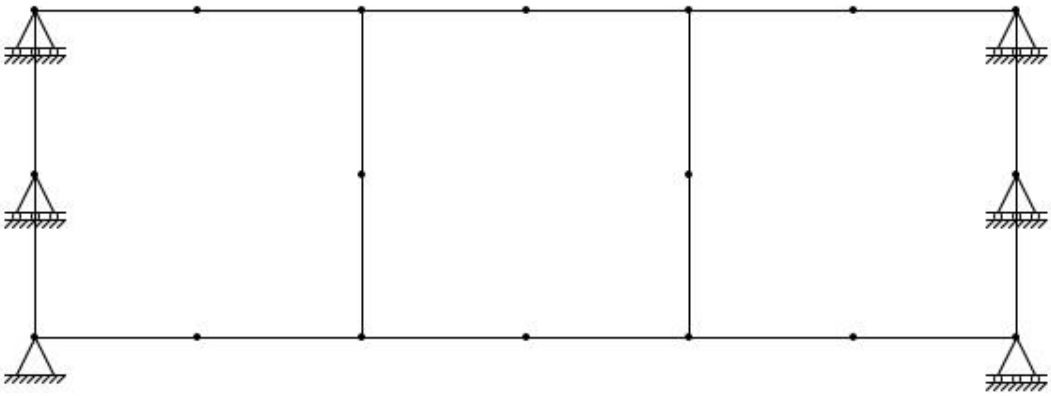


Figura 6.27: Malha com 3 elementos *Q8*

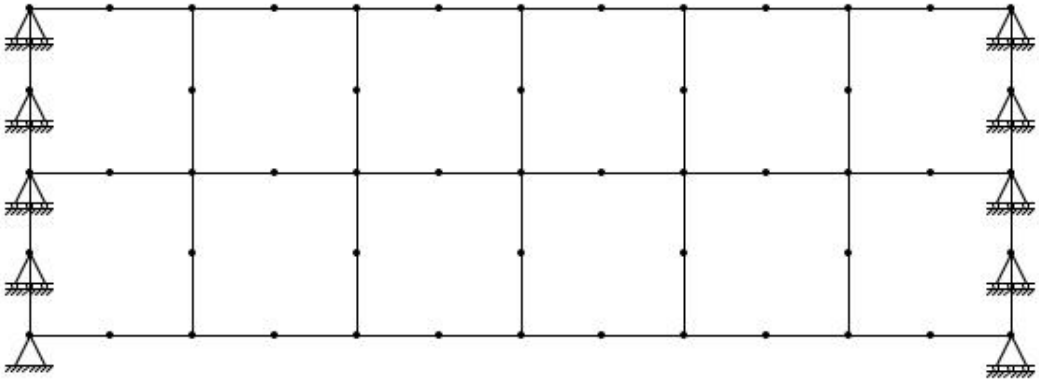


Figura 6.28: Malha com 12 elementos *Q8*

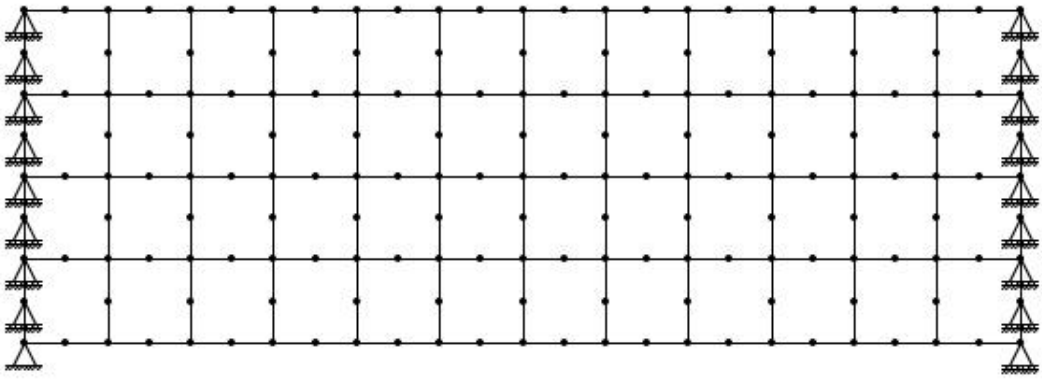


Figura 6.29: Malha com 48 elementos *Q8*

Tabela 6.8: Deslocamentos para as malhas de elementos *Q8*

Malha	$\delta\ (mm)$	Erro percentual relativo
3 *Q8*	0,7622	3,90
12 *Q8*	0,7989	0,53
48 *Q8*	0,7908	0,29

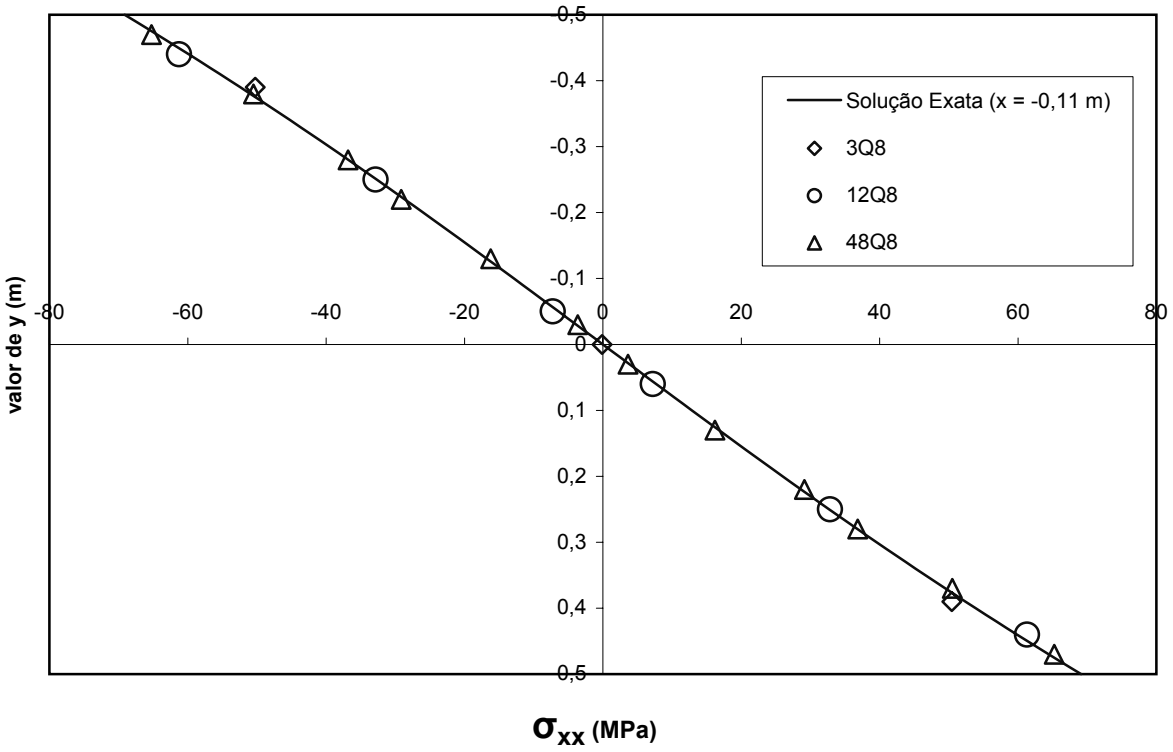


Figura 6.30: Variação das tensões σ_{xx} para as malhas de elementos *Q8*

6.4.6 Malhas com Elementos Quadrilaterais de Nove Nós

Para discretizar o modelo proposto da viga parede com elementos *Q9* são utilizadas as malhas mostradas nas figuras 6.31, 6.32 e 6.33 representando 3, 12 e 48 elementos, respectivamente. Os resultados para o deslocamento δ no meio da viga aparecem na tabela 6.9. A variação das tensões σ_{xx} ao longo da altura da viga é mostrada no gráfico da figura 6.34 e referem-se aos valores de σ_{xx} nos pontos de Gauss localizados em uma reta vertical nos elementos à esquerda do eixo de simetria das malhas. Para estas três malhas são utilizados 3×3 pontos de integração de Gauss em cada elemento finito.

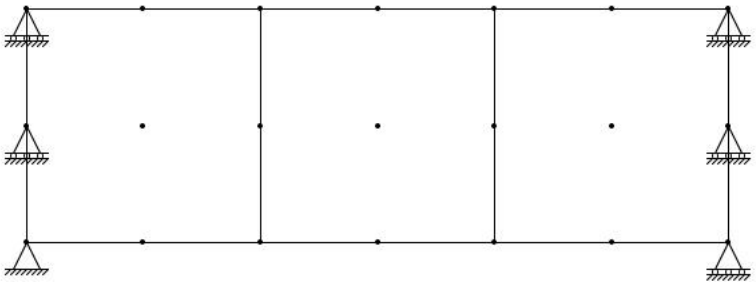


Figura 6.31: Malha com 3 elementos *Q9*

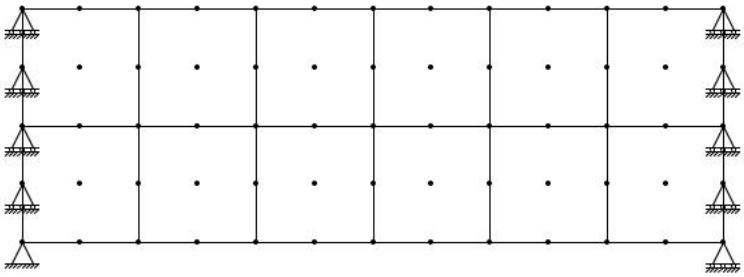


Figura 6.32: Malha com 12 elementos *Q9*

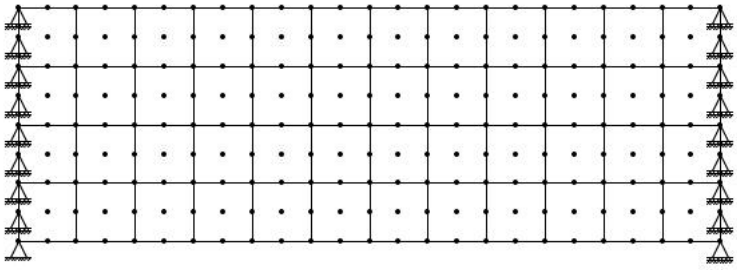


Figura 6.33: Malha com 48 elementos *Q9*

Tabela 6.9: Deslocamentos para as malhas de elementos *Q9*

Malha	δ (mm)	Erro percentual relativo
3 *Q9*	0,7629	3,80
12 *Q9*	0,7889	0,53
48 *Q9*	0,7908	0,29

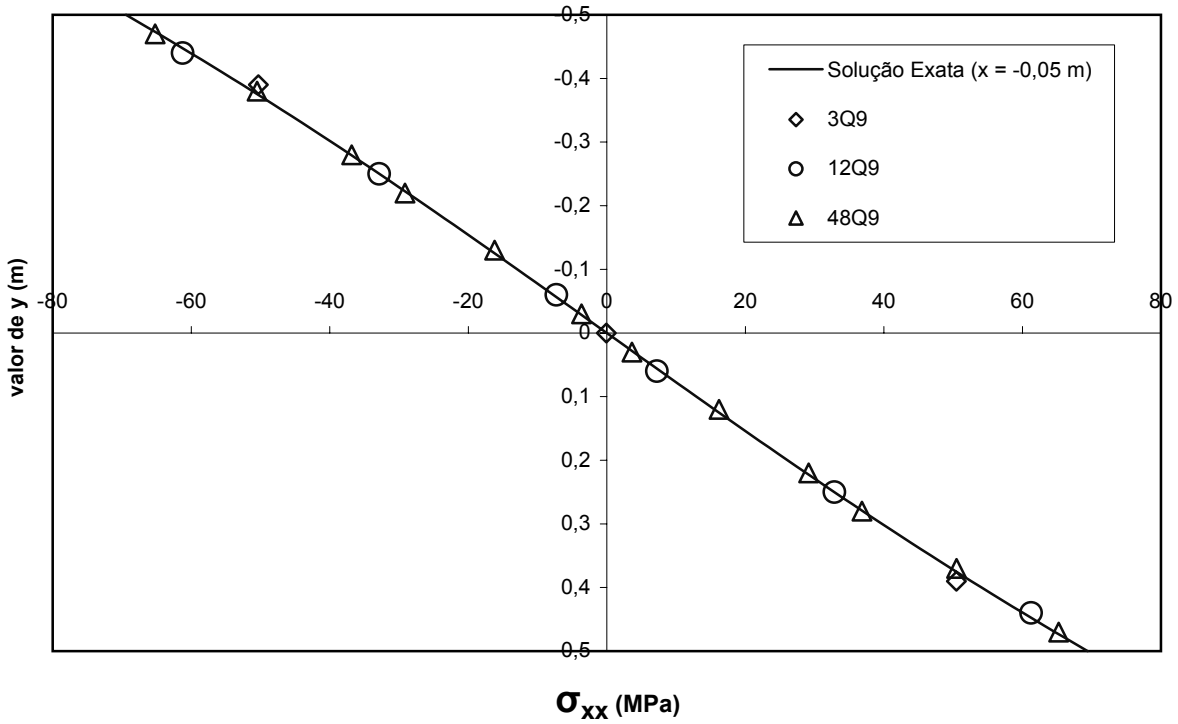


Figura 6.34: Variação das tensões σ_{xx} para as malhas de elementos *Q9*

Observando os resultados deste exemplo pode-se constatar que, ao utilizar diferentes elementos triangulares e quadrilaterais, para modelar um mesmo problema, ocorre convergência dos resultados de tensões σ_{xx} para a solução exata quando as malhas são refinadas. Constata-se também que o percentual de erro no deslocamento vertical da seção estudada diminui ao refinar a malha.

6.5 Viga de Concreto Armado

Neste exemplo modela-se uma viga de concreto armado com aço, submetida a um carregamento distribuído constante vertical conforme a figura 6.35. A viga tem comprimento $2L = 150\ uc$, base $b = 20\ uc$, altura $2c = 50\ uc$, área de aço $A_s = 10\,uc^2$, carregamento distribuído constante $q = 1{,}75\ uf/uc$ e condições de apoio tais que os deslocamentos verticais das faces laterais da viga são nulos. Para o concreto considera-se módulo de elasticidade $E_c = 4,4 \times 10^3\,uf/uc^2$ e coeficiente de Poisson $\nu_c = 0{,}2$; para o aço adota-se módulo de elasticidade $E_s = 2,1 \times 10^4\,uf/uc^2$. Neste exemplo são utilizados o modelo de análise *PlaneStressAnalysisM*, carregamento do tipo *LineElementForce*, elementos quadrilaterais *Q4* para modelar a viga de concreto e elementos de linha *L2* para modelar a armadura de aço. Todos os elementos *L2* são dispostos ao longo da reta horizontal y = 20 *uc* para modelar a armadura inferior da viga com cobrimento igual a 5 *uc*.

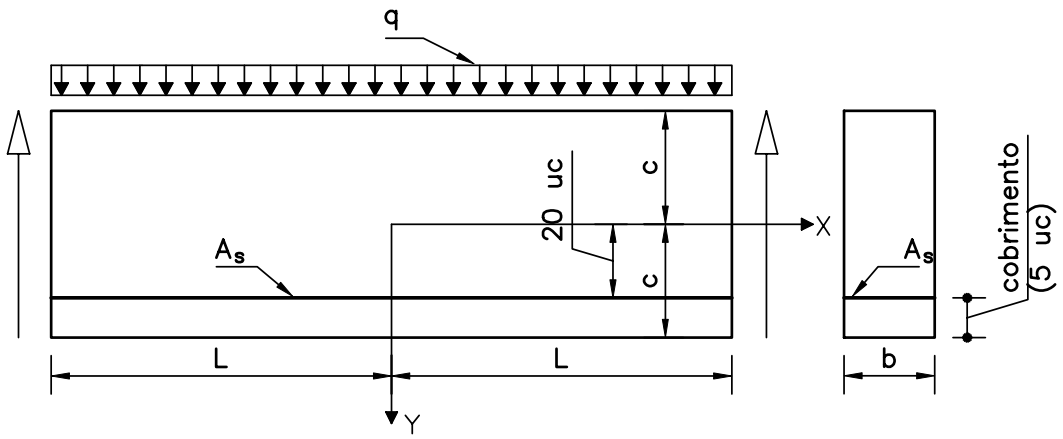


Figura 6.35: Viga armada proposta

A discretização da viga armada é mostrada na figura 6.36. Os resultados para as tensões σ_{xx} ao longo da reta x = -1,06 *uc*, coincidentes com uma fila de pontos de Gauss são obtidos no programa e aparecem no gráfico da figura 6.37, para as vigas com e sem armadura.

No gráfico da figura 6.37 pode-se verificar tensões de tração na parte inferior da viga e de compressão na parte superior conforme esperado. No mesmo gráfico verifica-se a diminuição da tensão σ_{xx} no concreto ao incluir no modelo o efeito das barras de armadura. Percebe-se também o valor da tensão de tração σ_{xx} em y = 20 *uc*, correspondendo à tensão

na barra de aço.

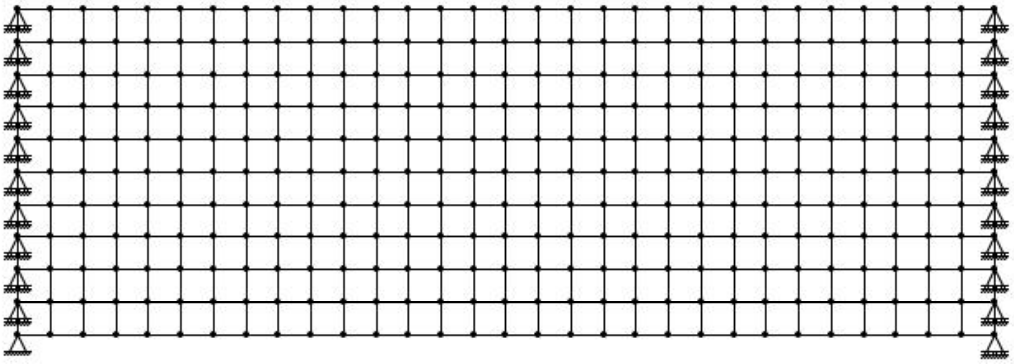


Figura 6.36: Discretização da viga armada com elementos *Q4* e *L2*

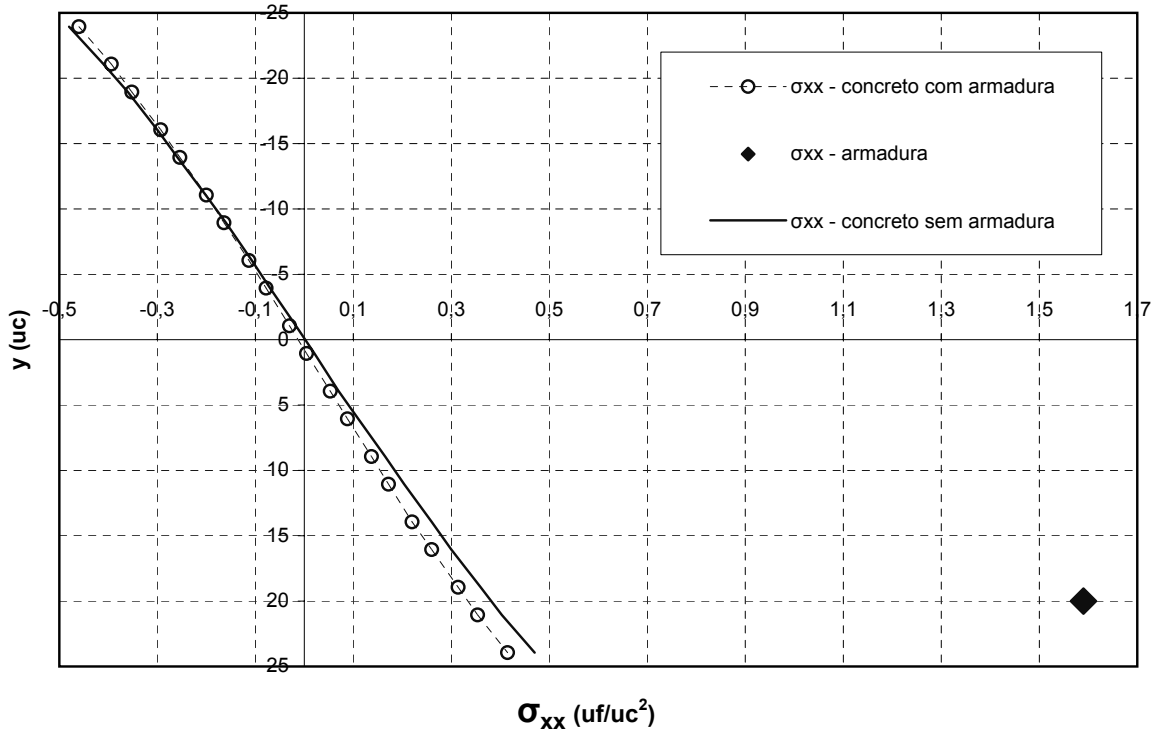


Figura 6.37: Tensões nos pontos de Gauss na reta x = -1,06 uc - viga da figura 6.35

6.6 Chapa com Furo Circular

Nesta seção o programa implementado é utilizado para obter a solução discreta do problema de uma chapa com furo circular central, submetida a um carregamento de tração em suas extremidades verticais, mostrada na figura 6.38. São utilizados os dados propostos na referência (Dawe 1983), ou seja, $E = 2 \times 10^4\, uf/uc^2$, $\nu = 0{,}25$ e espessura da chapa $t = 1\ uc$. O carregamento considerado é $q = 7 \times 10^{-4}\ uf/uc$. A solução exata para tensões σ_{xx} deste problema pode ser obtida pela teoria da elasticidade em (Timoshenko & Goodier 1980), sendo mostrada na equação 6.6.1. Utiliza-se o modelo de análise *PlaneStressAnalysisM*, carregamento do tipo *LineElementForce* e os elementos quadrilaterais *Q8*. Tirando-se proveito da dupla simetria do problema pode-se discretizar apenas um quarto da chapa furada e considerar condições de apoio coerentes com os dois eixos de simetria. Assim, é discretizado o quarto superior da chapa, correspondente a x e y positivos, com as malhas de 11, 25, 40 e 80 elementos do tipo *Q8*, mostradas na figura 6.39.

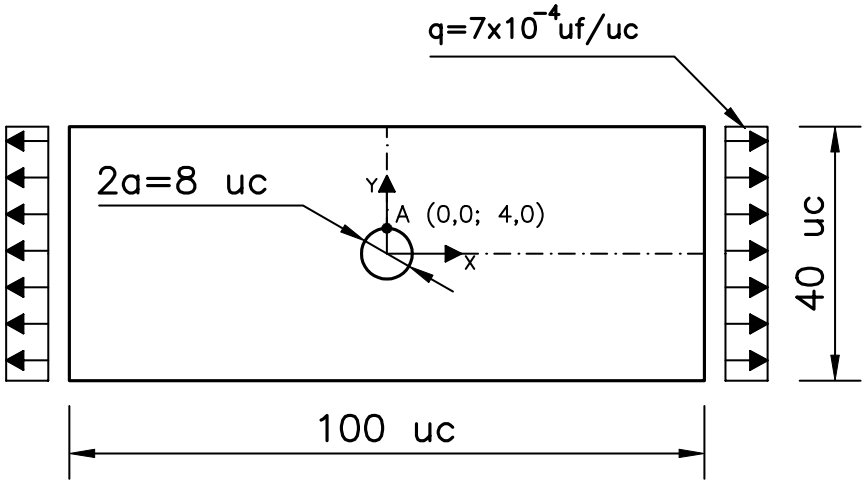
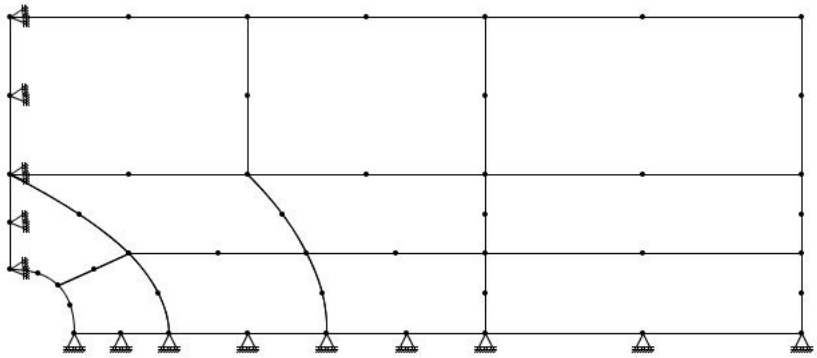


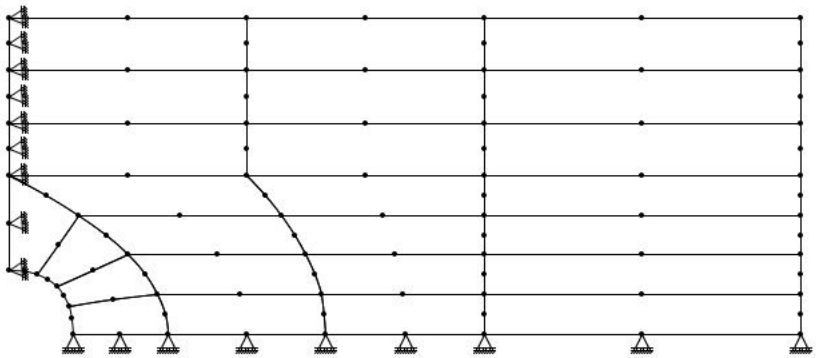
Figura 6.38: Chapa com furo circular

$$\sigma_{xx} = \frac{q}{2t}\,(2 + \frac{a^2}{y^2} + 3\,\frac{a^4}{y^4}) \tag{6.6.1}$$

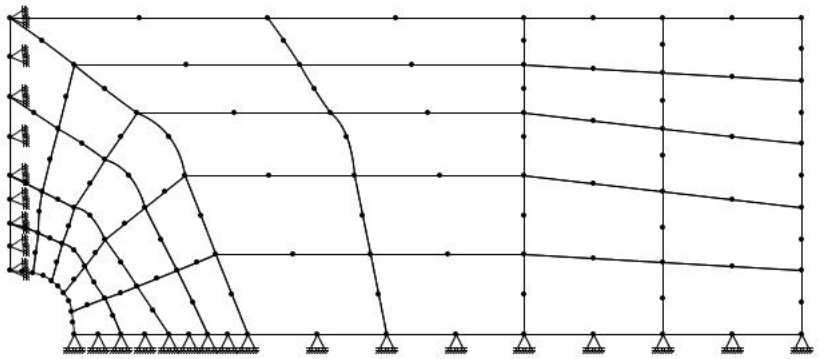
Os valores da tensão σ_{xx} obtidos no ponto A (x = 0,0; y = 4,0) da figura 6.38 para as quatro malhas são apresentados, na forma do quociente $(\sigma_{xx} \cdot t/\,q)$, no gráfico da figura 6.40.



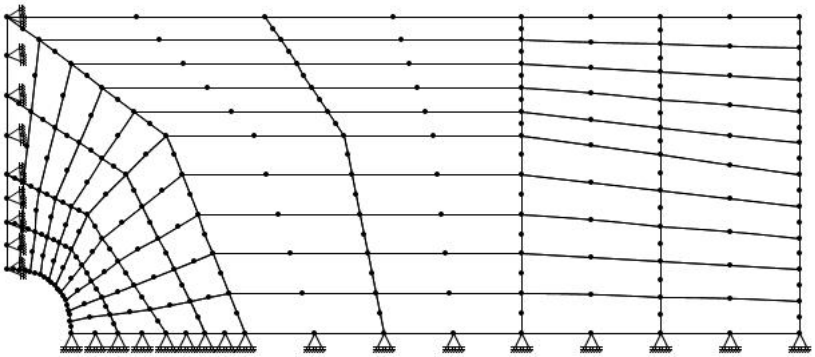
(a) 11 elementos *Q8*



(b) 25 elementos *Q8*



(c) 40 elementos *Q8*



(d) 80 elementos *Q8*

Figura 6.39: Discretizações para a chapa com furo central com elementos *Q8*

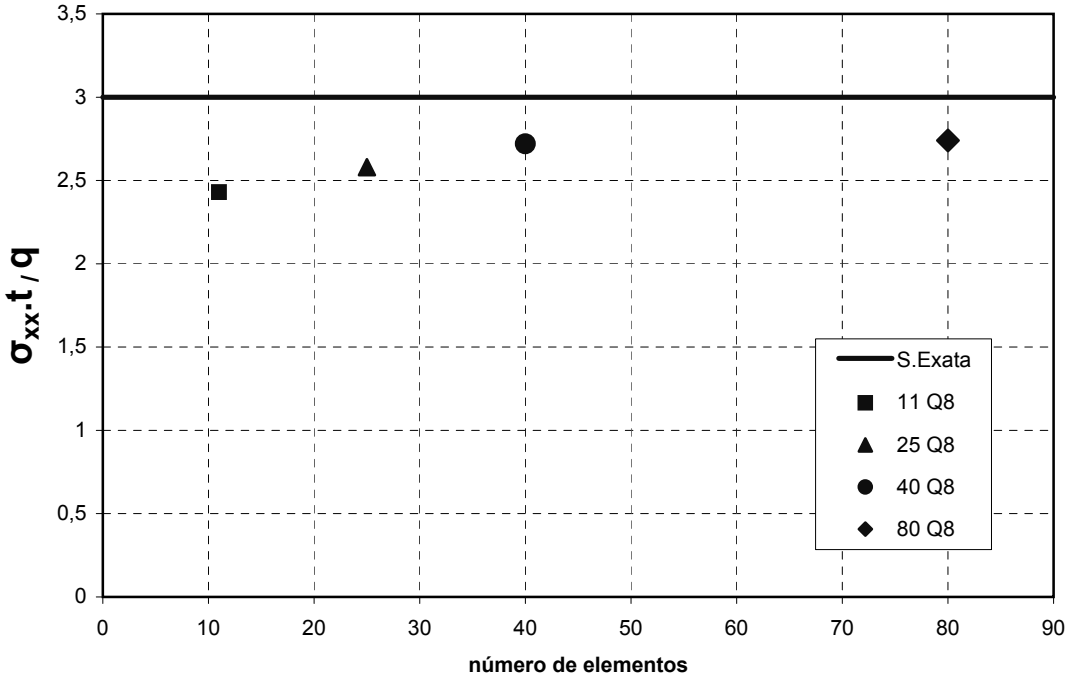


Figura 6.40: Valores da tensão σ_{xx} no ponto A para as malhas da figura 6.39

Observando o gráfico da figura 6.40 percebe-se que ocorre convergência da tensão σ_{xx} para a solução exata ao se refinar as malhas de elementos quadrilaterais de oito nós.

6.7 Cunha

O exemplo a seguir tem por objetivo obter as tensões radiais σ_r atuantes em uma cunha de espessura unitária submetida à força concentrada P no seu vértice.

As configurações geométrica e de cargas da cunha estão mostradas na figura 6.41, sendo $L = 3\ m$, $P = 20\ kN$, módulo de elasticidade $E = 1\ kN/m^2$ e coeficiente de Poisson ν = 0,3. A solução exata para as tensões radiais σ_r deste problema é fornecida pela teoria da elasticidade em (Timoshenko & Goodier 1980), sendo mostrada na equação 6.7.1. Neste exemplo utiliza-se o modelo de análise *PlaneStrainAnalysisM* e o elemento triangular *T3*.

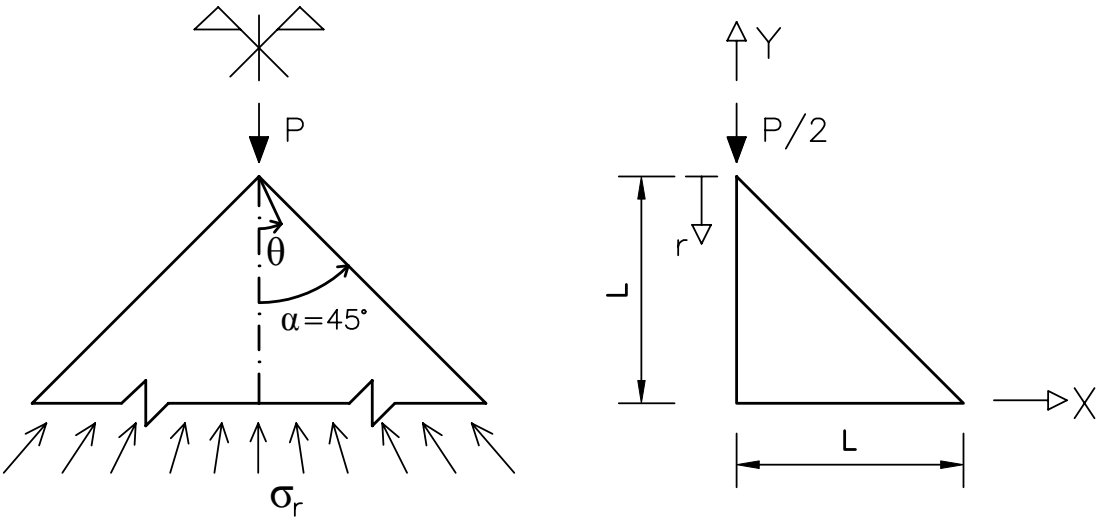
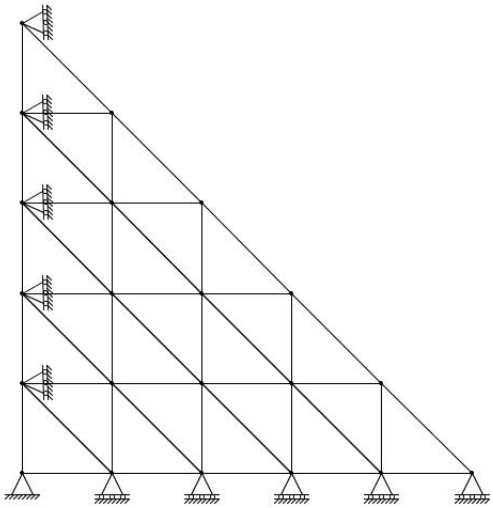


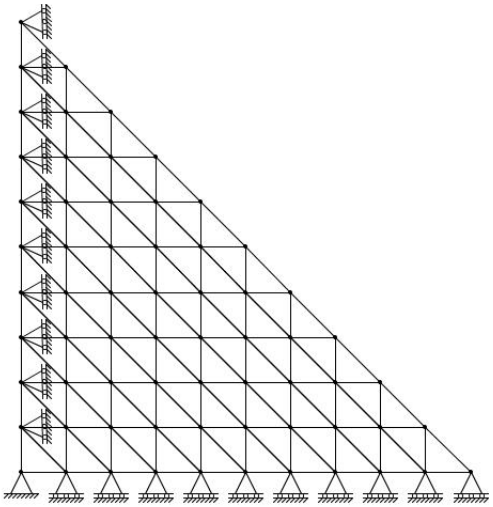
Figura 6.41: Cunha submetida a força concentrada

$$\sigma_r = -\frac{P\cos\theta}{r(\alpha + \frac{1}{2}sen2\alpha)} \tag{6.7.1}$$

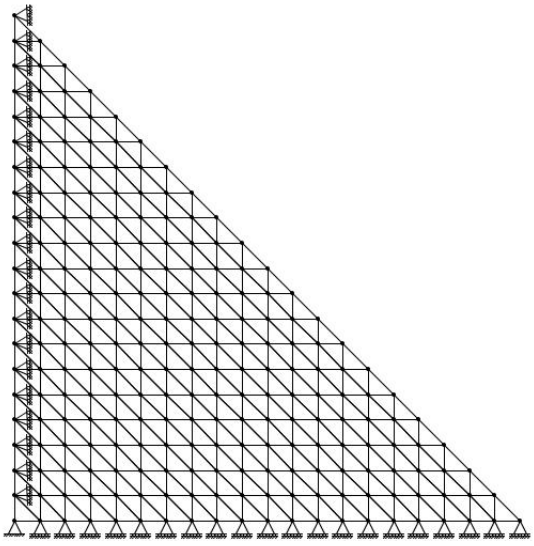
Tirando-se proveito da simetria do problema, são utilizadas as malhas com 25, 100 e 400 elementos *T3* mostradas na figura 6.42. Como todas as discretizações deste exemplo usam o elemento *T3*, é utilizado apenas um ponto de Gauss para obter a matriz de rigidez de cada elemento finito. Os valores da tensão σ_r obtidos nos pontos de Gauss já citados, próximos de θ = 0,0, estão representados no gráfico da figura 6.43, juntamente com a solução exata.



(a) 25 *T3*



(b) 100 *T3*



(c) 400 *T3*

Figura 6.42: Discretizações com elementos *T3* para a cunha

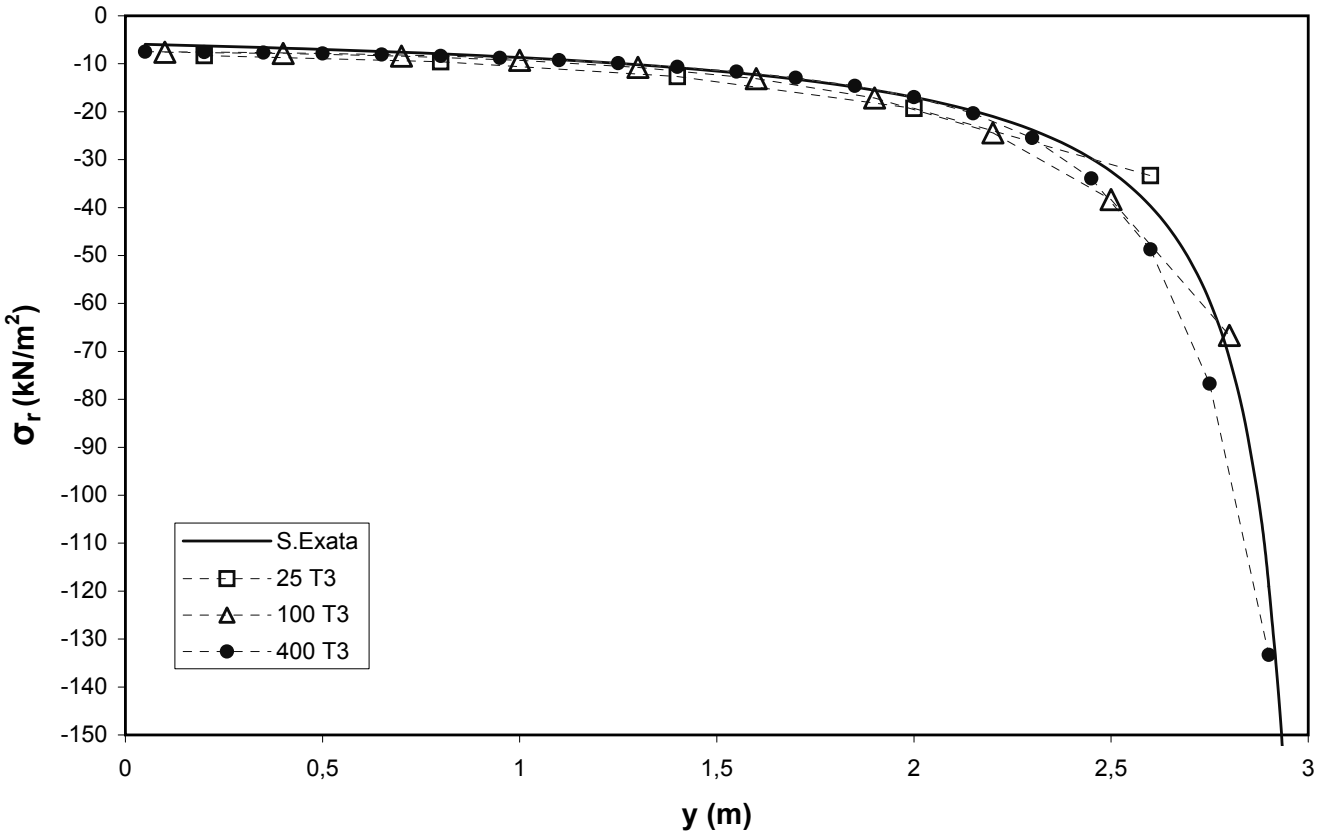


Figura 6.43: Tensões σ_r para a cunha da figura 6.41

Observando o gráfico da figura 6.43 conclui-se que ocorre convergência da tensão σ_r para a solução exata ao se refinar a malha de elementos triangulares de três nós.

6.8 Barragem

Neste exemplo serão mostrados alguns resultados da modelagem de uma barragem, obtida na referência (Soriano & Lima 1999), e reproduzida na figura 6.44 abaixo. Considera-se carregamento hidrostático $q(y) = 180 - 10y$ com valor máximo de $180\,kN/m^2$. O material tem módulo de elasticidade $E = 20800 \times 10^3\ kN/m^2$, coeficiente de Poisson $\nu = 0,2$ e espessura unitária. Utiliza-se o modelo de análise *PlaneStrainAnalysisM*, carregamento variável do tipo *LineElementForce* e os elementos triangulares de seis nós (*T6*) e quadrilaterais de oito nós (*Q8*).

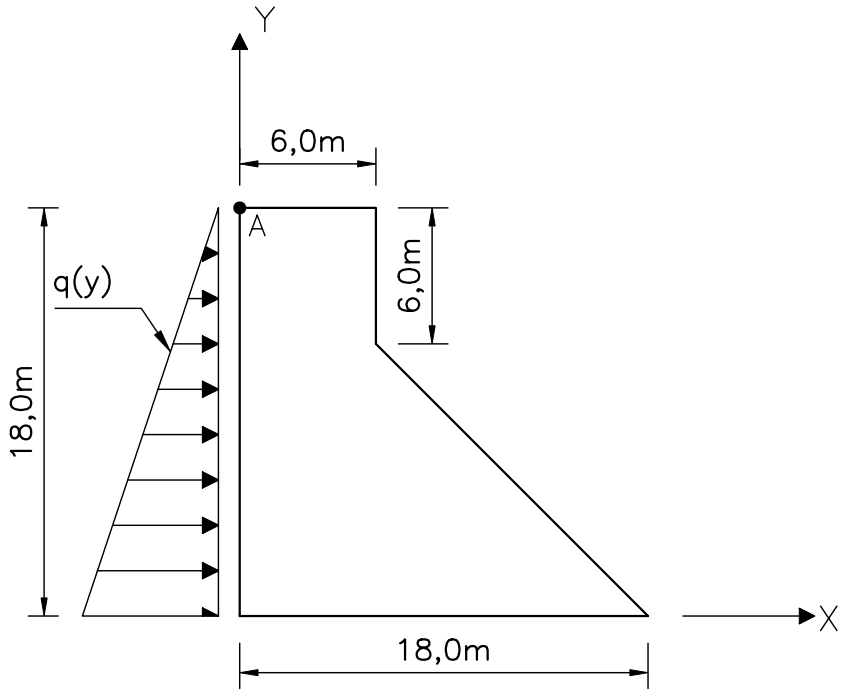
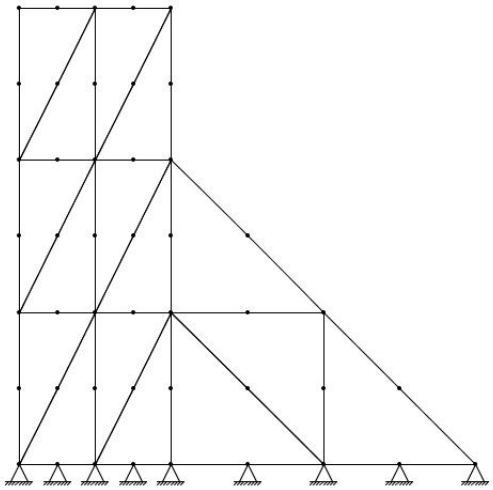
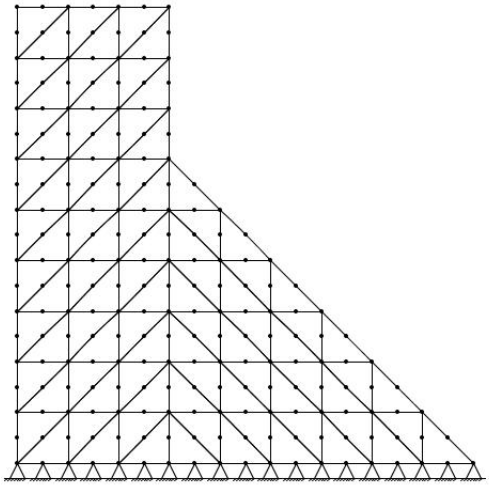


Figura 6.44: Seção transversal da barragem proposta

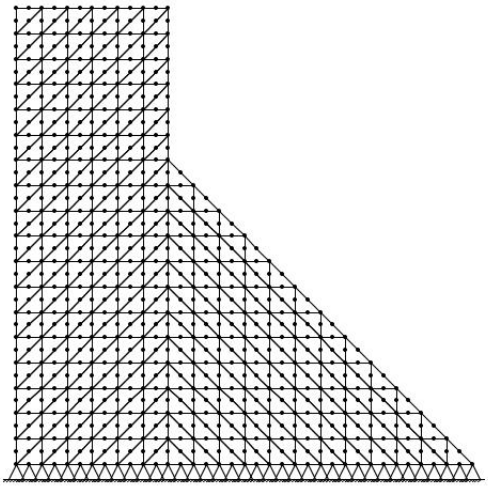
A seção da barragem é discretizada com quatro malhas. As três primeiras são compostas de elementos triangulares e estão mostradas nas figuras 6.45(a), 6.45(b) e 6.45(c). A outra malha contém 108 elementos *Q8*, discretizando a parte retangular, juntamente com 144 elementos *T6*, discretizando a parte triangular da seção da barragem, conforme a figura 6.45(d). No cálculo da matriz de rigidez de cada elemento *T6* são utilizados três pontos de integração e para os elementos *Q8* foram utilizados 3×3 pontos, onde as tensões são obtidas.



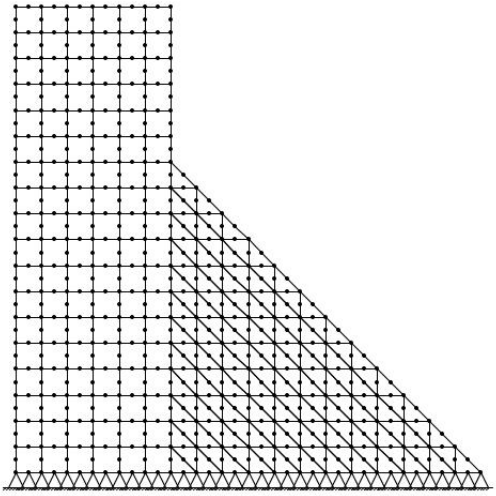
(a) 16 elementos *T6*



(b) 90 elementos *T6*



(c) 360 elementos *T6*



(d) 108 elementos *Q8* e 144 elementos *T6*

Figura 6.45: Discretizações da seção da barragem

Os deslocamentos horizontal u_A e vertical v_A do ponto nodal A da figura 6.44 calculados pelo programa são apresentados na tabela 6.10. No gráfico da figura 6.46 estão representados os valores das tensões σ_{yy} nos pontos de Gauss localizados na reta horizontal Y = 12,0 m.

Tabela 6.10: Deslocamentos do ponto A da figura 6.44

Malha	$u_A \times 10^{-4}$ $\boldsymbol{m}$	$v_A \times 10^{-4}$ $\boldsymbol{m}$
16 *T6*	2,993	0,992
90 *T6*	3,041	1,009
360 *T6*	3,050	1,012
108 *Q8* e 144 *T6*	3,050	1,012

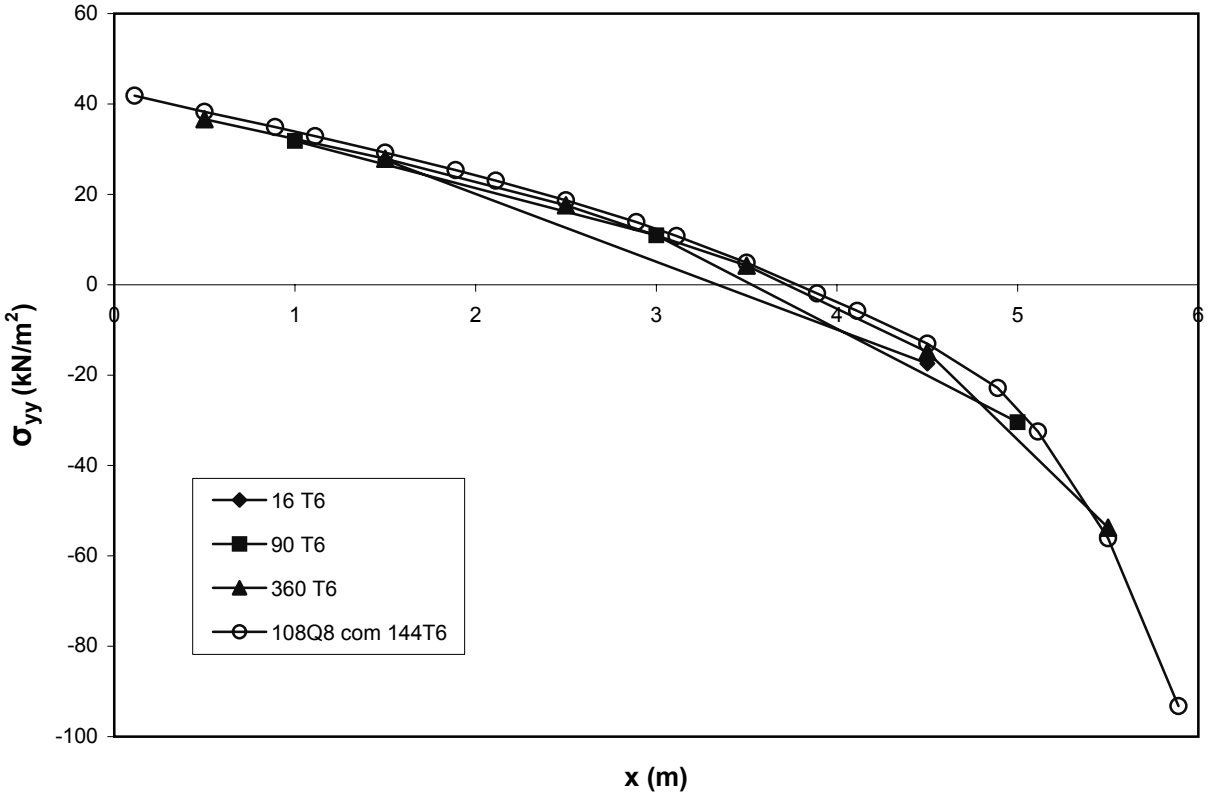


Figura 6.46: Variação das tensões σ_{yy} na seção Y = 12 m

Neste exemplo verifica-se que os deslocamentos (horizontais e verticais) e tensões σ_{yy} convergem ao refinar as malhas de elementos triangulares de seis nós; e ao utilizar uma combinação de elementos triangulares de seis nós e elementos quadrilaterais de oito nós também ocorre convergência para a solução exata.

6.9 Fundação

Neste exemplo serão apresentados os resultados obtidos pelo programa com o uso de integração reduzida. Para tal é considerado o problema obtido em (Zienkiewicz et al. 1986) que se refere a uma estrutura rígida apoiada sobre o solo. A estrutura com módulo de elasticidade $E = 10^6\ uf/uc^2$ e coeficiente de Poisson $\nu = 0,3$ é submetida a uma carga concentrada de compressão $P = 50\ uf$ conforme a discretização mostrada na figura 6.47. Para o solo considera-se módulo de elasticidade $E = 100\ uf/uc^2$ e coeficiente de Poisson $\nu = 0,3$. Ao modelar o problema, considera-se espessura unitária tanto para o solo quanto para a estrutura, utilizando o modelo de análise plana *PlaneStrainAnalysisM* e o elemento quadrilateral *Q8*. Neste modelo cada elemento quadrilateral da figura 6.47 possui largura de 2 *uc* e altura de 1 *uc*.

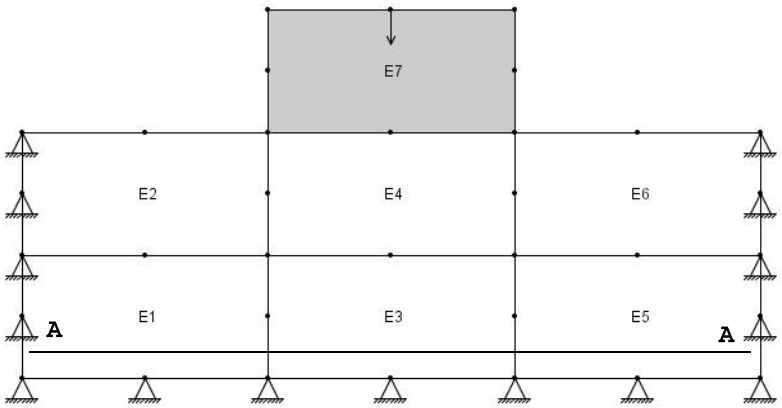
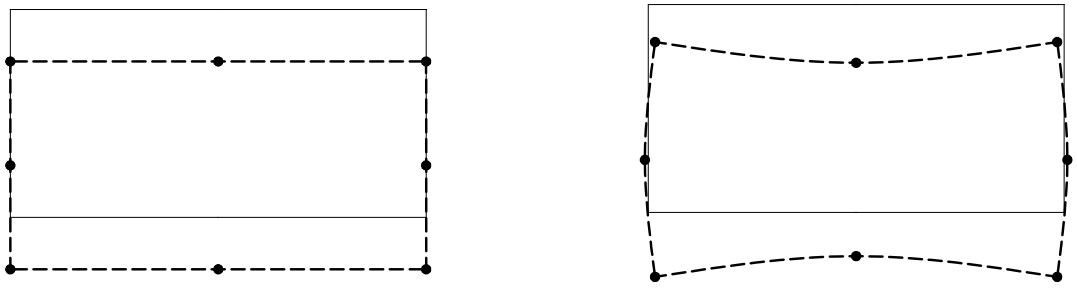


Figura 6.47: Discretização do problema com elementos *Q8*

Na figura 6.48(a) é mostrada a configuração deformada do elemento E7, usado para modelar a estrutura, quando foram utilizados 3×3 pontos de integração e na figura 6.48(b) pode-se perceber a esperada indução de modo espúrio de energia, quando utiliza-se a integração reduzida com 2×2 pontos de Gauss.

Para a malha com 9 pontos de integração por elemento, as tensões normais verticais nos pontos de integração ao longo da linha AA (figura 6.47) estão mostrados na figura 6.49.



(a) 9 pontos de Gauss (b) 4 pontos de Gauss

Figura 6.48: Configurações deformadas do elemento E7

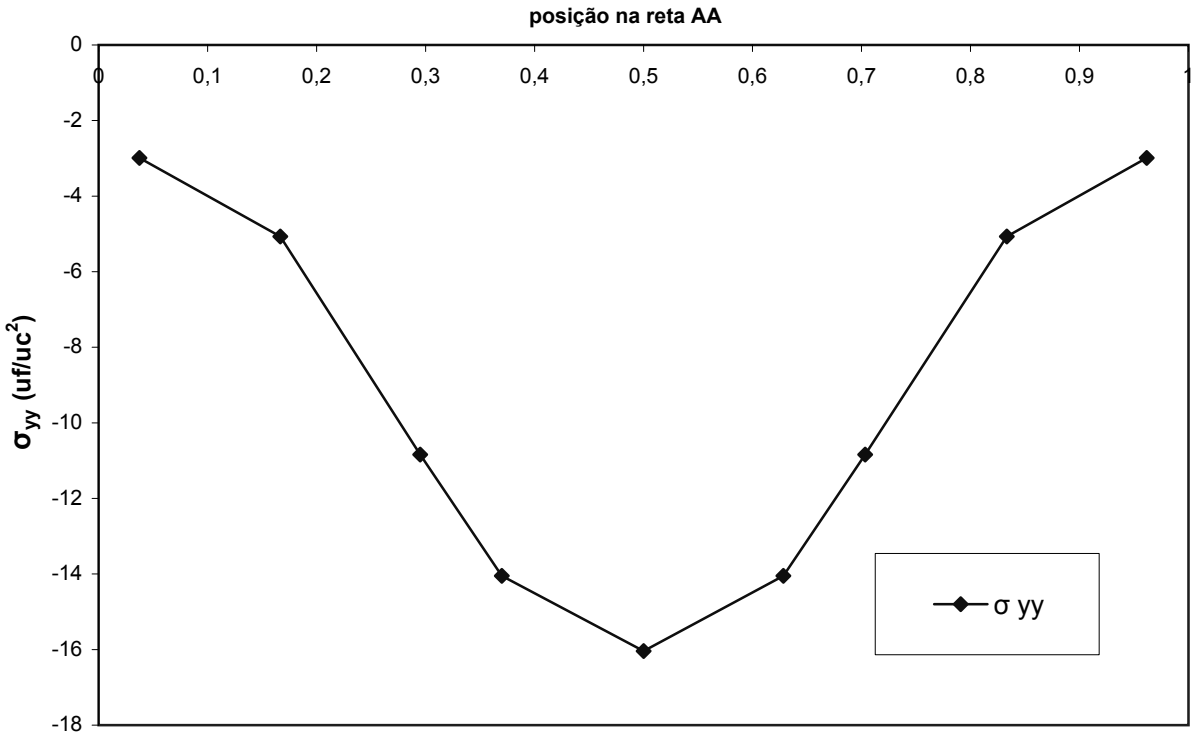


Figura 6.49: Variação da tensão σ_{yy} ao longo da linha AA da figura 6.47

Observando os resultados obtidos neste exemplo percebe-se a esperada indução do modo espúrio de energia ao utilizar-se integração reduzida.

6.10 Disco Axissimétrico

Nesta seção serão apresentados os resultados das tensões obtidas pelo programa na discretização do disco axissimétrico encontrado na referência (Oñate 1995) e reproduzido na figura 6.50 abaixo. Considera-se raio interno $r_i = 10\,uc$, raio externo $r_e = 20\,uc$, altura $h = 1\ uc$, $q = 20\ uf/uc$, $E = 1000\ uf/uc^2$ e $\nu = 0,3$. Tendo em vista que este problema é axissimétrico utiliza-se modelo de análise *AxiSymetricAnalysisM* e o elemento *AxiT3*. O carregamento é considerado do tipo *LineElementForce*.

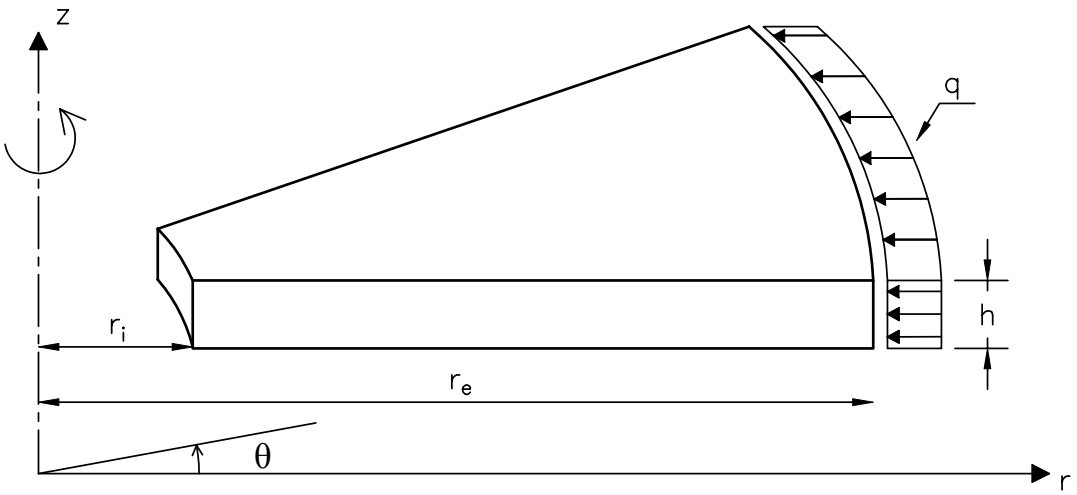


Figura 6.50: Disco axissimétrico

A solução exata para as tensões σ_r e σ_θ deste problema pode ser obtida, segundo a teoria da elasticidade (Timoshenko & Goodier 1980), através das equações

$$\sigma_r = \frac{A}{r^2} + 2\,C \tag{6.10.1}$$

$$\sigma_\theta = -\frac{A}{r^2} + 2\,C \tag{6.10.2}$$

sendo que

$$A = \frac{q\,r_e^2\,r_i^2}{r_e^2 - r_i^2} \quad e \quad C = \frac{-q\,r_e^2}{2(r_e^2 - r_i^2)} \tag{6.10.3}$$

O disco axissimétrico é discretizado com a malha de 20 elementos *AxiT3* mostrada na figura 6.51. Os valores das tensões σ_r, σ_θ e σ_z obtidos nos pontos de Gauss são mostrados

no gráfico da figura 6.52, juntamente com suas respectivas soluções exatas.

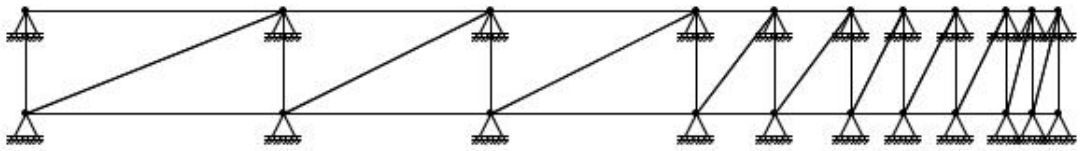


Figura 6.51: Malha utilizada para discretizar o disco axissimétrico

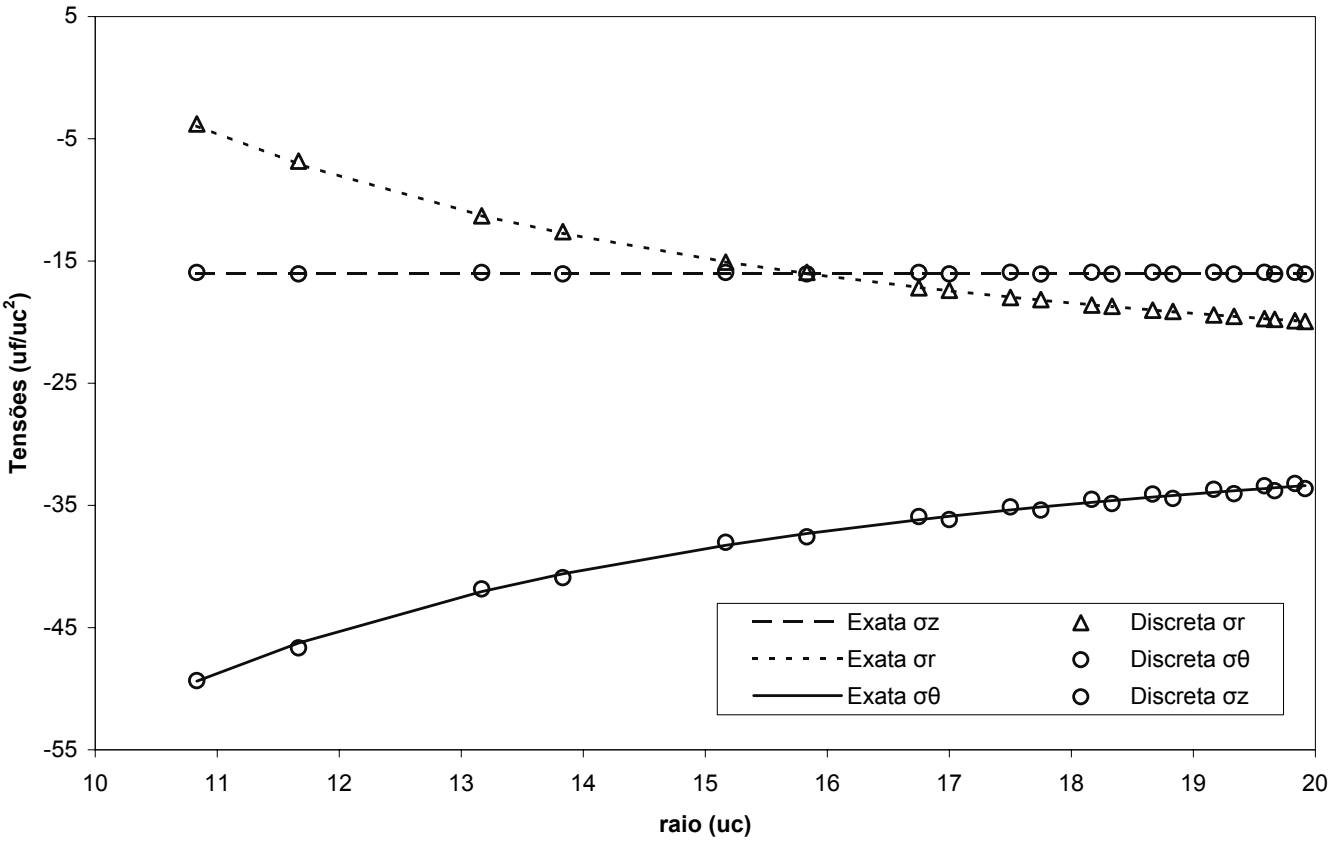


Figura 6.52: Resultados das tensões nos pontos de Gauss do disco axissimétrico

Observando o gráfico da figura 6.52 percebe-se que os resultados das tensões σ_z, σ_r e σ_θ obtidos neste exemplo coincidem com a solução exata.

6.11 Tubo Axissimétrico

Este exemplo mostra os valores das tensões σ_θ obtidas ao discretizar um tubo submetido à pressão interna P_i. O tubo em questão é mostrado na figura 6.53 e foi proposto por (Weaver & Johnston 1984), da qual serão utilizados os mesmos dados. Este problema é modelado como axissimétrico com eixo de simetria z conforme a figura 6.53.

Discretiza-se apenas a porção do tubo correspondente à fatia L da figura 6.53, considerando condições de contorno que garantam deslocamento nulo da porção L paralelamente ao eixo z. É considerada pressão interna $P_i = 1\ uf/uc^2$, comprimento $L = 1\ uc$, raio externo $r_e = 11\ uc$, raio interno $r_i = 10\ uc$, módulo de elasticidade $E = 1 \times 10^4\ uf/uc^2$ e coeficiente de Poisson $\nu = 0{,}3$. Utiliza-se também o modelo de análise *AxiSymetricAnalysisM* e o carregamento do tipo *LineElementForce* para representar a pressão interna P_i.

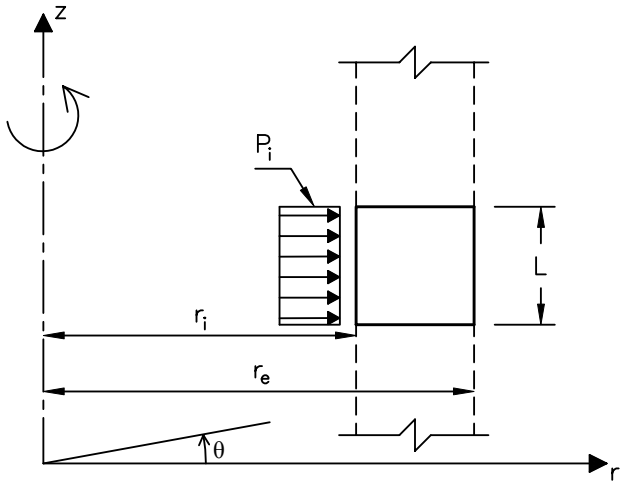


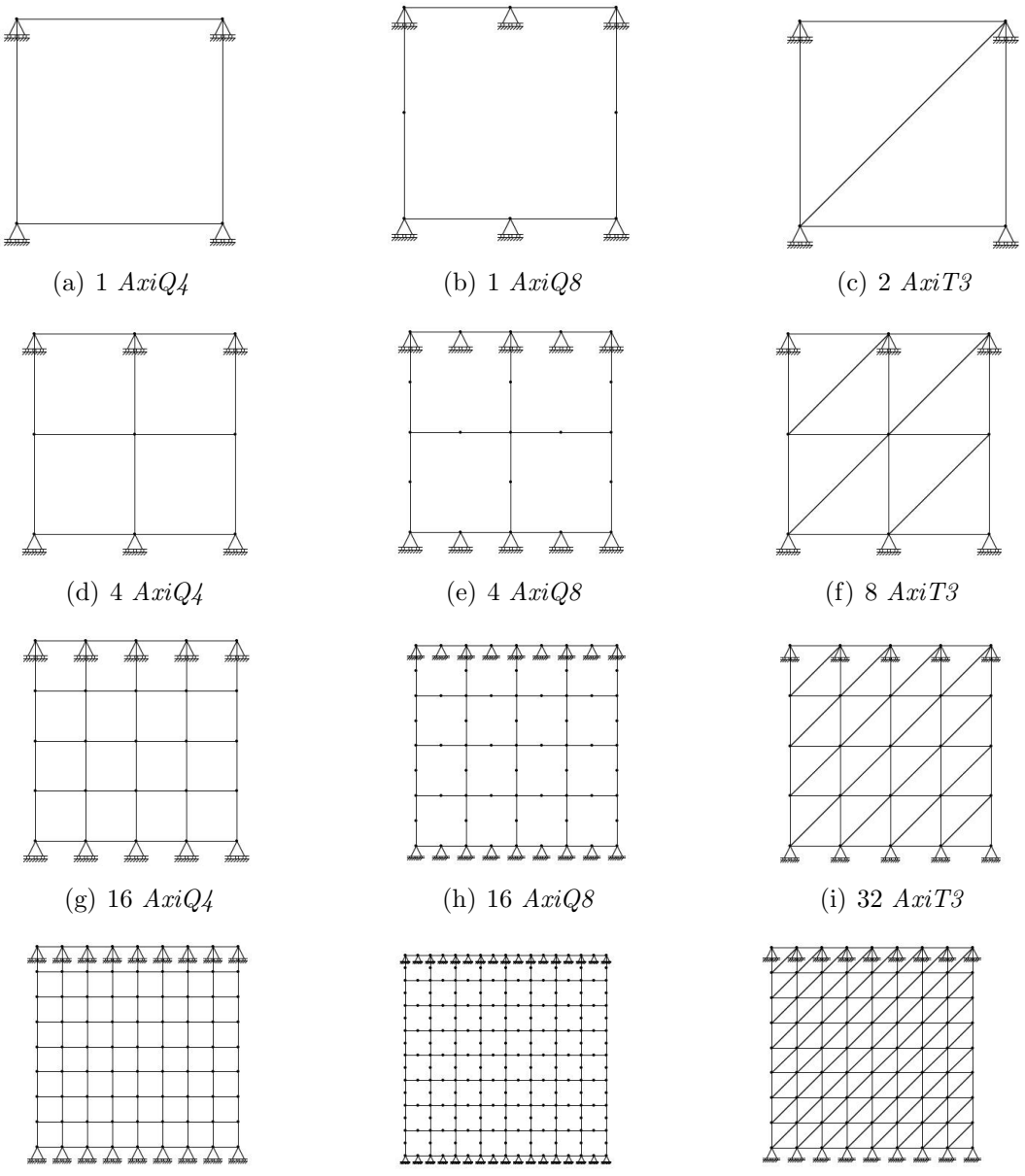
Figura 6.53: Tubo submetido à pressão interna

A solução exata para este problema pode ser encontrada em (Timoshenko & Goodier 1980) sendo dada por

$$\sigma_\theta = \frac{P_i\,(r_i^2\,r_e^2)}{r^2\,(r_e^2 - r_i^2)} + \frac{P_i\,r_i^2}{r_e^2 - r_i^2} \tag{6.11.1}$$

Com o propósito de demonstrar a convergência das tensões σ_θ para o valor exato são processadas várias malhas de elementos quadrilaterais *AxiQ4* e *AxiQ8* e de elementos triangulares *AxiT3*. A ordem de integração utilizada para os elementos *AxiQ4*, *AxiQ8*

e *AxiT3* é 2 × 2, 3 × 3 e 1 × 1 pontos, respectivamente, em cada direção. Em todas as malhas de elementos triangulares e quadrilaterais, cada discretização subseqüente é formada da divisão de cada elemento da malha anterior em outros quatro elementos de tamanhos iguais. Assim são obtidas as malhas mostradas na figura 6.54.



(a) 1 *AxiQ4* (b) 1 *AxiQ8* (c) 2 *AxiT3*

(d) 4 *AxiQ4* (e) 4 *AxiQ8* (f) 8 *AxiT3*

(g) 16 *AxiQ4* (h) 16 *AxiQ8* (i) 32 *AxiT3*

(j) 64 *AxiQ4* (k) 64 *AxiQ8* (l) 128 *AxiT3*

Figura 6.54: Malhas usadas na discretização do tubo

Apresenta-se a seguir os valores obtidos nas discretizações discutidas acima para a tensão σ_θ nos pontos de Gauss mais próximos de $r = 10$ *uc*, ou seja, mais próximos da parede interna do tubo. A tabela 6.11 apresenta os resultados obtidos para as referidas tensões e a figura 6.55 mostra graficamente os resultados desta tabela.

Tabela 6.11: Percentual relativo da tensão circunferencial

Malha	**Valor Exato** σ_θ uf/uc^2	**Valor Obtido** σ_θ uf/uc^2	**Erro percentual relativo**
1 AxiQ4	10,288	10,500	2,06
4 AxiQ4	10,410	10,520	1,05
16 AxiQ4	10,467	10,523	0,54
64 AxiQ4	10,489	10,524	0,33
1 AxiQ8	10,399	10,404	0,04
4 AxiQ8	10,455	10,460	0,04
16 AxiQ8	10,489	10,490	0,02
64 AxiQ8	10,508	10,510	0,02
2 AxiT3	10,524	10,807	2,69
8 AxiT3	10,524	10,714	1,81
32 AxiT3	10,524	10,630	1,00
128 AxiT3	10,524	10,580	0,53

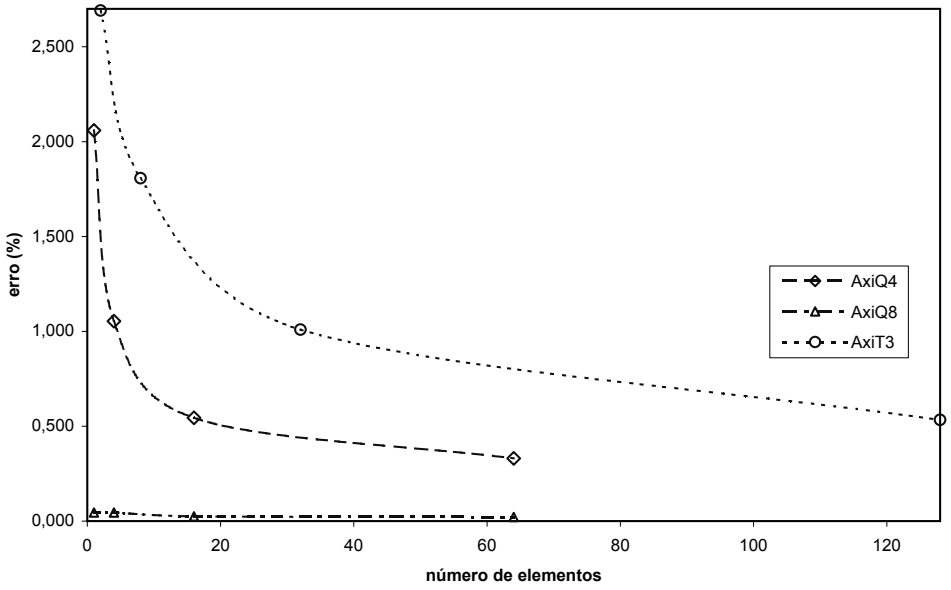


Figura 6.55: Variação do erro percentual relativo com o número de elementos

Observando os resultados obtidos conclui-se que o erro cometido pelo modelo discreto na avaliação da tensão σ_θ diminui ao se refinar a malha.

6.12 Problema de Boussinesq

Nesta seção o programa implementado é utilizado para obter a solução discreta de um problema clássico da elasticidade, conhecido como problema de Boussinesq (Timoshenko & Goodier 1980), que descreve o comportamento de um sólido semi-infinito submetido a uma carga concentrada.

Considera-se que a influência da carga concentrada a uma distância de 4 uc é pouco relevante, desde que sejam aplicadas restrições adequadas no contorno do sólido. Logo, ao discretizar o problema, são consideradas as condições de contorno mostradas na figura 6.56, carga concentrada $P = 4 \times 10^4$ *uf*, módulo de elasticidade $E = 1000\ uf/uc^2$ e coeficiente de Poisson $\nu = 0{,}2$. Utiliza-se o modelo de análise *AxiSymetricAnalysisM* e elementos *AxiQ8* na discretização.

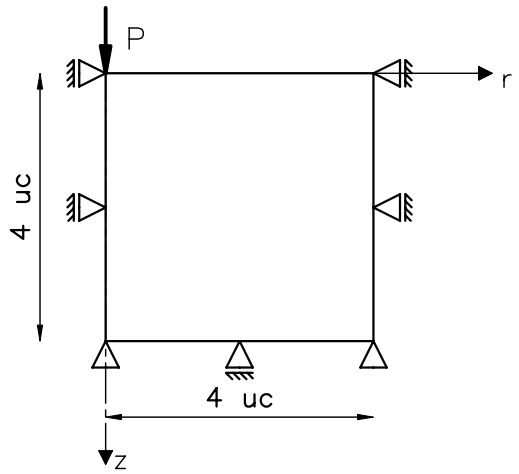


Figura 6.56: Modelo para o problema de Boussinesq

Na figura 6.57 é mostrada a discretização gerada com elementos *AxiQ8*. Com o objetivo de validar os resultados dos deslocamentos obtidos no programa implementado, este mesmo exemplo é processado no programa de elementos finitos *Ansys* (Ansys-INC 2004), utilizando elementos quadrilaterais de oito nós e modelo de análise axissimétrico.

Os resultados obtidos para os deslocamentos u e w, nas direções radial e vertical, respectivamente, ao longo da reta z = 2,0 uc estão plotados nos gráficos das figuras 6.58 e 6.59. São mostrados nestes gráficos os deslocamentos obtidos no programa implementado juntamente com os deslocamentos calculados pelo *Ansys*. O gráfico da figura 6.60

apresenta a variação da tensão σ_z, calculada pelo programa implementado, nos pontos de Gauss localizados ao longo da reta z = 0,6 uc, juntamente com a solução exata obtida em (Timoshenko & Goodier 1980).

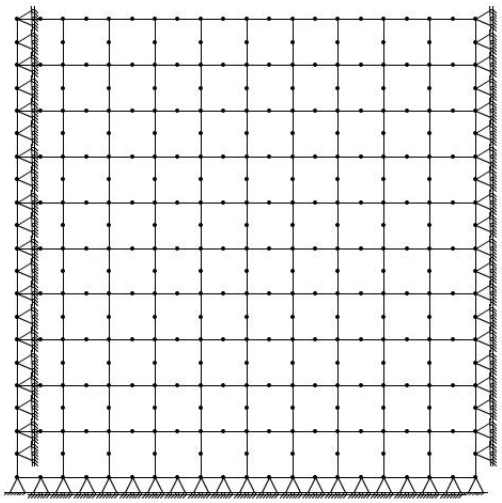


Figura 6.57: Discretização do problema de Boussinesq

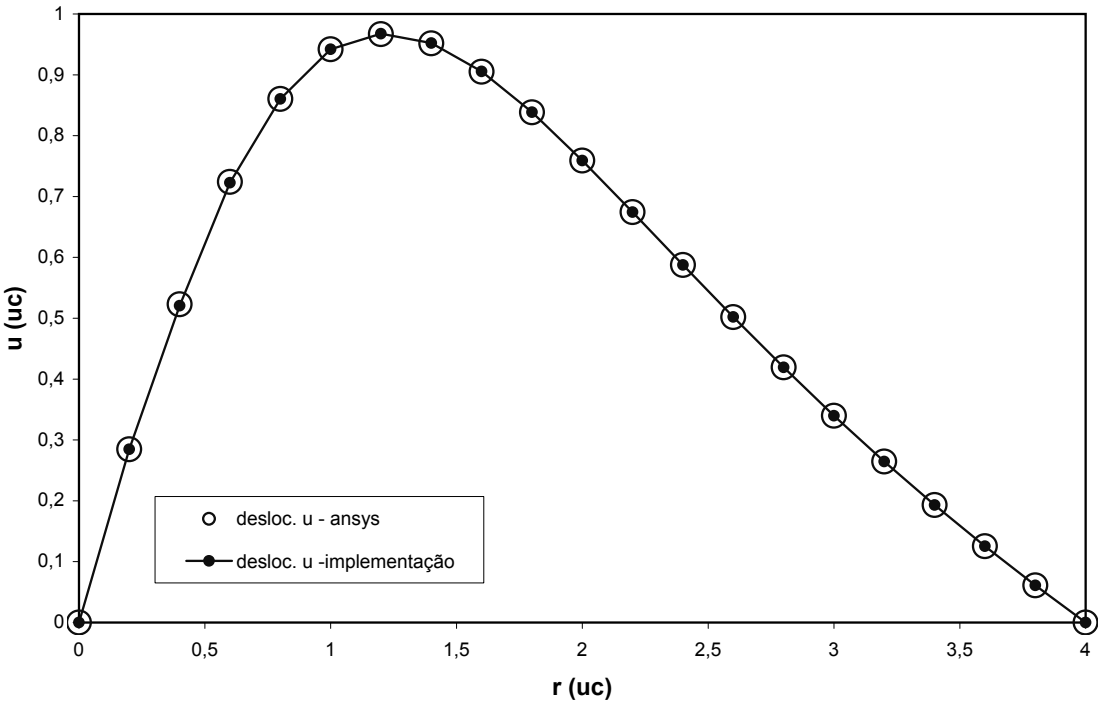


Figura 6.58: Deslocamentos horizontais ao longo da reta z = 2,0 uc

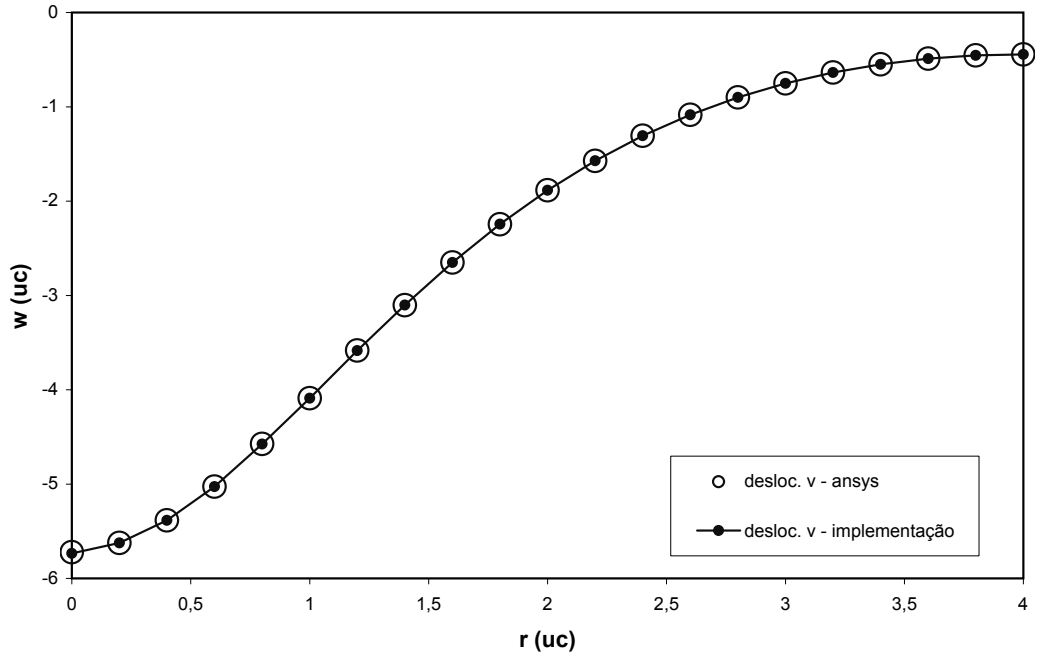


Figura 6.59: Deslocamentos verticais ao longo da reta z = 2,0 uc

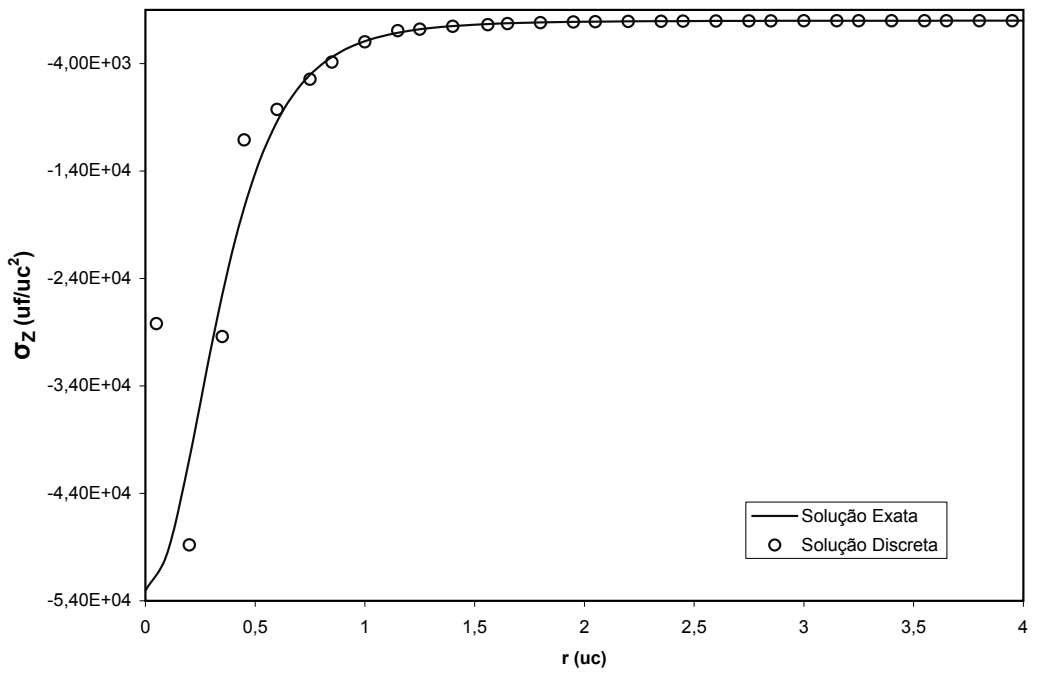


Figura 6.60: Tensões σ_z ao longo da reta z = 0,6 uc

Neste exemplo verifica-se a validade dos deslocamentos horizontais e verticais do modelo, a partir da comparação com os resultados fornecidos pelo program Ansys (Ansys-INC 2004). As tensões σ_z calculadas nos pontos de Gauss do modelo também coincidem com a solução exata, exceto no ponto de Gauss mais próximo da projeção do ponto de aplicação da carga, onde o valor da tensão σ_z tende para o infinito. Por isso o valor obtido pelo modelo não coincide com a solução exata.

6.13 Barra Prismática

Neste exemplo são apresentados os resultados obtidos pelo programa para o modelo de uma barra prismática, submetida a diferentes carregamentos. Tais carregamentos são aplicados separadamente na barra, sendo os resultados de cada um apresentados em diferentes sub-ítens desta seção.

A barra está mostrada na figura 6.61, sendo engastada em todo o plano x = 120 *uc*, com dimensões $L = 120$ *uc*, $b = h = 12$ *uc*, módulo de elasticidade $E = 10\ uf/uc^2$ e coeficiente de Poisson $\nu = 0{,}3$.

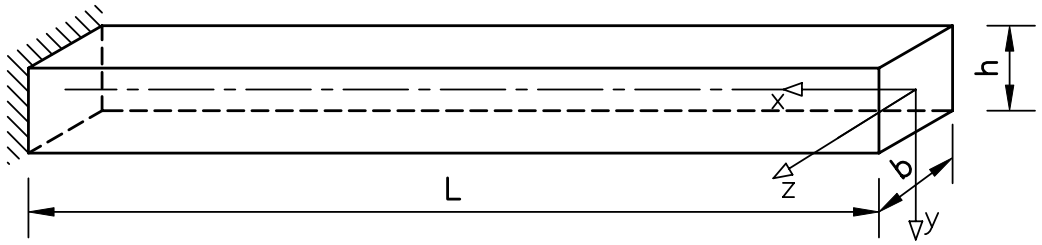


Figura 6.61: Barra tridimensional analisada

O INSANE (ver seção 1.1) ainda não possui um gerador automático de malhas tridimensionais. Portanto as informações referentes a todas as malhas tridimensionais, tais como coordenadas, restrições, carregamentos nodais, incidência dos elementos etc são apropriadamente editadas obedecendo a mesma estrutura de documentos *xml* utilizada nas malhas bidimensionais (ver apêndice A).

Para modelar a barra da figura 6.61 utiliza-se a discretização mostrada na figura 6.62, sendo considerado o modelo de análise tridimensional *SolidAnalysisM* e quatro elementos hexaédricos *H20* com quadratura de Gauss de $3 \times 3 \times 3$ pontos em cada elemento hexaédrico. O carregamento pode ser do tipo *LineElementForce*, *SurfaceElementForce* ou *VolumeElementForce*, dependendo de cada um dos casos de carregamento discutidos a seguir.

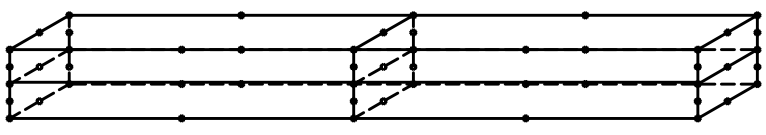


Figura 6.62: Discretização da barra tridimensional com 4 elementos *H20*

6.13.1 Carga vertical na Extremidade Livre

Aplica-se uma carga de cisalhamento uniformemente distribuída no plano correspondente a extremidade livre da barra tridimensional, ou seja, no plano $x = 0$. A carga tem módulo P $= 8,33 \times 10^{-2}\, uf/uc^2$ (Resultante $= 12\ uf$), sendo necessário utilizar o carregamento do tipo *SurfaceElementForce* para representá-la. Na figura 6.63 pode ser vista a representação da carga de cisalhamento na barra.

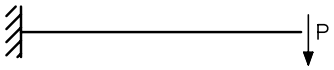


Figura 6.63: Carga de cisalhamento

A solução exata para os deslocamentos verticais v do eixo da barra deste problema pode ser obtida em (Timoshenko & Goodier 1980), sendo dado por

$$v(x) = \frac{Px^3}{6EI} - \frac{PL^2\,x}{2EI} + \frac{PL^3}{3EI} \tag{6.13.1}$$

O gráfico da figura 6.64 apresenta os deslocamentos verticais v obtidos pelo programa implementado juntamente com a solução exata.

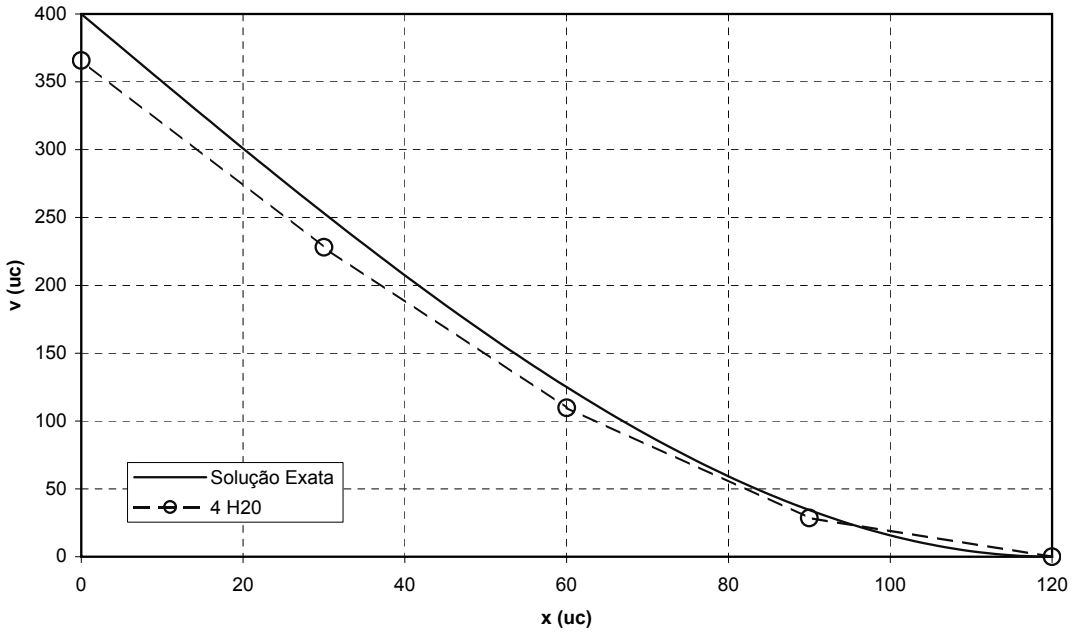


Figura 6.64: Deslocamentos verticais para carga de cisalhamento P

6.13.2 Momento Fletor na Extremidade Livre

Aplica-se um binário de forças concentradas $P = 1\ uf$ em dois nós do plano $x = 0$ para representar um momento fletor. A figura 6.65 mostra este momento fletor cujo módulo é $M = 12$ P $uf \cdot uc$.

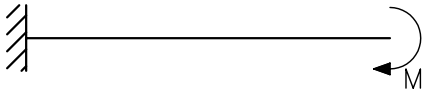


Figura 6.65: Momento fletor atuante

A solução exata deste problema é obtida analiticamente utilizando o modelo matemático unidimensional, sendo dada por

$$v(x) = \frac{Mx^2}{2EI} - \frac{MLx}{EI} + \frac{ML^2}{2EI} \tag{6.13.2}$$

Os deslocamentos verticais v do eixo da barra obtidos pelo programa implementado são mostrados juntamente com a solução exata no gráfico da figura 6.66.

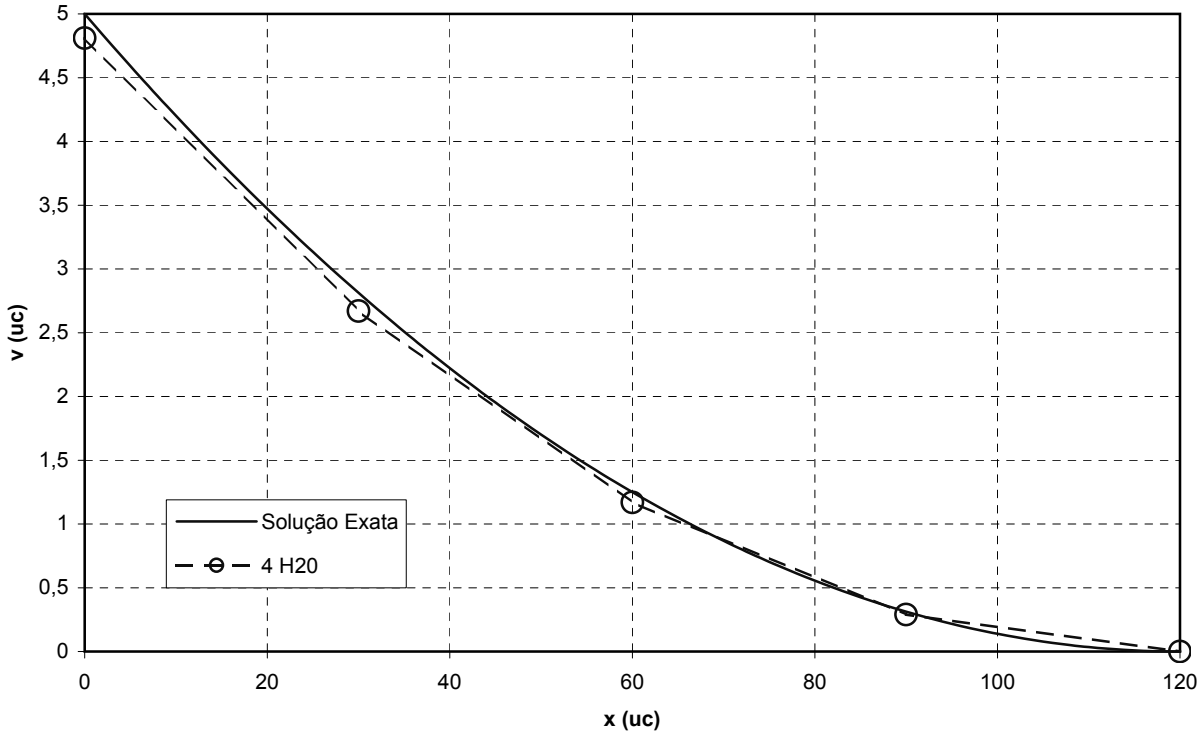


Figura 6.66: Deslocamentos verticais devidos ao momento M

6.13.3 Carga Distribuída Constante

Para representar uma carga distribuída constante atuando na face superior da barra tridimensional da figura 6.61 utiliza-se o carregamento do tipo *SurfaceElementForce*. Tal carregamento é aplicado na direção y em toda a área da face superior da barra, ou seja, no plano $y = -6\,uc$. A carga tem módulo $Q = 0,01\,uf/uc^2$. Sendo $b = 12\,uc$ a largura da barra obtêm-se $q = Q \times b = 0,01 \times 12 = 0,12\,uf/uc$. Na figura 6.67 pode ser vista uma representação esquemática da carga distribuída na barra.

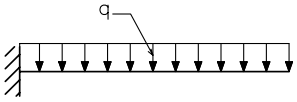


Figura 6.67: Carga distribuída constante q

A solução exata deste problema é obtida analiticamente utilizando o modelo matemático unidimensional, sendo dada por

$$v(x) = \frac{qx^4}{24EI} - \frac{qL^3x}{6EI} + \frac{qL^4}{8EI} \tag{6.13.3}$$

Os deslocamentos verticais v do eixo da barra obtidos pelo programa implementado são mostrados juntamente com a solução exata no gráfico da figura 6.68.

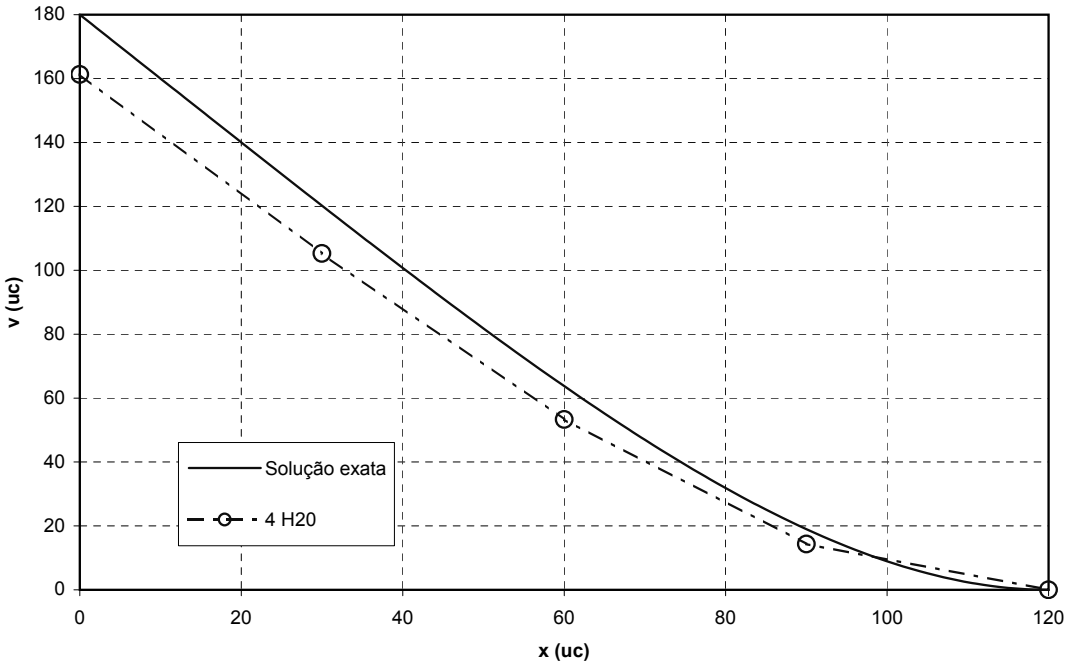


Figura 6.68: Deslocamentos verticais devidos à carga distribuída constante q

6.13.4 Carga Distribuída Variável

Para representar uma carga distribuída variável triangular atuando na face superior da barra tridimensional da figura 6.61 utiliza-se o carregamento do tipo *SurfaceElementForce*. Tal carregamento varia linearmente com x conforme a equação $q(x) = (q_0/L)\,x$, sendo aplicado na direção y em toda a área da face superior da barra, ou seja, no plano $y = -6\,uc$. O valor máximo da carga atuante é $Q = 0,012\,uf/uc^2$. Sendo $b = 12\,uc$ a largura da barra obtém-se $q_0 = Q \times b = 0,012 \times 12 = 0,144\,uf/uc$. Na figura 6.69 pode ser vista uma representação esquemática da carga distribuída na barra.

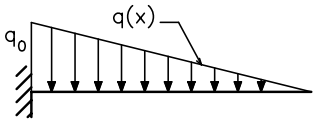


Figura 6.69: Carga distribuída variável *q(x)*

A solução exata deste problema é obtida analiticamente utilizando o modelo matemático unidimensional, sendo dada por

$$v(x) = \frac{q_0 x^5}{120EIL} - \frac{q_0 L^3 x}{24EI} + \frac{q_0 L^4}{30EI} \tag{6.13.4}$$

Os deslocamentos verticais v do eixo da barra obtidos pelo programa implementado são mostrados juntamente com a solução exata no gráfico da figura 6.70.

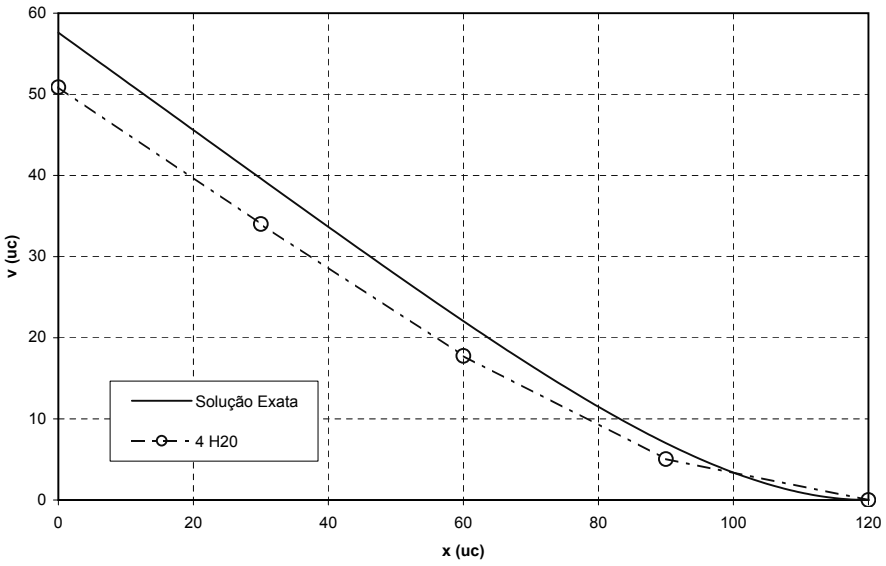


Figura 6.70: Deslocamentos verticais devidos a carga distribuída variável *q(x)*

6.13.5 Peso Próprio

Para representar a carga relativa ao peso próprio da barra da figura 6.61, é necessário utilizar o carregamento do tipo *VolumeElementForce*. O módulo do peso próprio considerado é $\rho g = 0,01\ uf/uc^3$, onde ρ é a densidade do material da barra e g é a aceleração da gravidade agindo ao longo do eixo x da barra. A carga de massa é aplicada em todo o volume da barra.

Na figura 6.71 pode ser vista uma representação esquemática deste carregamento.

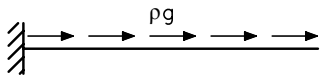


Figura 6.71: Peso próprio ρg

A solução exata para os deslocamentos horizontais u do eixo da barra deste problema pode ser obtida em (Timoshenko & Goodier 1980), sendo dada por

$$u(x) = -\frac{\rho g}{2E}(L^2 - x^2) \tag{6.13.5}$$

O gráfico da figura 6.72 apresenta a solução exata juntamente com os deslocamentos horizontais u obtidos.

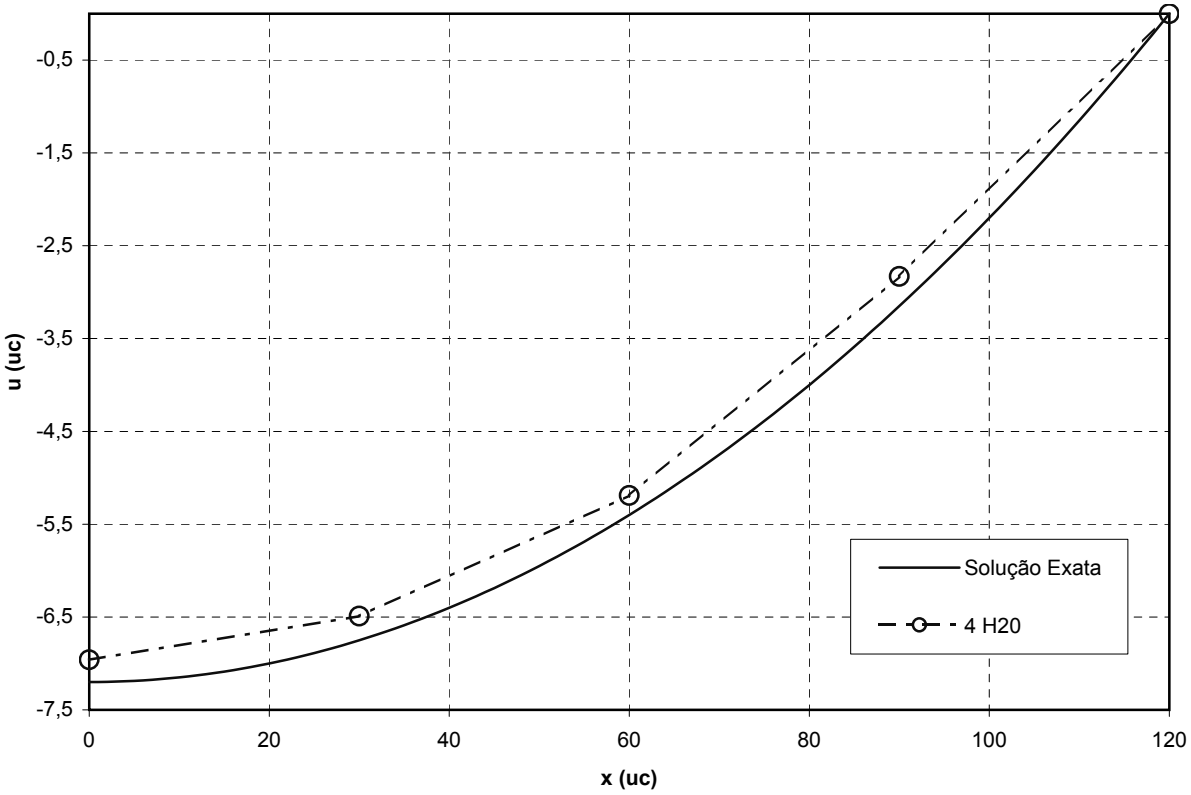


Figura 6.72: Deslocamentos horizontais devidos ao peso próprio ρg

Nos exemplos desta seção, modela-se uma barra prismática submetida a carregamentos de linha, área e volume utilizando os elementos hexaédricos de vinte nós. Em todos os modelos os valores obtidos se aproximam da solução exata.

6.14 Viga Biapoiada

Neste exemplo são apresentados alguns resultados obtidos pelo programa implementado, modelando-se uma viga tridimensional, com a mesma geometria do exemplo anterior, modificando-se apenas as condições de contorno.

Tais condições de contorno são alteradas para representar as restrições de apoio para uma viga tridimensional biapoiada. Ou seja, os nós do plano x = 0 são impedidos de deslocar na direção y e os nós do plano x = 120 *uc* são impedidos de deslocar nas direções x e y. Os outros dados do problema, tais como comprimento longitudinal, base, altura, módulo de elasticidade e coeficiente de Poisson não são alterados. É considerado o caso de um carregamento distribuído uniforme de módulo $Q = 0,01\ uf/uc^2$ atuando em toda a face superior da viga como uma força por unidade de área. Sendo $b = 12\,uc$ a largura da viga obtém-se $q = Q \times b = 0,01 \times 12 = 0,12\,uf/uc$. Neste modelamento é utilizada a mesma malha de elementos hexaédricos *H20* da figura 6.62.

Na figura 6.73 é mostrada a representação esquemática do carregamento na viga biapoiada.

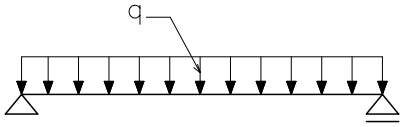


Figura 6.73: Carga distribuída constante q

A solução exata deste problema é obtida analiticamente utilizando o modelo matemático unidimensional, sendo dada por:

$$v(x) = \frac{qx}{24EI}(x^3 - 2Lx^2 + L^3) \tag{6.14.1}$$

Os deslocamentos verticais v do eixo da viga obtidos pelo programa implementado são mostrados, juntamente com a solução da equação 6.14.1, no gráfico da figura 6.74.

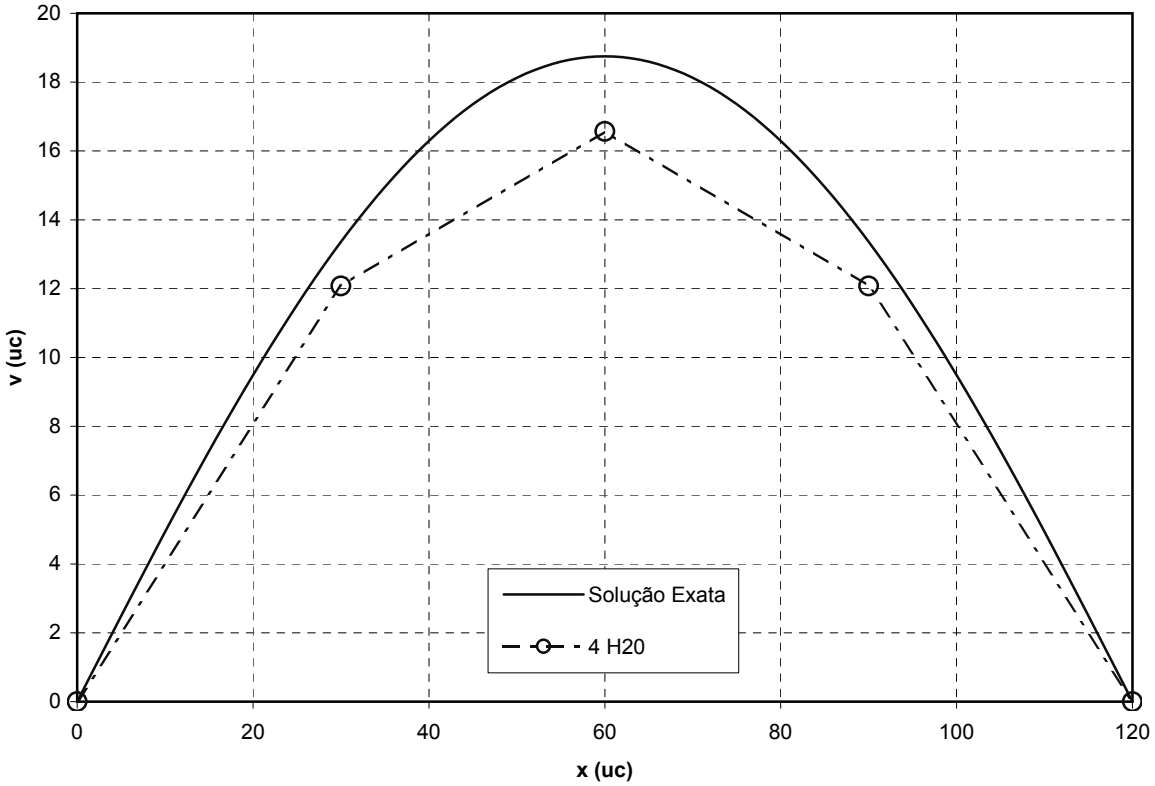


Figura 6.74: Deslocamentos verticais devidos ao carregamento q na viga biapoiada

6.15 Barra Curva

Este exemplo tem o propósito de mostrar alguns resultados obtidos pelo programa implementado ao modelar uma barra de eixo curvo com força cisalhante na extremidade, conforme a figura 6.75. A barra curva proposta é submetida a um carregamento cisalhante $q = 1\ uf/uc$ resultando em $P = 5\ uf$ e possui raio externo $b = 50\ uc$, raio interno $a = 45\ uc$, $h = 5\ uc$, espessura unitária, módulo de elasticidade $E = 1000\ uf/uc^2$ e coeficiente de Poisson $\nu = 0,3$.

A barra em estudo é modelada com os elementos bidimensionais quadrilaterais *Q8* e com os elementos tridimensionais hexaédricos *H20*. No modelamento bidimensional utiliza-se o modelo de análise *PlaneStressAnalysisM*, carregamento do tipo *LineElementForce* e quadratura de Gauss com 3×3 pontos. Para o caso tridimensional utiliza-se o modelo de análise *SolidAnalysisM*, carregamento do tipo *SurfaceElementForce* e quadratura de Gauss com 2×2 pontos.

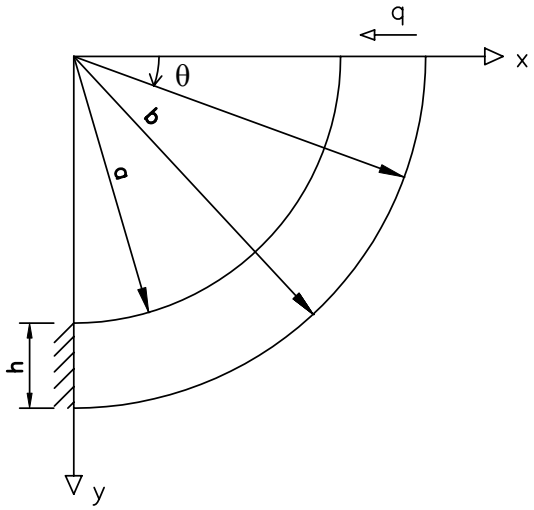


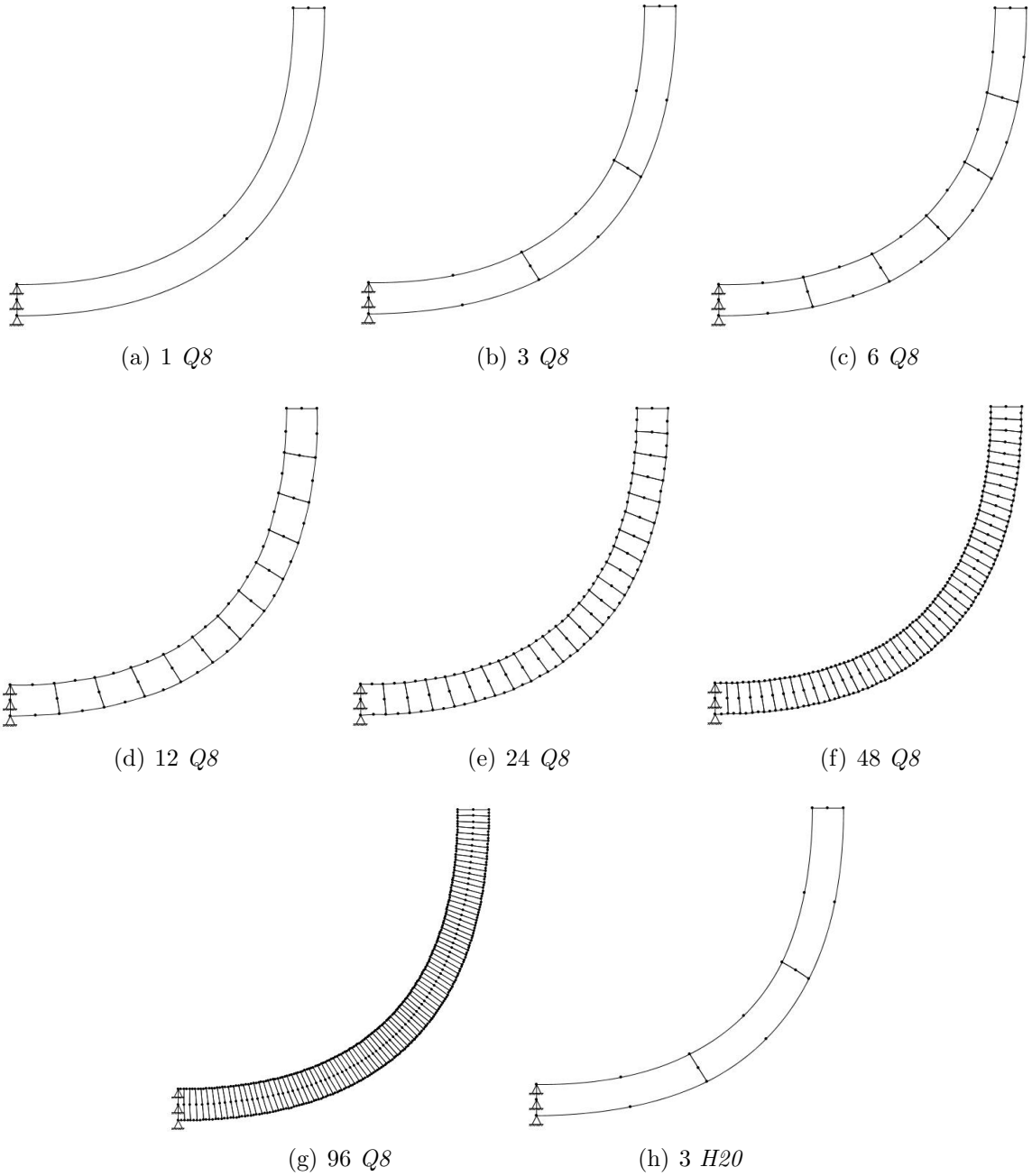
Figura 6.75: Barra de eixo curvo

A solução exata para o deslocamento horizontal u da extremidade livre da barra em questão foi obtida em (Timoshenko & Goodier 1980), sendo dada por:

$$u_{(\theta=0)} = -\frac{P\pi(a^2 + b^2)}{E[(a^2 - b^2) + (a^2 + b^2) \times ln(b/a)]} \tag{6.15.1}$$

As malhas utilizadas para modelar a barra com elementos *Q8* consistem de 1, 3, 6, 12, 24, 48 e 96 elementos, enquanto a malha de elementos *H20* consiste de 3 elementos.

Na figura 6.76 são mostradas as malhas adotadas. Os resultados obtidos pelo programa para os deslocamentos horizontais (u) na extremidade livre da barra são mostrados no gráfico da figura 6.77.



(a) 1 *Q8* (b) 3 *Q8* (c) 6 *Q8*

(d) 12 *Q8* (e) 24 *Q8* (f) 48 *Q8*

(g) 96 *Q8* (h) 3 *H20*

Figura 6.76: Malhas adotadas para a barra curva

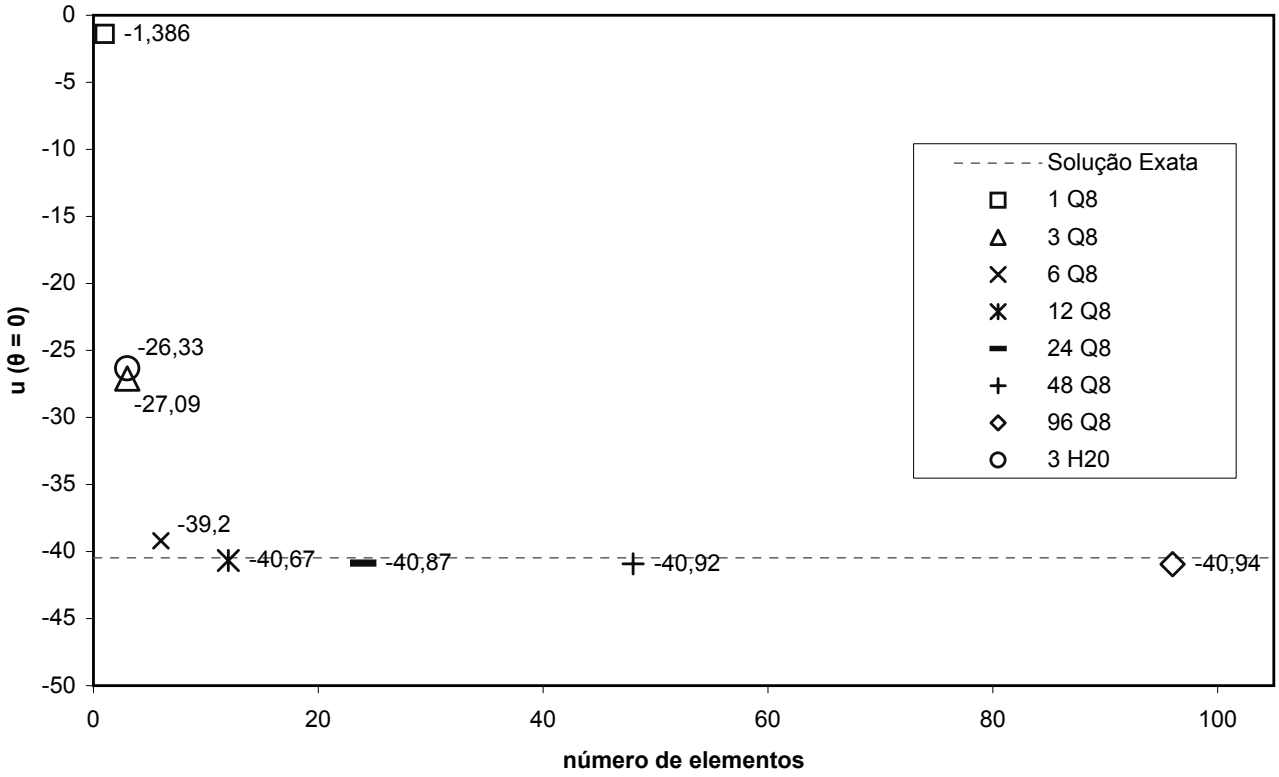


Figura 6.77: Deslocamentos horizontais da extremidade livre da barra curva

Neste exemplo verifica-se que os resultados obtidos para os deslocamentos horizontais convergem para a solução exata à medida em que as malhas são refinadas.

6.16 Dente de Engrenagem

Este exemplo mostra os resultados obtidos ao discretizar um dente de engrenagem submetido a uma carga cisalhante agindo em sua extremidade conforme a figura 6.78. O dente proposto aqui foi sugerido por (Weaver & Johnston 1984) na qual foram descritas as coordenadas dos nós localizados nas faces curvas do dente. É considerado carregamento cisalhante de linha $q = 10\ kN/cm$, $L = 1\ cm$, módulo de elasticidade $E = 7 \times 10^3\ kN/cm^2$ e coeficiente de Poisson $\nu = 0,3$. Existe simetria em relação ao plano xy da figura 6.78, possibilitando a discretização de metade do problema ao restringir os deslocamentos w, na direção z, de todos os nós do plano xy. O engastamento da base do dente é garantido ao restringir todos os deslocamentos dos nós localizados no plano xz.

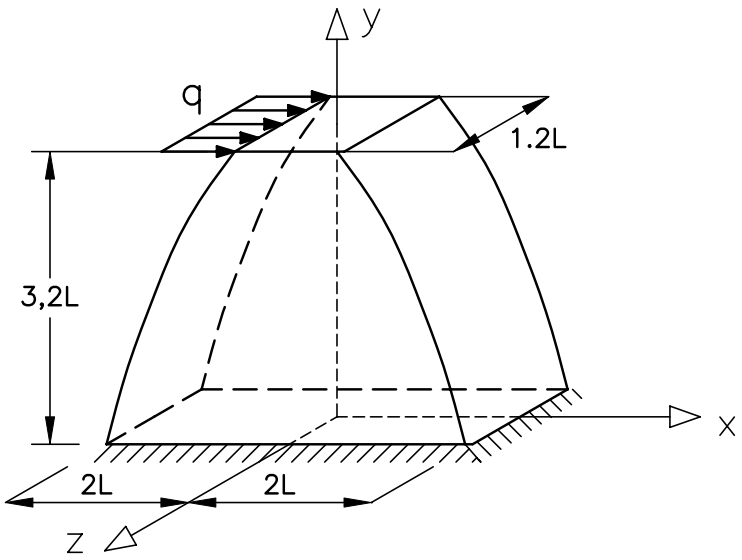


Figura 6.78: Dente de engrenagem

O dente é discretizado com os elementos bidimensionais quadrilaterais *Q8* e com os elementos tridimensionais hexaédricos *H20*. Para o caso bidimensional é utilizado o modelo de análise *PlaneStressAnalysisM*, carregamento concentrado nodal para representar a força cisalhante q e quadratura de Gauss com 3×3 pontos. Enquanto no caso tridimensional utiliza-se o modelo *SolidAnalysisM*, carregamento distribuído constante de linha do tipo *LineElementForce* representando a força cisalhante q e quadratura de Gauss com $3 \times 3 \times 3$ pontos.

Na figura 6.79 é mostrada a malha de elementos *Q8* e na figura 6.80 é representada a malha de elementos *H20*.

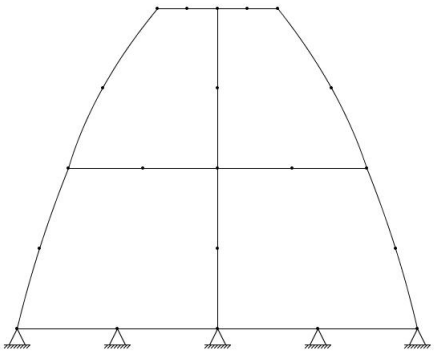


Figura 6.79: Malha de 4 elementos *Q8*

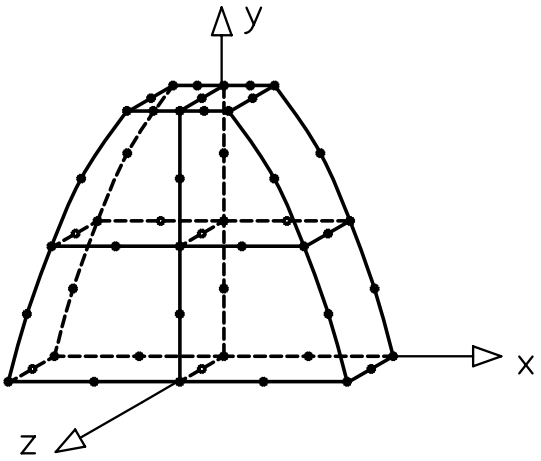


Figura 6.80: Malha de 4 elementos *H20*

No gráfico da figura 6.81 são mostrados os resultados obtidos pelo programa implementado para as duas malhas, consistindo dos deslocamentos horizontais u dos nós localizados sobre o eixo y do dente, bem como os resultados retirados da referência usada (Weaver & Johnston 1984).

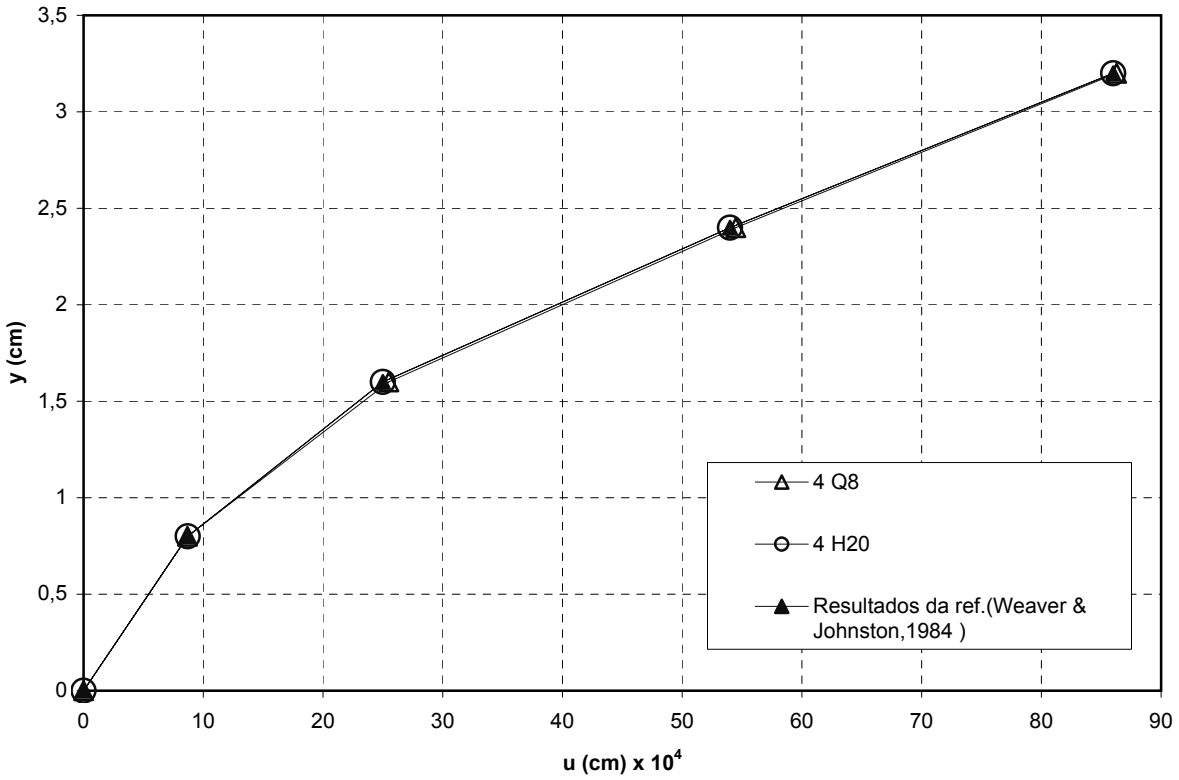


Figura 6.81: Deslocamentos horizontais dos nós localizados sobre o eixo y

Neste exemplo verifica-se que os deslocamentos horizontais obtidos para o modelo bidimensional e para o modelo tridimensional coincidem. Verifica-se também que os dois modelos (bidimensional e tridimensional) coincidem com os valores fornecidos pela referência utilizada como fonte.

Capítulo 7

Considerações Finais

Uma das propostas do projeto INSANE é trazer para a comunidade acadêmica soluções tecnológicas para o desenvolvimento de aplicações que auxiliem as pesquisas na área de métodos numéricos e computacionais. A dissertação aqui apresentada contribui para o objetivo citado ao disponibilizar um aplicativo de fácil expansão, pronto para atender às crescentes necessidades da pesquisa de modelos discretos de análise estrutural.

Os resultados obtidos neste trabalho, validados pelos diversos exemplos apresentados, só foram possíveis graças às soluções tecnológicas empregadas (consideradas no item 7.1) e ao desenvolvimento colaborativo de vários sub-projetos (citados no item 7.2)

7.1 Soluções Tecnológicas

A programação orientada a objetos foi uma ferramenta indispensável neste trabalho, uma vez que proporcionou grande agilidade e versatilidade, possibilitando o desenvolvimento dos diversos recursos disponibilizados no programa, destacando-se os vários elementos, modelos de análise e carregamentos implementados.

Os diversos conceitos da programação orientada a objetos (POO) foram muito bem aproveitados na implementação computacional do sistema, principalmente devido a adoção da formulação paramétrica do MEF, cuja generalidade permite a reutilização dos mesmos métodos e procedimentos repetidas vezes para obtenção das propriedades de diferentes entidades. Como exemplo desta vantagem, pode-se citar o procedimento para obtenção da matriz de rigidez de um elemento finito. Após a criação de um método geral para o cálculo da matriz de rigidez pôde-se utilizar este método para obter a matriz de

rigidez de qualquer elemento finito.

Outro benefício da utilização da POO, observado durante a realização deste trabalho, foi a grande adaptabilidade dos módulos de software a futuras mudanças. Isto foi possível graças ao encapsulamento dos dados, que permitiu a alteração de detalhes de partes do programa sem prejudicar os demais módulos.

A escolha da linguagem Java mostrou-se totalmente acertada, uma vez que pôde-se explorar o grande potencial desta linguagem no desenvolvimento do trabalho. Como aspectos que corroboram a adequação de Java, pode-se citar: a utilização de várias bibliotecas de classes prontas e testadas, desenvolvidas gratuitamente por membros da comunidade de programadores; o suporte à persistência de dados, viabilizando a comunicação entre os segmentos do projeto INSANE; e ainda a independência de plataforma, evitando que todo o processo de implementação seja realizado novamente, sempre que for preciso migrar para outra plataforma.

Outro fato que atesta o acerto na escolha de Java é o seu crescente uso pela comunidade de software livre. Estatísticas de dois dos maiores portais para hospedagem de projetos livres ($www.freshmeat.net/appindex/development/languages.html$) e ($www.sourceforge.net/softwaremap/trove\_list.php?form\_cat = 160$) mostram que Java está chegando ao primeiro posto desta competição, muito próxima da campeã C++.

A persistência adequada dos dados também foi um requisito importante, plenamente atingido neste trabalho. A adoção de um padrão que permitisse a auto-descrição dos dados, a extensibilidade desta descrição sem dificultar a autoria e a fácil transmissão dos dados pela internet, foi atendida com o padrão XML.

7.2 Desenvolvimento Colaborativo

O desenvolvimento do projeto INSANE é feito de forma colaborativa, envolvendo alunos em diferentes estágios de conhecimento ((Fonseca & Pitangueira 2004), (Fonseca, Pitangueira & Filho 2004), (Gonçalves & Pitangueira 2004*a*) e (Gonçalves & Pitangueira 2004*b*)). Alguns trabalhos, já em andamento, e sugestões para trabalhos futuros são listados a seguir.

7.2.1 Trabalhos em Andamento

- Implementação do pós-processador para análise gráfica de resultados;

- Integração do processador desenvolvido neste trabalho com o pré-processador existente;

- Implementação de um processador voltado para o ensino do MEF;

- Expansão do projeto INSANE para contemplar análise dinâmica;

- Expansão do projeto INSANE para contemplar análise não linear;

- Desenvolvimento de um servidor WEB para o sistema;

- Implementar generalizações no sistema para contemplar outros problemas como: transferência de calor, mecânica dos fluidos, problemas de campo etc;

- Iniciar o uso dos testes automatizados para o código através do *JUnit* (*http://junit.org/index.html*);

- Utilização de ferramentas para gerenciamento do sistema como o *CVS* (*http://www.cvshome.org/docs/manual*) e o *Maven* (*http://maven.apache.org*).

7.2.2 Sugestões para Trabalhos Futuros

- Implementação de elementos finitos para placas e cascas;

- Implementação de modelos de elementos finitos de fissuras distribuídas;

- Implementação de modelos de elementos finitos de fissuras discretas;

- Implementação de modelos de elementos finitos para plasticidade;

- Implementação de modelo para métodos sem malha;

- Implementação de modelo para o método dos elementos de contorno.

Apêndice A

Formato do Arquivo XML para Persistência

Conforme já discutido na seção 1.1, a interface entre o usuário e o processador implementado é realizada por meio da linguagem padronizada de arquivos XML (ver seção 3.3). Será apresentado a seguir um exemplo da estrutura XML adotada para representar os dados referentes a um modelo paramétrico de elementos finitos. O exemplo apresentado refere-se à malha de 2 elementos axissimétricos triangulares de três nós, usada para o modelo do tubo axissimétrico da seção 6.11. A malha é mostrada novamente na figura A.1.

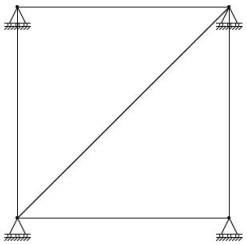


Figura A.1: Malha de 2 elementos *AxiT3* utilizada na seção 6.11

O formato do arquivo XML é mostrado nas figuras A.2 e A.3. É possível visualizar este mesmo arquivo em um formato resumido como mostra a figura A.4.

Após análise e resolução do problema, o processador cria um novo arquivo, no formato XML, contendo todos os resultados obtidos. Este arquivo é mostrado nas figuras A.5 e A.6.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE Model (View Source for full doctype...)>
- <Model class="ParametricModel">
  - <NodeList>
    - <Node label="1">
        <Coord>10.0 0.0 0.0</Coord>
        <Restraints>false true false false false false</Restraints>
        <Load>0.000 0.000 0.000 0.000 0.000 0.000</Load>
        <PreDisplacements>0.000E00 0.000E00 0.000E00 0.000E00 0.000E00
          0.000E00</PreDisplacements>
        <mySpring>0.000E00 0.000E00 0.000E00 0.000E00 0.000E00
          0.000E00</mySpring>
        <Angle>0.0</Angle>
      </Node>
    - <Node label="2">
        <Coord>10.0 1.0 0.0</Coord>
        <Restraints>false true false false false false</Restraints>
        <Load>0.000 0.000 0.000 0.000 0.000 0.000</Load>
        <PreDisplacements>0.000E00 0.000E00 0.000E00 0.000E00 0.000E00
          0.000E00</PreDisplacements>
        <mySpring>0.000E00 0.000E00 0.000E00 0.000E00 0.000E00
          0.000E00</mySpring>
        <Angle>0.0</Angle>
      </Node>
    - <Node label="3">
        <Coord>11.0 0.0 0.0</Coord>
        <Restraints>false true false false false false</Restraints>
        <Load>0.000 0.000 0.000 0.000 0.000 0.000</Load>
        <PreDisplacements>0.000E00 0.000E00 0.000E00 0.000E00 0.000E00
          0.000E00</PreDisplacements>
        <mySpring>0.000E00 0.000E00 0.000E00 0.000E00 0.000E00
          0.000E00</mySpring>
        <Angle>0.0</Angle>
      </Node>
    - <Node label="4">
        <Coord>11.0 1.0 0.0</Coord>
        <Restraints>false true false false false false</Restraints>
        <Load>0.000 0.000 0.000 0.000 0.000 0.000</Load>
        <PreDisplacements>0.000E00 0.000E00 0.000E00 0.000E00 0.000E00
          0.000E00</PreDisplacements>
        <mySpring>0.000E00 0.000E00 0.000E00 0.000E00 0.000E00
          0.000E00</mySpring>
        <Angle>0.0</Angle>
      </Node>
    </NodeList>
  - <MaterialList>
    - <Material class="Isotropic" label="mat">
        <Elasticity>10000.0</Elasticity>
        <Poisson>0.3</Poisson>
      </Material>
    </MaterialList>
  - <ElementList>
    - <Element class="ParametricElement.Triangular.AxiT3" label="E1">
        <Type>T S</Type>
        <Incidence>3 4 1</Incidence>
        <Thickness>1.0</Thickness>
```

Figura A.2: Formato do arquivo XML utilizado para representar o modelo da figura A.1 (1ª parte)

```
      <IntegrationOrder>1 1 0</IntegrationOrder>
      <MyMaterial>mat</MyMaterial>
      <MyAnalysisModel>AxiSymetricAnalysisM</MyAnalysisModel>
      <MyElmPointForce />
      <MyElementForce />
    </Element>
  - <Element class="ParametricElement.Triangular.AxiT3" label="E2">
      <Type>T S</Type>
      <Incidence>4 2 1</Incidence>
      <Thickness>1.0</Thickness>
      <IntegrationOrder>1 1 0</IntegrationOrder>
      <MyMaterial>mat</MyMaterial>
      <MyAnalysisModel>AxiSymetricAnalysisM</MyAnalysisModel>
    - <MyLineForce>
      - <LineElementForce>
        - <PointForce>
            <PointForceCoords>10.0 1.0 0.0</PointForceCoords>
            <Force>1 0 0 0 0 0</Force>
          </PointForce>
        - <PointForce>
            <PointForceCoords>10.0 0.0 0.0</PointForceCoords>
            <Force>1 0 0 0 0 0</Force>
          </PointForce>
        </LineElementForce>
      </MyLineForce>
      <MyElmPointForce />
      <MyElementForce />
    </Element>
  </ElementList>
  <GlobalAnalysisModel>AxiSymetricAnalysisM</GlobalAnalysisModel>
  <Solution>OnePointEq</Solution>
  <Driver>StructuralMech</Driver>
</Model>
```

Figura A.3: Formato do arquivo XML utilizado para representar o modelo da figura A.1 (2ª parte)

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE Model (View Source for full doctype...)>
- <Model class="ParametricModel">
  - <NodeList>
    + <Node label="1">
    + <Node label="2">
    + <Node label="3">
    + <Node label="4">
    </NodeList>
  - <MaterialList>
    + <Material class="Isotropic" label="mat">
    </MaterialList>
  - <ElementList>
    + <Element class="ParametricElement.Triangular.AxiT3" label="E1">
    + <Element class="ParametricElement.Triangular.AxiT3" label="E2">
    </ElementList>
    <GlobalAnalysisModel>AxiSymetricAnalysisM</GlobalAnalysisModel>
    <Solution>OnePointEq</Solution>
    <Driver>StructuralMech</Driver>
  </Model>
```

Figura A.4: Forma resumida de visualização

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE JMefData (View Source for full doctype...)>
- <JMefData>
  - <Driver class="class model.discrete.driver.StructuralMech">
      <Solution class="class model.discrete.solution.OnePointEq" />
    - <FemModel class="class model.discrete.femmodel.FemModel">
        <NumberOfEquations>4</NumberOfEquations>
        <NumberOfRestraints>4</NumberOfRestraints>
      - <NodeList>
        - <Node label="1">
            <Coord>1.000E01 0.000E00 0.000E00</Coord>
            <Restraints>false true false false false false</Restraints>
            <Displacements>9.943E-03 0.000E00 0.000E00 0.000E00 0.000E00 0.000E00</Displacements>
            <Load>0.000 0.000 0.000 0.000 0.000 0.000</Load>
            <PreDisplacements>0.000E00 0.000E00 0.000E00 0.000E00 0.000E00 0.000E00</PreDisplacements>
            <mySpring>0.000E00 0.000E00 0.000E00 0.000E00 0.000E00 0.000E00</mySpring>
            <ForceOnSpring>-0.000 -0.000 -0.000 -0.000 -0.000 -0.000</ForceOnSpring>
            <Reactions>0.000 -91.590 0.000 0.000 0.000 0.000</Reactions>
            <Angle>0.0</Angle>
          </Node>
        - <Node label="2">
            <Coord>1.000E01 1.000E00 0.000E00</Coord>
            <Restraints>false true false false false false</Restraints>
            <Displacements>9.961E-03 0.000E00 0.000E00 0.000E00 0.000E00 0.000E00</Displacements>
            <Load>0.000 0.000 0.000 0.000 0.000 0.000</Load>
            <PreDisplacements>0.000E00 0.000E00 0.000E00 0.000E00 0.000E00 0.000E00</PreDisplacements>
            <mySpring>0.000E00 0.000E00 0.000E00 0.000E00 0.000E00 0.000E00</mySpring>
            <ForceOnSpring>-0.000 -0.000 -0.000 -0.000 -0.000 -0.000</ForceOnSpring>
            <Reactions>0.000 91.489 0.000 0.000 0.000 0.000</Reactions>
            <Angle>0.0</Angle>
          </Node>
        - <Node label="3">
            <Coord>1.100E01 0.000E00 0.000E00</Coord>
            <Restraints>false true false false false false</Restraints>
            <Displacements>9.528E-03 0.000E00 0.000E00 0.000E00 0.000E00 0.000E00</Displacements>
            <Load>0.000 0.000 0.000 0.000 0.000 0.000</Load>
            <PreDisplacements>0.000E00 0.000E00 0.000E00 0.000E00 0.000E00 0.000E00</PreDisplacements>
            <mySpring>0.000E00 0.000E00 0.000E00 0.000E00 0.000E00 0.000E00</mySpring>
            <ForceOnSpring>-0.000 -0.000 -0.000 -0.000 -0.000 -0.000</ForceOnSpring>
            <Reactions>0.000 -96.905 0.000 0.000 0.000 0.000</Reactions>
            <Angle>0.0</Angle>
          </Node>
        - <Node label="4">
            <Coord>1.100E01 1.000E00 0.000E00</Coord>
            <Restraints>false true false false false false</Restraints>
            <Displacements>9.512E-03 0.000E00 0.000E00 0.000E00 0.000E00 0.000E00</Displacements>
            <Load>0.000 0.000 0.000 0.000 0.000 0.000</Load>
            <PreDisplacements>0.000E00 0.000E00 0.000E00 0.000E00 0.000E00 0.000E00</PreDisplacements>
            <mySpring>0.000E00 0.000E00 0.000E00 0.000E00 0.000E00 0.000E00</mySpring>
            <ForceOnSpring>-0.000 -0.000 -0.000 -0.000 -0.000 -0.000</ForceOnSpring>
            <Reactions>0.000 97.006 0.000 0.000 0.000 0.000</Reactions>
            <Angle>0.0</Angle>
          </Node>
        </NodeList>
      - <MaterialList>
        - <Material class="class model.discrete.material.Isotropic" label="mat">
            <Elasticity>10000.0</Elasticity>
            <Poisson>0.3</Poisson>
          </Material>
        </MaterialList>
      - <CrossSectionList>
        - <CrossSection class="class model.discrete.crosssection.CrossSection" label="cs_1.0">
            <Thickness>1.0</Thickness>
          </CrossSection>
        </CrossSectionList>
      - <AnalysisModelList>
          <AnalysisModel class="class model.discrete.analysismodel.AxiSymetricAnalysisM"
            label="AxiSymetricAnalysisM" />
        </AnalysisModelList>
      - <ElementList>
        - <Element class="class model.discrete.element.ElmAxiT3" label="E1">
            <Incidence>3 4 1</Incidence>
            <IntegrationOrder>3 3 0</IntegrationOrder>
            <MyMaterial>mat</MyMaterial>
            <MyAnalysisModel>AxiSymetricAnalysisM</MyAnalysisModel>
          - <GaussPointList>
            - <GaussPointResults>
                <GPCartesianCoords>x y z</GPCartesianCoords>
                <GPStrains>epsilonR epsilonZ epsilonTeta gammaRTeta</GPStrains>
                <GPStresses>sigmaR sigmaZ sigmaTeta tauRTeta</GPStresses>
              </GaussPointResults>
            - <GaussPoint label="GP-1">
                <GPCartesianCoord>1.0500E01 5.0000E-01 0.0000E00</GPCartesianCoord>
```

Figura A.5: Formato do arquivo XML criado pelo processador para representar a solução do problema (1ª parte)

```
            <GPStrain>-4.152E-04 0.000E00 9.264E-04 -1.611E-05</GPStrain>
            <GPStress>-2.446831E-01 2.949322E00 1.007576E01 -6.197177E-02</GPStress>
          </GaussPoint>
        - <GaussPoint label="GP-2">
            <GPCartesianCoord>1.0500E01 0.0000E00 0.0000E00</GPCartesianCoord>
            <GPStrain>-4.152E-04 0.000E00 9.272E-04 -1.611E-05</GPStrain>
            <GPStress>-2.402566E-01 2.953749E00 1.008609E01 -6.197177E-02</GPStress>
          </GaussPoint>
        - <GaussPoint label="GP-3">
            <GPCartesianCoord>1.1000E01 5.0000E-01 0.0000E00</GPCartesianCoord>
            <GPStrain>-4.152E-04 0.000E00 8.655E-04 -1.611E-05</GPStrain>
            <GPStress>-5.965163E-01 2.597489E00 9.254813E00 -6.197177E-02</GPStress>
          </GaussPoint>
        </GaussPointList>
      - <NodalStressesAndStrains>
          <NodeCoords>x y z</NodeCoords>
          <NodeStrains>epsilonR epsilonZ epsilonTeta gammaRTeta</NodeStrains>
          <NodeStresses>sigmaR sigmaZ sigmaTeta tauRTeta</NodeStresses>
          <Node label="3" />
          <NodeCoord>1.100E01 0.000E00 0.000E00</NodeCoord>
          <NodeStrain>-4.152E-04 0.000E00 8.647E-04 -1.611E-05</NodeStrain>
          <NodeStress>-6.007E-01 2.593E00 9.245E00 -6.197E-02</NodeStress>
          <Node label="4" />
          <NodeCoord>1.100E01 1.000E00 0.000E00</NodeCoord>
          <NodeStrain>-4.152E-04 0.000E00 9.943E-04 -1.611E-05</NodeStrain>
          <NodeStress>1.470E-01 3.341E00 1.099E01 -6.197E-02</NodeStress>
          <Node label="1" />
          <NodeCoord>1.000E01 0.000E00 0.000E00</NodeCoord>
          <NodeStrain>-4.152E-04 0.000E00 8.662E-04 -1.611E-05</NodeStrain>
          <NodeStress>-5.923E-01 2.602E00 9.265E00 -6.197E-02</NodeStress>
        </NodalStressesAndStrains>
      </Element>
    - <Element class="class model.discrete.element.ElmAxiT3" label="E2">
        <Incidence>4 2 1</Incidence>
        <IntegrationOrder>3 3 0</IntegrationOrder>
        <MyMaterial>mat</MyMaterial>
        <MyAnalysisModel>AxiSymetricAnalysisM</MyAnalysisModel>
      - <GaussPointList>
        - <GaussPointResults>
            <GPCartesianCoords>x y z</GPCartesianCoords>
            <GPStrains>epsilonR epsilonZ epsilonTeta gammaRTeta</GPStrains>
            <GPStresses>sigmaR sigmaZ sigmaTeta tauRTeta</GPStresses>
          </GaussPointResults>
        - <GaussPoint label="GP-1">
            <GPCartesianCoord>1.0000E01 5.0000E-01 0.0000E00</GPCartesianCoord>
            <GPStrain>-4.488E-04 0.000E00 9.952E-04 1.744E-05</GPStrain>
            <GPStress>-2.996667E-01 3.152441E00 1.080780E01 6.707934E-02</GPStress>
          </GaussPoint>
        - <GaussPoint label="GP-2">
            <GPCartesianCoord>1.0500E01 5.0000E-01 0.0000E00</GPCartesianCoord>
            <GPStrain>-4.488E-04 0.000E00 9.264E-04 1.744E-05</GPStrain>
            <GPStress>-6.963620E-01 2.755746E00 9.882181E00 6.707934E-02</GPStress>
          </GaussPoint>
        - <GaussPoint label="GP-3">
            <GPCartesianCoord>1.0500E01 1.0000E00 0.0000E00</GPCartesianCoord>
            <GPStrain>-4.488E-04 0.000E00 9.273E-04 1.744E-05</GPStrain>
            <GPStress>-6.915706E-01 2.760537E00 9.893360E00 6.707934E-02</GPStress>
          </GaussPoint>
        </GaussPointList>
      - <NodalStressesAndStrains>
          <NodeCoords>x y z</NodeCoords>
          <NodeStrains>epsilonR epsilonZ epsilonTeta gammaRTeta</NodeStrains>
          <NodeStresses>sigmaR sigmaZ sigmaTeta tauRTeta</NodeStresses>
          <Node label="4" />
          <NodeCoord>1.100E01 1.000E00 0.000E00</NodeCoord>
          <NodeStrain>-4.488E-04 0.000E00 9.961E-04 1.744E-05</NodeStrain>
          <NodeStress>-2.946E-01 3.157E00 1.082E01 6.708E-02</NodeStress>
          <Node label="2" />
          <NodeCoord>1.000E01 1.000E00 0.000E00</NodeCoord>
          <NodeStrain>-4.488E-04 0.000E00 9.943E-04 1.744E-05</NodeStrain>
          <NodeStress>-3.047E-01 3.147E00 1.080E01 6.708E-02</NodeStress>
          <Node label="1" />
          <NodeCoord>1.000E01 0.000E00 0.000E00</NodeCoord>
          <NodeStrain>-4.488E-04 0.000E00 8.647E-04 1.744E-05</NodeStrain>
          <NodeStress>-1.052E00 2.400E00 9.051E00 6.708E-02</NodeStress>
        </NodalStressesAndStrains>
      </Element>
    </ElementList>
    <GlobalAnalysisModel>AxiSymetricAnalysisM</GlobalAnalysisModel>
  </FemModel>
</Driver>
</JMefData>
```


Figura A.6: Formato do arquivo XML criado pelo processador para representar a solução do problema (2ª parte)

Referências Bibliográficas

Alvim, P. (2003), 'Open source: Os novos desafios de negócios e a indústria de ti', *Developers Magazine* .

Ansys-INC (2004), *ANSYS - Finite Element Program Users*, Cononsburg/USA. V. 8.1.

Barros, J. C. P. (1994), 'Análise térmica pelo método de elementos finitos implementada através de uma filosofia orientada a objetos, dissertação de mestrado', *Pontifícia Universidade Católica do Rio de Janeiro, Puc-RJ* .

Braz, M. R. (2003), 'Tecnologia de web services: Definições e perspectivas', *Developers Magazine* (80), 22–24.

Camarão, C. & Figueiredo, L. (2003), *Programação de Computadores em Java*, LTC.

Cook, R. D., Malkus, D. S. & Plesha, M. E. (1989), *Concepts and Applications of Finite Element Analysis*, third edn, John Wiley & Sons.

Dawe, D. J. (1983), *Matrix and Finite Element Displacement Analysis of Structures*, Oxford University Press.

Deitel, H. M. & Deitel, P. J. (2001), *Java Como Programar, 3ª ed.*, Bookman.

Fonseca, F. T., Pitangueira, R. L. S. & Filho, A. V. (2004), 'Implementação de modelos estruturais de barras como casos particulares do método de elementos finitos', *SIMMEC/2004, Itajubá* .

Fonseca, F. T. & Pitangueira, R. L. S. (2004), 'Um programa gráfico interativo para modelos estruturais de barras', *XXV CILAMCE, Recife* .

Fowler, M. & Scott, K. (2000), *UML Essencial*, Bookman.

Fuina, J. S. (2004), 'Métodos de controle de deformações para análise não linear de estruturas, dissertação de mestrado', *Escola de Engenharia da UFMG, Belo Horizonte-MG* .

Gamma, E., Helm, R., Johnson, R. & Vlissides, J. (1995), *Design Paterns - Elements of Reusable Object-Oriented Software*.

Gonçalves, M. A. B. & Pitangueira, R. L. S. (2004*a*), 'O padrão model-view-controller para um gerador de malhas bidimensionais de elementos finitos', *SIMMEC/2004, Itajubá* .

Gonçalves, M. A. B. & Pitangueira, R. L. S. (2004*b*), 'Padrões de projeto de software para um gerador de malhas bidimensionais de elementos finitos', *XXV CILAMCE, Recife* .

Gonçalves, M. A. B. (2004), 'Geração de malhas bidimensionais de elementos finitos baseada em mapeamentos transfinitos, dissertação de mestrado', *Escola de Engenharia da UFMG, Belo Horizonte-MG* .

Goodrich, M. T. & Tamassia, R. (2002), *Estruturas de Dados e Algoritmos em Java*, Bookman.

Guimarães, L. G. S. (1992), 'Disciplina de programação orientada a objetos para análise e visualização bidimensionais de modelos de elementos finitos, dissertação de mestrado', *Pontifícia Universidade Católica do Rio de Janeiro, Puc-RJ* .

Holanda, A. S. (2000), 'Análise do equilíbrio e estabilidade de placas com restrições de contato, tese de doutorado', *Pontifícia Universidade Católica do Rio de Janeiro, Puc-RJ* .

Júnior, E. P. (2000), 'Análise de sensibilidade e otimização de forma de estruturas geometricamente não lineares, tese de doutorado', *Pontifícia Universidade Católica do Rio de Janeiro, Puc-RJ* .

Lages, E. N. (1997), 'Modelagem de localização de deformações com teorias de contínuo generalizado, tese de doutorado', *Pontifícia Universidade Católica do Rio de Janeiro, Puc-RJ* .

Lichao, Y. & Ashok, V. K. (2001), 'An object-oriented modular framework for implementing the finite element method', *Computers and Structures* (79), 919–928.

Liesenfeld, R. (2002), 'Processando xml em java', *Java Magazine* pp. 48–52.

Lozano, F. (2003), 'Entenda a tecnologia xml', *Revista do Linux* pp. 42–49.

Martha, L. F., Menezes, I. F., Lages, E. N., Parente, J. E. P. & Pitangueira, R. L. S. (1996), 'An oop class organization for materially nonlinear finite element analysis', *XVII CILAMCE, Padova Itália* pp. 229–232.

Neto, J. B. G. (1994), 'Simulação auto-adaptativa baseada em enumeração espacial recursiva em modelos bidimensionais de elementos finitos, dissertação de mestrado', *Pontifícia Universidade Católica do Rio de Janeiro, Puc-RJ* .

Nikishkov, G. P., Nikishkov, Y. G. & Savchenko, V. V. (2003), 'Comparasion of c and java performance in finite element computations', *Computers and Structures* (81), 2401–2408.

Noronha, M. A. M. (1998), 'Técnicas avançadas de integração numérica e programação orientada a objetos aplicadas e métodos de elementos de contorno, tese de doutorado', *Pontifícia Universidade Católica do Rio de Janeiro, Puc-RJ* .

Oñate, E. (1995), *Cálculo de Estructuras por el Método de Elementos Finitos - Analisis Estático Lineal*, CIMNE, Barcelona.

Pietro, G. A. (2001), 'Utilização de padrões de projeto na reengenharia de sistemas, dissertação de mestrado', *UFSCar, São Carlos - SP* .

Pitangueira, R. L. S. (1998), 'Mecânica de estruturas de concreto com inclusão de efeitos de tamanho e heterogeneidade, tese de doutorado', *Pontifícia Universidade Católica do Rio de Janeiro, Puc-RJ* .

Pitangueira, R. L. S. (2000), *Introdução ao Método dos Elementos Finitos, Notas de Aula, Depto Eng. Estruturas da UFMG.*

Santos, R. (2003), *Introdução à Programação Orientada a Objetos Usando Java*, Campus.

Silva, P. P. R. (2001), 'Implementação de modelos de microplanos para análise de estruturas de concreto, uma abordagem orientada a objetos, dissertação de mestrado', *Escola de Engenharia da UFMG, Belo Horizonte-MG* .

Simão, W. I. S. (2003), 'Modelos de armadura e aderência para análise não linear de estruturas de concreto armado, dissertação de mestrado', *Escola de Engenharia da UFMG, Belo Horizonte-MG* .

Soriano, H. L. & Lima, S. S. (1999), *Método de Elementos Finitos em Análise de Estrutura*, Universidade Federal do Rio de Janeiro.

Sybine, W. (1997), 'Uma modelagem de fenômenos de contato com impacto utilizando o método de elementos finitos numa implementação orientada a objetos, dissertação de mestrado', *Pontifícia Universidade Católica do Rio de Janeiro, Puc-RJ* .

Timoshenko, S. P. & Goodier, J. N. (1980), *Teoria da Elasticidade, 3ª ed.*, Guanabara Dois.

Weaver, W. J. & Johnston, P. R. (1984), *Finite Elements for Structural Analysis*, Prentice-Hall.

Zienkiewicz, O. C., Chan, A. C. H., Taylor, R. L. & Simo, J. C. (1986), 'The patch test - a condition for assessing fem convergence', *International Journal For Numerical Methods In Engineering* **22**, 39–62.

Zienkiewicz, O. C. (1979), *The Finite Element Method*, McGraw Hill.