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

SISTEMA GRÁFICO INTERATIVO PARA ENSINO DE ANÁLISE ESTRUTURAL ATRAVÉS DO MÉTODO DOS ELEMENTOS FINITOS

Renata Nicoliello Moreira

Orientador: Roque Luiz da Silva Pitangueira

Belo Horizonte
Setembro de 2006

Aprender é a única coisa de que a mente nunca se cansa, nunca tem medo e nunca se arrepende.

(Leonardo da Vinci*)*

Dedico este trabalho às pessoas que dão sentido a cada passo da minha caminhada:

minha mãe, Francisca; meu pai, Lúcio Flávio; meus irmãos, Bruno e Arthur; e ao meu amor, Gustavo.

Índice

Lista de Tabelas

Lista de Figuras

Lista de Abreviaturas, Siglas e Símbolos

API	Aplication program interface
DOF	Degrees of freedom
INSANE	Interactive structural analysis environment
JVM	Java virtual machine
MEF	Método dos elementos finitos
MVC	Model-view-controller
POO	Programação orientada a objetos
uc	Unidade de comprimento
uf	Unidade de força
UML	Unified modelling language
XML	Extensible markup language
WEB	Rede internacional de computadores

Resumo

Esta dissertação de mestrado refere-se à expansão do `INSANE` (INteractive Structural ANalysis Environment): um sistema computacional, em linguagem java e programação orientada a objetos (POO) para o processamento de modelos discretos do método de elementos finitos (MEF). O trabalho consiste numa aplicação gráfica interativa para auxiliar o ensino de análise estrutural através do MEF.

Apresenta-se um estudo dos diversos enfoques para a abordagem do MEF em cursos de engenharia, identificando suas generalidades e as possibilidades que o `INSANE` oferece para facilitar o processo de aprendizagem. Discute-se, então, sugestões disponíveis na literatura para caracterização de etapas para a solução de problemas do MEF.

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 para exposição do núcleo numérico do sistema computacional ao usuário, de modo que o mesmo possa visualizar informações referentes à resolução de modelos do MEF.

Este trabalho documenta e destaca a importância das fases de Análise, Projeto e Implementação Orientados a Objetos. Padrões de projeto de *software* reconhecidamente eficientes são adotados na implementação desta aplicação. Visando separar o modelo de sua representação gráfica, a implementação é baseada na metáfora de programação denominada *Model-View-Controller* (*MVC*). Tal enfoque permite a apresentação gráfica, interativa e didática do processamento de modelos do MEF. O projeto orientado a objetos da implementação é, então, apresentado com o auxílio de diagramas UML *(Unified Modelling Language)* apropriados.

Os recursos disponibilizados na nova interface são apresentados através de exemplos e as possibilidades de enriquecimento do processo de aprendizagem são, então, ilustradas.

Abstract

This master's thesis refers to the expansion of `INSANE` (INteractive Structural ANalysis Environment): a computational system for finite element method (FEM) structural analysis discrete models, in Java language and object oriented programming (OOP). The work consists in an interactive graphic application to assist the FEM teaching for structural engineering.

A study of the diverse approaches for FEM in engineering courses is presented, identifying its generalities and the possibilities that `INSANE` offers to facilitate learning process. It is argued, then, available suggestions in literature for characterization of stages for the solution of FEM problems.

It is verified that the OOP is quite appropriated for the implementation and the use of language Java has great advantages. An object oriented analysis to identify the main necessary classes for exposition of the numerical nucleus of the computational system to the user is done.

This work documents and focuses on object-oriented Analysis, Project and Implementation. Admittedly efficient software design patterns are adopted in the implementation of the application. To separate the model from its representation, the implementation is based on *Model-View-Controller* (*MVC*) programming metaphor. Such approach allows the graphical, interactive and didactic presentation of the processing of FEM models.

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

The resources of the new graphical interface are presented through examples and the possibilities of enrichment of the learning process are shown.

Agradecimentos

Serei eternamente grata a todos que contribuíram para a realização deste trabalho.

A Deus, por me permitir compartilhar as alegrias e dividir as preocupações com pessoas tão especiais. Por iluminar os passos da minha jornada e me fazer forte, para superar os obstáculos.

A minha mãe, e maior amiga, por nunca deixar de acreditar neste sonho. Agradeço pelo café coado de madrugada e por todo amor e apoio, fundamentais na realização desta conquista.

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

Ao meu irmão Bruno, pela amizade, companheirismo e horas de conversa.

Ao meu irmão Arthur, por renovar minha alegria.

Ao Gustavo, meu grande amor, pela paciência nos momentos em que não pude estar presente, por aliviar minhas preocupações e ouvir com carinho as minhas queixas e lamentações. Por se mostrar sempre feliz com minhas conquistas. E ainda, por me inspirar a ser uma pessoa cada vez melhor.

Aos meus familiares, amigos e amigas, por compreenderem a minha ausência, e ainda, sempre me receberem com palavras de incentivo.

Ao meu orientador, Roque, por acreditar na minha capacidade. Pela incansável disponibilidade em me ajudar, e dividir comigo seus conhecimentos e experiências de vida. Por me mostrar que um trabalho bem feito necessita de muita paciência e concentração.

Aos "Insanos", por dividirem comigo cada novo aprendizado.

Aos professores e funcionários do Departamento de Engenharia de Estruturas pela disponibilidade e atenção em todos os momentos.

À FUMEC, pela oportunidade de aplicar meus conhecimentos.

Ao CNPq por viabilizar financeiramente a concretização deste trabalho.

Capítulo 1

Introdução

A revolução proporcionada pelos computadores e pela facilidade de acesso à informação, além de impulsionar o mercado científico e tecnológico, pode impactar positivamente o processo de ensino-aprendizagem nas diversas áreas do conhecimento.

Com a necessidade de se formar profissionais cada vez mais capacitados, é indispensável que o ensino de engenharia e recursos computacionais facilitem o aprendizado de conceitos complexos. Neste sentido, o computador pode potencializar o ensino focado no aprendiz, proporcionando atendimento individualizado e adaptativo, e permitindo o aprendizado através da prática.

No caso da engenharia estrutural, o uso de recursos computacionais, especialmente softwares e recursos de programação, não é novidade. Entretanto, este uso ainda se restringe à consulta final de resultados. É urgente, portanto, que se utilize as modernas tecnologias informáticas, especialmente a computação gráfica, para criar um ambiente mais dinâmico e de melhor proveito, auxiliando professor e aluno no processo de aprendizagem. Isto pode ocorrer na medida em que professor e aluno usem recursos que permitam, não só obter os resultados, como na prática corrente, mas também interagir e interferir no processo de solução dos problemas.

O Método dos Elementos Finitos (MEF) é um recurso atualmente bastante difundido para se resolver os problemas de análise estrutural. Antes do aparecimento do MEF, a análise dos meios contínuos era, quando possível, efetuada por resolução direta dos sistemas de equações diferenciais parciais que regem o fenômeno, tendo em consideração as

necessárias condições de contorno. Com o grande desenvolvimento que o MEF teve na década de 60, e com a facilidade de acesso ao computador, este passou a ser prática corrente na análise de estruturas de geometria arbitrária, constituídas por múltiplos materiais e sujeitas a qualquer tipo de carregamento.

Nos cursos de graduação e pós-graduação de engenharia, onde a análise estrutural é tratada, é tradicional começar-se por ensinar modelos unidimensionais para vigas, pórticos, treliças e grelhas, genericamente designados modelos reticulados, por serem representativos de peças, cuja seção transversal apresenta dimensões muito inferiores ao comprimento. As estruturas que não possuem esta característica são, em geral, estudadas como meios contínuos (paredes, lajes, cascas, sólidos, etc.). Para ambos os tipos de modelos estruturais, o MEF se mostra bastante eficiente.

O ensino do MEF, por sua vez, pode se tornar bastante árido, à medida que exige do aluno conceitos físicos e matemáticos aprofundados, e abstrações na formulação dos modelos.

Para facilitar o aprendizado do MEF, é fundamental dispor do computador como uma ferramenta favorável ao processo. Assim, em lugar do uso tradicional, em que as soluções são obtidas automaticamente a partir da inserção dos dados iniciais do problema, propõe-se aqui, um sistema computacional, em que a solução aconteça passo a passo, e o aluno possa interagir e, interferindo no processo, aprender a metodologia.

1.1 Objetivos Gerais - 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 define 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 a 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 - *www.dees.ufmg.br/insane*) objetiva o desenvolvimento de um ambiente computacional com as características acima citadas. A figura 1.1 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 disponibiliza, 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 (eXtensible Marckup Language) e/ou objetos Java.

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

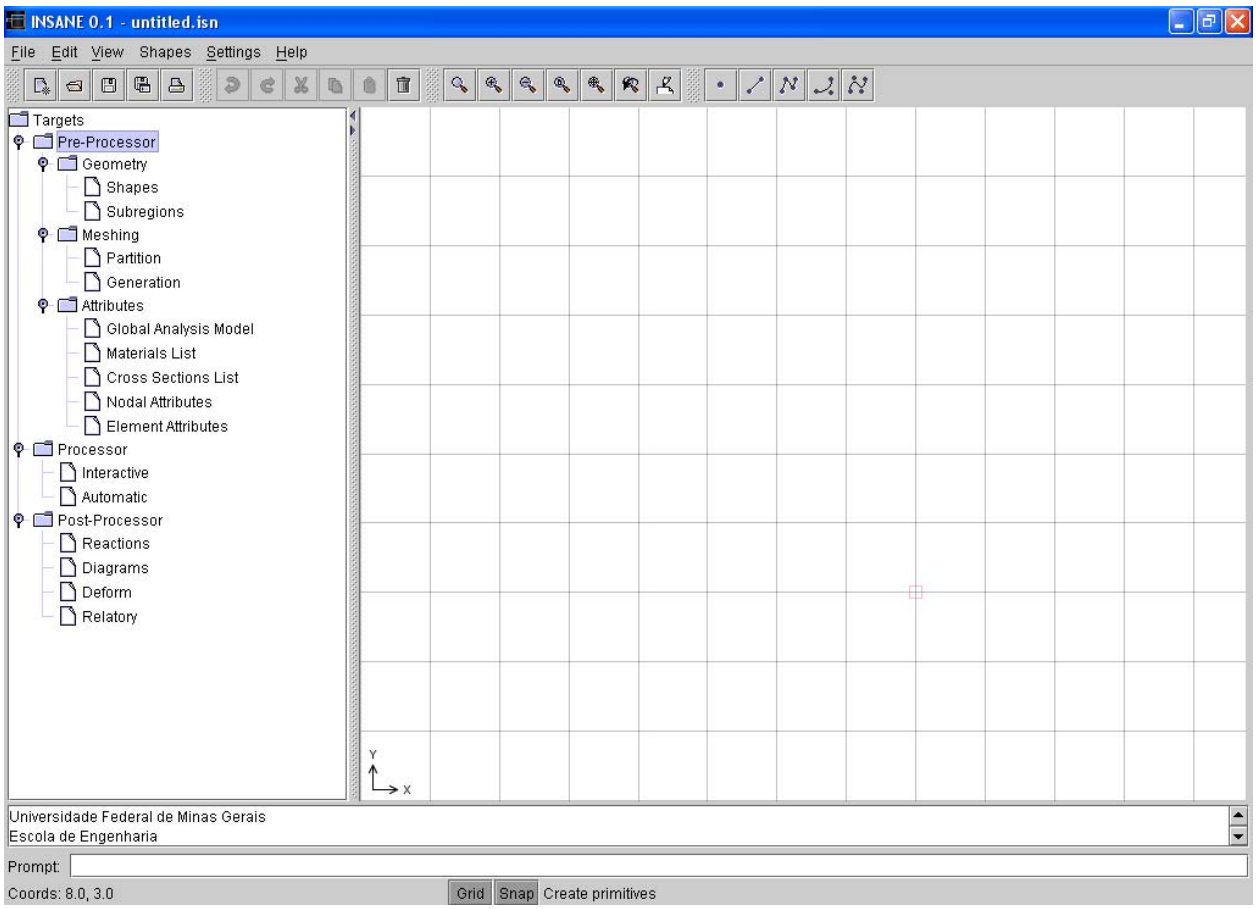


Figura 1.1: Interface Gráfica do `INSANE`

implementadas segundo POO, utilizam o padrão de projeto de software (Gamma, Helm, Johnson e Vlissides 1995*a*) 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*b*), 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 Access 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

O objetivo específico da dissertação de mestrado que aqui se apresenta é a implementação computacional, de uma aplicação gráfica interativa na qual possa-se acompanhar e visualizar todas as etapas da resolução de modelos discretos do Método dos Elementos Finitos, auxiliando assim, o ensino desta matéria.

1.3 Organização do Texto

Esta dissertação está apresentada em 7 capítulos.

No capítulo 2 discute-se diversos enfoques para a abordagem do MEF em cursos de engenharia, identificando suas generalidades e as possibilidades que o `INSANE` oferece para facilitar o processo de aprendizagem, apresentando a análise de requisitos da aplicação.

O capítulo 3 discute as tecnologias de desenvolvimento de *software* empregadas. Após apresentação dos principais conceitos do paradigma de programação orientada a objetos, discute-se os padrões de projetos de *software* adotados e justifica-se a escolha de Java como linguagem de implementação.

No capítulo 4 procede-se a análise do projeto orientado a objetos existente no `INSANE` e a expansão de classes da aplicação, para contemplar o processador interativo do MEF. Finalmente, as classes criadas e as associações entre elas são apresentadas utilizando diagramas *UML* (*Unified Modeling Language*) apropriados.

O capítulo 5 apresenta a interface gráfica da implementação do sistema através de um exemplo, que ilustra as possíveis interações entre o usuário e as principais etapas da solução de um modelo do MEF.

No capítulo 6 são apresentados vários exemplos de malhas de elementos finitos que utilizam os diversos recursos disponibilizados no sistema. Estes exemplos apresentam as principais etapas do processamento através do MEF.

Finalmente, no capítulo 7, os possíveis benefícios da aplicação para o processo de

aprendizagem do MEF são discutidos, enfatizando-se o sistema `INSANE` como fomentador do desenvolvimento de novos modelos discretos, evitando o recomeço do processo de implementação e permitindo maior agilidade e criatividade da pesquisa na área.

Capítulo 2

Ensino do MEF e Análise de Requisitos da Aplicação

O MEF é uma eficaz ferramenta numérica de resolução de problemas de meio contínuo. O método é eficientemente aplicado na análise de estruturas, área onde teve sua origem e mais se desenvolveu. Hoje não se imagina projetar estruturas inovadoras e arrojadas sem fazer uso do MEF. Este assunto, portanto, faz parte dos cursos de engenharia.

A literatura especializada no assunto é vasta, na sua maioria estrangeira, e complexa para os iniciantes. Assim, grande parte dos cursos voltados ao ensino do MEF estão em constante desenvolvimento e diversas abordagens têm sido aplicadas.

Partindo-se de uma discussão sobre estas abordagens, este capítulo apresenta a análise de requisitos da aplicação.

2.1 Análise Estrutural Através do MEF

A análise estrutural tem por objetivo determinar deslocamentos e tensões ao longo de uma estrutura em equilíbrio, submetida à ações externas (Pitangueira 2000).

Orientando-se por experiência de projeto, o problema de meio contínuo da estrutura real é substituído por um modelo matemático, utilizando-se hipóteses simplificadoras. Tal modelo matemático é expresso por equações diferenciais (ordinárias ou parciais) cujas soluções, ditas soluções analíticas, são conhecidas apenas para alguns poucos casos simples.

A solução analítica do modelo matemático do contínuo, apesar de aproximada em

relação ao problema físico real, é normalmente denominada solução exata.

Para evitar a resolução das equações diferenciais associadas às soluções analíticas e superar as limitações de tais soluções, adota-se um modelo numérico aproximado dito modelo discreto. Dentre os métodos discretos, o MEF é o mais difundido.

A análise estrutural via MEF pressupõe a divisão do domínio de estudo em subdomínios interconectados, denominados elementos finitos. Na formulação do MEF em deslocamentos, cada um dos elementos finitos tem uma função de aproximação de deslocamentos a ele associada. Nestes elementos também é necessário estabelecer hipóteses relativas ao regime de deformações (relações Deformações×Deslocamentos) e ao comportamento do material (relações Tensões×Deformações). Utilizando princípios físicos apropriados, estas três hipóteses são consideradas na obtenção das equações de equilíbrio nodais do elemento, gerando assim, sua matriz de rigidez.

Os vários elementos de uma discretização do MEF são interconectados entre si através de nós. A imposição das condições de equilíbrio a cada um dos nós da malha, considerando as equações de equilíbrio do elemento e a compatibilidade de deslocamentos nos nós, permite obter-se o sistema de equações de equilíbrio do modelo. Este sistema, inicialmente singular, pode ser resolvido após a imposição das condições de contorno.

Obtidos os deslocamentos nodais, pode-se reutilizar as três hipóteses fundamentais de qualquer um dos elementos finitos, para a obtenção de grandezas internas (deslocamentos, deformações e tensões).

É importante ressaltar que, desde que o campo de deslocamentos de cada elemento seja aproximado de forma adequada, a solução do modelo discreto se aproxima da solução analítica à medida que se aumenta o número de elementos da análise.

Entretanto deve-se atentar para a acurácia do modelo discreto, a adequabilidade do modelo matemático, e a definição do meio contínuo associado à estrutura real, antes de se utilizar os resultados da análise. A figura 2.1 ilustra esta questão (Soriano e Lima 1999).

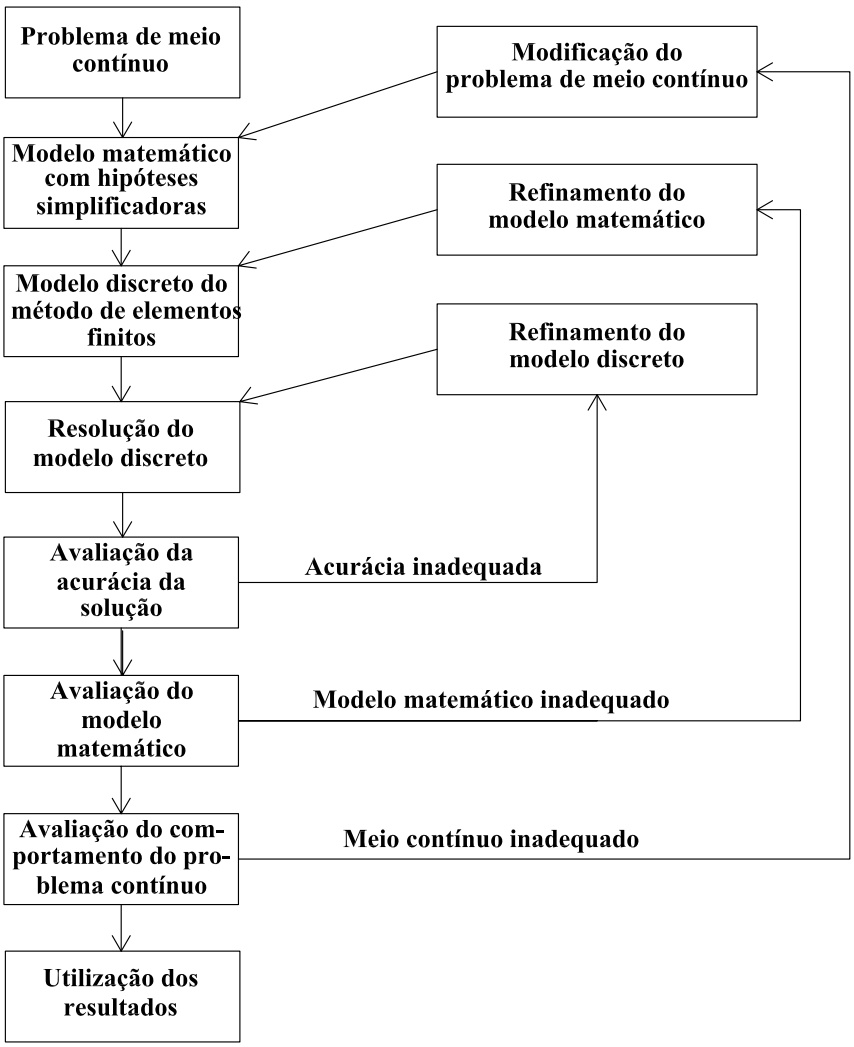


Figura 2.1: Estabelecimento e análise de um problema de meio contínuo com o MEF.

2.2 Modelos Discretos do MEF

O MEF surgiu como evolução da análise matricial dos modelos reticulados, com a disponibilidade de computadores digitais e devido à necessidade de se projetar estruturas de modelos contínuos.

Qualquer que seja a causa de forças e deformações internas em uma estrutura, três condições básicas precisam ser observadas:

i. o equilíbrio de forças;

ii. a compatibilidade dos deslocamentos; e

iii. as relações constitutivas do material.

A primeira condição simplesmente exige que as forças internas estejam em equilíbrio com as cargas externas aplicadas. Embora o uso de apenas esta condição seja suficiente para resolver um problema estaticamente determinado, as condições de compatibilidade e as leis do comportamento do material sendo então automaticamente satisfeitas, para estruturas hiperestáticas ela não fornece suficiente informação que possibilite fazer-se uma análise completa.

Nestas circunstâncias a condição de compatibilidade precisa ser invocada separadamente. A compatibilidade requer que a estrutura deformada ajuste-se em conjunto adequadamente, isto é que as deformações dos elementos sejam compatíveis.

Para isto é necessário conhecer a relação entre a carga e a deformação para cada componente da estrutura. Esta relação, em problemas de elasticidade linear, é descrita pela Lei de Hooke.

Definidos os parâmetros da malha, e escolhendo-se um sistema de coordenadas adequado, a generalização da análise pode ser descrita analiticamente, através de alguns passos básicos, ilustrados abaixo, para o caso de análise bidimensional de tensões.

1. Escolher uma função deslocamento:

$$\{d(x,y)\} = [f(x,y)]\{a\} \tag{2.2.1}$$

onde $\{f(x,y)\}$ representa os termos do polinômio aproximador e $\{a\}$ os coeficientes do mesmo.

2. Obter o estado de deslocamentos em um ponto qualquer em termos dos deslocamentos nodais $\{d\}$:

$$\{d(x,y)\} = [f(x,y)][A]^{-1}\{d\} \tag{2.2.2}$$

onde as linhas de $[A]$ são determinadas através do valor de $\{f(x,y)\}$ em cada ponto nodal.

3. Relacionar as deformações em um ponto qualquer a $\{d(x,y)\}$ e, conseqüentemente, a $\{d^e\}$, os deslocamentos do elemento finito:

$$\{\varepsilon(x,y)\} = [B]\{d^e\} \tag{2.2.3}$$

onde $[B]$ define as relações diferenciais entre deformações e deslocamentos.

4. Relacionar as tensões em um ponto qualquer a $\{\varepsilon(x,y)\}$ e, conseqüentemente, a $\{d^e\}$:

$$\{\sigma(x,y)\} = [D][B]\{d^e\} \tag{2.2.4}$$

onde $[D]$ contém as propriedades do material.

5. Substituir $\{\sigma(x,y)\}$ por cargas nodais equivalentes $\{f^e\}$ relacionando, portanto, $\{f^e\}$ a $\{d^e\}$:

$$\{f^e\} = \int_{Ve} [B]^T[D][B]\, dVe\{d^e\} \tag{2.2.5}$$

Estabelecendo a relação de equilíbrio do elemento

$$\{f^e\} = [k^e]\{d^e\} \tag{2.2.6}$$

vê-se que:

$$[k^e] = \int_{Ve} [B]^T [D][B] \, dVe \tag{2.2.7}$$

A partir destas equações algébricas, que podem ser obtidas através das formulações direta (apresentada acima), variacional ou de resíduos, que regem o comportamento aproximado de cada um dos sub-domínios (elementos finitos), monta-se o sistema de equações da malha de elementos como um todo, denominado *sistema global*, que juntamente com as condições de contorno ainda não atendidas ao se arbitrar o campo das variáveis nos sub-domínios, permite a determinação dos valores nodais de definição desse campo. Pode-se, então, retornar à análise de cada elemento isoladamente para determinação de incógnitas em qualquer um de seus pontos.

A figura 2.2 mostra o inter-relacionamento dos métodos de resolução (formulações direta, variacional e de resíduos) da análise via MEF.

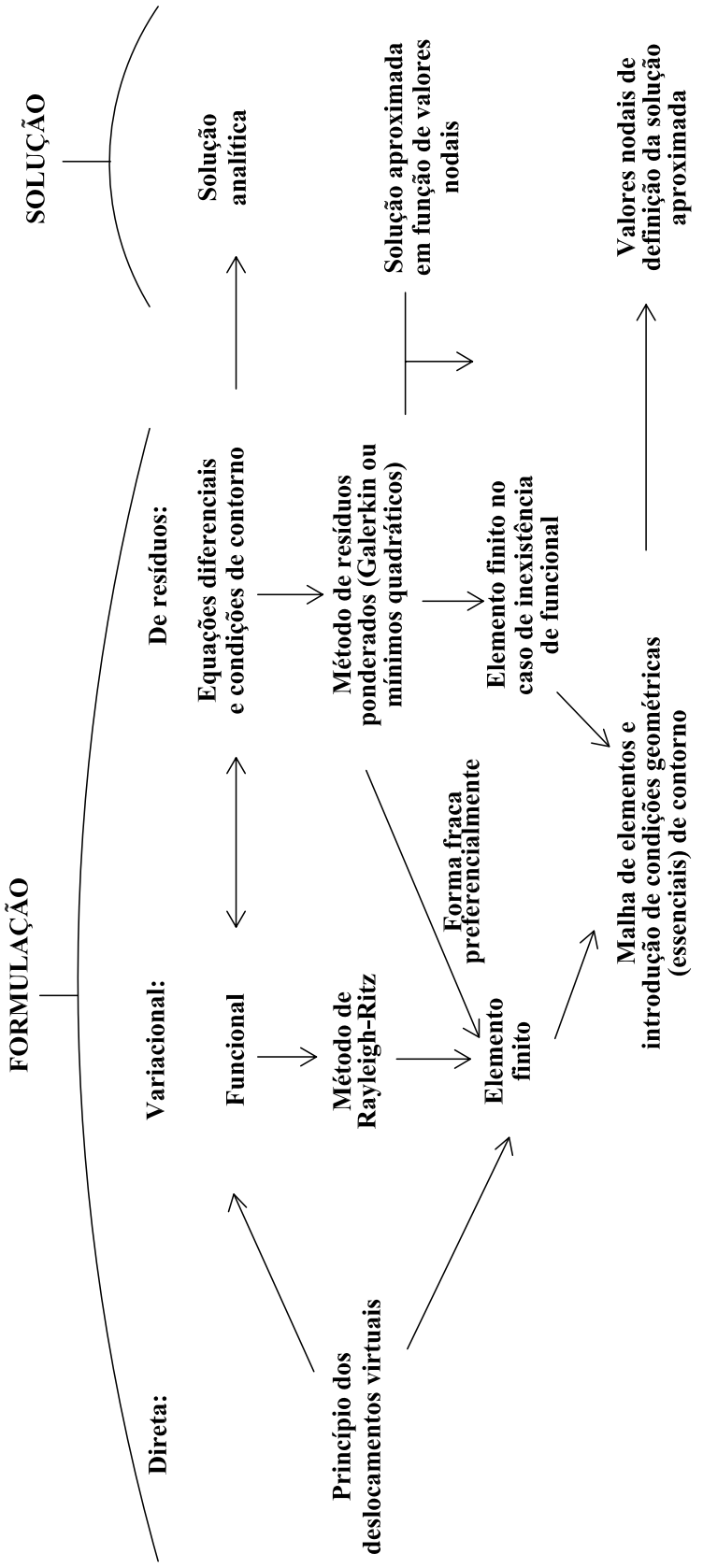


Figura 2.2: Inter-relacionamento das formulações direta, variacional e de resíduos do MEF.

2.3 Ensino na Graduação

O ensino do MEF nos cursos de graduação na área de engenharia estrutural ainda é recente e se encontra em desenvolvimento. Em geral, começa-se por ensinar modelos unidimensionais para vigas, pórticos, treliças e grelhas, genericamente designados modelos reticulados, por serem representativos de peças cuja seção transversal apresenta dimensões muito inferiores ao comprimento.

Adota-se, geralmente, a formulação direta para a apresentação do método, tomando-se como ponto de partida a análise estrutural clássica do Método dos Deslocamentos.

Ainda na graduação, são estudados modelos estruturais bidimensionais, através de elementos finitos cuja geometria é simples (triangular e retangular). Nesta fase, não se aborda a formulação paramétrica do MEF, visto que a mesma exige conceitos aprofundados, que usualmente são estudados na pós-graduação.

Ainda que seja de grande importância, o ensino do MEF na graduação não é habitual, sendo realizado em poucas instituições de ensino de engenharia no Brasil.

2.4 Ensino na Pós-Graduação

Nos cursos de pós-graduação são adotados diferentes enfoques para o ensino do MEF. Alguns focos podem ser escolhidos na abordagem de um curso de pós-graduação (Soriano e Lima 1999), a saber:

1. Apresentação do método através de seus fundamentos e formulações.

   Esta abordagem conduz à formulação analítica do modelo de deslocamentos do MEF, anteriormente ilustrado na figura 2.2.

2. Orientação ao uso de sistemas computacionais com implementação do método.

   Este enfoque prioriza o ensino da utilização de softwares relacionados ao MEF.

3. Desenvolvimento de programas automáticos.

   Esta abordagem visa formar os alunos para o desenvolvimento de programas de aplicação do MEF.

2.5 Análise de Requisitos da Aplicação

Dentro das diversas perspectivas de abordagem do MEF nos cursos de pós-graduação, citadas acima, o programa `INSANE` pode ser empregado como eficiente complementação dos recursos didáticos.

No caso da abordagem analítica (1), atualmente o programa possibilita apenas a verificação dos resultados obtidos analiticamente e a interpretação das variações do resultados gerados a partir de possíveis alterações no modelo discreto.

Para o enfoque de usuários de programas (2), o `INSANE` é uma ferramenta didática bastante útil, por possuir uma interface gráfica "amigável", de simples compreensão.

Por se tratar de um programa de código aberto, o `INSANE` possibilita que alunos interessados em desenvolver programas de aplicação do MEF participem de sua implementação (3).

Com este trabalho pretende-se ampliar os recursos didáticos do `INSANE`, de maneira que o mesmo possa atender todo desenvolvimento de um curso do MEF.

Partindo-se deste princípio, e sabendo-se que as possibilidades para o ensino do MEF são bastante variadas e que os cursos de graduação e pós-graduação têm particularidades inevitáveis, buscou-se uma sugestão de programa para um curso de elementos finitos em nível de pós-graduação, que reunisse grande parte dos conceitos relacionados à análise estrutural através do MEF, de maneira a auxiliar na análise de requisitos da aplicação.

Assim, apresenta-se a seguir o atual roteiro do curso do MEF do Programa de Pós-graduação em Engenharia de Estruturas da UFMG.

1. Introdução
   (a) Análise Estrutural através do MEF;
   (b) Bases do MEF;
   (c) História do MEF;
2. Elementos Finitos Unidimensionais
   (a) Formulação do MEF segundo o Princípio dos Trabalhos Virtuais;

(b) Elementos Finitos de treliça, viga, pórtico e grelha;

(c) Transformação de Eixos;

(d) Método da Rigidez Direta;

(e) Critérios de Convergência;

3. Elementos Finitos Planos Lineares da Elasticidade

(a) Tensões e Deformações em duas Dimensões;

(b) Elementos Triangulares;

(c) Elementos Retangulares;

4. Formulação Isoparamétrica do MEF

(a) Coordenadas Naturais;

(b) Integração Numérica;

(c) Elementos Quadrilaterais;

(d) Elementos Triangulares;

(e) Convergência de Elementos Paramétricos;

5. Aplicações Diversas

(a) Sólidos Axissimétricos;

(b) Flexão de Placas;

6. Cálculo Variacional

(a) Métodos Aproximados de Minimização de Funcionais;

(b) Métodos Aproximados de Resíduos Ponderados;

(c) Formulações Variacionais do MEF;

Desta forma estabeleceu-se um paralelo entre a sugestão de curso descrita anteriormente e os requisitos de um ”software” educativo, conforme descrito na tabela 2.1.

Tabela 2.1: Requisitos da Aplicação

Item	Requisitos da Aplicação
1	Utilização do ”software” sem preocupações conceituais
2	Apresentação do MEF como uma generalização da análise estrutural clássica
3	Apresentação dos diversos conceitos da formulação clássica
4	Apresentação dos diversos conceitos da formulação isoparamétrica
5	Aplicações diversas com ponderação conceitual
6	Associação dos resultados obtidos com a formulação variacional

A versão inicial do `INSANE` já atendia às expectativas referentes aos ítens 1, 2 e 6. A expansão, portanto, contemplou os ítens 3, 4 e 5, que compreendem a etapa de solução de um problema do MEF.

2.6 Proposta da Expansão

A proposta desta expansão foi baseada na sugestão feita por Logan (2001) de resolução de problemas do MEF, mostrada na figura 2.3.

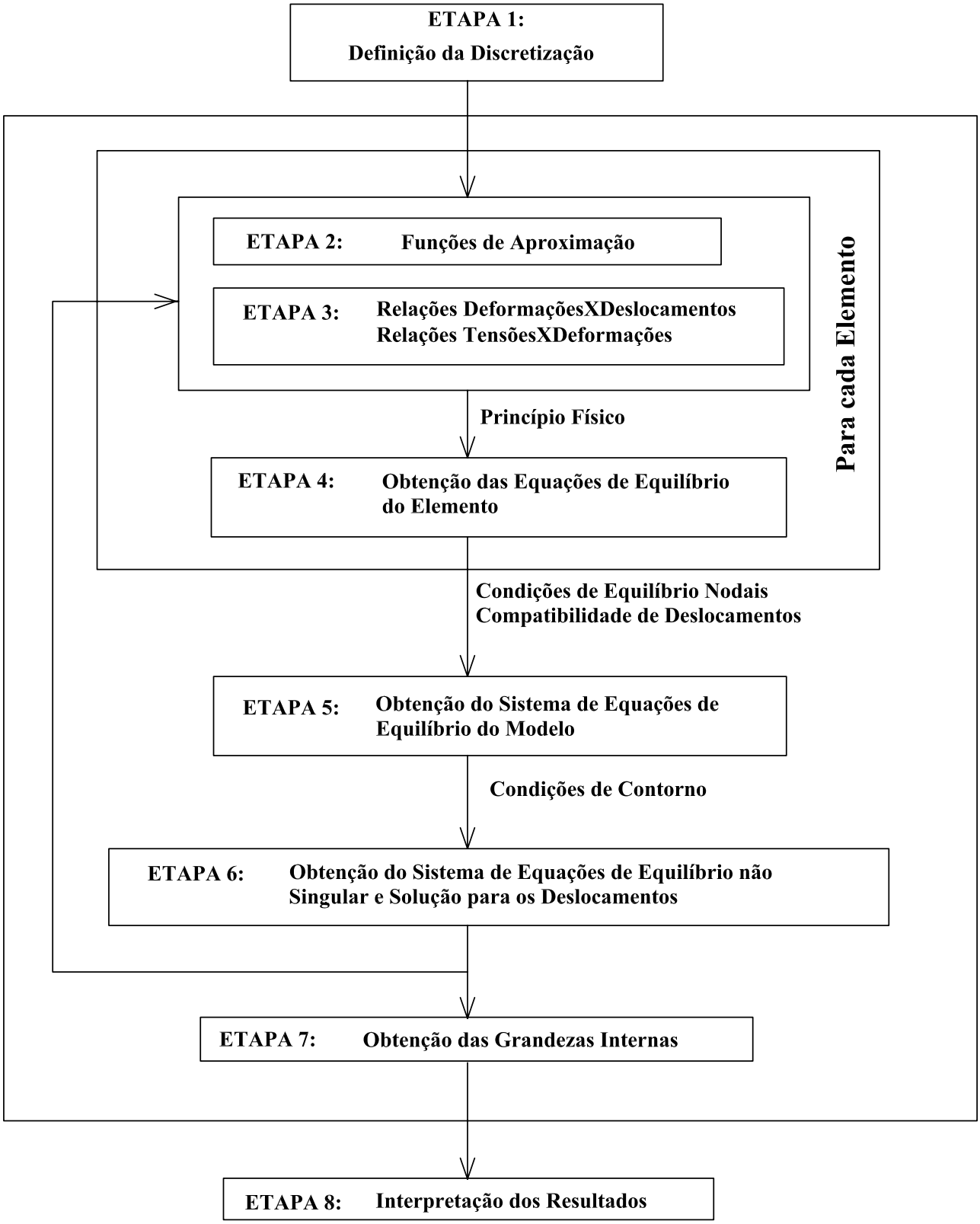


Figura 2.3: Etapas de análise através do MEF.

As etapas mostradas na figura 2.3 podem ser agrupadas em três segmentos, assim como a análise via MEF se caracteriza no projeto `INSANE`:

i. Pré-processador (Etapa 1)

ii. Processador (Etapas 2 a 7)

iii. Pós-processador (Etapa 8)

Os pré e pós-processadores (etapas 1 e 8) são aplicações gráficas interativas, que disponibilizam recursos diversos para diferentes modelos discretos. O processador (etapas 2 a 7) é uma aplicação, que representa o núcleo numérico do sistema. Este núcleo é responsável pela obtenção dos resultados de diferentes modelos discretos de análise estrutural.

As etapas 1 e 8, caracterizadas pelo pré-processador e pós-processador, não foram alteradas, uma vez que já funcionavam atendendo às expectativas didáticas. Já as etapas de 2 a 7 foram apresentadas detalhadamente ao usuário, de maneira que a compreensão da solução do problema fosse facilitada.

A tabela 2.2 mostra os novos recursos do programa, estabelecidos de acordo com as etapas do processamento.

Tabela 2.2: Interação com o usuário nas etapas de análise via MEF

Etapa	**Interação com o Usuário**
Etapa 2	Apresentação da matriz das funções de forma do elemento Apresentação dos deslocamentos referentes a um dado ponto do elemento a partir dos deslocamentos nodais Apresentação da teoria referente ao assunto
Etapa 3	Apresentação da matriz B, referente às relações de Deformações X Deslocamentos Apresentação das deformações do elemento a partir dos deslocamentos nodais Apresentação da matriz D, referente às relações de Tensões X Deformações Apresentação das tensões no elemento a partir das deformações obtidas Apresentação da teoria referente ao assunto
Etapa 4	Apresentação do equilíbrio de cada elemento da discretização Apresentação da matriz de rigidez completa de cada elemento Apresentação da matriz de rigidez reduzida de cada elemento Apresentação da matriz de forças nodais equivalentes completa de cada elemento Apresentação da matriz de forças nodais equivalentes reduzida de cada elemento Apresentação da teoria referente ao assunto
Etapa 5	Apresentação da matriz de rigidez do modelo através da contribuição de cada elemento Apresentação da matriz de forças do modelo através da contribuição de cada elemento Apresentação da matriz de forças nodais do modelo Apresentação do equilíbrio do modelo Apresentação da teoria referente ao assunto
Etapa 6	Apresentação da solução do sistema Apresentação da teoria referente ao assunto
Etapa 7	Apresentação das grandezas internas dos elementos

Capítulo 3

Recursos Utilizados no Desenvolvimento da Aplicação

Este capítulo apresenta os principais recursos utilizados no desenvolvimento da aplicação. Paradigma de Programação Orientado a Objetos (POO), Padrões de Projeto de Software, Linguagem Java, Persistência de Dados e Linguagem de Representação Gráfica para POO são discutidos, justificando-se a escolha dos mesmos.

3.1 Paradigma de Programação Orientada a Objetos

Muitas aplicações para as quais deseja-se 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.

3.1.1 Coleções de Objetos

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 e 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 e Figueiredo 2003).

3.1.3 Abstração

Abstrair, no contexto da POO, 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 e 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 esta interface gráfica com o 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 e Tamassia 2002).

3.1.5 Modularidade

Além da 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 e 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 e 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 com 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 e Tamassia 2002).

3.2 Padrões de Projeto de *Software*

Padrões de projeto podem ser definidos como a estrutura básica de um projeto de *software* bem-sucedido, capaz de fornecer um esquema para os subsistemas ou componentes de um sistema de *software* a ser projetado. Esse esquema deve ser específico para resolver o problema em questão e suficientemente genérico para atender a futuros problemas e requisitos. A utilização de Padrões de Projeto propicia evitar ou minimizar o re-projeto.

A utilização de Padrões de Projeto (Gamma, Helm, Johnson e Vlissides 1995*b*) em sistemas orientados a objetos torna estes sistemas mais flexíveis e reutilizáveis. Os Padrões de Projeto ajudam os projetistas de *software* a reutilizar bons projetos ao basear os novos projetos em experiências anteriores. O uso de determinado Padrão de Projeto garante que uma grande quantidade de decisões de projeto decorra automaticamente, porque o sistema em desenvolvimento será baseado em um sistema que está em funcionamento.

Entre as vantagens de se utilizar padrões de projeto no desenvolvimento de *software* pode-se citar: aumento de produtividade, uniformidade na estrutura do *software*, incremento da padronização no desenvolvimento de *software*, aplicação imediata por outros desenvolvedores, redução da complexidade do sistema (Pietro 2001*a*).

3.2.1 Padrão *Model-View-Controller*

Visando separar o modelo de sua representação, a implementação do `INSANE` é baseada no padrão de projeto *Model-View-Controller (MVC)*. A utilização desta metáfora de programação permite que o controle de uma alteração, através de interação com o usuário, e a visualização da mesma sejam implementados independentemente do modelo adotado, minimizando as tarefas de manutenção e expansão da aplicação. A implementação segundo o padrão *MVC* permite o aperfeiçoamento gradual da aplicação através de mudança de plataforma, criação de diversas vistas sincronizadas com o modelo, substituição ou atualização das diversas vistas e disponibilização "on-line" do sistema.

Existe um ciclo de vida para cada uma das atividades executadas pelo programa. Este ciclo permite que o usuário faça alterações no modelo e visualize o resultado a

cada alteração, até que consiga o resultado desejado. O referido ciclo compõe-se de: especificação do usuário, atualização do modelo e visualização.

O padrão de projeto *Model-View-Controller* pode ser usado para a implementação do ciclo de vida de cada atividade. Este padrão divide a aplicação em três componentes: `modelo`, `vista` e `controlador`. A Figura 3.1 ilustra o padrão.

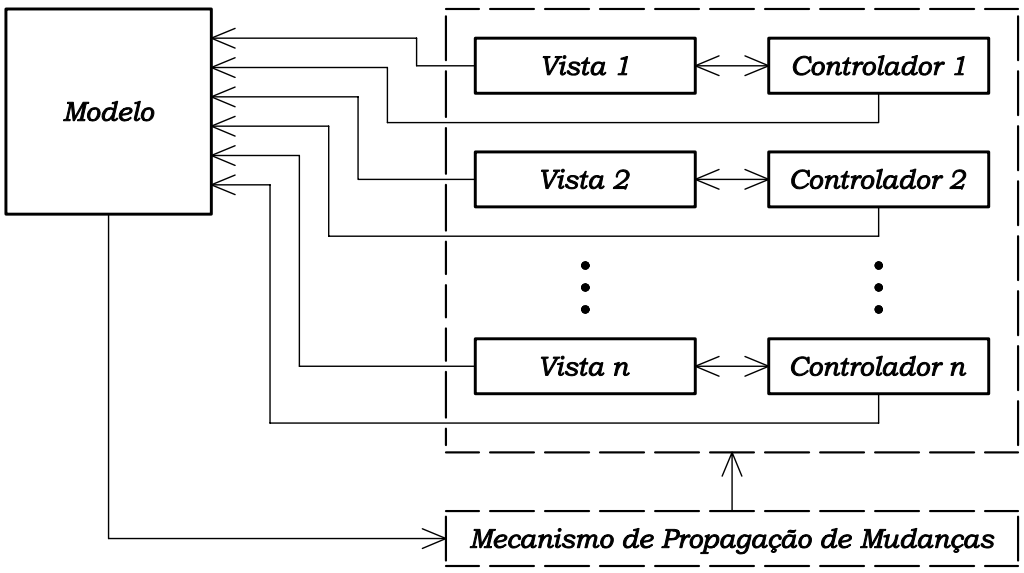


Figura 3.1: Componentes do padrão `MVC`

O `modelo` contém o núcleo dos dados e da funcionalidade do sistema, sendo independente das saídas e entradas de dados. A `vista` apresenta para o usuário as informações armazenadas no `modelo`. Cada `controlador` é associado a um componente `vista`, sendo o responsável pela percepção das entradas do usuário e tradução das mesmas em requisições de serviços para os componentes `modelo` e `vista`. Todas as requisições dos usuários devem ser feitas através dos controladores (Buschmann, Meunier, Rohnert, Sommerlad e Stal 1995).

Existe um mecanismo de propagação de mudanças que garante a consistência e a comunicação entre os componentes `controlador` e `vista` com o componente `modelo`. No momento em que um componente `controlador` ou `vista` é criado, o mecanismo de propagação de mudanças efetua seu registro, ligando-o ao `modelo` do qual ele é dependente. Os componentes `vista` e `controlador` dependem desse registro para serem informados das atualizações do `modelo`.

O mecanismo de propagação de mudanças é disparado a cada mudança de estado do

`modelo`, acarretando na execução do procedimento de atualização do componente `vista`, que exibe ao usuário a informação atualizada. Cada `vista` é associada a um único `controlador` e fornece a este a funcionalidade necessária para manipular a exibição de dados.

3.2.2 Padrão `Command`

O uso do padrão `Command` na implementação do `INSANE` permite o encapsulamento de rotinas de execução em objetos, a associação destes objetos a elementos de interface gráfica com o usuário (GUI) e dispositivos de entrada (teclado e mouse), a execução de uma mesma rotina disparada por diferentes elementos de GUI e possibilita um incremento na modularidade de seu código. O encapsulamento das rotinas de execução possibilita também que a realização de alterações nas mesmas não provoque modificações nas classes existentes.

O padrão `Command` baseia-se em uma classe abstrata de mesmo nome, a qual declara uma interface para execução de operações. Na sua forma mais simples, esta interface inclui uma operação abstrata `execute()`. As subclasses concretas de `Command` especificam um par receptor-ação através do armazenamento do receptor como uma variável de instância e pela implementação de `execute()`, para invocar a solicitação. O receptor tem o conhecimento necessário para poder executar a solicitação.

Este padrão desacopla o objeto que invoca a operação daquele que tem o conhecimento para executá-la. Isto proporciona grande flexibilidade no projeto da interface de usuário. Uma aplicação pode oferecer tanto uma interface gráfica com *menus* como uma interface gráfica com *botões* para algum recurso seu, simplesmente fazendo com que o *menu* e o *botão* compartilhem uma instância da mesma classe que implementa `Command`. O padrão `Command` possibilita a substituição dinâmica de comandos, o que é muito útil para interfaces gráficas sensíveis ao contexto. Pode-se ainda concatenar comandos para compor comandos maiores, propiciando a redução da complexidade destes. Todos estes recursos são possíveis porque o objeto que emite a solicitação não precisa conhecer a execução da solicitação.

O tempo de vida de um objeto `Command` é independente de sua solicitação original, permitindo armazenar comandos e executar suas rotinas em momentos distintos. A operação `execute()`, de `Command`, pode armazenar estados para que o comando possa reverter seus efeitos. Para suportar a operação desfazer a interface de `Command` deve acrescentar a operação `undo()`, que reverte os efeitos de uma chamada anterior de `execute()`. Os comandos executados devem ser armazenados em uma lista histórica. O nível ilimitado de desfazer e refazer operações é obtido percorrendo esta lista para trás e para frente, chamando operações `undo()` e `execute()`, respectivamente (Gamma et al. 1995*b*).

A figura 3.2 apresenta os componentes envolvidos com o padrão `Command` e ilustra o

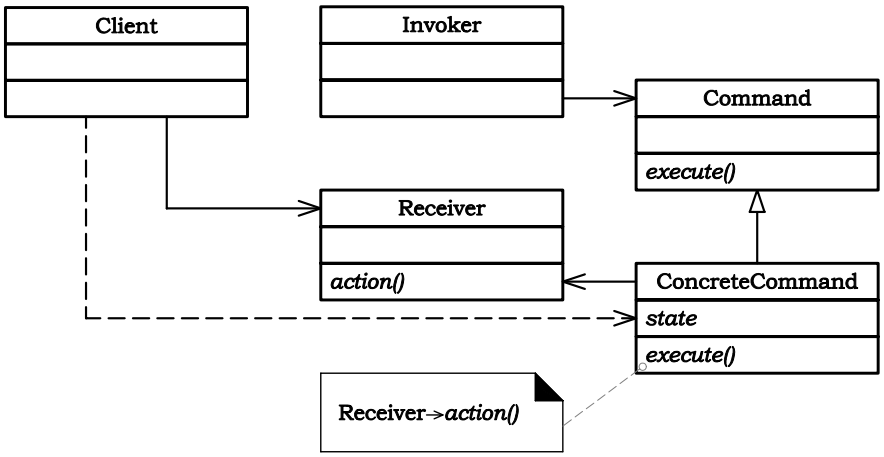


Figura 3.2: Estrutura do padrão *Command* (Gamma et al. 1995)

relacionamento entre eles. `Receiver` é o componente que sabe como executar as operações associadas a uma solicitação, qualquer classe pode funcionar como um `Receiver`. A classe `Command` declara uma interface para a execução de uma operação. `ConcreteCommand` implementa o processo `execute()` através da invocação das operações correspondentes no `Receiver` e define uma vinculação entre um objeto `Receiver` e uma ação. Quando os comandos podem ser desfeitos, `ConcreteCommand` armazena estados para desfazer o comando antes de invocar `execute()`. O componente `Invoker` é responsável por disparar o processo `execute` de seu comando associado (Gamma et al. 1995*b*). O componente `Client` cria um objeto `ConcreteCommand`, o associa a um `Invoker` e estabelece o seu receptor (`Receiver`).

3.3 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 do `INSANE`. Estes aspectos são: independência de sistema operacional; performance numérica; capacidade de reutilização do software e suporte à persistência dos dados.

3.3.1 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 e Figueiredo 2003).

3.3.2 Comparação de Performance entre Java e C++

Nikishkov e Savchenko (2003) compararam 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 3.3 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 3.4 mostra os resultados obtidos para montagem da matriz de rigidez esparsa. Novamente JVM 1.2 mostra-se mais eficiente do que o compilador C++.

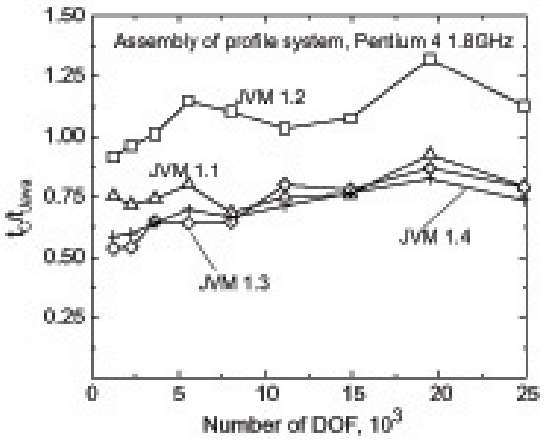


Figura 3.3: Eficiência para cálculo da matriz de rigidez

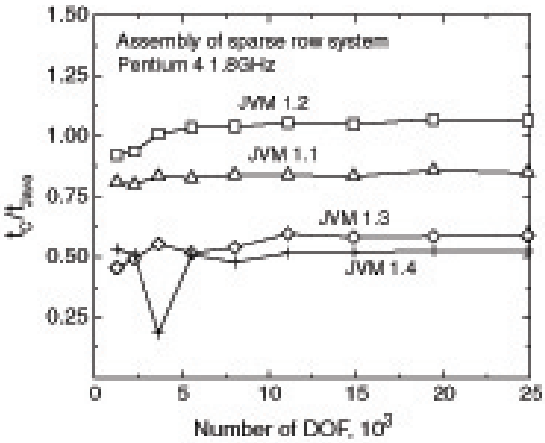


Figura 3.4: Eficiência para montagem da matriz de rigidez esparsa

3.3.3 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 e 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 e Deitel 2001).

3.4 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 e 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).

3.5 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 3.5 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.

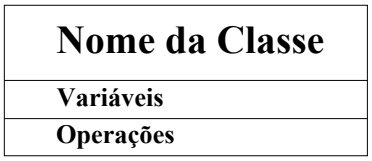


Figura 3.5: Diagrama de classe na UML

A figura 3.6 mostra um diagrama de herança, no qual pode-se visualizar duas subclasses derivando da superclasse.

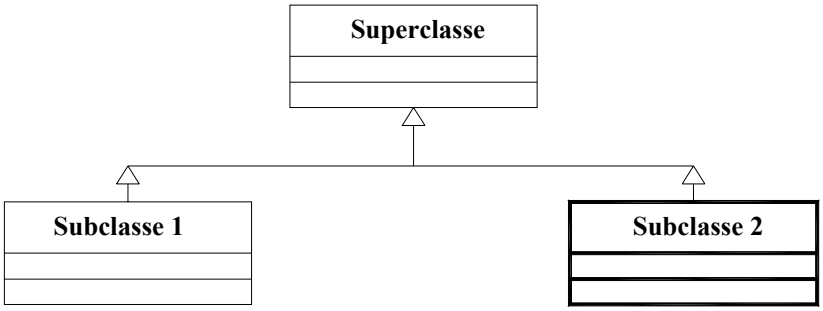


Figura 3.6: Diagrama de herança UML

A figura 3.7 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:

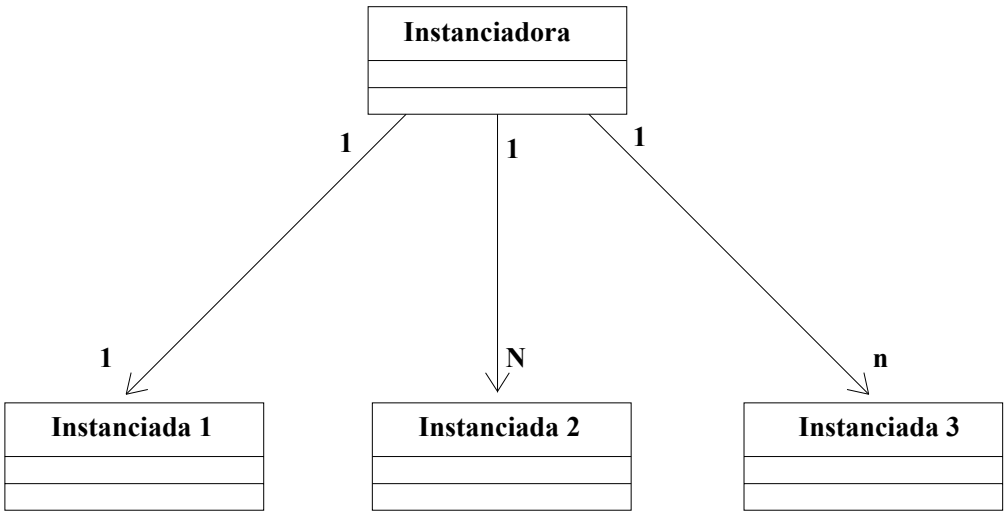


Figura 3.7: Diagrama de instâncias na UML

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

Projeto Orientado a Objetos da Aplicação

O processo de desenvolvimento de *software* orientado a objetos compreende três fases principais. Inicialmente, na fase de *análise*, procura-se enfatizar a descoberta e descrição dos objetos - ou conceitos - do domínio do problema. Em seguida, na fase de *projeto*, procura-se definir os elementos lógicos de *software*, seus atributos e métodos. Finalmente, durante a fase de *construção*, os componentes do projeto serão implementados em uma linguagem de programação que suporte o paradigma da progamação orientada a objetos.

Este capítulo apresenta o projeto orientado a objetos da expansão do `INSANE` para contemplar o processador interativo do MEF.

4.1 Arquitetura em Camadas e Padrões de Projetos de *Software*

Visando separar o modelo de sua representação, a implementação do `INSANE` é baseada no padrão de projeto *Model-View-Controller*. Suas principais vantagens consistem em facilitar a manutenção e a expansão da aplicação, permitindo a inclusão de novas telas, alteração na seqüência de telas, criação de caminhos alternativos de navegação em uma aplicação etc.

O *Model-View-Controller* (*MVC*) orienta a organização do código, definindo responsabilidades para cada componente da aplicação. É muito comum confundir o *MVC* com o que se convencionou chamar de *Programação em Três Camadas*, que propõe uma divisão da aplicação em componentes de *Apresentação*, *Negócios* e *Persistência*. Fazendo um

paralelo entre estes dois padrões de projeto, o componente *Modelo* do *MVC* abrange as camadas de *Negócio* e *Persistência*, e a camada de *Apresentação* incorpora os componentes *Vista* e *Controlador* do *MVC* (figura 4.1).

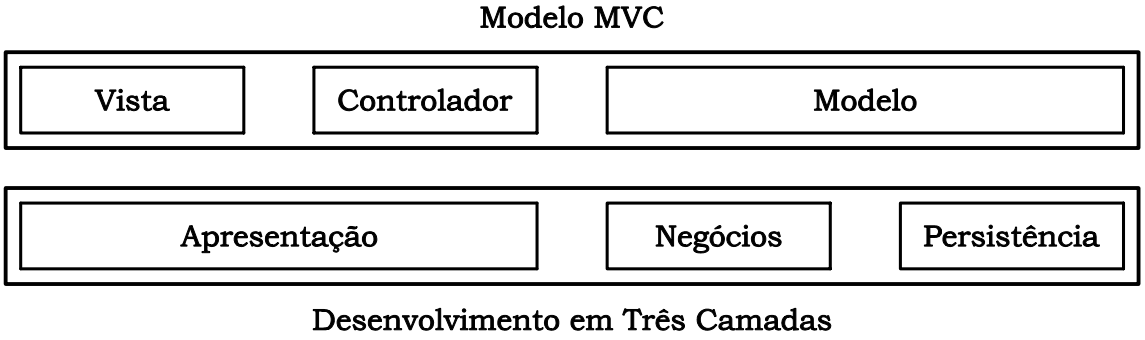


Figura 4.1: Componentes dos padrões MVC e Três Camadas

O uso dos padrões MVC e a Programação em Três Camadas é vantajoso na maioria das aplicações. A fusão dos dois padrões gera uma *Programação em Quatro Camadas*: *vista*, *controlador*, *negócios* e *persistência* (figura 4.2). Essa divisão em camadas deve nortear a forma como se constrói o código (Lozano 2004).

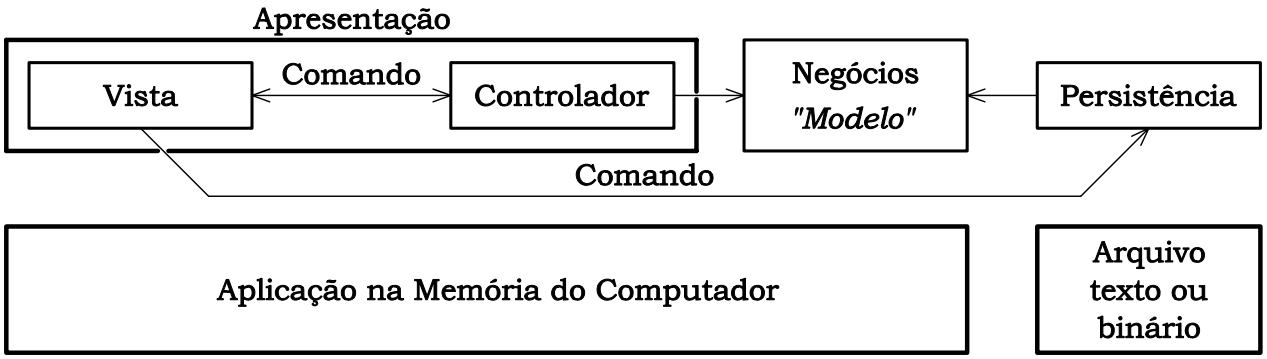


Figura 4.2: Programação em Quatro Camadas

Na parte superior da figura 4.2 pode-se ver as quatro camadas lógicas da aplicação. A camada de *Negócios*, objetos de negócio, representa entidades tangíveis, em uma aplicação, as quais os usuários podem criar, acessar e manipular. Os objetos de negócio possuem tipicamente estado, são persistentes e tem vida longa. Eles contém dados e modelam o comportamento do negócio (Gupta 2004). Optou-se denominar a camada de *Negócios* por *Modelo*, mais adequado neste caso. Na parte inferior da figura 4.2 observa-se as camadas físicas (somente duas, nesta versão da aplicação): arquivos XML ou objetos

Java persistidos em disco e toda a lógica da aplicação na memória do computador.

O inter-relacionamento entre as camadas é conseguido, principalmente, através da implementação do padrão de projeto de *software* denominado *Comando* (Grand 1998), já discutido no item 3.2.2.

A figura 4.3 exemplifica este relacionamento para o caso da tarefa de adição de uma entidade geométrica ao modelo corrente e sua visualização. Como pode ser visto na figura, o fluxo de informações para a realização de tal tarefa ocorre em quatro etapas. Na

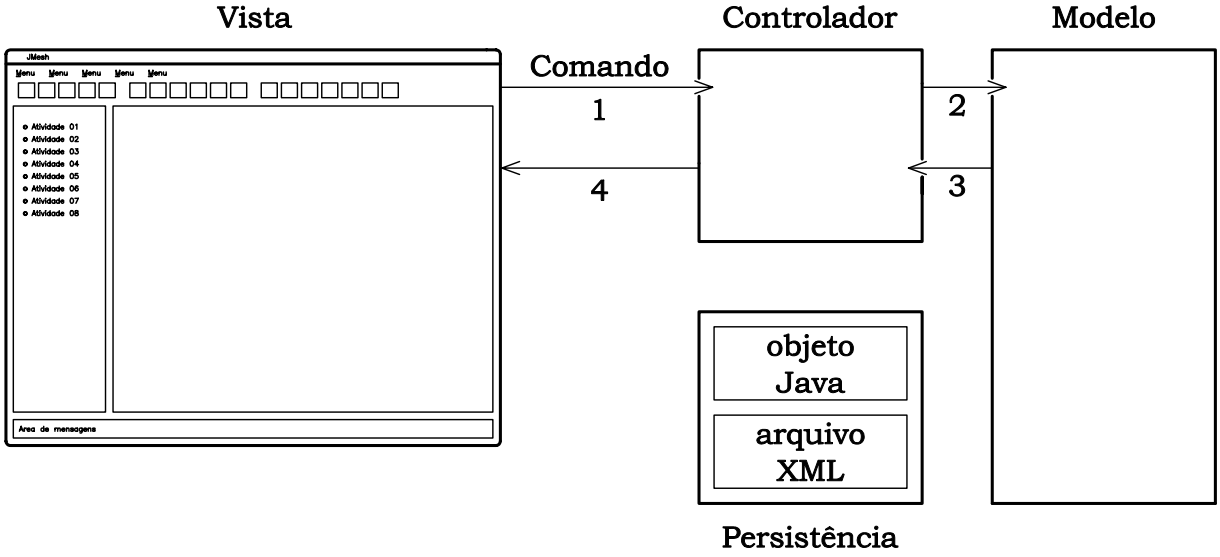


Figura 4.3: Relacionamento entre camadas para adição de uma entidade geométrica

primeira etapa, o objeto `Comando`, responsável pela tarefa, aciona o `Controlador` ativo informando a requisição. A seguir (2), o `Controlador` cria o objeto correspondente à entidade geométrica e o adiciona ao `Modelo` pertinente. Na etapa 3, o `Controlador` cria objetos de desenho representativos dos objetos do `Modelo`. Finalmente, na etapa 4, os objetos de desenho pertencentes ao `Controlador` são apresentados na área de desenho da `Vista`.

4.2 Projeto Orientado a Objetos Existente

A combinação dos padrões de projeto MVC e Command foi implementada nos trabalhos de Gonçalves (2004) , Fonseca (2004) e (2004) e Almeida (2005). Gonçalves (2004) e Fonseca (2004) trataram em seus trabalhos da interface gráfica com o usuário

nas etapas de pré e pós-processamento, enquanto Almeida (2005) e Fonseca (2004) se encarregaram do processamento através de modelos do MEF.

4.2.1 Interface Gráfica como Usuário

As classes responsáveis pelas camadas de apresentação (camadas vista e controlador na figura 4.2), bem como as classes que implementam as interações entre estas camadas, estão mostradas na figura 4.4.

A classe `Interface` (figura 4.4) possui um objeto da classe `Model`, um da classe

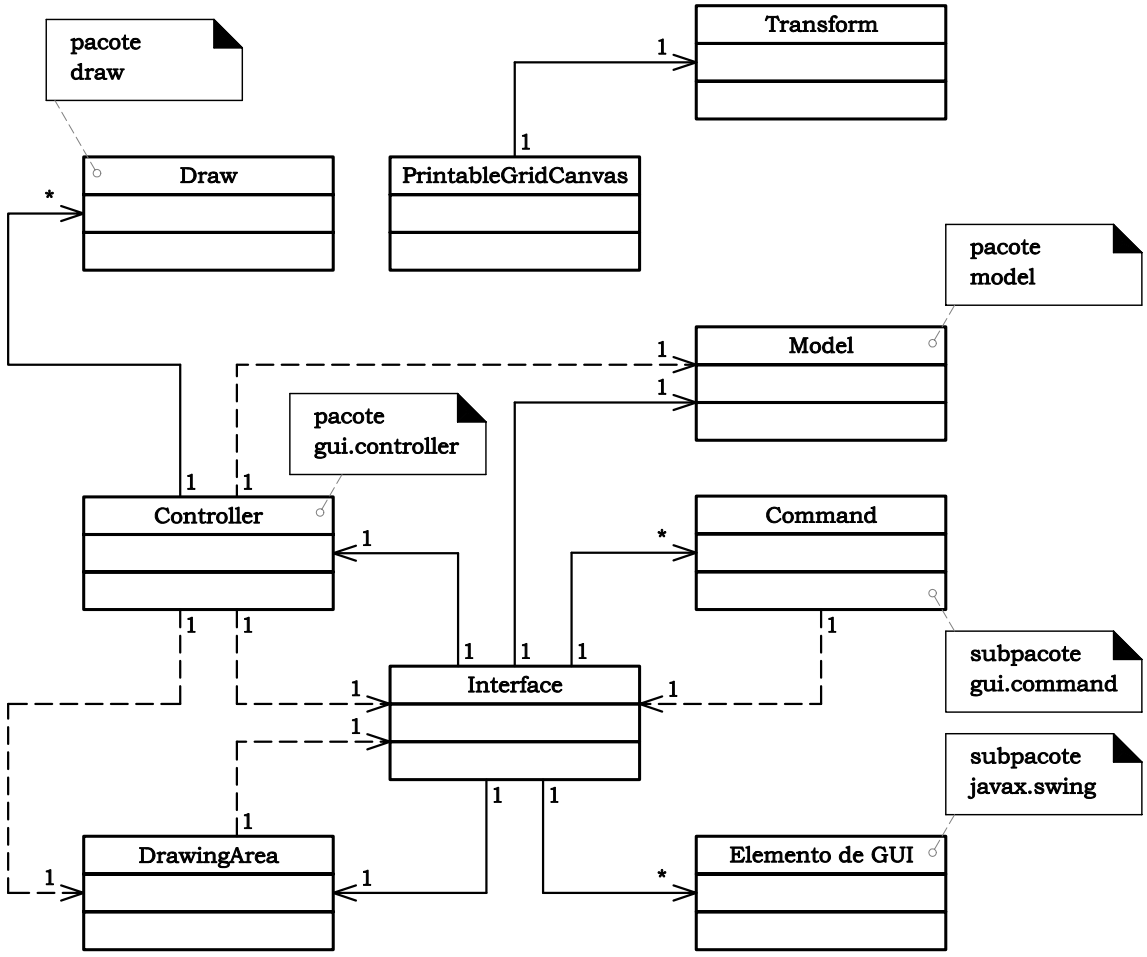


Figura 4.4: Diagrama de instâncias das classes do pacote `gui`

`Controller`, um da classe `DrawingArea`, vários objetos da classe `Command` e vários objetos que compõem a interface gráfica com o usuário (GUI - "Graphical User Interface").

Os objetos `DrawingArea` e `Elemento de GUI` (figura 4.4) são intâncias de classes da *API Java Swing* (Horstmann e Cornell 2001*b*). O *Swing*, que faz parte da biblioteca *Java Foundation Classes*, oferece uma maneira de fornecer uma interface gráfica com o

usuário em programas Java e de receber entradas do usuário com o teclado, o mouse ou outros dispositivos de entrada. Através do *Swing* pode-se criar aplicativos que apresentam uma interface gráfica com o usuário, utilizando-se os componentes: quadros, contêineres, botões, rótulos, campos de texto e áreas de texto, listas suspensas, caixas de verificação e botões de rádio.

A classe `DrawingArea` é derivada da classe `PrintableGridCanvas` que é derivada de *JComponent*, que é superclasse para a maioria dos componentes de interface gráfica. A classe `PrintableGridCanvas` possui uma instância da classe `Transform` (figura 4.4) que é a classe responsável por fazer transformações entre os sistemas de coordenadas do dispositivo e do modelo.

O objeto `DrawingArea` herda de *JComponent*, um componente do *Swing*, onde um objeto *Graphics2D*, composto por intâncias de objetos de desenho do `Controller`, é desenhado.

As classes, representando objetos de desenhos necessários à visualização do processo de modelagem foram agrupadas no subpacote `gui.draw`. O diagrama de herança destas classes está mostrado na figura 4.5.

A *API Java2D* (*Application Programming Interface*) fornece recursos gráficos bidimensionais avançados que exigem manipulações gráficas complexas e detalhadas, muito amplos para serem trabalhados neste texto. O desenho com a *API Java2D* é realizado com uma instância da classe Graphics2D. A classe *Graphics2D* é uma subclasse de *Graphics*, portanto ela herda todos os recursos gráficos disponíveis na classe *Graphics*, que implementa métodos para desenho, manipulação de fontes, manipulação de cor, entre outros (Deitel e Deitel 2001).

A classe `Controller` (do subpacote `gui.controller`), como recomenda Grand (1998), é uma classe totalmente abstrata (em *Java*, uma interface) que faz referência ao `Model`, à `Interface` e à `DrawingArea`.

Cada uma das classes que implementam a interface `Controller` (figura 4.6) possui listas (instâncias de classes da *API Collections* (Horstmann e Cornell 2001*a*)) contendo instâncias de `Objetos de Desenho`, que são extensões de classes da *API* gráfica

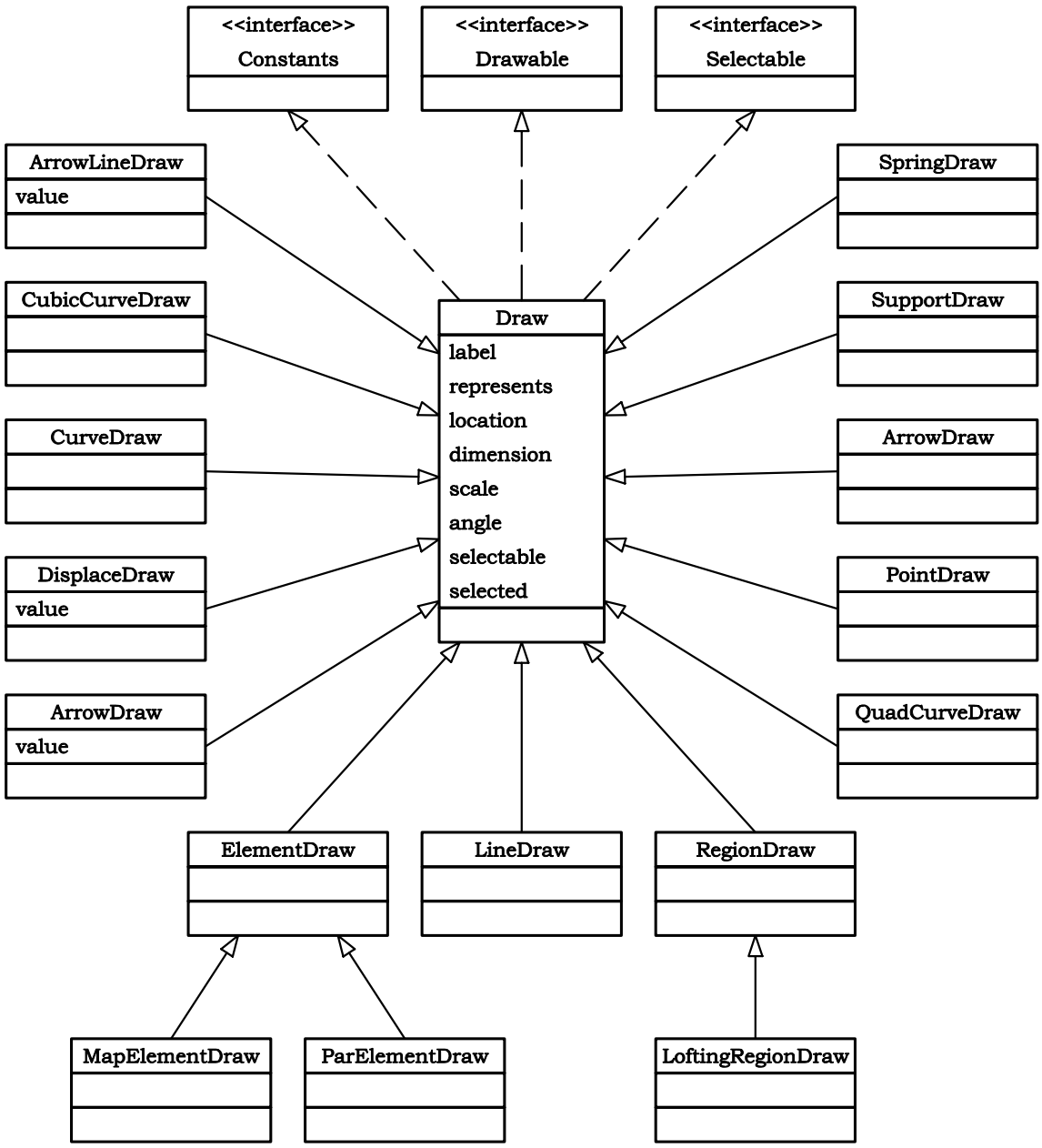


Figura 4.5: Diagrama de herança das classes do subpacote `gui.draw`

Java2D (Rowe 2001).

A *API Collections* oferece ao programador acesso a estruturas de dados pré-empacotadas e a algoritmos para manipular essas estruturas. As coleções são padronizadas de modo que os aplicativos possam compartilhá-las facilmente, sem a preocupação com detalhes de sua implementação. Essas coleções são escritas para ampla reutilização e ajustadas para rápida execução e utilização eficiente da memória.

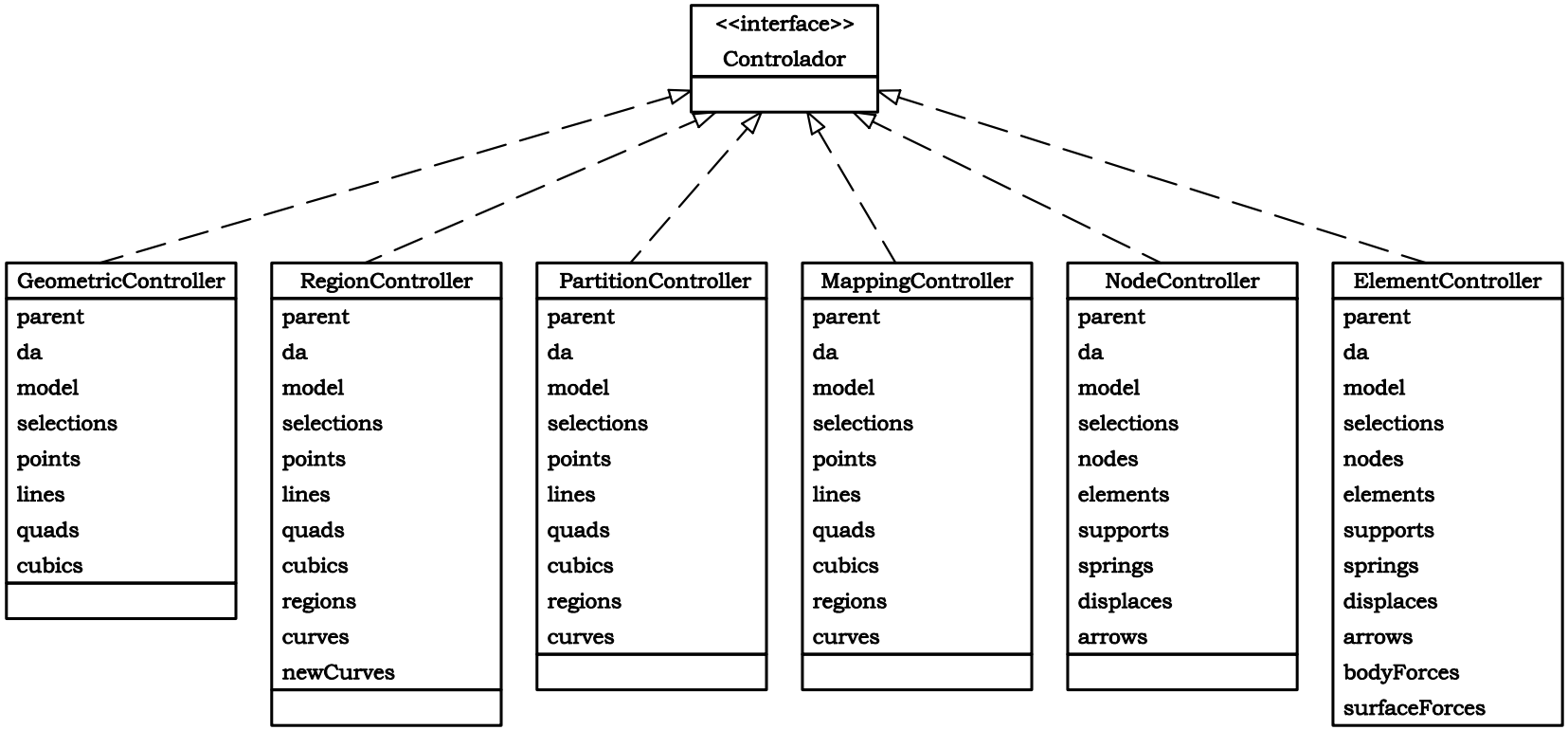


Figura 4.6: Diagrama de herança das classes do pacote `gui.controller`

Finalmente, as requisições que o usuário faz através dos elementos de interface gráfica (botões, menus, etc) foram tratadas através de classes que implementam a interface Command (figura 4.4), pertencentes ao subpacote `gui.command`. Cada subclasse de `Command` realiza uma operação específica e, por isso, apresenta grupos distintos de atributos. Elas são muito especializadas, o que dificulta a generalização que se observa nos diagramas anteriores. Na figura 4.7, são apresentadas várias subclasses de `Command`, observa-se um único tipo de relação de herança e relações de instância distintas.

Algumas classes, principalmente classes especializadas de `Command`, instanciam subclasses da interface `TabbedDialog` para possibilitarem a interação com o usuário. Essas subclasses apresentam diálogos específicos para recuperar entradas do usuário que são convertidas em parâmetros utilizados em suas rotinas. Estes diálogos formam o subpacote `gui.dialog`.

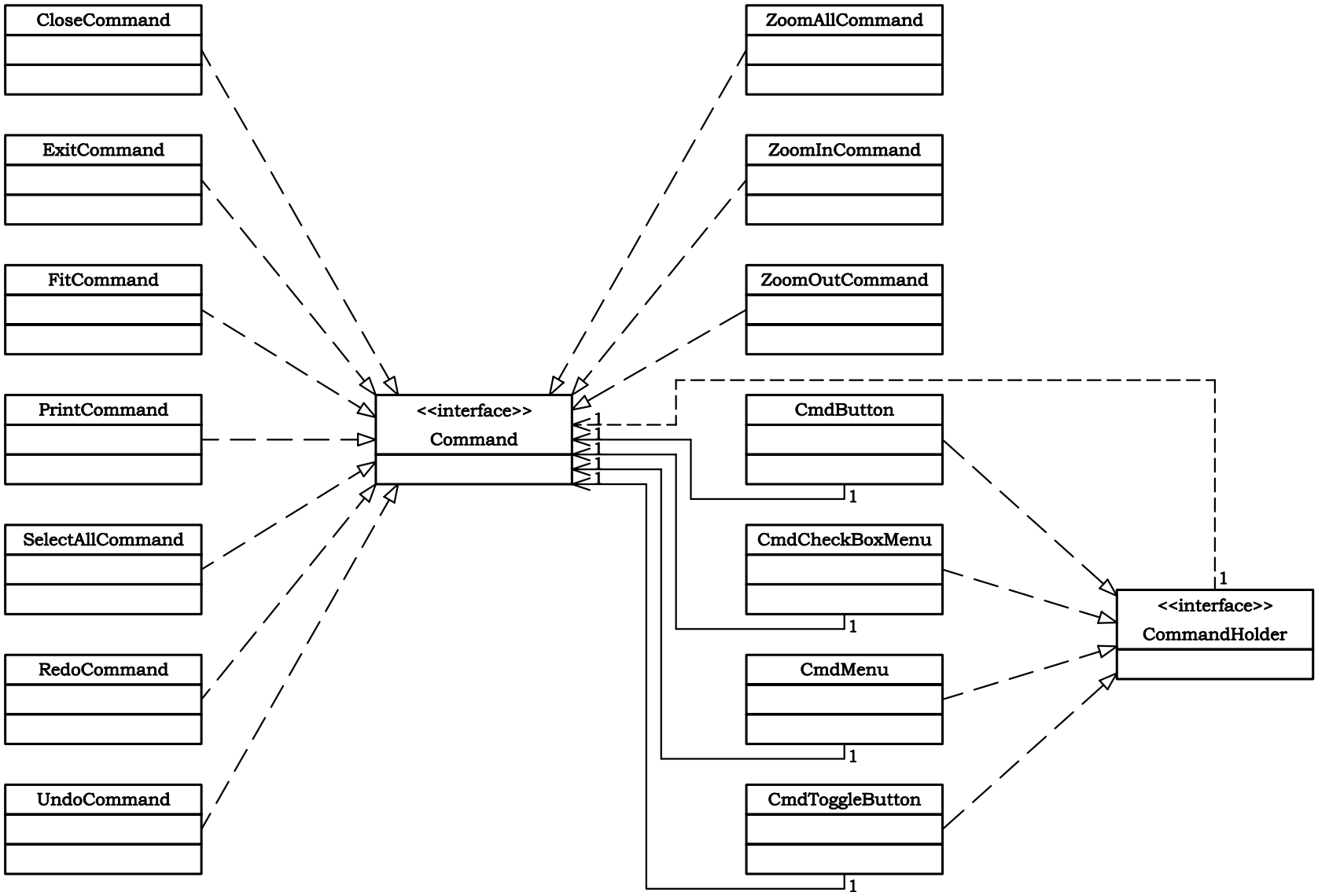


Figura 4.7: Diagrama de herança e instância das classes do subpacote `gui.command`

4.2.2 Modelos do MEF

No que se refere ao núcleo numérico do sistema, a camada modelo (ver figuras 4.1, 4.2) é representada pela classe `FemModel` cujo diagrama de instâncias está mostrado na figura 4.8. 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*.

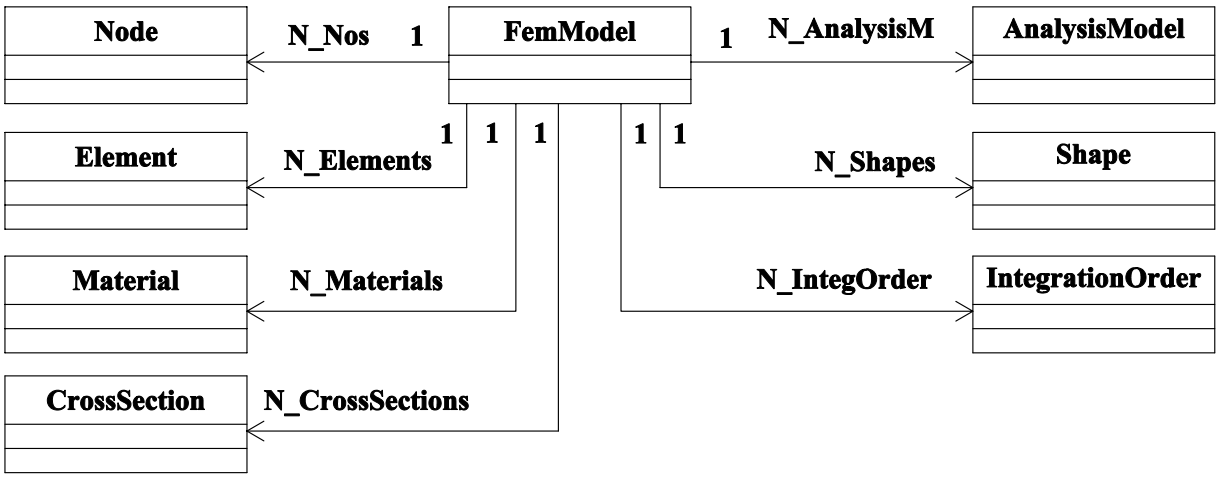


Figura 4.8: Objetos instanciados pela classe *FemModel*

Cada `Modelo` discreto adotado possui determinadas características (nós, elementos, materiais, modelo de análise, funções de forma, ordem de integração, etc.), que são armazenadas em listas de atributos deste `Modelo`.

A figura 4.9 mostra o diagrama de instâncias da classe *Element*. Cada objeto do tipo *Element* 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.

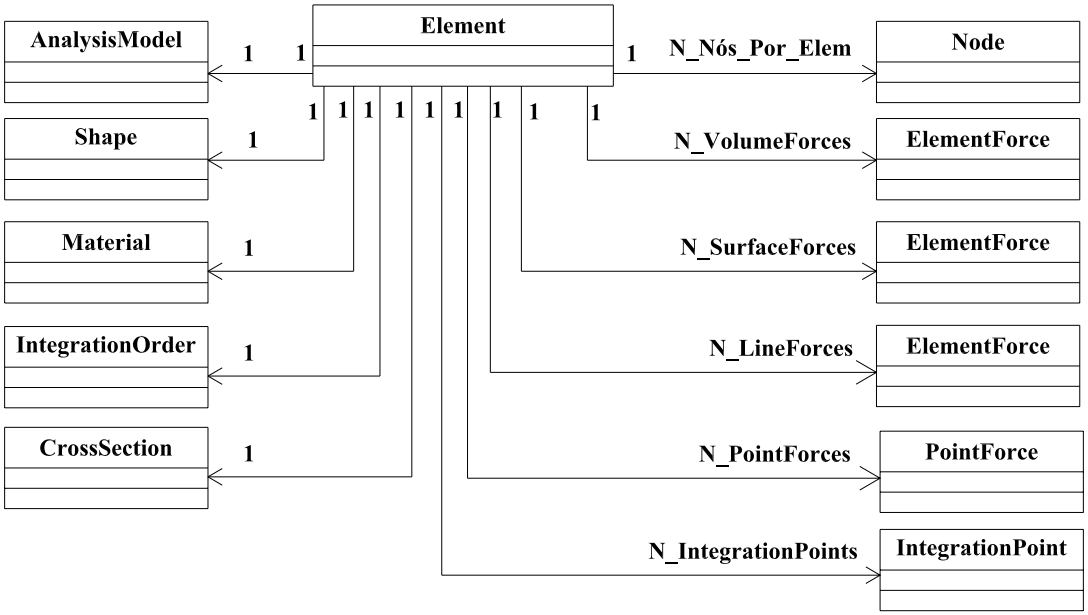


Figura 4.9: Objetos instanciados pela classe *PElement*

A figura 4.10 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*.

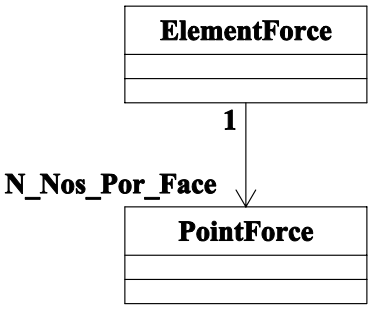


Figura 4.10: Objetos instanciados pela classe *ElementForce*

A figura 4.11 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.

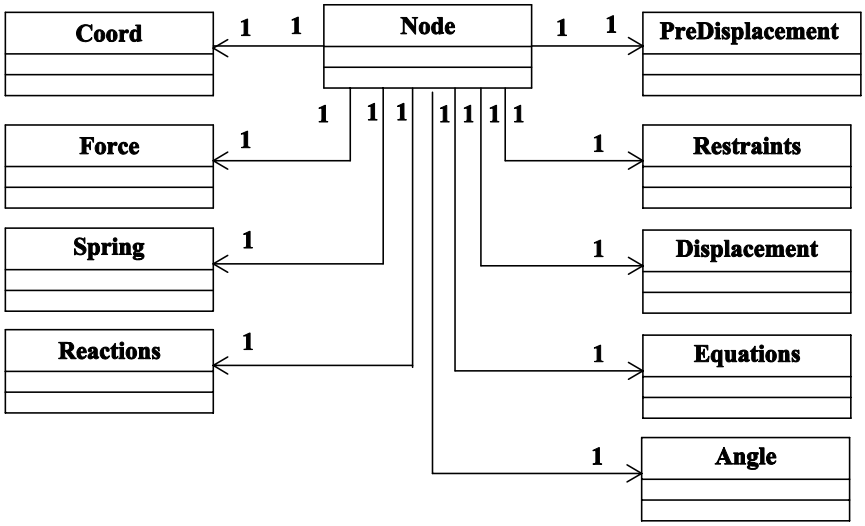


Figura 4.11: Objetos instanciados pela classe *Node*

A figura 4.12 mostra a hierarquia da classe *AnalysisModel* que tem por finalidade agrupar os tipos de análise 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 deformações (subclasse *PlaneStrainAnalysisM*).

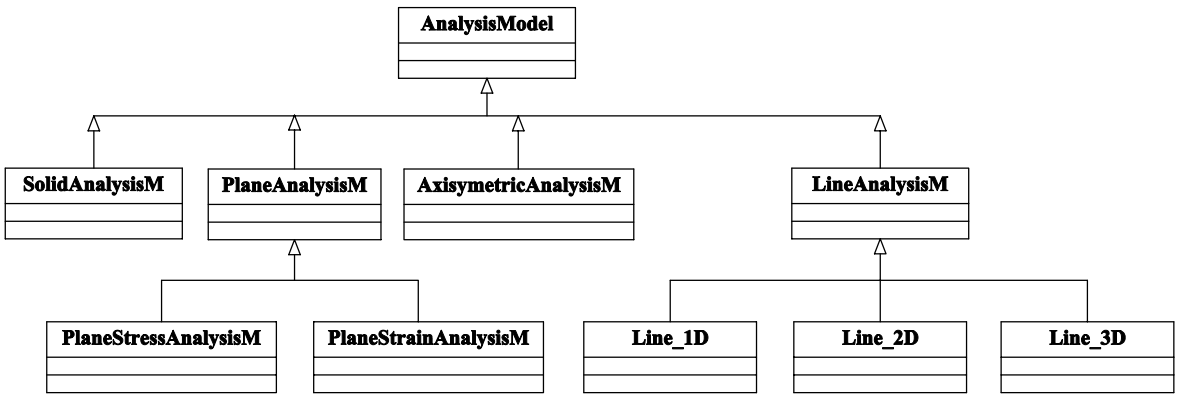


Figura 4.12: Hierarquia da classe *AnalysisModel*

A figura 4.13 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 4.13 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.

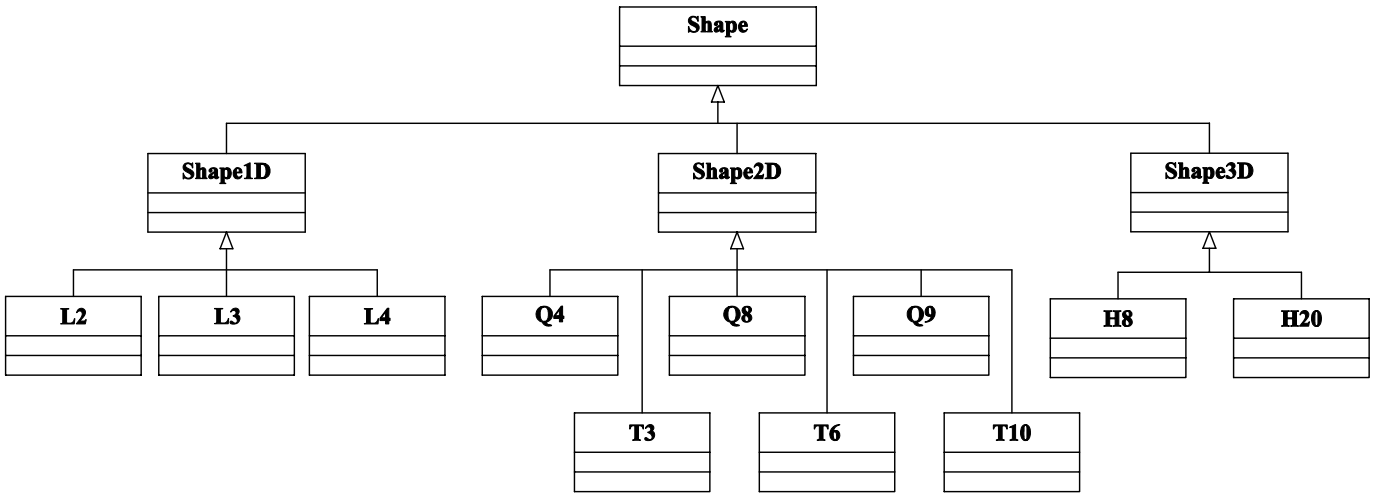


Figura 4.13: Hierarquia da classe *Shape*

A figura 4.14 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 e não-lineares.

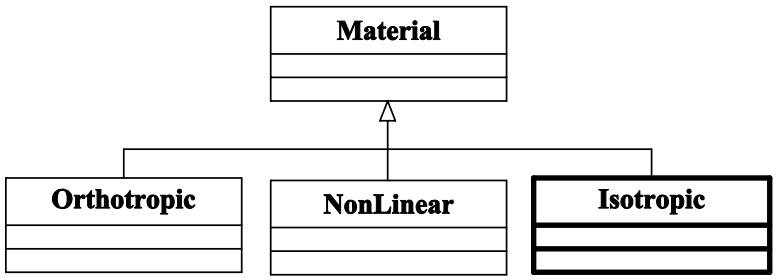


Figura 4.14: Hierarquia da classe *Material*

4.3 Projeto da Expansão

A expansão de classes do programa baseou-se nas etapas de análise via MEF descritas no item 2.6. Para cada etapa do processamento (figura 2.3 na página 18), foram criados pares vista-controlador, que permitem ao usuário consultar o Modelo, e obter as informações referentes à fase do processamento escolhida (tabela 2.2 na página 20). Assim, o projeto orientado a objetos existente, discutido no item 4.2, foi alterado como indicam os diagramas UML das figuras 4.15 e 4.16.

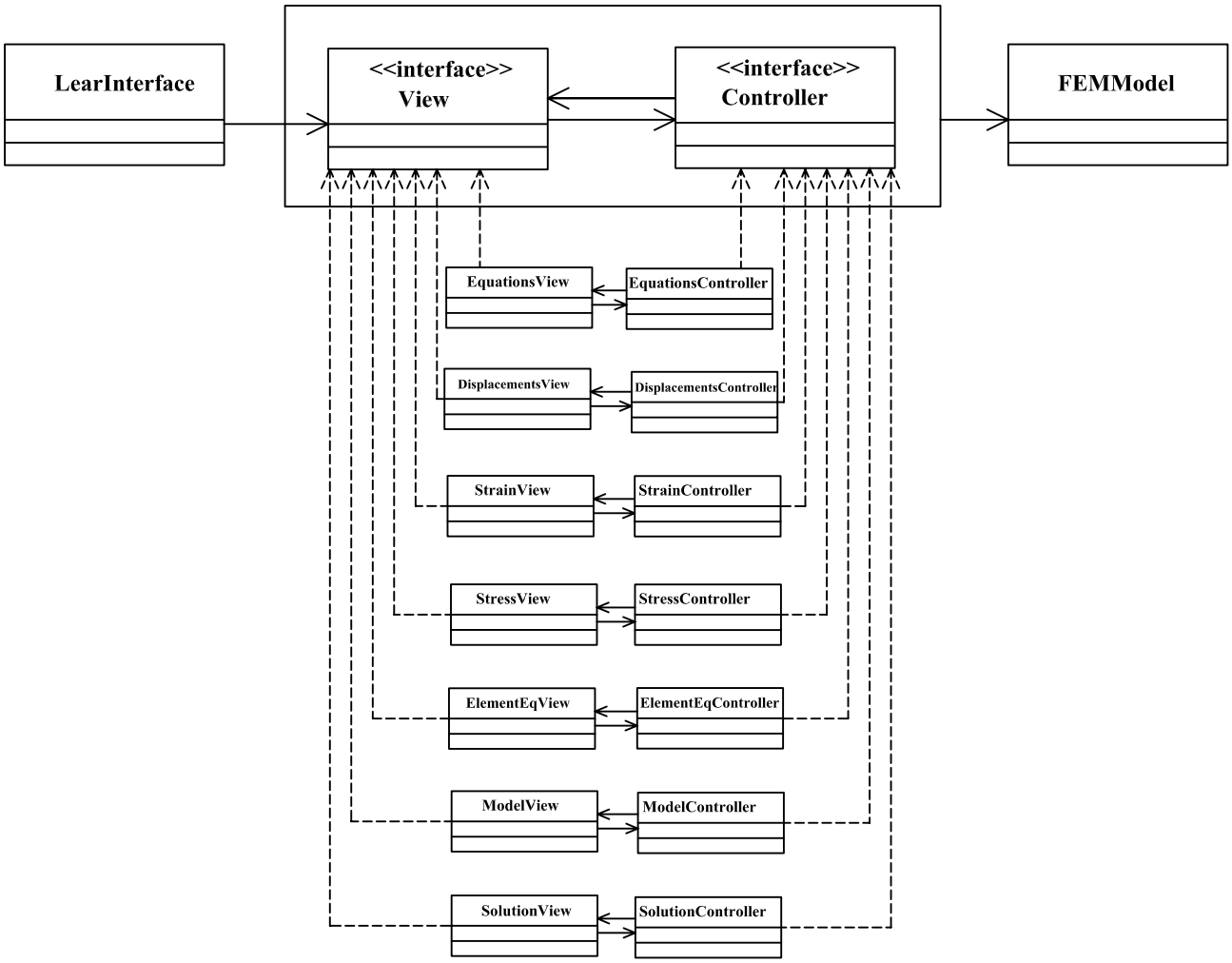


Figura 4.15: Pares de Vistas e Controladores do Processamento Interativo.

O pacote `learn` (figura 4.16) foi criado para agrupar estas classes responsáveis pelas camadas de apresentação (pares vista e controlador) da interface do processador interativo, bem como as classes que implementam as interações entre estas camadas.

A classe `LearnInterface` (figura 4.16) possui um único objeto da classe `FemModel`,

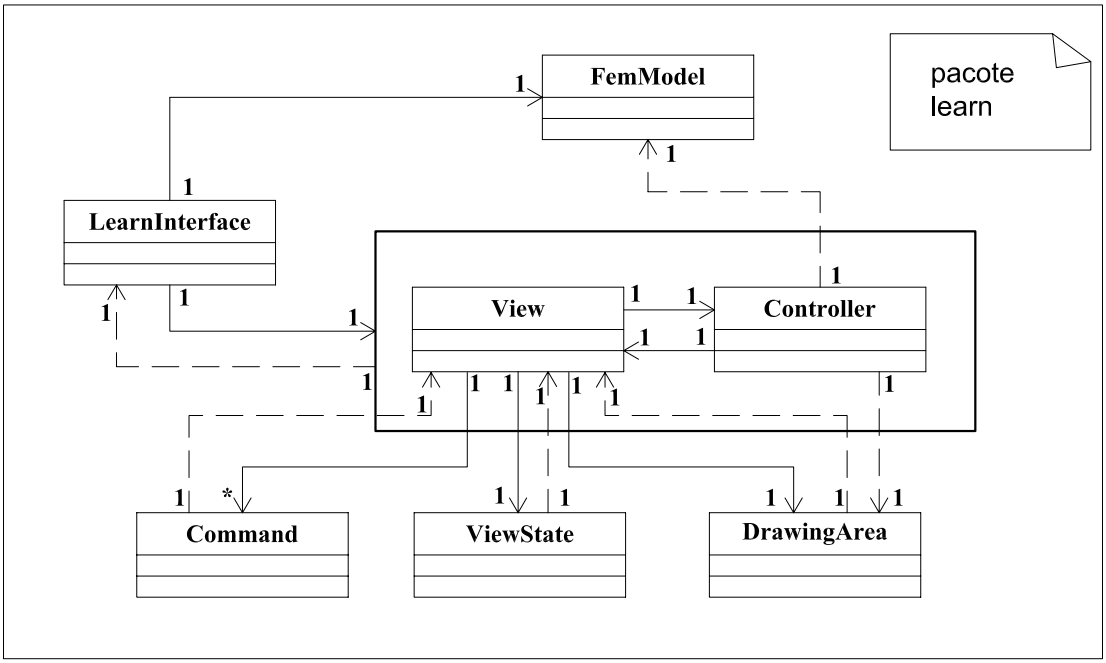


Figura 4.16: Diagrama de instâncias das classes do pacote `learn`.

já que para cada análise, existe apenas um modelo discreto, e um par vista-controlador, com objetos das classe `View` e `Controller`.

A classe `View`, por sua vez, possui um objeto da classe `DrawingArea`, um objeto da classe `ViewState`, e vários objetos da classe `Command`.

O objeto `DrawingArea` (figura 4.16) é instância de classes da *API Java Swing* (Horstmann e Cornell 2001*b*).

A classe Controller, como recomenda Grand (1998), é uma classe totalmente abstrata (em *Java*, uma interface) que faz referência ao `FemModel`, à `LearnInterface` e à `DrawingArea`.

Cada uma das classes que implementam a interface `Controller` (figura 4.17) possui listas (instâncias de classes da *API Collections* (Horstmann e Cornell 2001*a*)) contendo instâncias de `Objetos de Desenho`, que são extensões de classes da *API* gráfica *Java2D* (Rowe 2001).

A classe `View` também é uma classe abstrata, que faz referência à `LearnInterface`.

Cada uma das classes que implementam `View` (figura 4.18) possui um objeto das classes `DrawingArea` e `ViewState`, e vários objetos da classe `Command`.

As classes que implementam `View` possuem elementos gráficos (botões, menus, etc.),

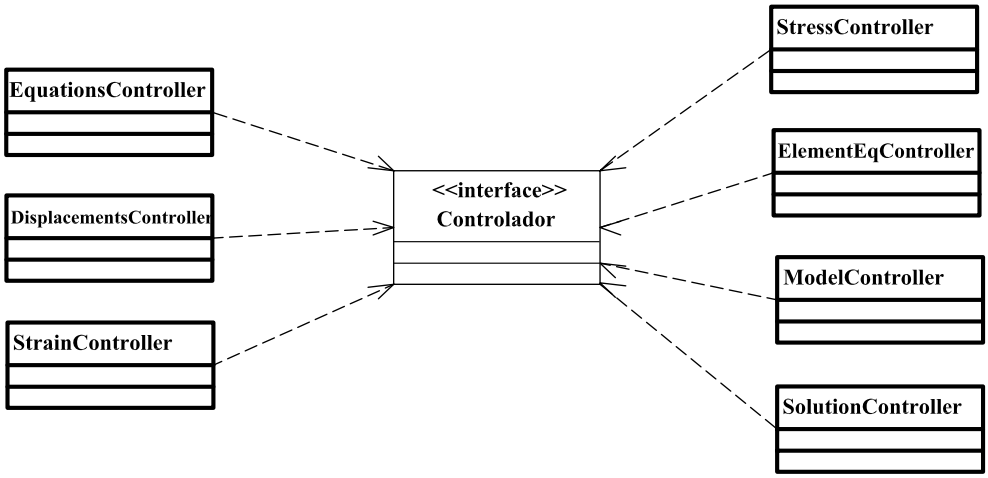


Figura 4.17: Herança dos controladores do pacote `learn`.

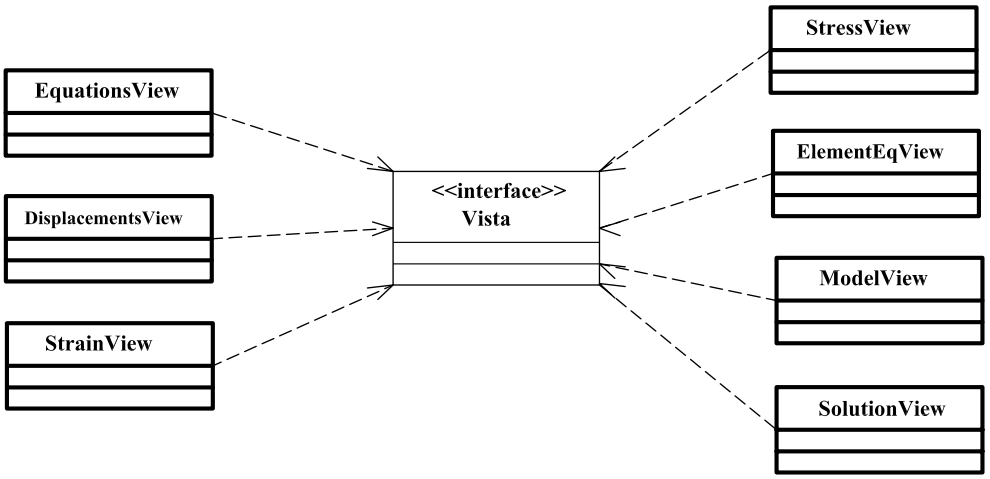


Figura 4.18: Herança das vistas do pacote `learn.view`.

que atendem às solicitações do usuário para cada etapa da solução do problema.

Estas requisições, que o usuário faz através dos elementos de interface gráfica, foram tratadas através de classes que implementam a interface Command (figura 4.16), pertencentes a diversos subpacotes. Cada subclasse de `Command` realiza uma operação específica e, por isso, apresenta grupos distintos de atributos. Elas são muito especializadas, o que dificulta a generalização que se observa nos diagramas anteriores. Nas figuras 4.19, 4.20, 4.21, 4.22, 4.23, 4.24 e 4.25, são apresentadas as subclasses de `Command` de acordo com a herança estabelecida para a classe abstrata `View`.

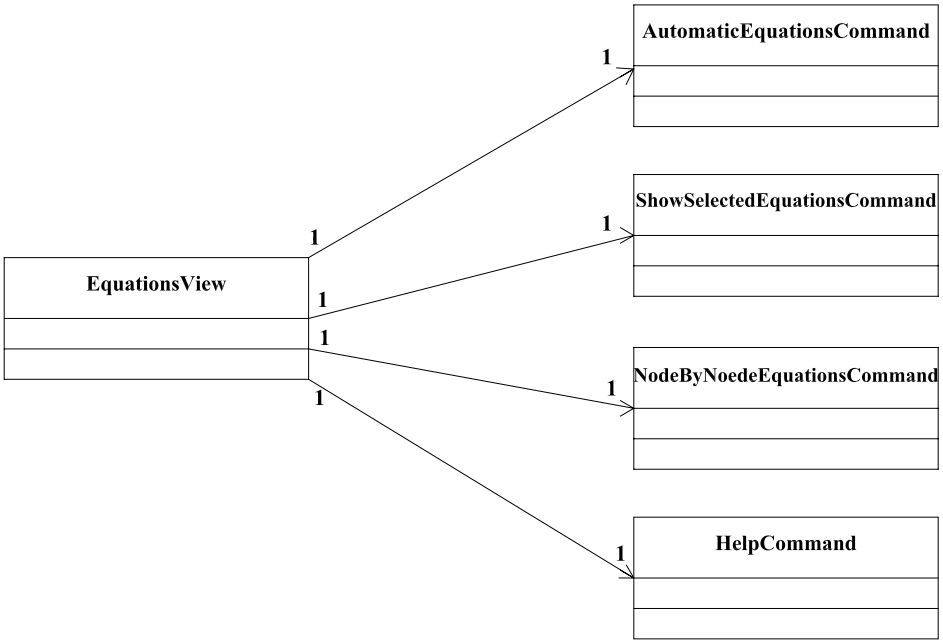


Figura 4.19: Instâncias `Command` do pacote `learn.view.equation`.

Algumas classes, principalmente classes especializadas de `Command`, instanciam subclasses da interface `TabbedDialog`, para possibilitarem a interação com o usuário. Essas subclasses apresentam diálogos específicos para trazer os dados da solução ao usuário.

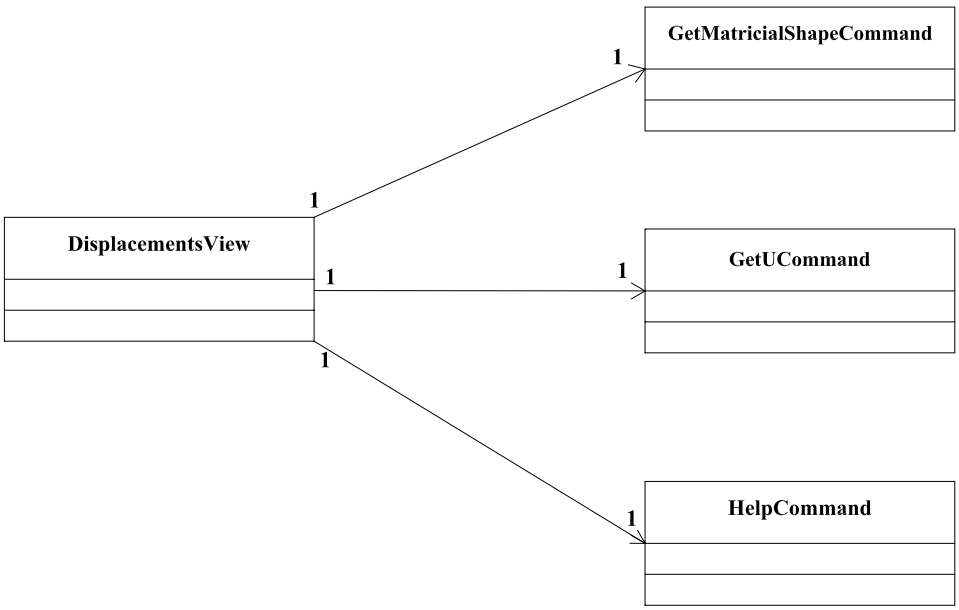


Figura 4.20: Instâncias `Command` do pacote `learn.view.displacements`.

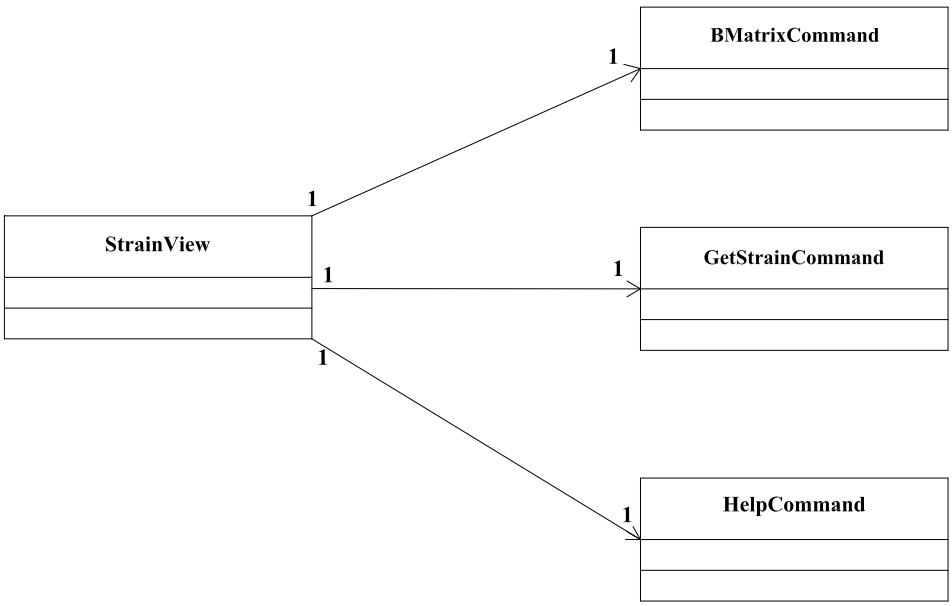


Figura 4.21: Instâncias `Command` do pacote `learn.view.strain`.

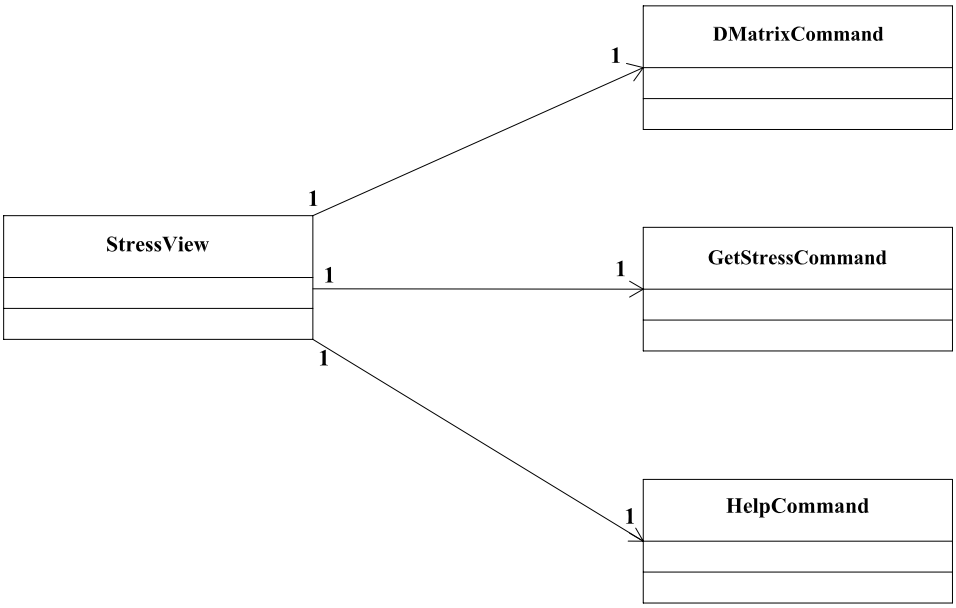


Figura 4.22: Instâncias `Command` do pacote `learn.view.stress`.

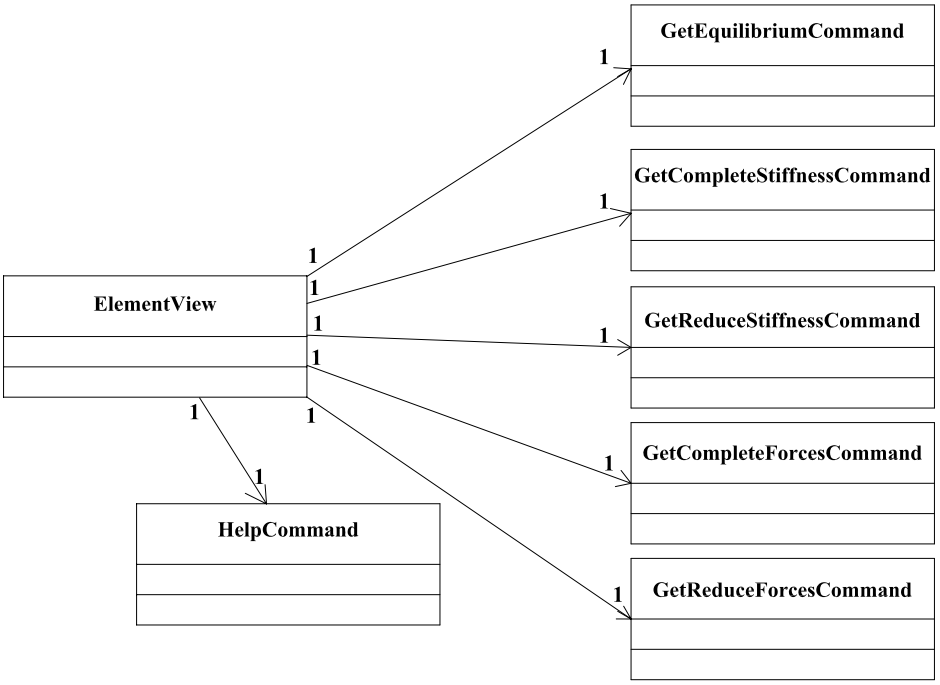


Figura 4.23: Instâncias `Command` do pacote `learn.view.element`.

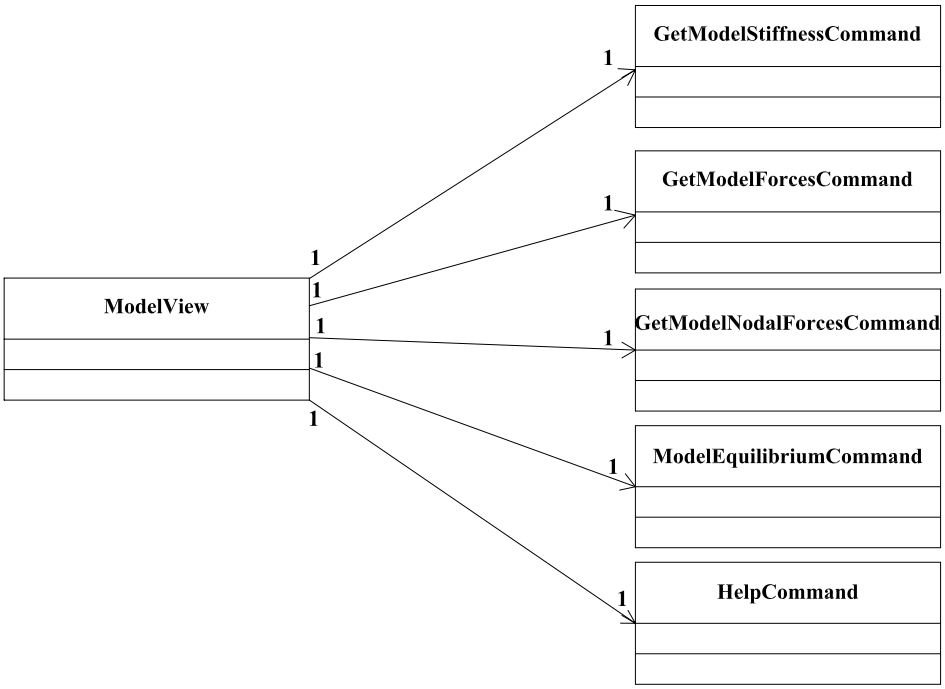


Figura 4.24: Instâncias `Command` do pacote `learn.view.model`.

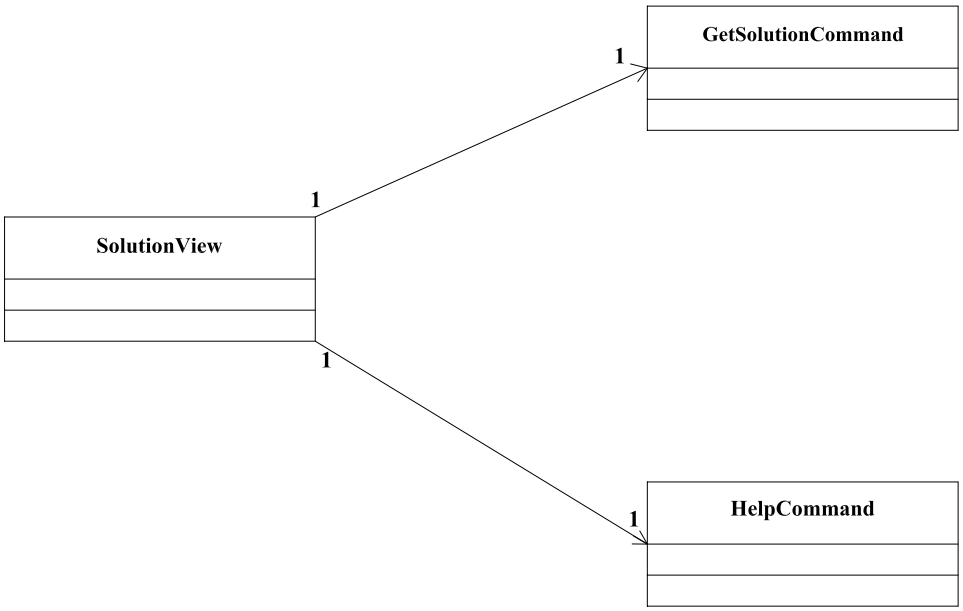


Figura 4.25: Instâncias `Command` do pacote `learn.view.solution`.

Capítulo 5

Funcionamento da Aplicação

Este capítulo tem o objetivo de apresentar os recursos do processador interativo. Ele ilustra as interações possíveis entre o usuário e as principais etapas da solução de um modelo do MEF.

O modelo adotado para esta ilustração é uma discretização simples, com poucos elementos, para uma membrana em estado plano de tensões (figura 5.1), submetida a um carregamento distribuído uniforme de tração ($po = 10\ uf/uc$). A configuração geométrica da placa está mostrada na figura 5.1 em uc, e as propriedades do material são $E = 30,0 \times 10^6\, uf/uc^2$ (módulo de elasticidade longitudinal) e $\nu = 0,25$ (coeficiente de Poisson), sendo uf = unidades de força e uc = unidades de comprimento.

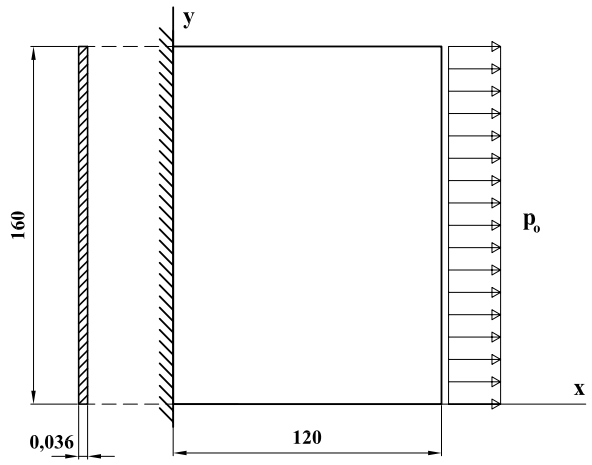


Figura 5.1: Membrana em estudo.

A figura 5.2 mostra a discretização adotada, obtida na fase de pré-processamento do programa `INSANE`.

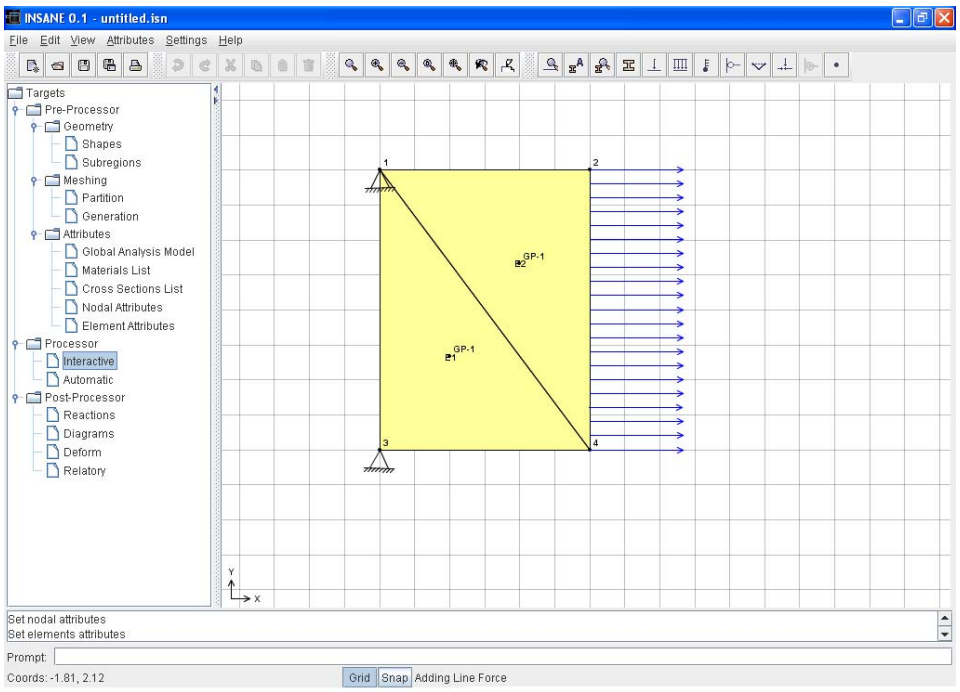


Figura 5.2: Modelo obtido no pré-processador `INSANE`.

O processamento interativo tem início com escolha da opção ”Interactive” da árvore de opções do programa. Um mensagem é exibida na tela, sugerindo ao usuário trabalhar com malhas de poucos elementos devido a finalidade didática deste recurso, conforme mostrado na figura 5.3.

A partir deste momento, uma nova interface gráfica é exibida, mostrando o modelo a ser analisado através do processador interativo (figura 5.4). Nesta interface existe uma árvore de opções estabelecida de acordo com as etapas descritas no item 2.6.

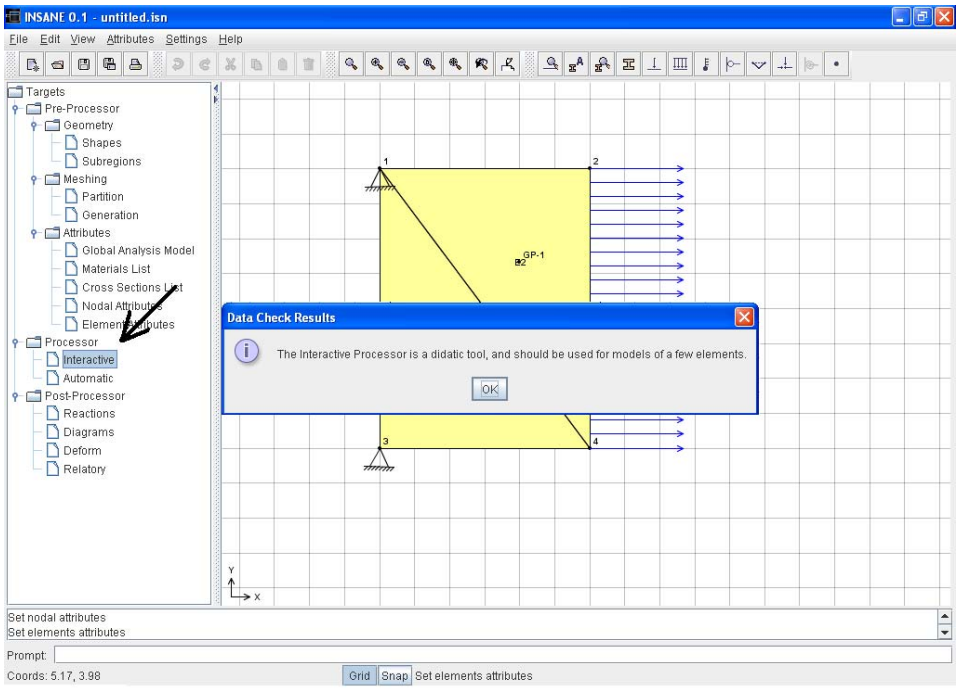


Figura 5.3: Seleção do processador interativo.

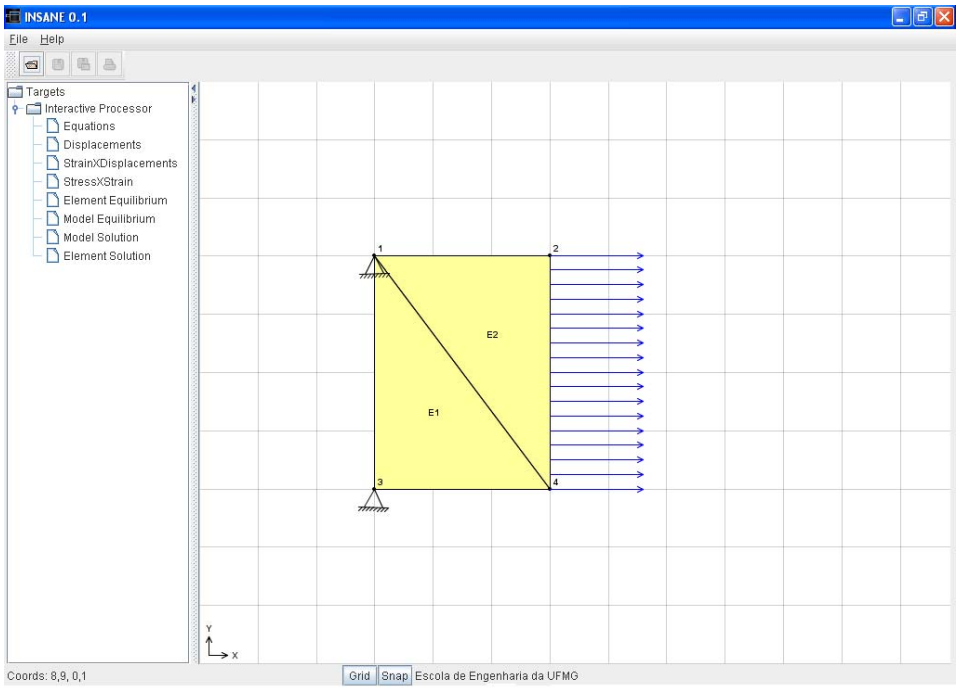


Figura 5.4: Interface do processador interativo.

O primeiro nó da árvore, denominado *Equations*, não é definido nas etapas do processamento (item 2.6), mas é um importante passo para a resolução do problema. Trata-se da numeração das equações do modelo, de acordo com os graus de liberdade dos elementos e as condições de contorno. Esta numeração interfere diretamente no processo de solução, e por isso foi adicionada às opções do processamento.

A figura 5.5 mostra a opção selecionada (*Equations*) e os botões de comando referentes às interações que podem ser estabelecidas com o usuário.

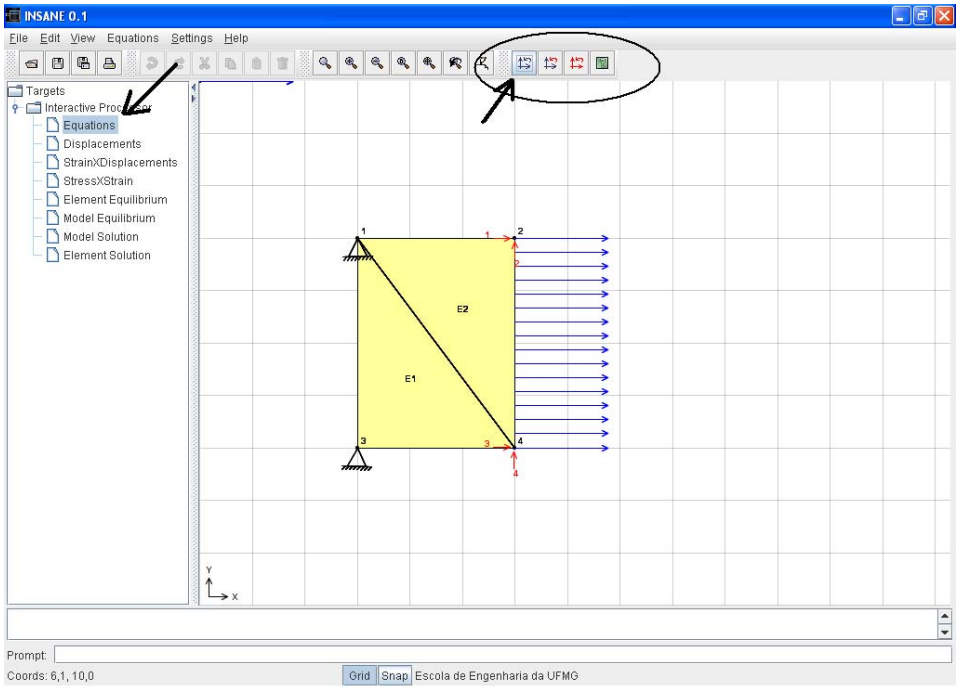


Figura 5.5: Numeração das equações do modelo.

O primeiro botão, *Automatic Equations*, executa a numeração automática das equações do modelo, sem que o usuário possa interferir na ordem das mesmas.

O segundo botão, *Show Selected Equations*, apenas mostra as equações já numeradas, sem refazer a ordem desta numeração, caso o usuário queira somente observar a numeração já feita.

O terceiro botão, *Node By Node Equations*, possibilita que o usuário determine a ordem da numeração das equações do modelo e verifique o efeito desta mudança no processo de solução.

Ao acionar o último botão, *Help*, o usuário obtém um breve relato a respeito da

teoria referente à etapa do processamento, e também uma explicação sobre as funções relativas aos comandos da etapa. A função *Help* está presente em todas as opções da árvore, auxiliando o usuário em possíveis dúvidas sobre a teoria e o funcionamento do processador interativo. Esta função será detalhada adiante.

O segundo nó da árvore, *Displacements*, refere-se à etapa 2 do processamento do MEF (item 2.6, figura 2.3) e permite ao usuário a visualização das funções de forma para cada ponto de Gauss dos elementos do modelo, como mostra a figura 5.6.

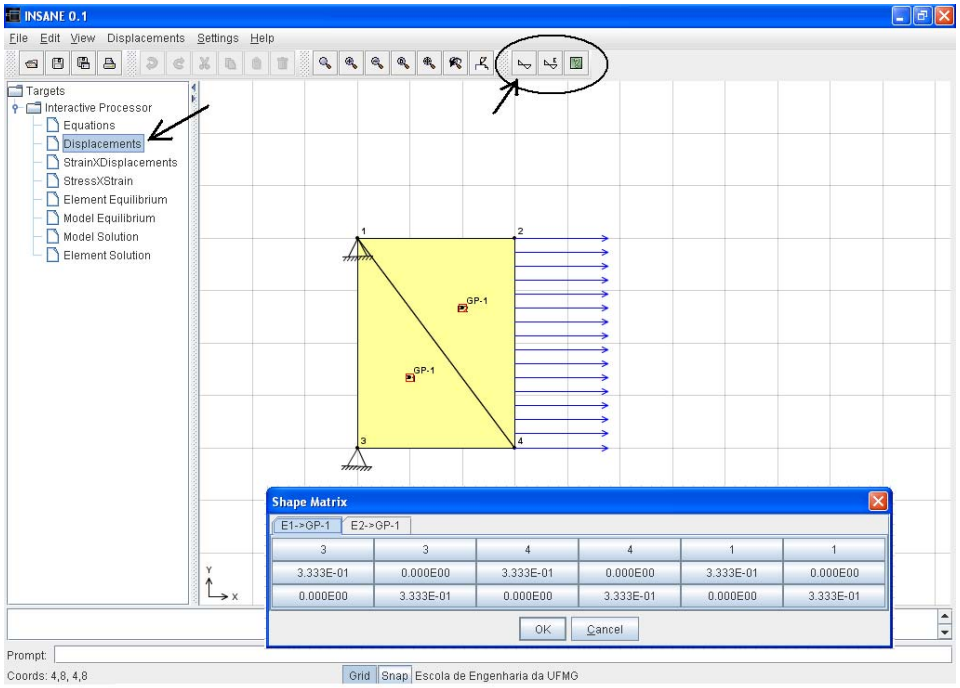


Figura 5.6: Matriz das funções de forma.

Nesta fase, também podem ser consultados, os deslocamentos referentes a cada ponto de Gauss do elemento, a partir da obtenção dos deslocamentos nodais, como mostra a figura 5.7.

O primeiro botão deste nó, *Matricial Shape*, apresenta a matriz das funções de forma calculada em cada ponto de Gauss do elemento (figura 5.6).

O segundo botão, *Get U* , apresenta o produto das funções de forma pelos deslocamentos nodais, resultando assim, nos valores dos deslocamentos em cada ponto de Gauss do elemento (figura 5.7).

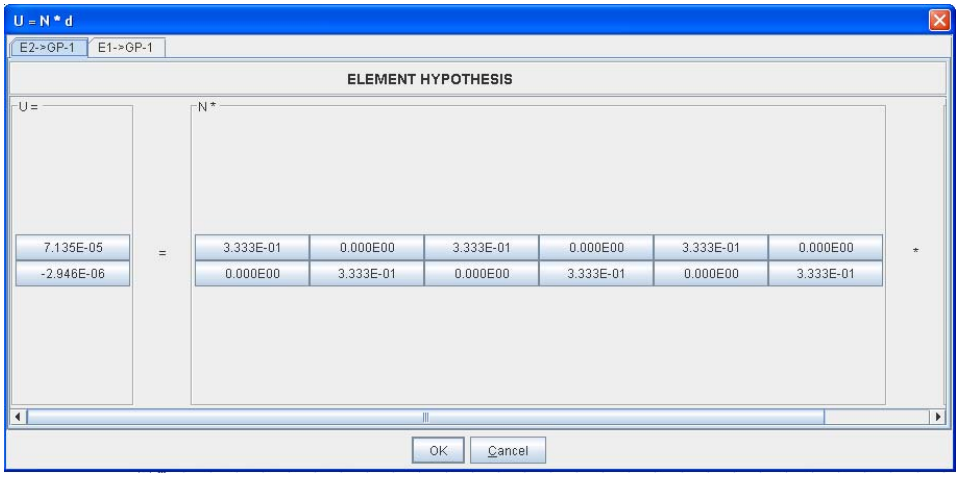


Figura 5.7: Deslocamentos nos pontos de Gauss selecionados.

O botão de *Help* desta etapa traz uma breve teoria e o esclarecimento sobre cada opção disponível (figura 5.8).

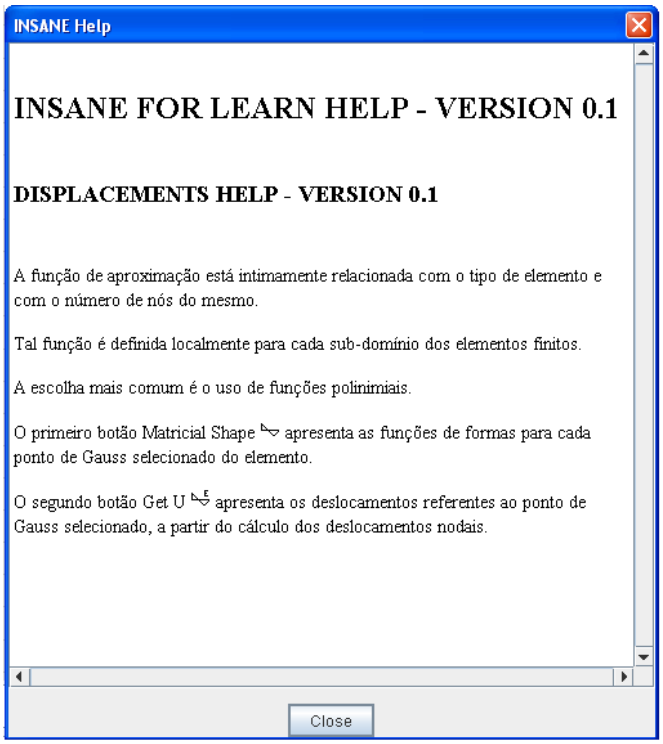


Figura 5.8: `INSANE` help.

Após a etapa de definição das funções de forma, são definidas as relações de *Deformação* × *Deslocamento* oriundas da hipótese de pequenos deslocamentos, referente à etapa 3 do processamento do MEF (item 2.6, figura 2.3). A nó da árvore *Strain* × *Displacement* representa estas relações.

O primeiro botão, *B Matrix*, apresenta a matriz B contendo as derivadas das funções

de forma em cada ponto de Gauss selecionado do elemento (figura 5.9).

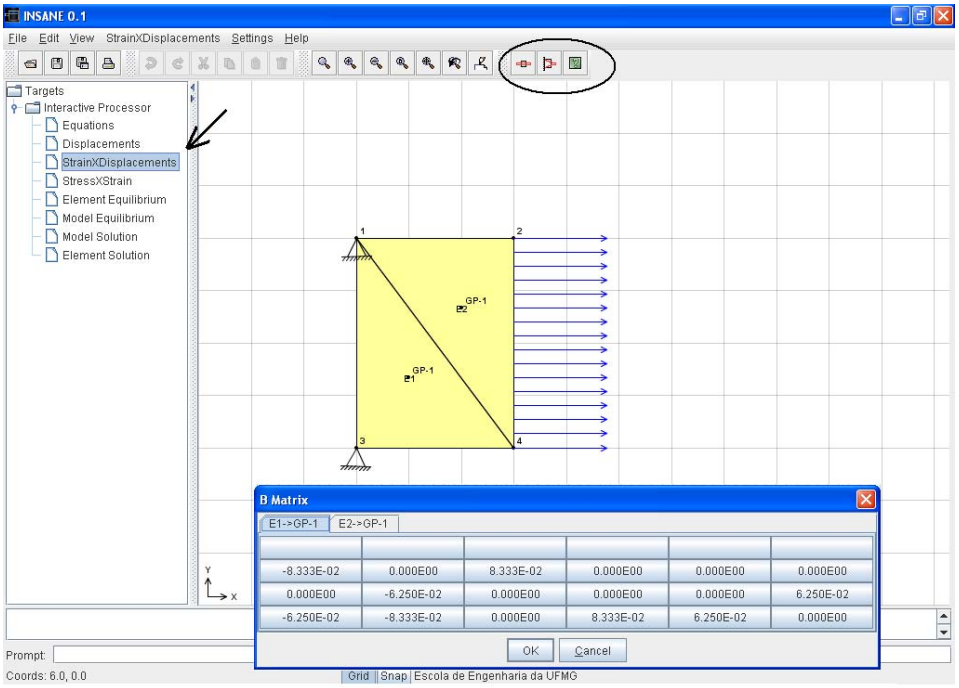


Figura 5.9: Matriz B - derivadas das funções de forma.

O segundo botão, *Get Strain*, apresenta o produto da matriz B pelos deslocamentos nodais calculados, obtendo-se portanto as deformações em cada ponto de Gauss selecionado do elemento (figura 5.10).

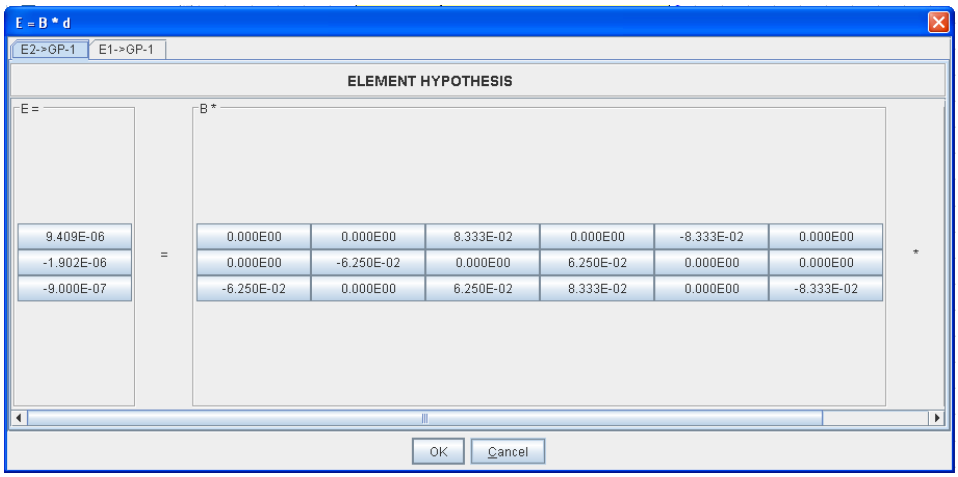


Figura 5.10: Deformações nos pontos de Gauss selecionados.

O comando *Help* também está disponível nesta etapa, assim como nas etapas seguintes.

Ainda na etapa 3 do processamento via MEF (item 2.6, figura 2.3), são definidas as relações de Tensão×Deformação. No caso deste processamento, adotou-se a hipótese de material elástico linear. O nó da árvore *Stress×Strain* apresenta estas relações (figura 5.11).

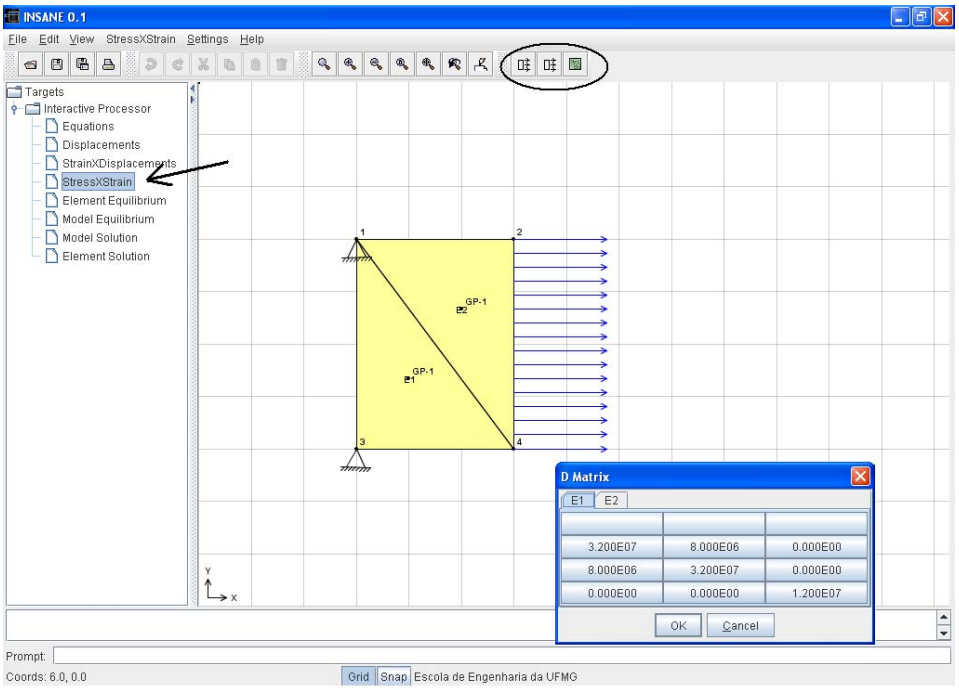


Figura 5.11: Matriz D - propriedades do material.

O primeiro botão, *D Matrix*, apresenta a matriz D contendo as propriedades do material selecionado para o elemento (figura 5.11). O segundo botão, *Get Stress*, apresenta o produto da matriz D pelas deformações calculadas em cada ponto de Gauss do elemento, obtendo-se, portanto, as tensões nos referidos pontos (figura 5.12).

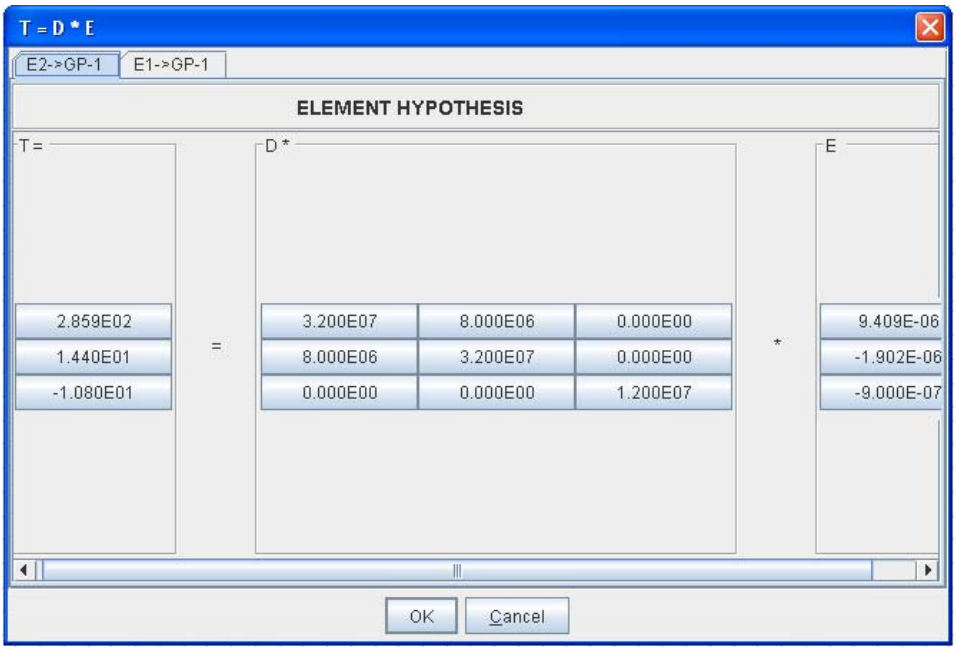


Figura 5.12: Tensões nos pontos de Gauss selecionados.

Na etapa 4 do processamento via MEF (item 2.6, figura 2.3), obtém-se as equações de equilíbrio para cada elemento da discretização. A figura 5.13 apresenta o nó *Element Equilibrium*, que agrupa os parâmetros referentes ao equilíbrio de cada elemento.

Nesta etapa são apresentados a matriz de rigidez e o vetor de forças nodais equivalentes para cada elemento da malha. A figura 5.13 mostra um diálogo contendo as matrizes de rigidez completas dos elementos.

A partir da aplicação das condições de contorno do problema, também é possível visualizar a matriz de rigidez reduzida de cada elemento, já com as linhas e colunas, com restrição de deslocamentos, eliminadas (figura 5.14).

Os vetores de carregamento nodal equivalente também podem ser observados na forma completa e reduzida, conforme mostrado na figura 5.15.

A partir da determinação da matriz de rigidez e do vetor de forças, obtém-se o equilíbrio para cada elemento do modelo. Neste caso, se os deslocamentos forem conhecidos, a figura 5.16 mostra o produto da matriz de rigidez pelos deslocamento e, subtraindo-se o vetor de carregamento nodal equivalente, tem-se as ações de extremidade para cada elemento.

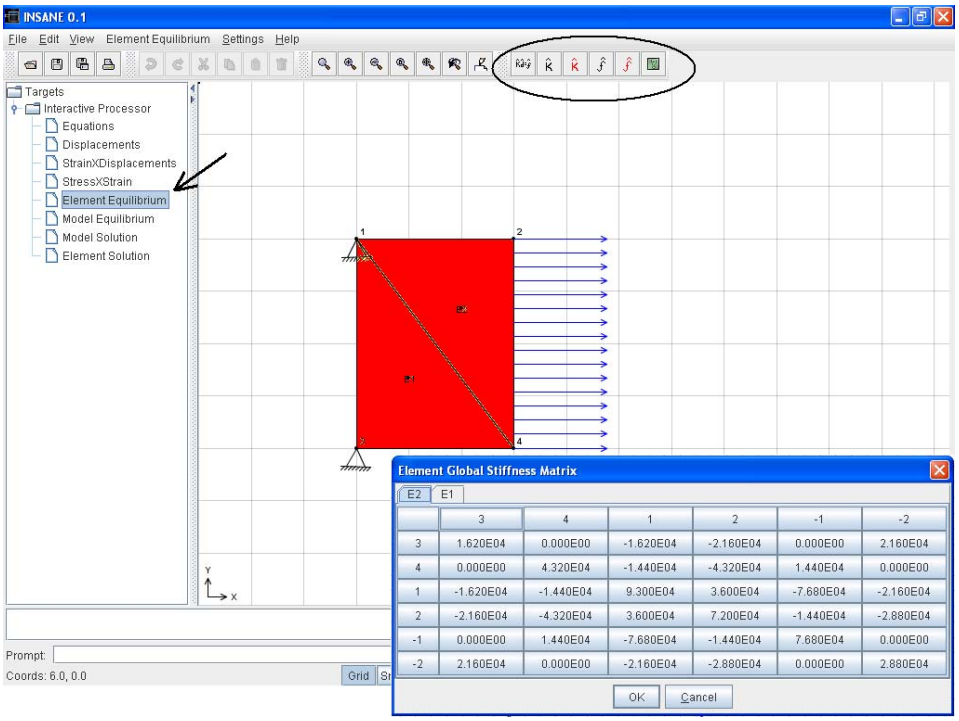


Figura 5.13: Matriz de rigidez completa do elemento.

Figura 5.14: Matriz de rigidez reduzida do elemento.

Figura 5.15: Vetores de carregamento nodal equivalente do elemento.

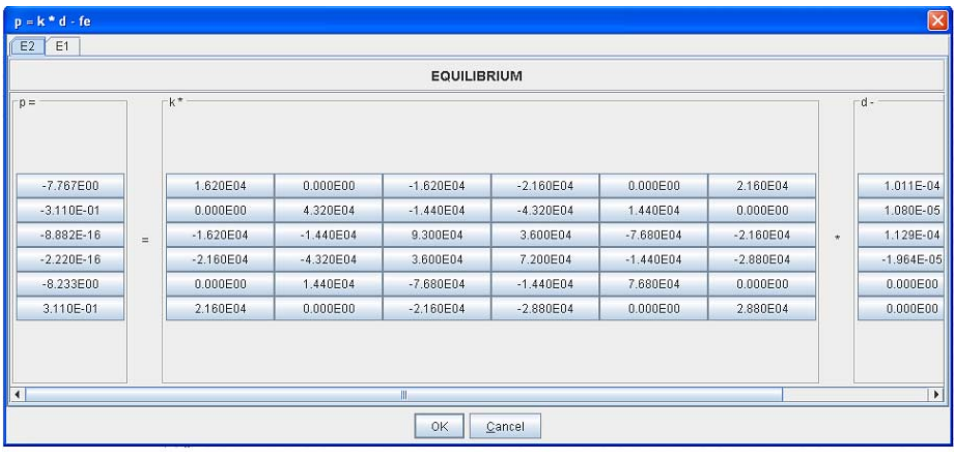


Figura 5.16: Equilíbrio do Elemento.

As matrizes de rigidez e os vetores de forças obtidos na etapa 4 para cada elemento agora são ”somados”, via Método da Rigidez Direta, para obtenção da matriz de rigidez e do vetor de cargas externas do modelo. Neste estágio a matriz de rigidez é singular, e para solução do modelo, as condições de contorno precisam ser impostas. Assim se caracteriza a etapa 5 (figura 5.17), em que verifica-se a montagem das equações de equilíbrio do modelo.

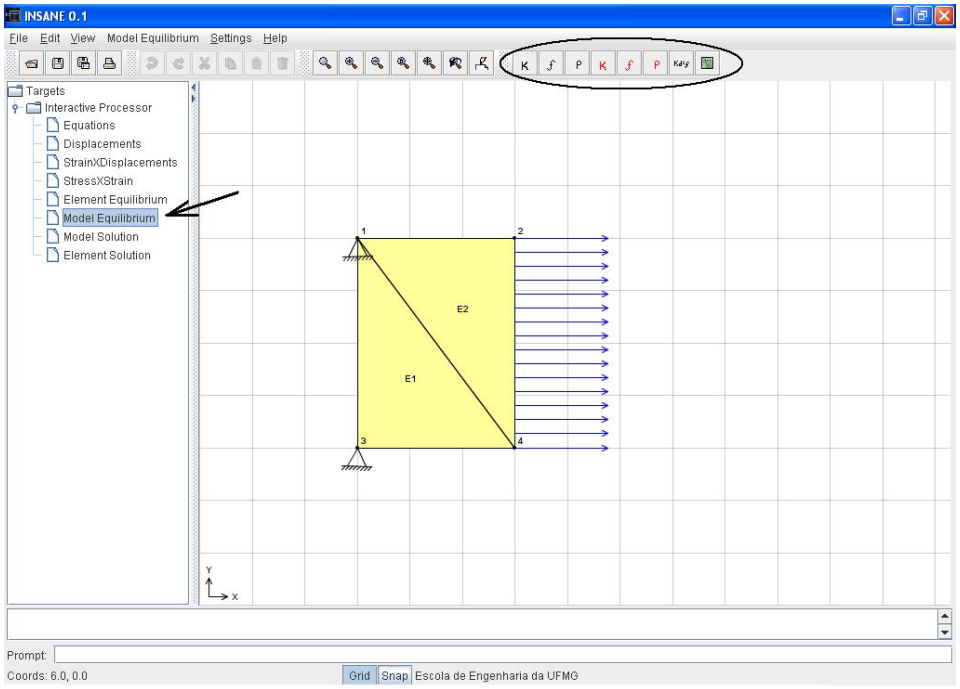


Figura 5.17: *Model Equilibrium.*

A figura 5.18 mostra a matriz de rigidez completa do modelo, onde em cada célula, é possível visualizar a contribuição de cada elemento.

A matriz de rigidez reduzida também pode ser consultada, com as devidas condições de contorno aplicadas, eliminando-se as linhas e colunas referentes aos graus de liberdade restritos. Para esta opção também é possível visualizar as contribuições de cada elemento. O mesmo pode ser aplicado á consulta dos vetores de força completo e reduzido (figura 5.19).

A figura 5.20 mostra o equilíbrio do elemento, com as condições de contorno aplicadas, e portanto, pode-se solucionar o sistema de equações, para determinação dos deslocamentos nodais incógnitos.

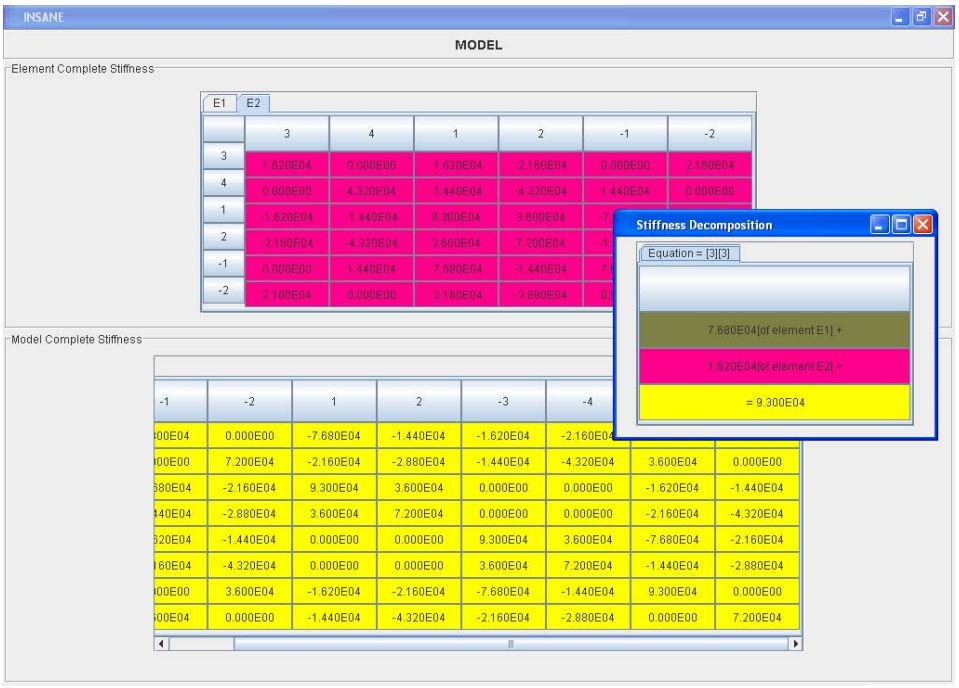


Figura 5.18: Matriz de rigidez completa do modelo.

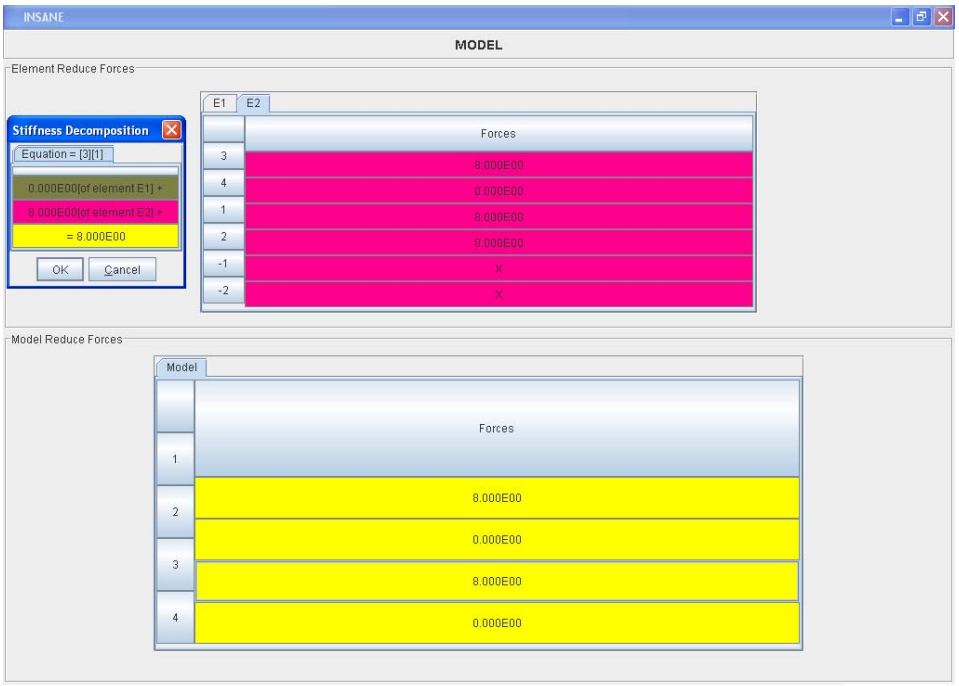


Figura 5.19: Vetor de forças do modelo.

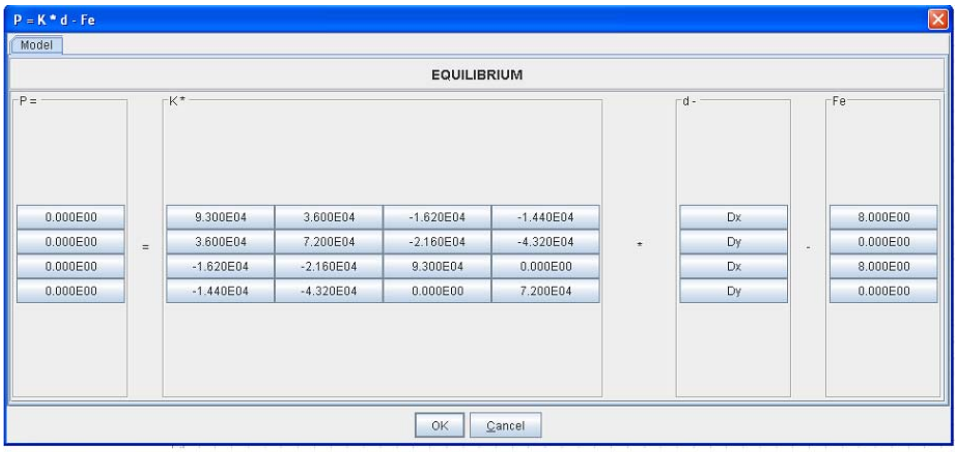


Figura 5.20: Equilíbrio do modelo.

Após a imposição das condições de contorno, o sistema de equações algébricas obtido na etapa 5 pode ser resolvido para a determinação das incógnitas cinemáticas. O nó da árvore *Model Solution* caracteriza a etapa 6 do processamento, quando ocorre a solução das equações e a obtenção dos deslocamentos nodais desconhecidos, conforme mostrado na figura 5.21.

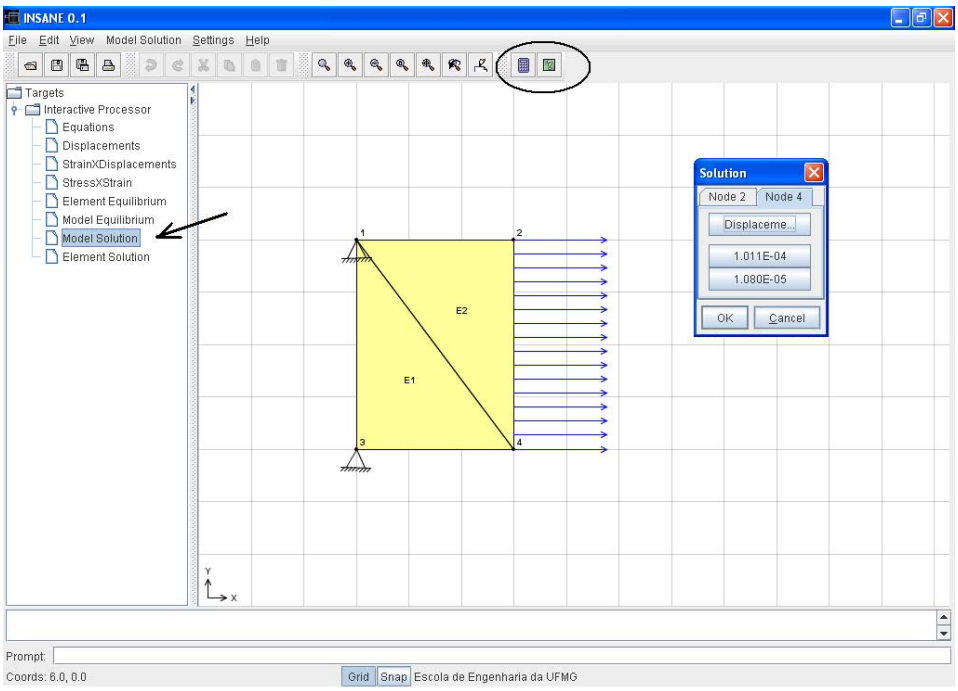


Figura 5.21: Solução dos deslocamentos.

Capítulo 6

Exemplos

6.1 Introdução

Neste capítulo são apresentados vários modelos de elementos finitos que utilizam os diversos recursos disponibilizados no sistema. São apresentados seis exemplos conforme indica a tabela 6.1. O exemplo 1 mostra um modelo de elementos de treliça plana. O exemplo 2 mostra um modelo de elementos de viga. O exemplo 3 mostra um modelo de elementos de pórtico plano. O exemplo 4 mostra um modelo de elementos de grelha. O exemplo 5 mostra um modelo de elementos de estado plano de tensões. O exemplo 6 mostra um modelo de elementos de estado plano de deformações.

Tabela 6.1: Exemplos

Exemplos	**Modelo de Análise**	**Seções no Capítulo**
1	Treliça Plana	6.2
2	Viga	6.3
3	Pórtico Plano	6.4
4	Grelha	6.5
5	Estado Plano de Tensões	6.6
6	Estado Plano de Deformação	6.7

6.2 Treliça Plana

Este exemplo tem como objetivo mostrar os elementos finitos unidimensionais de treliça plana, obtendo interativamente a solução de um exercício típico de um curso do MEF. O modelo em questão é mostrado na figura 6.1.

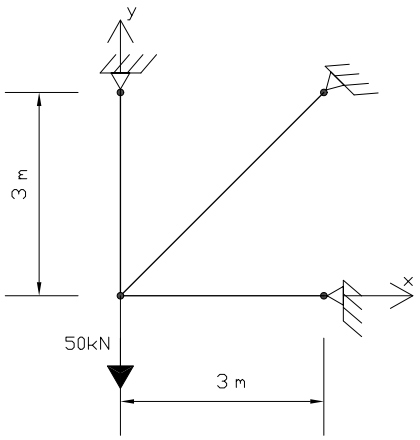


Figura 6.1: Treliça plana em estudo.

A figura 6.2 mostra o modelo em estudo, gerado pelo pré-processador `INSANE`, onde se pode observar a numeração dos nós, as condições de contorno e o carregamento. Na figura 6.3 são mostrados os sistemas locais de coordenadas adotados para os três elementos do modelo. Para processamento, adotou-se, área da seção transversal, $A = 1,2 \times 10^{-3}\, uc^2$, e módulo de elasticidade longitudinal, $E = 2,0 \times 10^{8}\, uf/uc^2$, para todos os elementos.

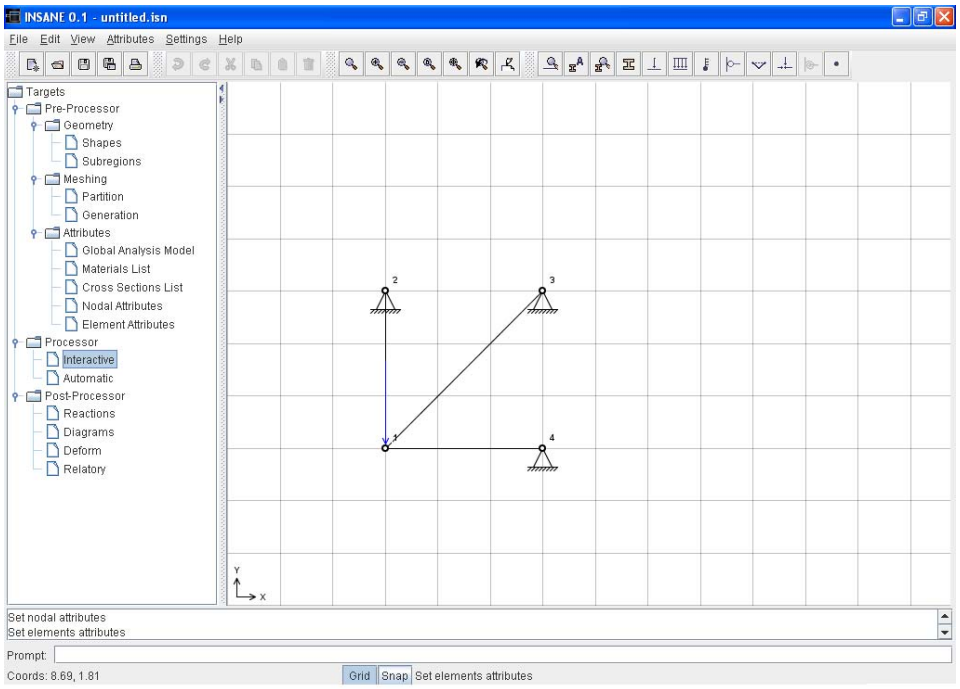


Figura 6.2: Modelo de treliça plana obtido no pré-processador `INSANE`.

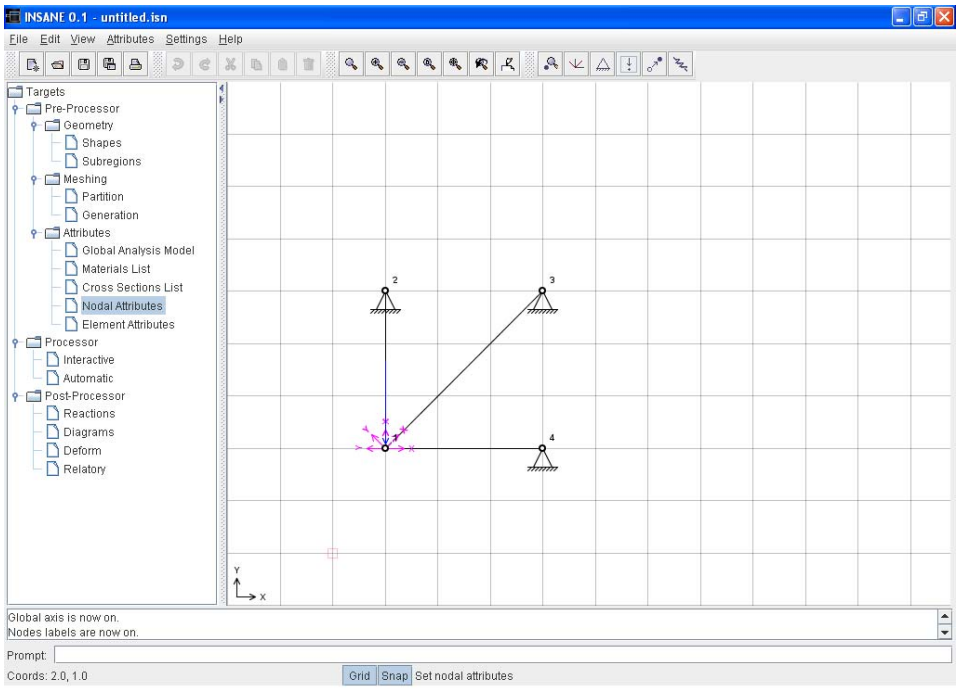


Figura 6.3: Sistemas locais de coordenadas.

A solução interativa inicia-se pela numeração das equações do modelo conforme mostra a figura 6.4 .

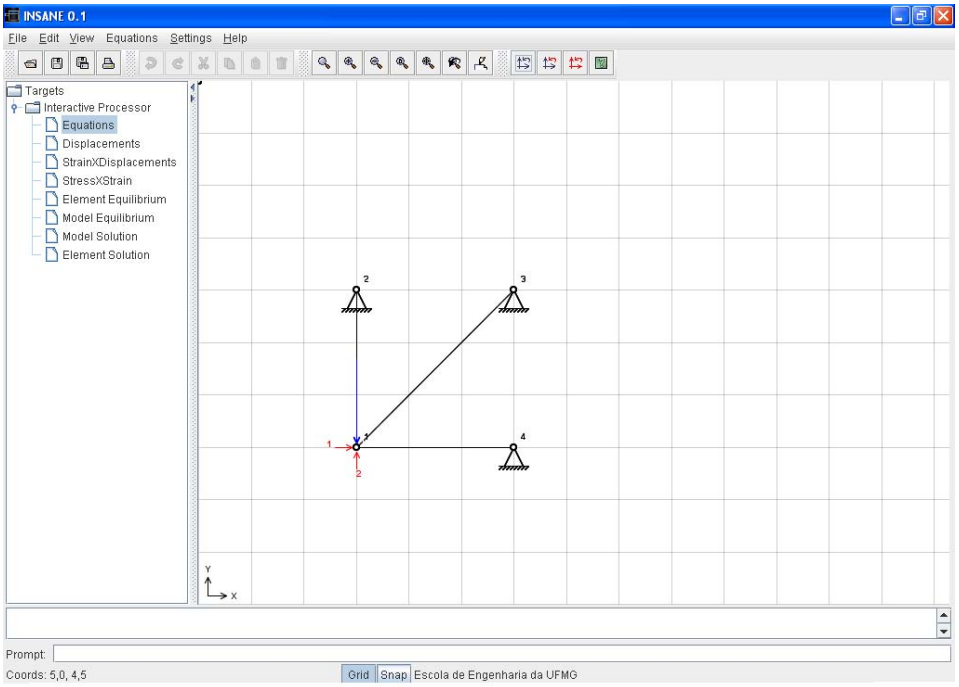


Figura 6.4: Numeração das equações do modelo.

Nos modelos de treliça plana, a partir da interpolação de deslocamentos $\{u\} = [N]\,\{\hat{d}\}^e$, tem-se:

$$\left\{ \begin{array}{c} u \\ v \end{array} \right\} = \left[\begin{array}{cccc} N_1 & 0 & N_2 & 0 \\ 0 & N_1 & 0 & N_2 \end{array} \right] \left\{ \begin{array}{c} \hat{d}_{1x} \\ \hat{d}_{1y} \\ \hat{d}_{2x} \\ \hat{d}_{2y} \end{array} \right\} \tag{6.2.1}$$

onde u e v são os deslocamentos nas direções x e y, $\hat{d}_{1x}$, $\hat{d}_{1y}$, $\hat{d}_{2x}$ e $\hat{d}_{2y}$ são , respectivamente, os deslocamentos dos nós inicial e final, e N_i a função de forma associada ao nó i, todos relativos ao sistema local de coordenadas do elemento (figura 6.3).

A matriz $[N]$ pode ser visualizada na solução interativa para cada ponto de um elemento, conforme mostra a figura 6.5, para $x = 2,0$, no elemento E1-3.

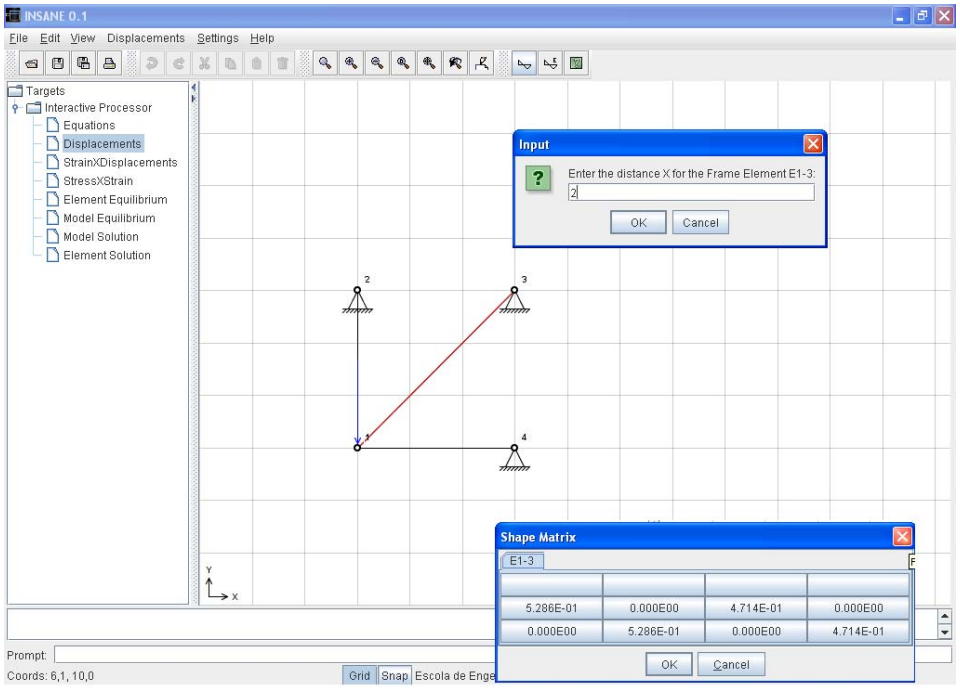


Figura 6.5: Funções de forma do elemento E1-3 no ponto x=2,0.

As deformações normais nas barras da treliça (ε_x) podem ser obtidas através de $\{\varepsilon_x\} = [B]\ \{\hat{d}\}$ ou :

$$\left\{ \varepsilon_x \right\} = \left[N_{1,x} \quad 0 \quad N_{2,x} \quad 0 \right] \left\{ \hat{d} \right\} \tag{6.2.2}$$

sendo $[B]$ a matriz das derivadas primeiras das funções de forma em relação a x, e $\{\hat{d}\}$ o vetor dos deslocamentos nodais.

A matriz $[B]$ pode ser visualizada na solução interativa para cada ponto do elemento, conforme mostra a figura 6.6, para $x = 2,0$, no elemento E1-2.

Os esforços normais nas barras da treliça podem ser obtidos através de $\{N\} = [EA]\ \{\varepsilon_x\}$. A matriz $[EA]$ pode ser visualizada na solução interativa, conforme mostra a figura 6.7.

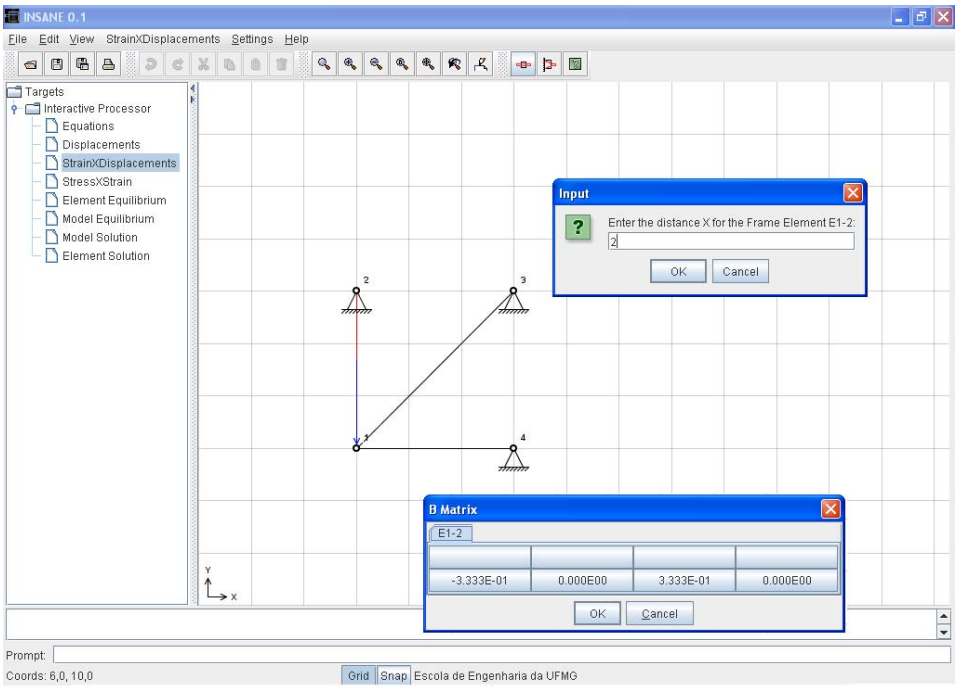


Figura 6.6: Matriz $[B]$ do elemento E1-2 no ponto x=2,0.

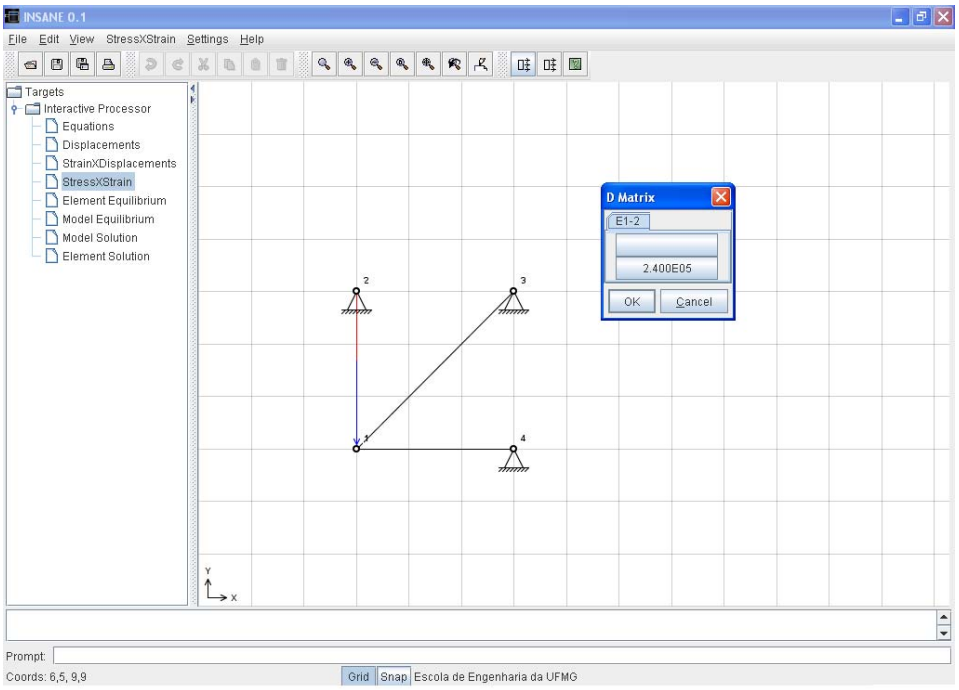


Figura 6.7: Matriz $[EA]$ do elemento E1-2.

Definidas as hipóteses dos elementos, a matriz de rigidez de cada elemento pode ser visualizada (figura 6.8), assim como o vetor de carregamento nodal equivalente.

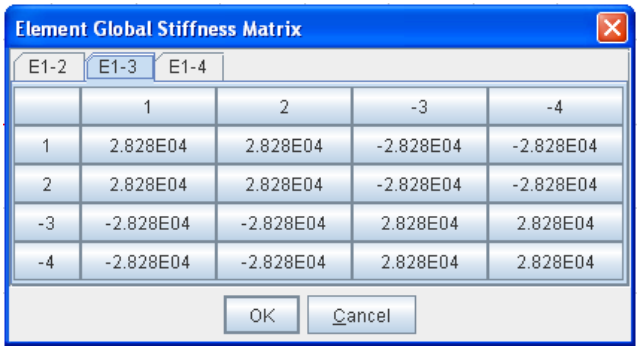


Figura 6.8: Matriz de rigidez completa de cada elemento do modelo.

Aplicando as condições de contorno, a matriz de rigidez reduzida de cada elemento é apresentada (figura 6.9), assim como o vetor de carregamento nodal equivalente reduzido (figura 6.10).

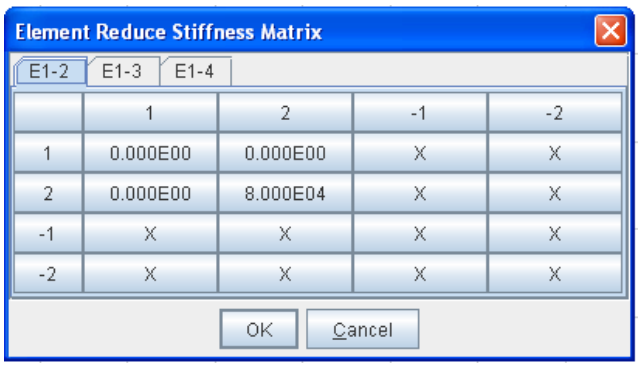


Figura 6.9: Matriz de rigidez reduzida de cada elemento do modelo.

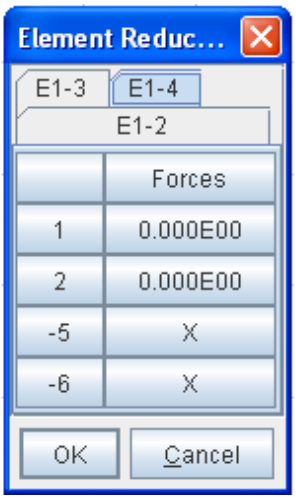


Figura 6.10: Vetor de carregamento nodal equivalente de cada elemento do modelo.

A partir do equilíbrio de cada elemento, é gerado o equilíbrio do modelo. Este também pode ser consultado, sendo mostrada a contribuição de cada elemento do problema à rigidez total do modelo. A figura 6.11 mostra a matriz de rigidez completa do modelo e a visualização da contribuição de cada elemento para formar um dos termos da mesma. A matriz de rigidez reduzida também pode ser consultada. Para esta opção também é possível visualizar as contribuições de cada elemento. O mesmo pode ser aplicado à consulta dos vetores de força completa e reduzida.

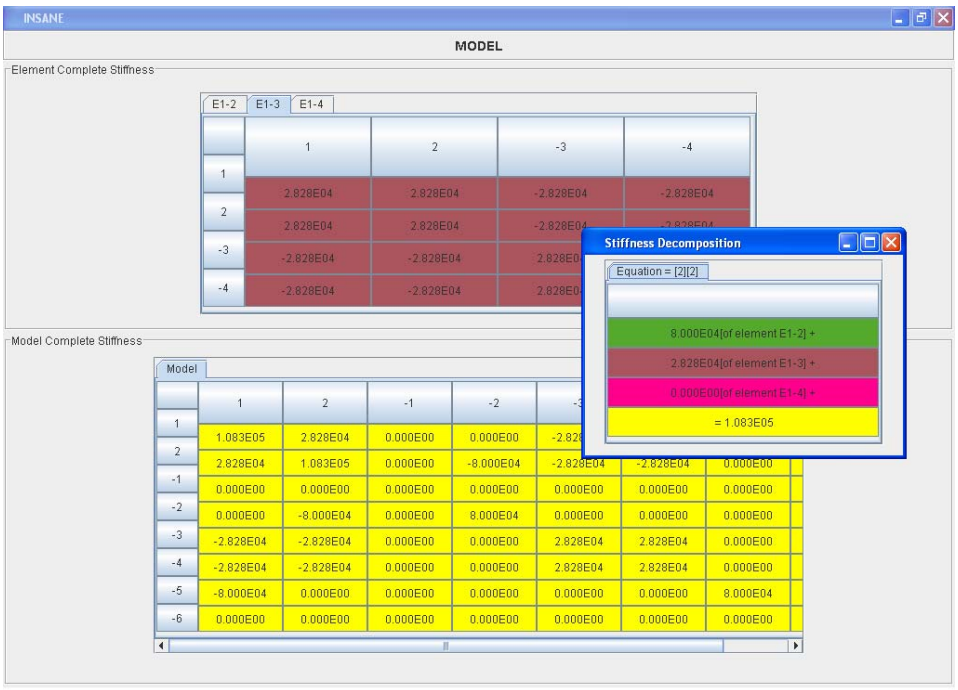


Figura 6.11: Matriz de rigidez completa do modelo.

Também é possível visualizar as equações de equilíbrio do modelo (figura 6.12). Após a montagem destas equações, pode-se solucionar o sistema, para os deslocamentos nodais incógnitos, que podem ser consultados, através da seleção de um nó do modelo (figura 6.13).

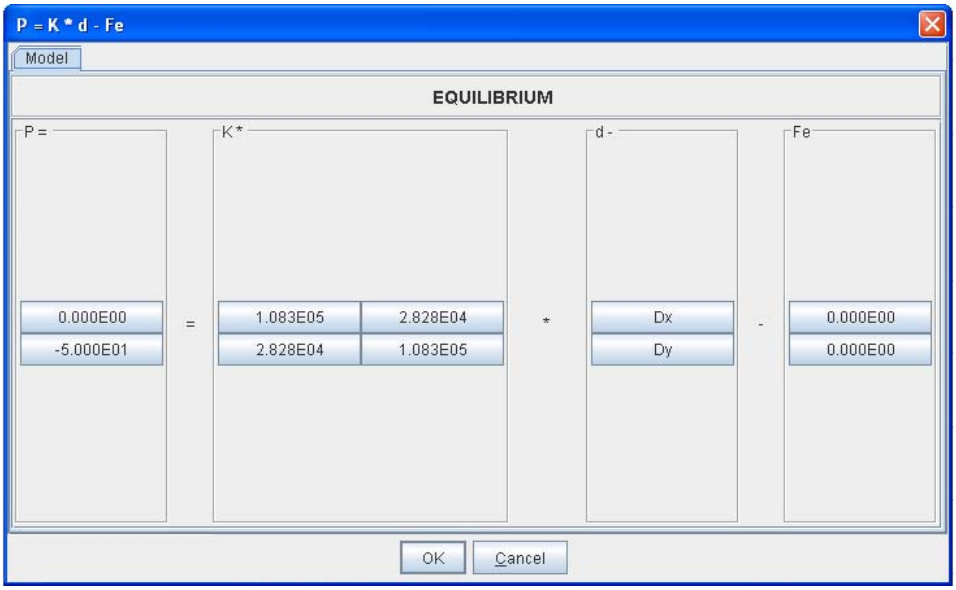


Figura 6.12: Equações de equilíbrio do modelo.

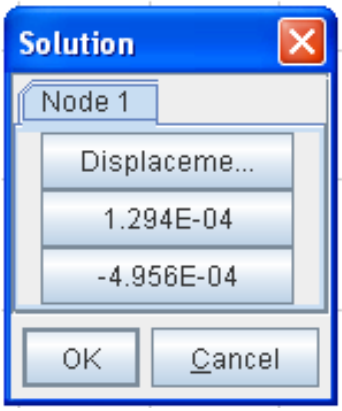


Figura 6.13: Deslocamentos nodais incógnitos.

Obtidos os deslocamentos nodais incógnitos, pode-se voltar às hipóteses dos elementos para conhecer as grandezas internas de cada elemento. Para um ponto qualquer interno ao elemento de treliça pode-se obter os deslocamentos (figura 6.14), a deformação axial (figura 6.15) e o esforço normal (figura 6.16). Também são apresentadas as forças de extremidade de cada elemento a partir dos deslocamentos nodais (figura 6.17).

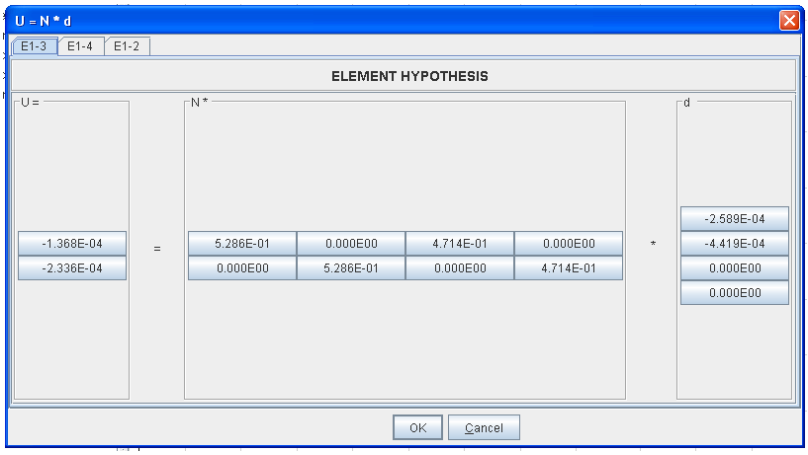


Figura 6.14: Deslocamentos em um ponto qualquer do elemento.

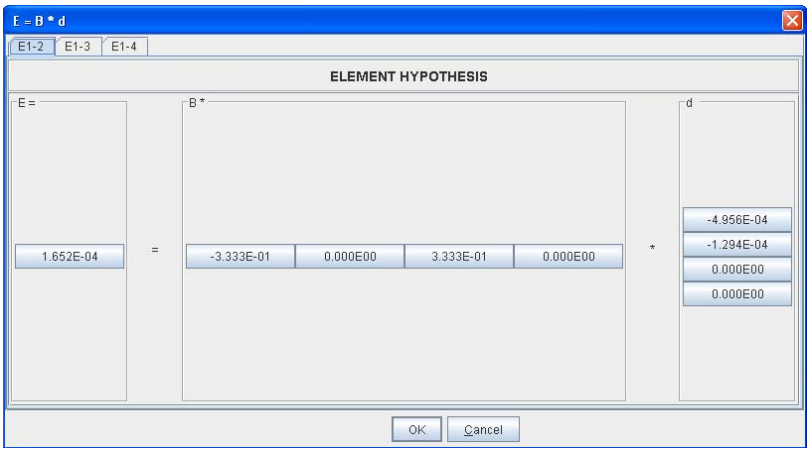


Figura 6.15: Deformação axial em um ponto qualquer do elemento.

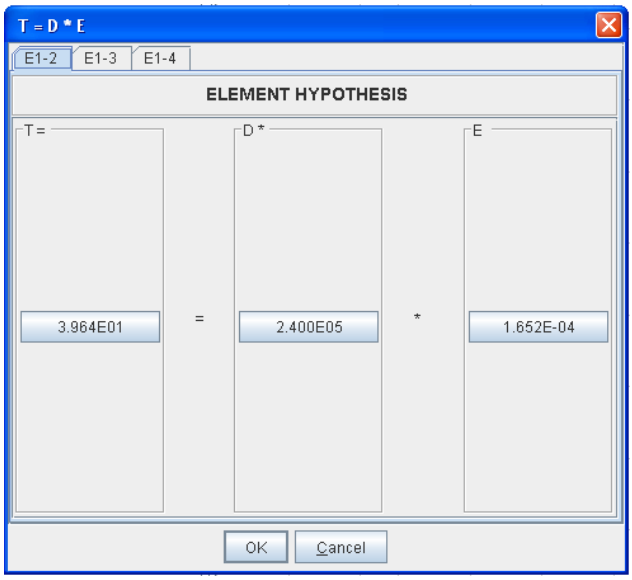


Figura 6.16: Força normal em um ponto qualquer do elemento.

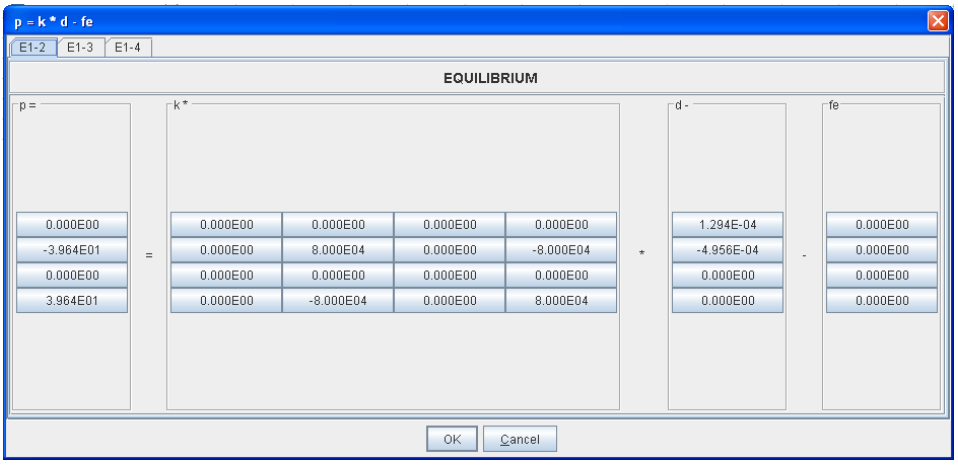


Figura 6.17: Forças de extremidade do elemento.

6.3 Viga

Este exemplo tem como objetivo mostrar os elementos finitos unidimensionais de viga. O modelo em questão está mostrado na figura 6.18.

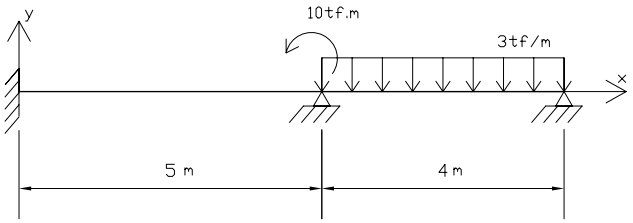


Figura 6.18: Viga em estudo.

A figura 6.19 mostra resultado do pré-processamento do modelo em estudo. Para processamento, adotou-se momento de inércia $I_z = 1,0 \times 10^{-2}\, m^4$ e módulo de elasticidade longitudinal $E = 1,0 \times 10^6\, tf/m^2$ para todos os elementos.

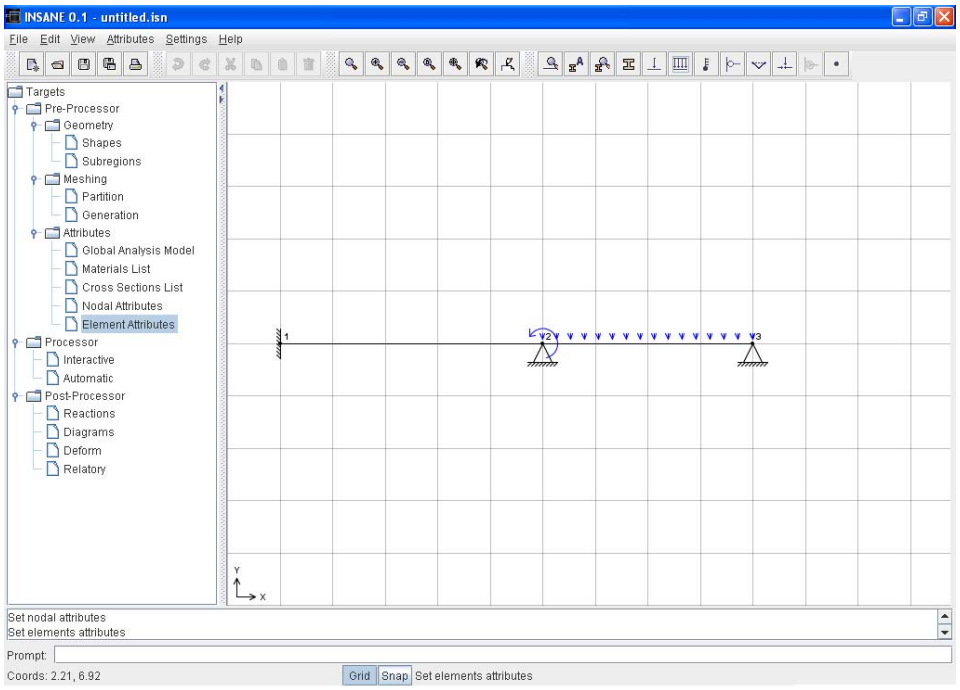


Figura 6.19: Modelo de viga obtido no pré-processador `INSANE`.

A solução interativa inicia-se pela numeração das equações do modelo conforme mostra a figura 6.20 .

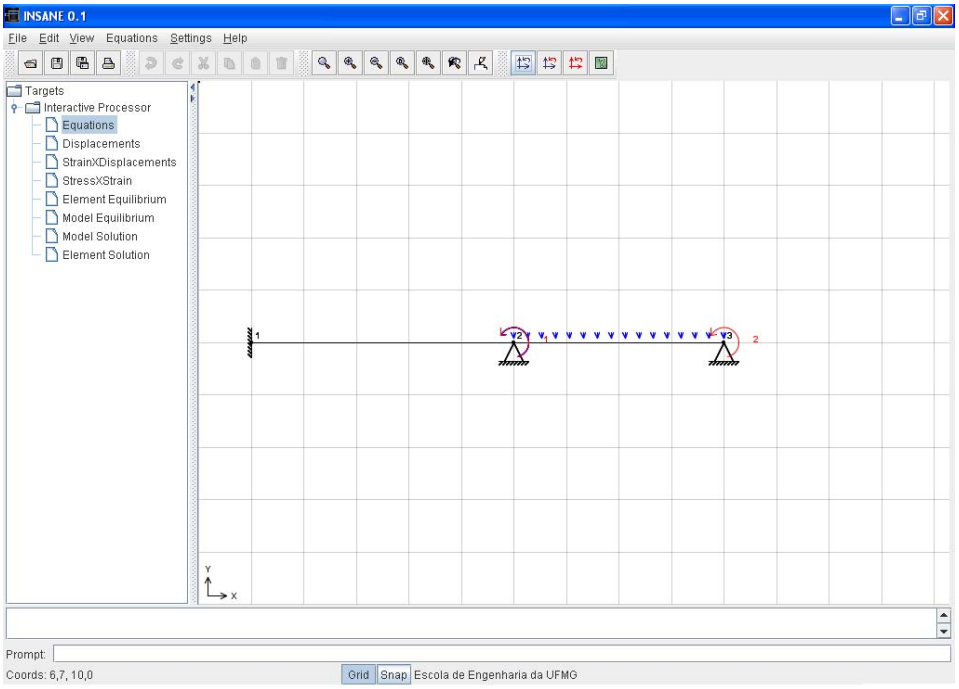


Figura 6.20: Numeração das Equações do modelo.

Nos modelos de viga, a partir da interpolação de deslocamentos $\{u\} = [N]\ \{\hat{d}\}^e$, tem-se:

$$\left\{ \begin{array}{c} v \\ \phi \end{array} \right\} = \left[\begin{array}{cccc} N_1 & \overline{N}_1 & N_2 & \overline{N}_2 \\ N_{1,x} & \overline{N}_{1,x} & N_{2,x} & \overline{N}_{2,x} \end{array} \right] \left\{ \begin{array}{c} \hat{d}_{1x} \\ \phi_1 \\ \hat{d}_{2x} \\ \phi_2 \end{array} \right\} \tag{6.3.1}$$

onde v e ϕ são os deslocamentos, $\hat{d}_{1x}$, ϕ_1, $\hat{d}_{2x}$ e ϕ_2 são , respectivamente, os deslocamentos dos nós inicial e final, e N_i a função de forma associada ao nó i, todos relativos ao sistema local de coordenadas do elemento.

A matriz $[N]$ pode ser visualizada na solução interativa para cada ponto do elemento, conforme mostra a figura 6.21.

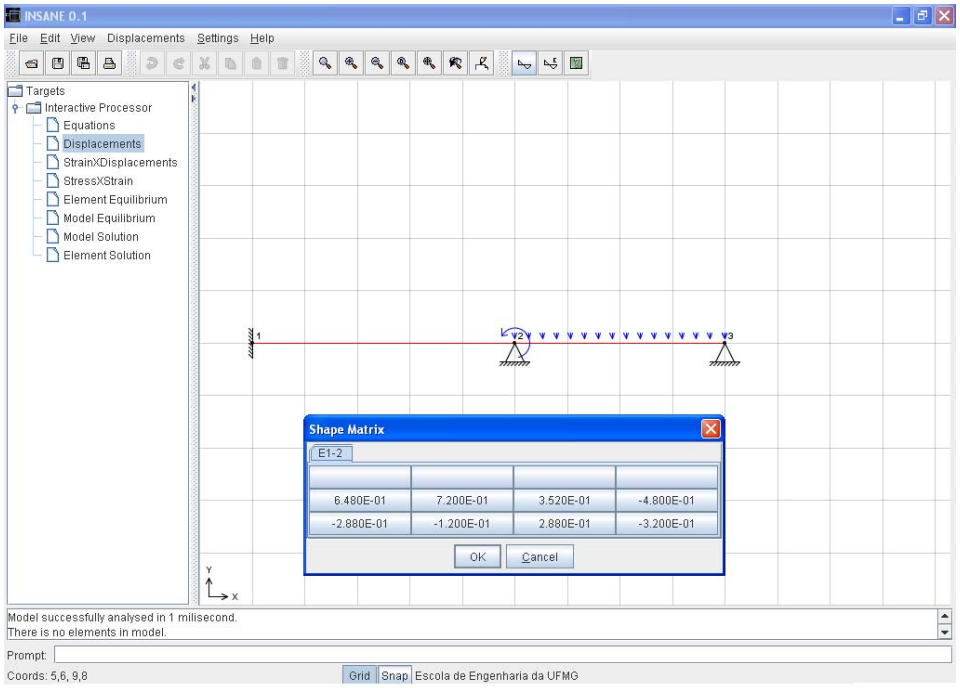


Figura 6.21: Funções de forma do elemento E1-2 no ponto x=2,0.

As deformações generalizadas (curvaturas) nas barras da viga podem ser obtidas através de $\{\chi\} = [B]\ \{\hat{d}\}^e$ ou:

$$\left\{ \chi \right\} = \left[N_{1,xx} \quad \overline{N}_{1,xx} \quad N_{2,xx} \quad \overline{N}_{2,xx} \right] \left\{ \hat{d} \right\} \tag{6.3.2}$$

sendo $[B]$ a matriz das derivadas primeiras das funções de forma em relação a x, e $\{\hat{d}\}$ o vetor dos deslocamentos nodais.

A matriz $[B]$ pode ser visualizada na solução interativa para cada ponto do elemento, conforme mostra a figura 6.22.

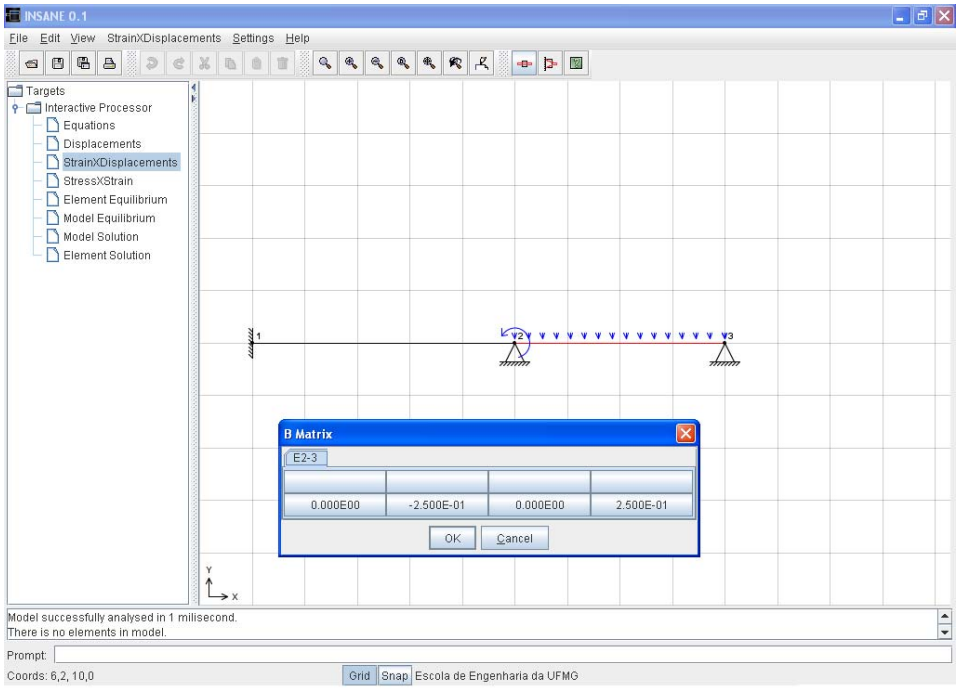


Figura 6.22: Matriz $[B]$ do elemento E2-3 no ponto x=2.

Os momentos fletores nas barras da viga podem ser obtidos através de $\{M\} = [EI_z]\{\chi\}$. A matriz $[EI_z]$ pode ser visualizada na solução interativa, conforme mostra a figura 6.23.

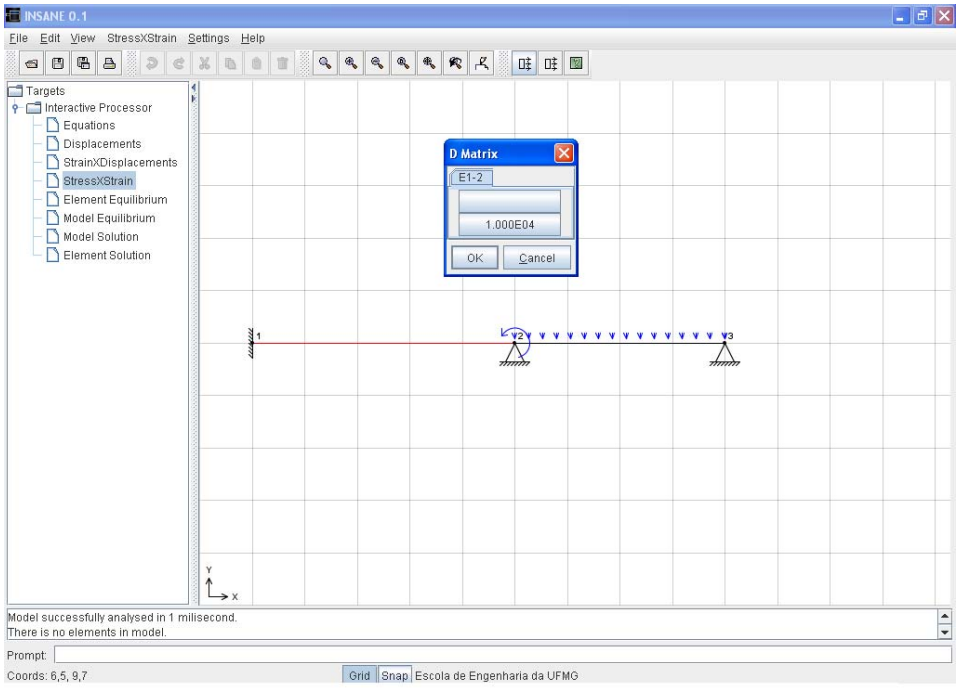
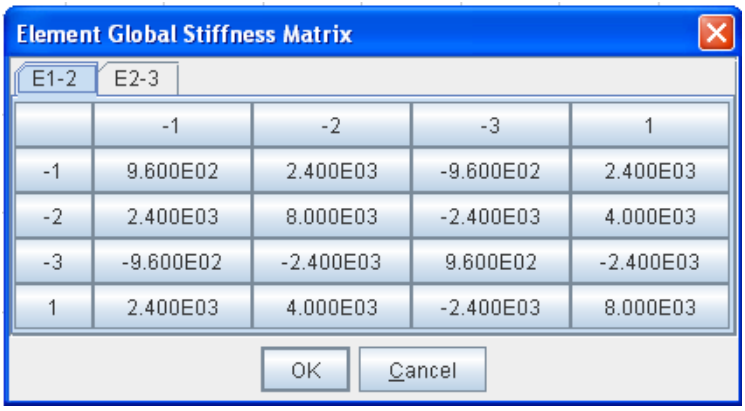


Figura 6.23: Matriz $[EI_z]$ do elemento E1-2.

Definidas as hipóteses dos elementos, a matriz de rigidez de cada elemento pode ser visualizada (figura 6.24), assim como o vetor de carregamento nodal equivalente (figura 6.25).



	-1	-2	-3	1
-1	9.600E02	2.400E03	-9.600E02	2.400E03
-2	2.400E03	8.000E03	-2.400E03	4.000E03
-3	-9.600E02	-2.400E03	9.600E02	-2.400E03
1	2.400E03	4.000E03	-2.400E03	8.000E03

Figura 6.24: Matriz de rigidez completa de cada elemento do modelo.

Aplicando as condições de contorno, a matriz de rigidez reduzida de cada elemento é apresentada (figura 6.26), assim como o vetor de carregamento nodal equivalente reduzido (figura 6.27).

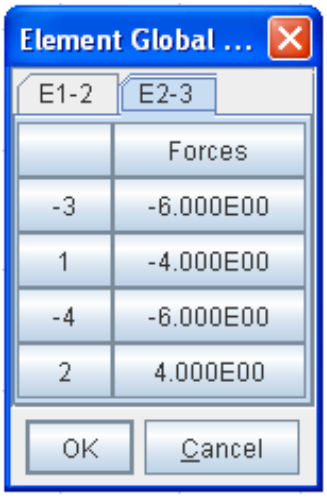


Figura 6.25: Vetor de carregamento nodal equivalente de cada elemento do modelo.

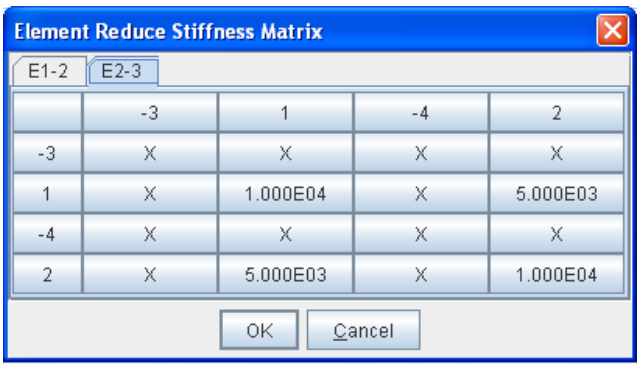


Figura 6.26: Matriz de rigidez reduzida de cada elemento do modelo.

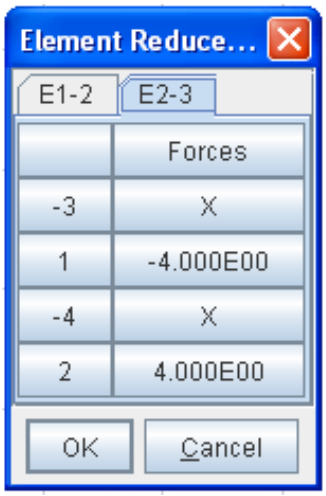


Figura 6.27: Vetor de carregamento nodal equivalente reduzido de cada elemento do modelo.

A partir do equilíbrio de cada elemento, é gerado o equilíbrio do modelo. Este também pode ser consultado, sendo mostrada a contribuição de cada elemento do problema, à rigidez total do modelo. A figura 6.28 mostra a matriz de rigidez completa do modelo e a visualização da contribuição de cada elemento para formar um dos termos da mesma. A matriz de rigidez reduzida também pode ser consultada. Para esta opção também é possível visualizar as contribuições de cada elemento. O mesmo pode ser aplicado à consulta dos vetores de força completa e reduzida.

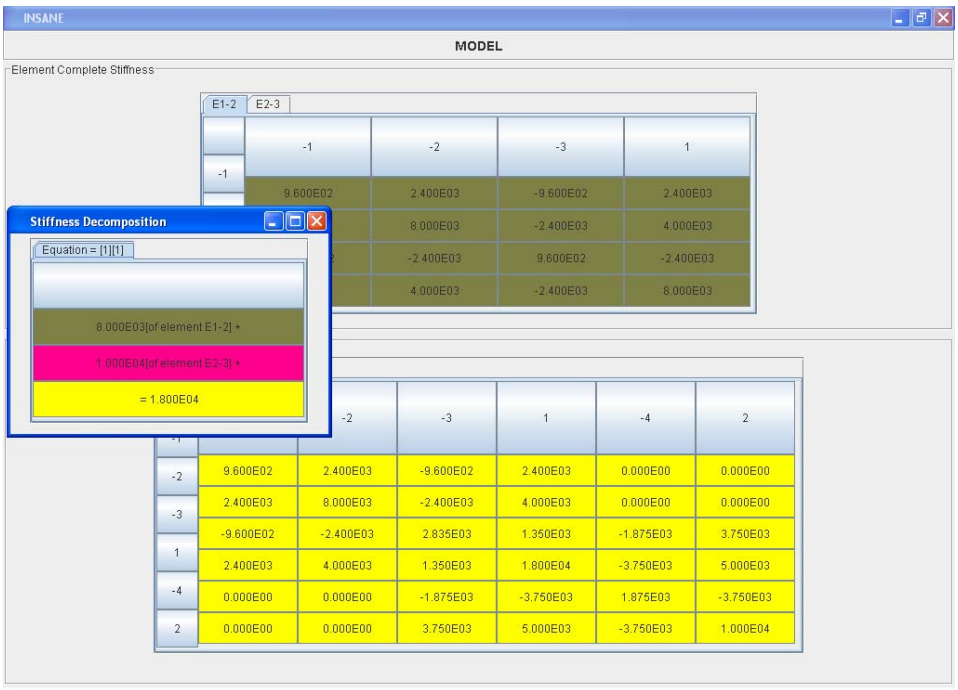


Figura 6.28: Matriz de rigidez completa do modelo.

Também é possível visualizar as equações de equilíbrio do modelo (figura 6.29). Após a montagem destas equações, pode-se solucionar o sistema, para os deslocamentos nodais incógnitos, que podem ser consultados, através da seleção de um nó do modelo (figura 6.30).

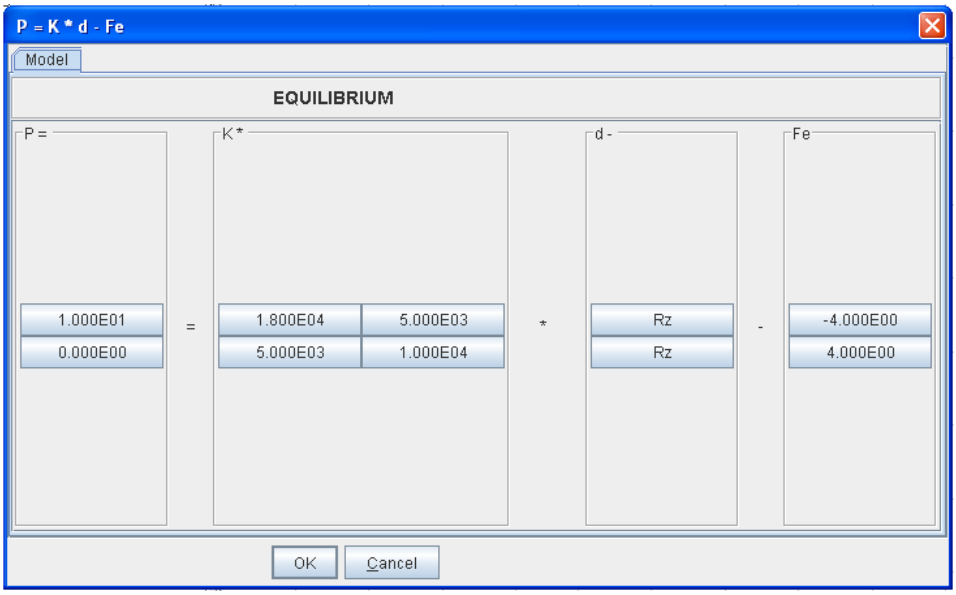


Figura 6.29: Equações de equilíbrio do modelo.

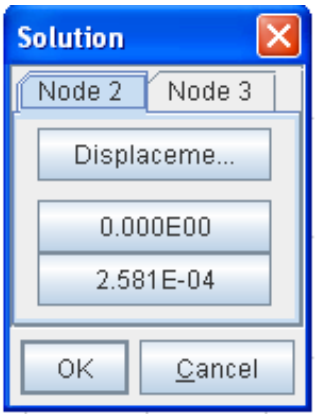


Figura 6.30: Deslocamentos nodais incógnitos.

Obtidos os deslocamentos nodais incógnitos, pode-se voltar às hipóteses dos elementos para conhecer as grandezas internas de cada elemento. Para um ponto qualquer do elemento de viga pode-se obter os deslocamentos (figura 6.31), a curvatura (figura 6.32) e o momento fletor aproximados (figura 6.33). Também são apresentadas as forças de extremidade de cada elemento a partir dos deslocamentos nodais (figura 6.34).

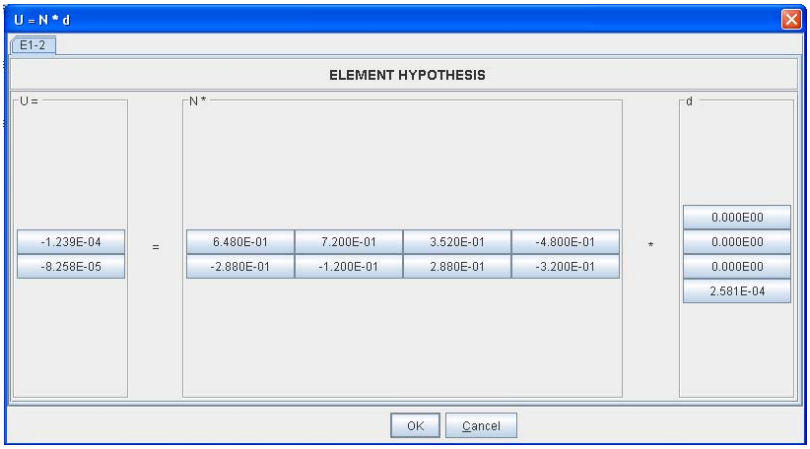


Figura 6.31: Deslocamentos em um ponto qualquer do elemento.

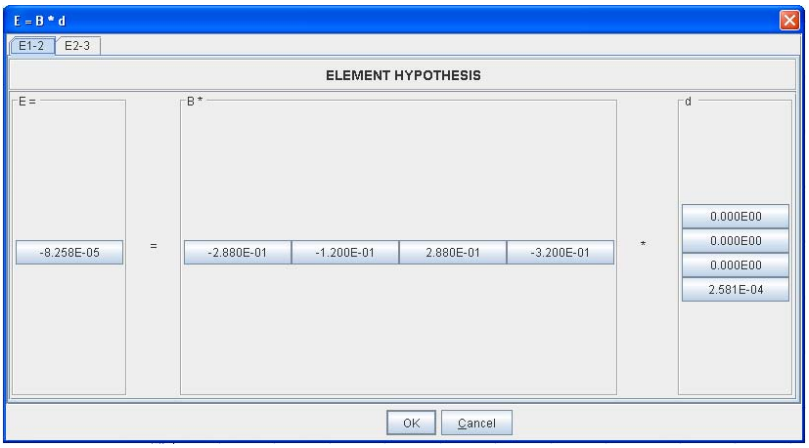


Figura 6.32: Curvatura em um ponto qualquer do elemento.

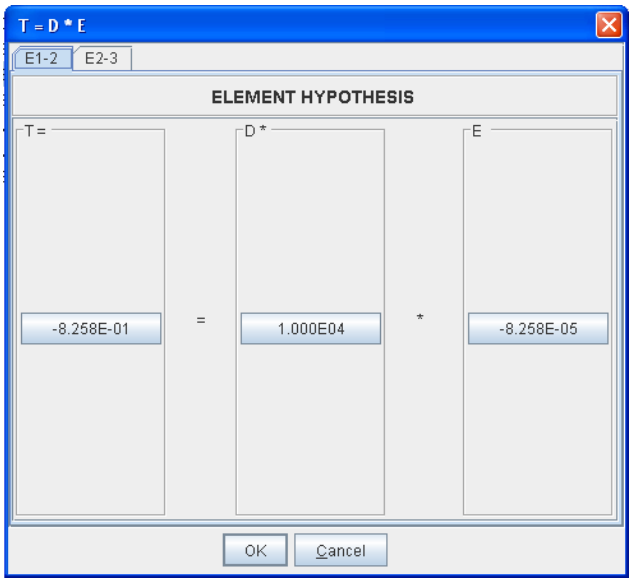


Figura 6.33: Momento Fletor em um ponto qualquer do elemento.

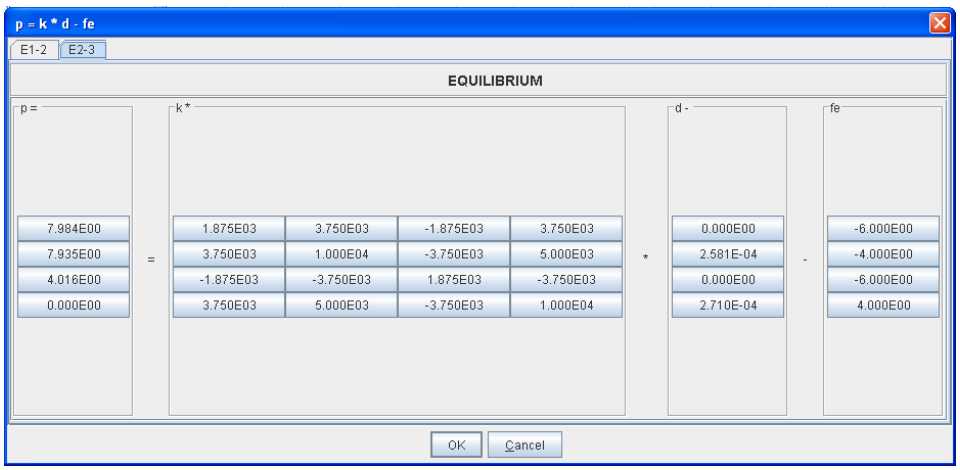


Figura 6.34: Forças de extremidade do elemento.

6.4 Pórtico Plano

Este exemplo tem como objetivo mostrar os elementos finitos unidimensionais de pórtico plano. O modelo em questão é mostrado na figura 6.35.

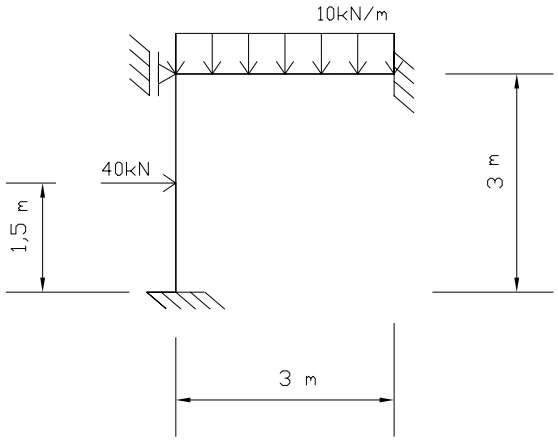


Figura 6.35: Pórtico Plano em estudo.

A figura 6.36 mostra o modelo em estudo, gerado pelo pré-processador `INSANE`, onde se pode visualizar os nós, elementos, cargas e condições de contorno. Para processamento, adotou-se área da seção transversal $A = 0{,}06\ m^2$, momento de inércia da seção transversal $I = 0{,}03\ m^4$ e módulo de elasticidade longitudinal do material $E = 2,1 \times 10^7\ kN/m^2$ para todos elementos do modelo.

Na figura 6.37 são mostrados os sistemas de coordenadas locais adotados para cada elemento do modelo.

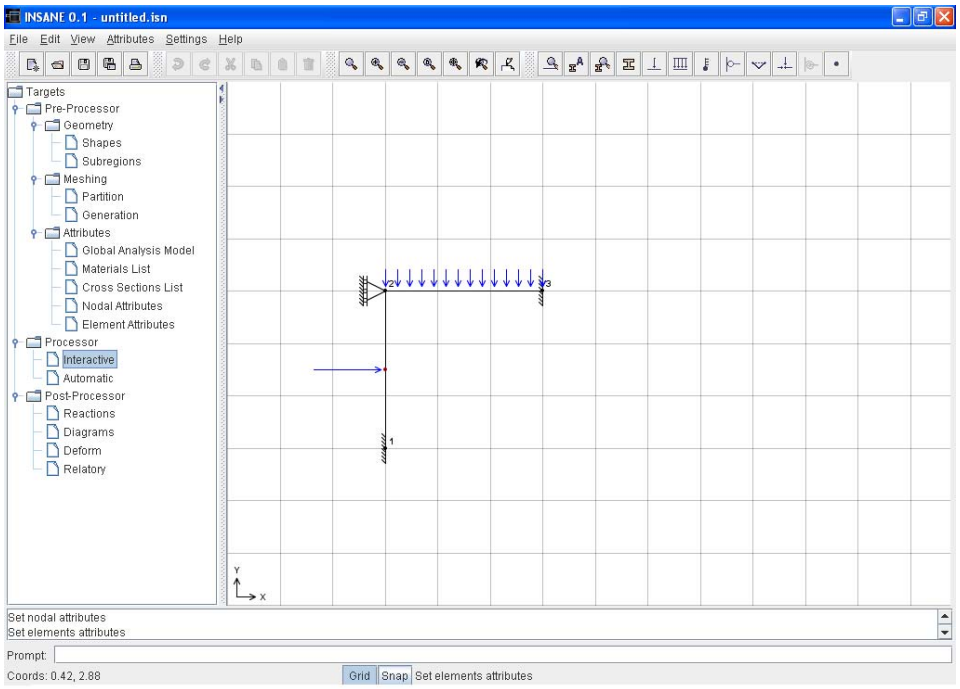


Figura 6.36: Modelo de pórtico plano obtido no pré-processador `INSANE`.

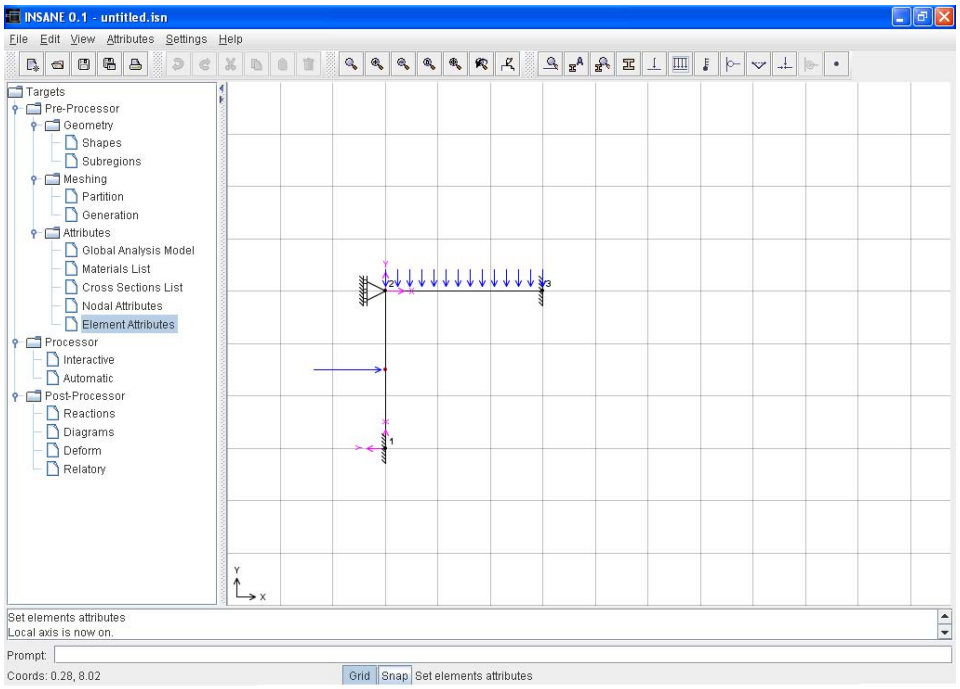


Figura 6.37: Sistema de coordenadas locais.

A solução interativa inicia-se pela numeração das equações do modelo conforme mostra a figura 6.38.

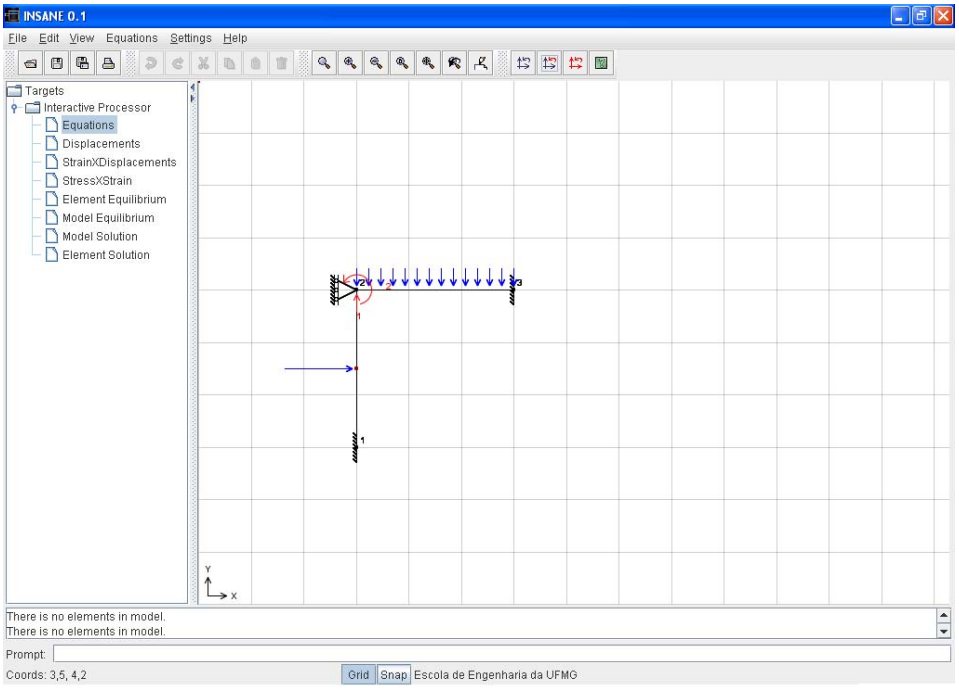


Figura 6.38: Numeração das Equações do modelo.

Nos modelos de pórtico plano, a partir da interpolação de deslocamentos $\{u\} = [N]\,\{\hat{d}\}^e$, tem-se:

$$\begin{Bmatrix} u \\ v \\ \phi \end{Bmatrix} = \begin{bmatrix} N_1 & 0 & 0 & N_2 & 0 & 0 \\ 0 & \overline{N}_1 & \overline{\overline{N}}_1 & 0 & \overline{N}_2 & \overline{\overline{N}}_2 \\ 0 & \overline{N}_{1,x} & \overline{\overline{N}}_{1,x} & 0 & \overline{N}_{2,x} & \overline{\overline{N}}_{2,x} \end{bmatrix} \begin{Bmatrix} \hat{d}_{1x} \\ \hat{d}_{1y} \\ \phi_1 \\ \hat{d}_{2x} \\ \hat{d}_{2y} \\ \phi_2 \end{Bmatrix} \tag{6.4.1}$$

onde u, v e ϕ são os deslocamentos, $\hat{d}_{1x}$, $\hat{d}_{1y}$, ϕ_1, $\hat{d}_{2x}$, $\hat{d}_{2y}$ e ϕ_2 são, respectivamente, os deslocamentos dos nós inicial e final, e N_i a função de forma associada ao nó i, todos relativos ao sistema local de coordenadas do elemento.

A matriz $[N]$ pode ser visualizada na solução interativa para cada ponto do elemento, conforme mostra a figura 6.39.

As deformações generalizadas nas barras do pórtico plano podem ser obtidas através

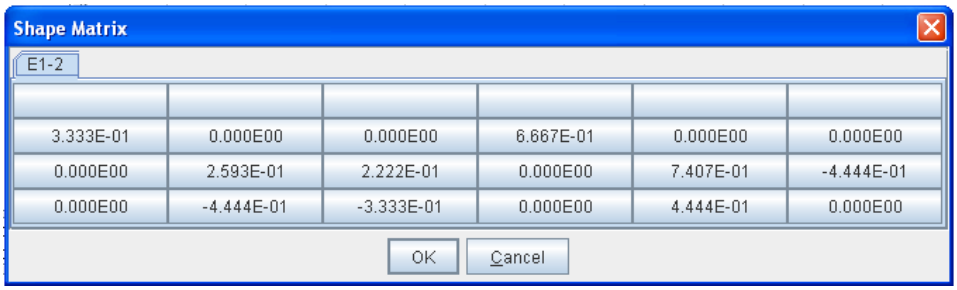


Figura 6.39: Funções de forma do elemento.

de $\{\chi\} = [B]\ \{\hat{d}\}^e$ ou:

$$\left\{ \begin{array}{c} \varepsilon_x \\ \chi \end{array} \right\} = \left[\begin{array}{cccccc} N_{1,x} & 0 & 0 & N_{2,x} & 0 & 0 \\ 0 & \overline{N}_{1,xx} & \overline{\overline{N}}_{1,xx} & 0 & \overline{N}_{2,xx} & \overline{\overline{N}}_{2,xx} \end{array} \right] \left\{ \begin{array}{c} \\ \hat{d} \\ \\ \end{array} \right\} \tag{6.4.2}$$

sendo $[B]$ a matriz das derivadas primeiras das funções de forma em relação a x, e $\{\hat{d}\}$ o vetor dos deslocamentos nodais.

A matriz $[B]$ pode ser visualizada na solução interativa para cada ponto do elemento, conforme mostra a figura 6.40.

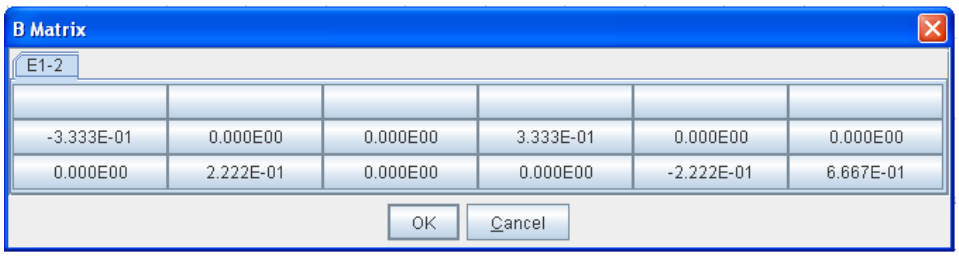


Figura 6.40: Matriz B do elemento.

A força normal (N) e o momento fletor (M) nas barras do pórtico plano podem ser obtidos através de:

$$\left\{ \begin{array}{c} N \\ M \end{array} \right\} = \left[\begin{array}{cc} EA & 0 \\ 0 & EI_z \end{array} \right] \left\{ \begin{array}{c} \varepsilon_x \\ \chi \end{array} \right\} \tag{6.4.3}$$

Os valores de EA e EI_z podem ser visualizados na solução interativa, através da matriz constitutiva, conforme mostra a figura 6.41.

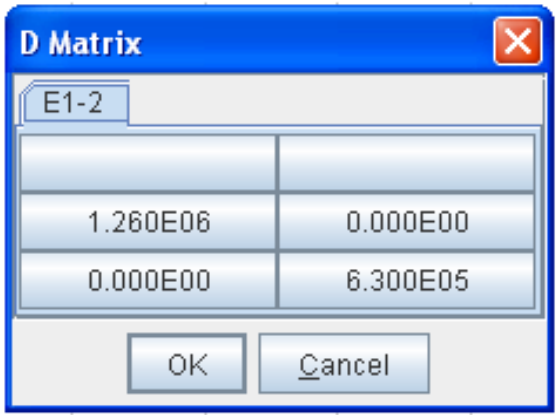
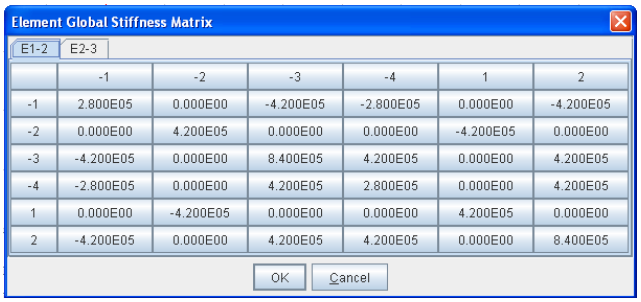


Figura 6.41: Matriz constitutiva do elemento.

Definidas as hipóteses dos elementos, a matriz de rigidez de cada elemento pode ser visualizada (figura 6.42), assim como o vetor de carregamento nodal equivalente (figura 6.43).



	-1	-2	-3	-4	1	2
-1	2.800E05	0.000E00	-4.200E05	-2.800E05	0.000E00	-4.200E05
-2	0.000E00	4.200E05	0.000E00	0.000E00	-4.200E05	0.000E00
-3	-4.200E05	0.000E00	8.400E05	4.200E05	0.000E00	4.200E05
-4	-2.800E05	0.000E00	4.200E05	2.800E05	0.000E00	4.200E05
1	0.000E00	-4.200E05	0.000E00	0.000E00	4.200E05	0.000E00
2	-4.200E05	0.000E00	4.200E05	4.200E05	0.000E00	8.400E05

Figura 6.42: Matriz de rigidez completa de cada elemento do modelo.

Aplicando as condições de contorno, a matriz de rigidez reduzida de cada elemento é apresentada (figura 6.44), assim como o vetor de carregamento nodal equivalente reduzido (figura 6.45).

Figura 6.43: Vetor de carregamento nodal equivalente de cada elemento do modelo.

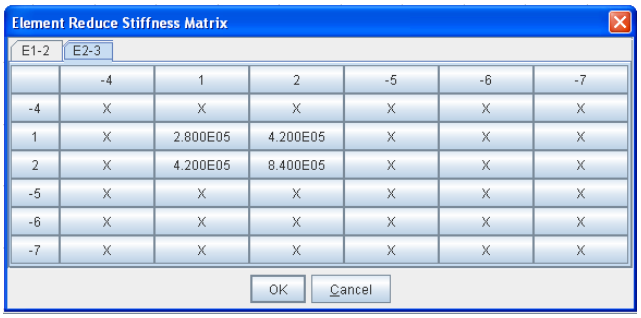


Figura 6.44: Matriz de rigidez reduzida de cada elemento do modelo.

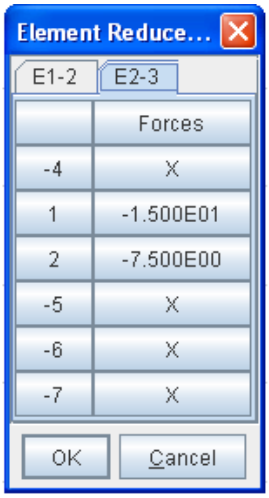


Figura 6.45: Vetor de carregamento nodal equivalente reduzido de cada elemento do modelo.

A partir do equilíbrio de cada elemento, é gerado o equilíbrio do modelo. Este também pode ser consultado, sendo mostrada a contribuição de cada elemento do problema, à rigidez total do modelo. A figura 6.46 mostra a matriz de rigidez completa do modelo e a visualização da contribuição de cada elemento para formar um dos termos da mesma. A matriz de rigidez reduzida também pode ser consultada. Para esta opção também é possível visualizar as contribuições de cada elemento. O mesmo pode ser aplicado à consulta dos vetores de força completa e reduzida.

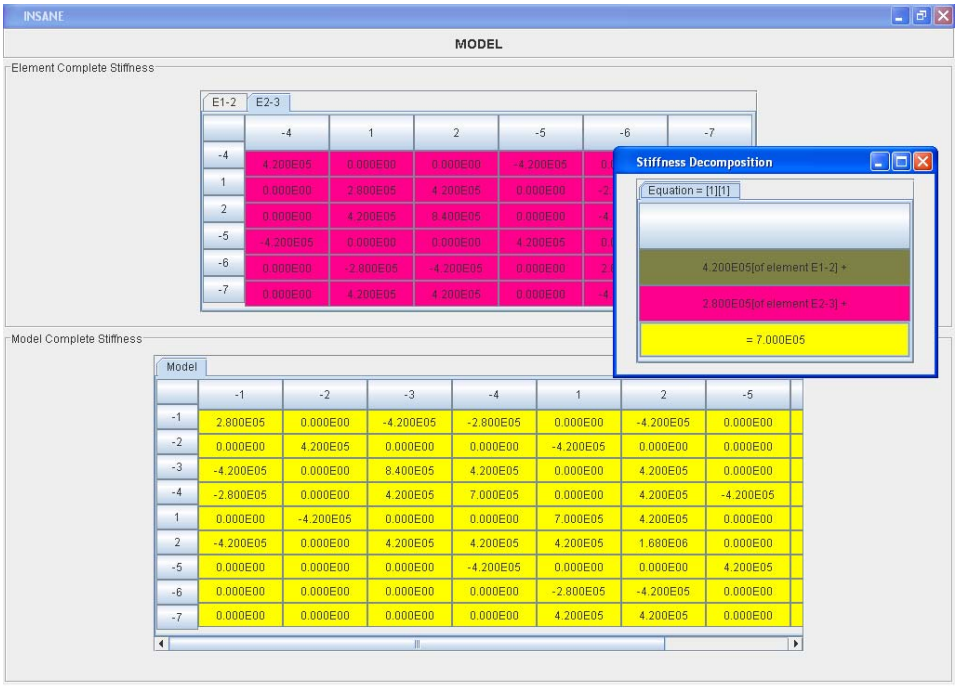


Figura 6.46: Matriz de rigidez completa do modelo.

Também é possível visualizar as equações de equilíbrio do modelo (figura 6.47). Após montagem destas equações, pode-se solucionar o sistema, para os deslocamentos nodais incógnitos, que podem ser consultados, através da seleção de um nó do modelo (figura 6.48).

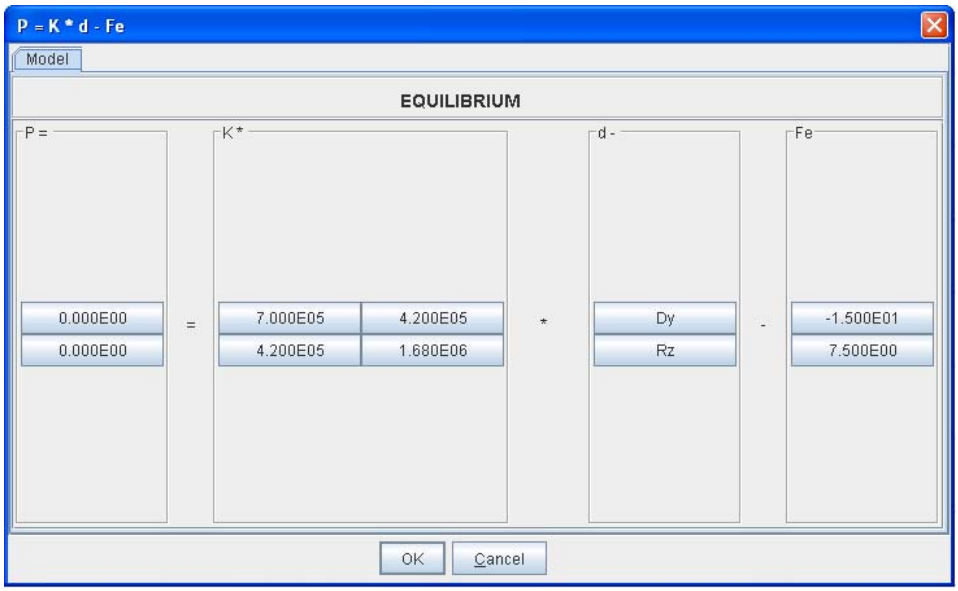


Figura 6.47: Equações de equilíbrio do modelo.

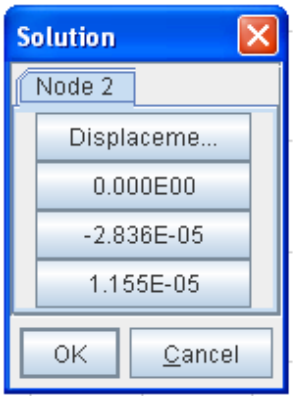


Figura 6.48: Deslocamentos nodais incógnitos.

Obtidos os deslocamentos nodais incógnitos, pode-se voltar às hipóteses dos elementos para conhecer as grandezas internas aproximadas de cada elemento. Para um ponto qualquer no interior do elemento pode-se obter os deslocamentos (figura 6.49), as deformações (figura 6.50) e os esforços (figura 6.51) aproximados.

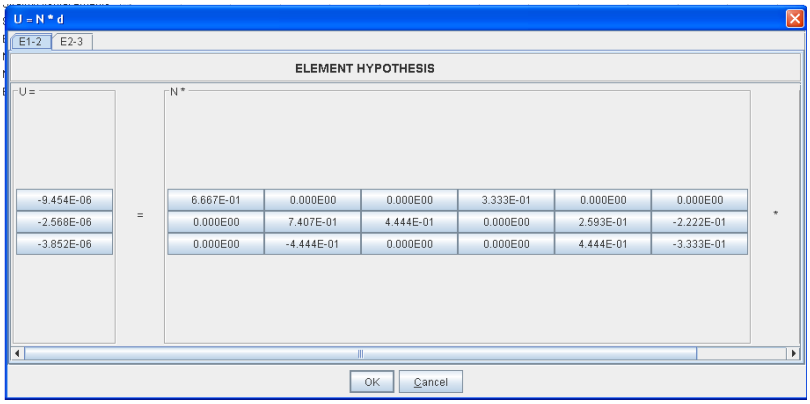


Figura 6.49: Deslocamentos em um ponto qualquer do elemento.

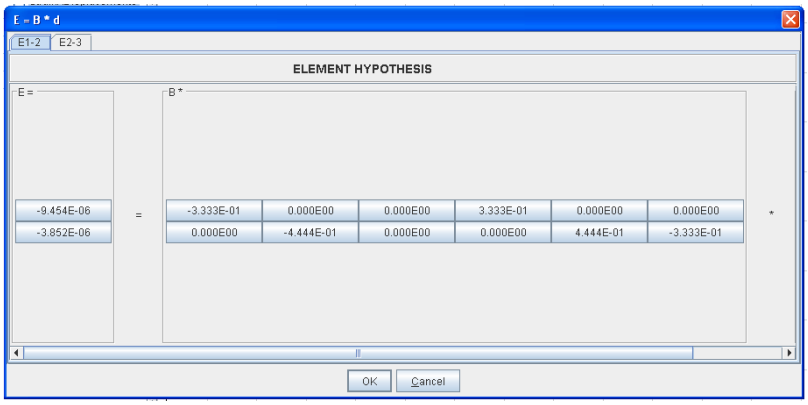


Figura 6.50: Deformações em um ponto qualquer do elemento.

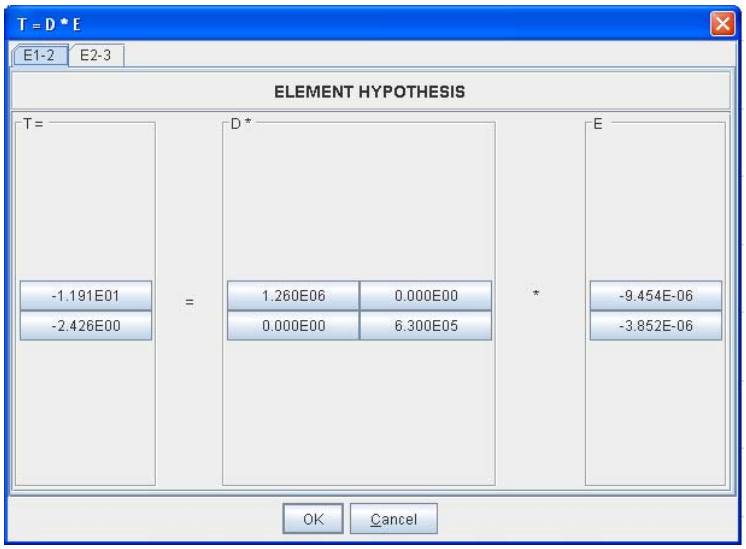


Figura 6.51: Esforços em um ponto qualquer do elemento.

6.5 Grelha

Este exemplo tem como objetivo mostrar os elementos finitos unidimensionais de grelha. O modelo em questão é mostrado na figura 6.52.

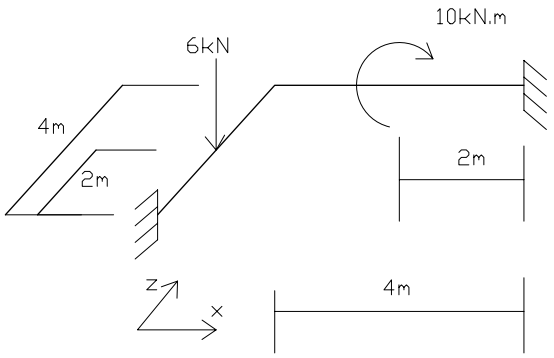


Figura 6.52: Grelha em estudo.

A figura 6.53 mostra o modelo em estudo, gerado pelo pré-processador `INSANE`, onde se pode observar, em projeção horizontal (no plano XZ) os nós, elementos, cargas e condições de contorno. Na figura são mostrados os sistemas locais de coordenadas adotados para os dois elementos do modelo.

Para o processamento, adotou-se constante de torção $Ix = 0{,}01\ m^4$, momento de inércia $Iz = 0{,}01\ m^4$, módulo de elasticidade longitudinal $E = 1,0 \times 10^6\, kN/m^2$ e módulo de elasticidade transversal $G = 0,5 \times 10^6\, kN/m^2$ para todas as barras.

Na figura 6.54 são mostrados os sistemas de coordenadas locais adotados para cada elemento do modelo.

A solução interativa inicia-se pela numeração das equações do modelo conforme mostra a figura 6.55.

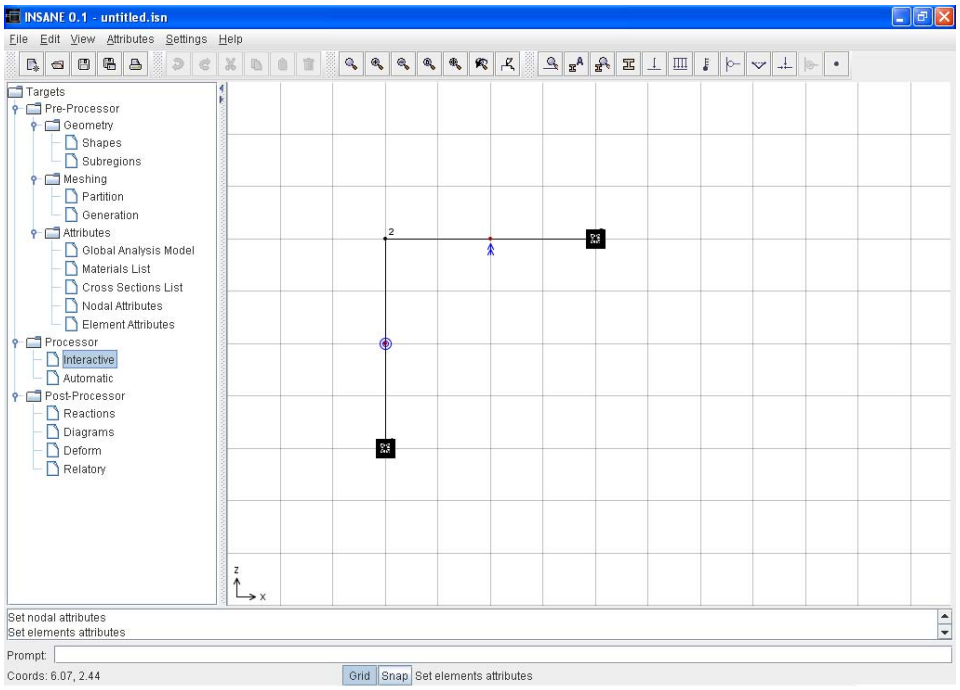


Figura 6.53: Modelo de grelha obtido no pré-processador `INSANE`.

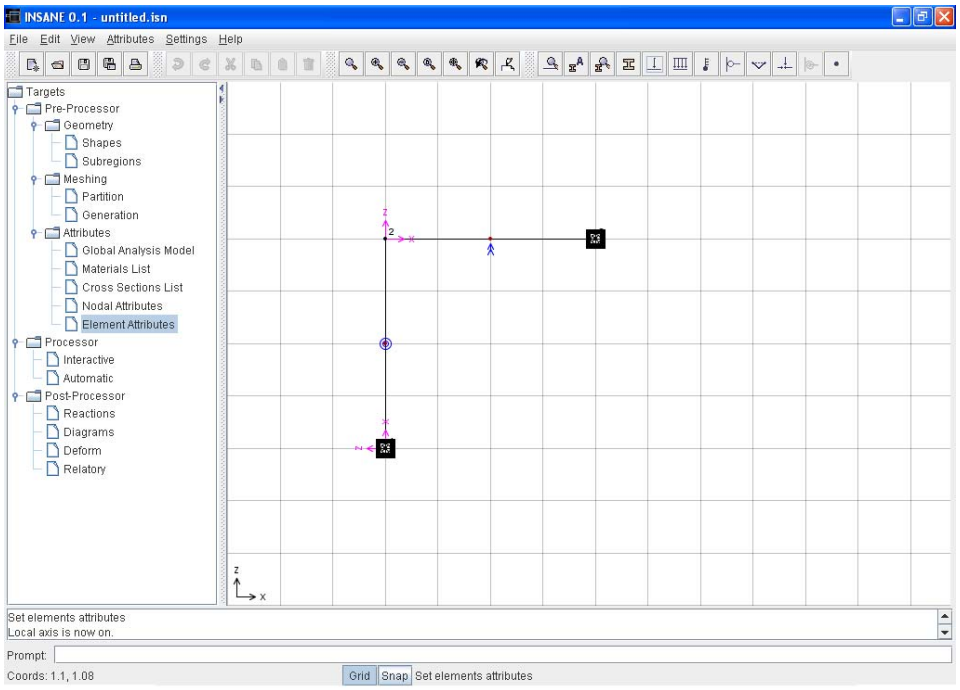


Figura 6.54: Sistema de coordenadas locais.

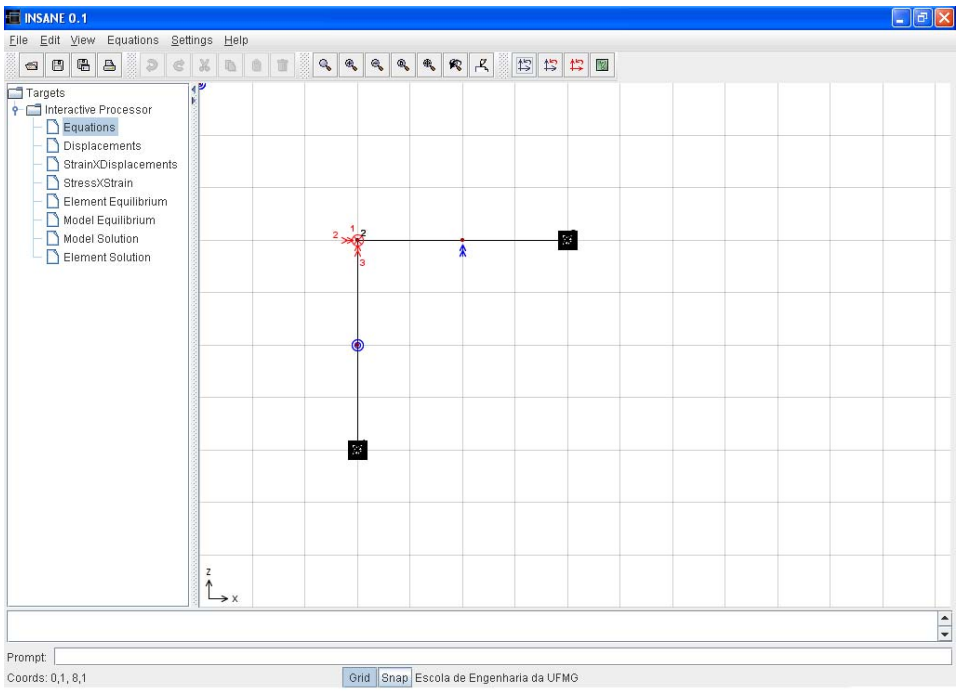


Figura 6.55: Numeração das Equações do modelo.

Nos modelos de grelha, a partir da interpolação de deslocamentos $u = [N]\ \{\hat{d}\}^e$ tem-se:

$$\begin{Bmatrix} \phi_x \\ v \\ \phi_z \end{Bmatrix} = \begin{bmatrix} N_1 & 0 & 0 & N_2 & 0 & 0 \\ 0 & \overline{N}_1 & \overline{\overline{N}}_1 & 0 & \overline{N}_2 & \overline{\overline{N}}_2 \\ 0 & 0 & N_1 & 0 & 0 & N_2 \end{bmatrix} \begin{Bmatrix} \phi_{1x} \\ \hat{d}_{1y} \\ \phi_{1z} \\ \phi_{2x} \\ \hat{d}_{2y} \\ \phi_{2z} \end{Bmatrix} \tag{6.5.1}$$

onde ϕ_x, v e ϕ_z são os deslocamentos, ϕ_{1x}, $\hat{d}_{1y}$, ϕ_{1z}, ϕ_{2x}, $\hat{d}_{2y}$ e ϕ_{2z} são, respectivamente, os deslocamentos dos nós inicial e final, e N_i a função de forma associada ao nó i, todos relativos ao sistema local de coordenadas do elemento.

A matriz $[N]$ pode ser visualizada na solução interativa para cada ponto do elemento, conforme mostra a figura 6.56.

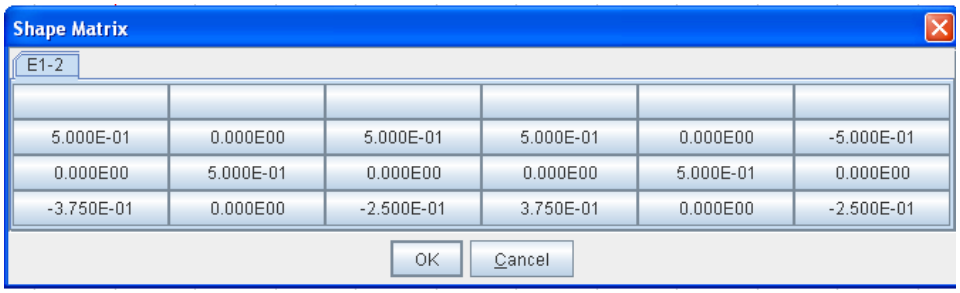


Figura 6.56: Funções de forma para o ponto x=2,0 do elemento E1-2.

As deformações generalizadas nas barras da grelha podem ser obtidas através de $\{\chi\} = [B]\ \{d\}^e$, ou:

$$\left\{ \begin{array}{c} \gamma \\ \chi \end{array} \right\} = \left[\begin{array}{cccccc} N_1' & 0 & 0 & N_2' & 0 & 0 \\ 0 & \overline{N}_{1,xx} & \overline{\overline{N}}_{1,xx} & 0 & \overline{N}_{2,xx} & \overline{\overline{N}}_{2,xx} \end{array} \right] \left\{ \begin{array}{c} \hat{d} \end{array} \right\} \tag{6.5.2}$$

sendo $[B]$ a matriz das derivadas primeiras das funções de forma em relação a x, e $\{\hat{d}\}$ o vetor dos deslocamentos nodais.

A matriz $[B]$ pode ser visualizada na solução interativa para cada ponto do elemento, conforme mostra a figura 6.57.

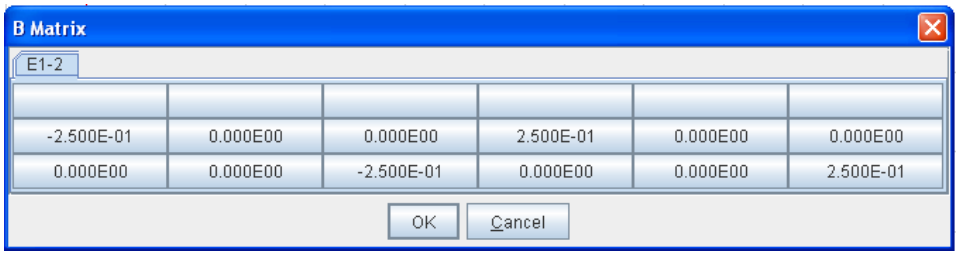


Figura 6.57: Matriz $[B]$ do elemento.

Os momentos de torção e de flexão nas barras da grelha podem ser obtidos através de:

$$\left\{ \begin{array}{c} T \\ M \end{array} \right\} = \left[\begin{array}{cc} GI_x & 0 \\ 0 & EI_z \end{array} \right] \left\{ \begin{array}{c} \gamma \\ \chi \end{array} \right\} \tag{6.5.3}$$

Os valores de GI_x e EI_z podem ser visualizados na solução interativa, através da matriz constitutiva do elemento, conforme mostra a figura 6.58.

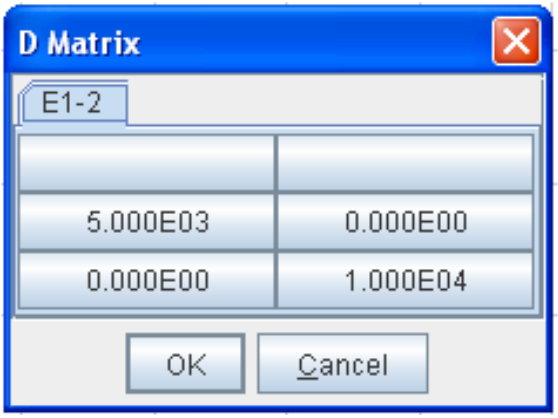
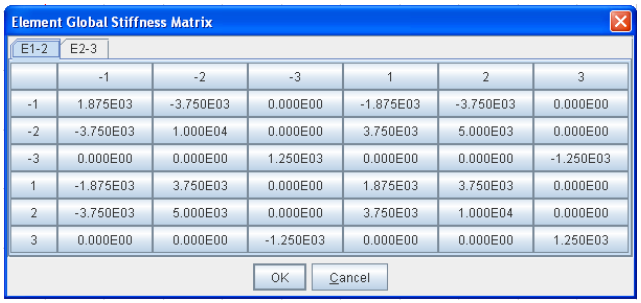


Figura 6.58: Matriz constitutiva do elemento E1-2.

Definidas as hipóteses dos elementos, a matriz de rigidez de cada elemento pode ser visualizada (figura 6.59), assim como o vetor de carregamento nodal equivalente (figura 6.60).



	-1	-2	-3	1	2	3
-1	1.875E03	-3.750E03	0.000E00	-1.875E03	-3.750E03	0.000E00
-2	-3.750E03	1.000E04	0.000E00	3.750E03	5.000E03	0.000E00
-3	0.000E00	0.000E00	1.250E03	0.000E00	0.000E00	-1.250E03
1	-1.875E03	3.750E03	0.000E00	1.875E03	3.750E03	0.000E00
2	-3.750E03	5.000E03	0.000E00	3.750E03	1.000E04	0.000E00
3	0.000E00	0.000E00	-1.250E03	0.000E00	0.000E00	1.250E03

Figura 6.59: Matriz de rigidez completa de cada elemento do modelo.

Aplicando as condições de contorno, a matriz de rigidez reduzida de cada elemento é apresentada (figura 6.61), assim como o vetor de carregamento nodal equivalente reduzido (figura 6.62).

Figura 6.60: Vetor de carregamento nodal equivalente de cada elemento do modelo.

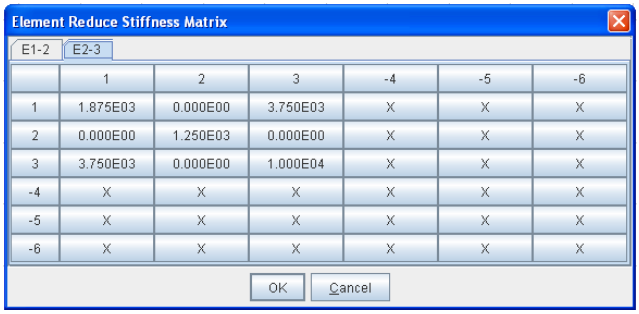


Figura 6.61: Matriz de rigidez reduzida de cada elemento do modelo.

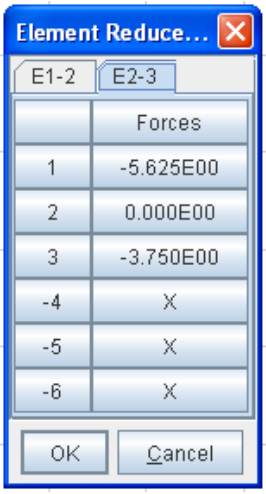


Figura 6.62: Vetor de carregamento nodal equivalente reduzido de cada elemento do modelo.

A partir do equilíbrio de cada elemento, é gerado o equilíbrio do modelo. Este também pode ser consultado, sendo mostrada a contribuição de cada elemento do problema, à rigidez total do modelo. A figura 6.63 mostra a matriz de rigidez completa do modelo e a visualização da contribuição de cada elemento para formar um dos termos da mesma. A matriz de rigidez reduzida também pode ser consultada. Para esta opção também é possível visualizar as contribuições de cada elemento. O mesmo pode ser aplicado à consulta dos vetores de força completa e reduzida.

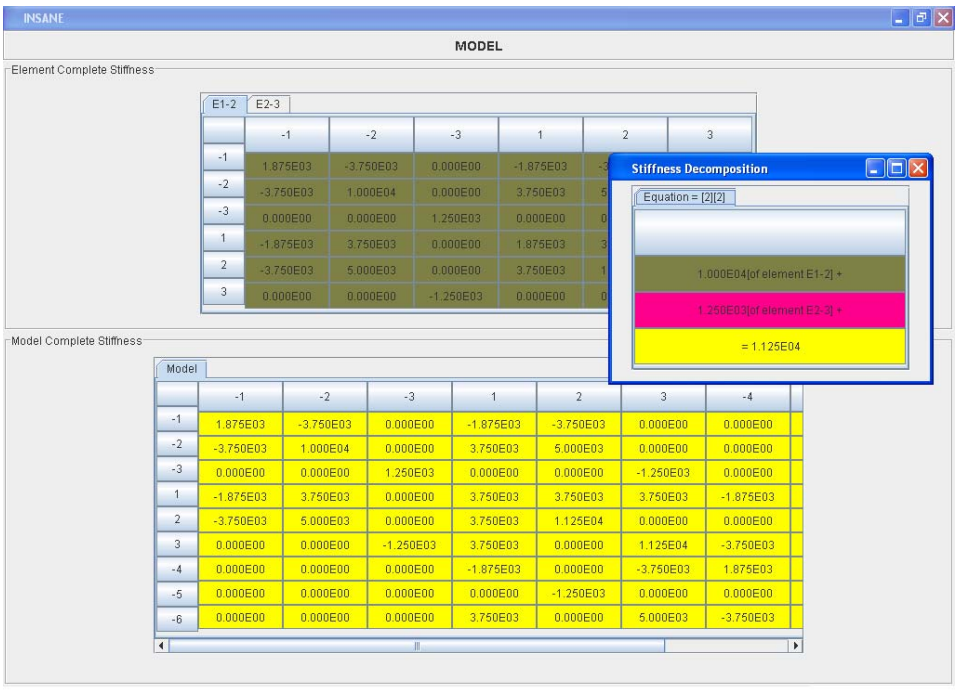


Figura 6.63: Matriz de rigidez completa do modelo.

Também é possível visualizar as equações de equilíbrio do modelo (figura 6.64). Após montagem destas equações, pode-se solucionar o sistema, para os deslocamentos nodais incógnitos, que podem ser consultados, através da seleção de um nó do modelo (figura 6.65).

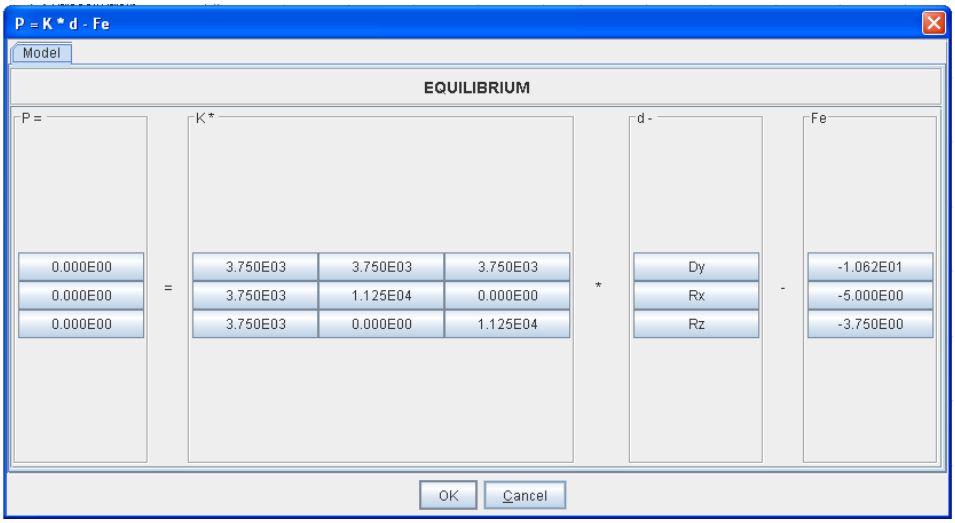


Figura 6.64: Equilíbrio do modelo.

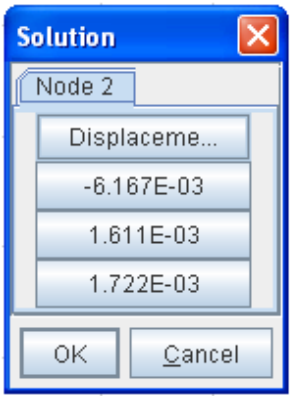


Figura 6.65: Deslocamentos nodais incógnitos.

Obtidos os deslocamentos nodais incógnitos, pode-se voltar às hipóteses dos elementos para conhecer as grandezas internas de cada elemento. Para um ponto qualquer do elemento de grelha pode-se obter os deslocamentos (figura 6.66), as deformações (figura 6.67) e os esforços (figura 6.68) aproximados.

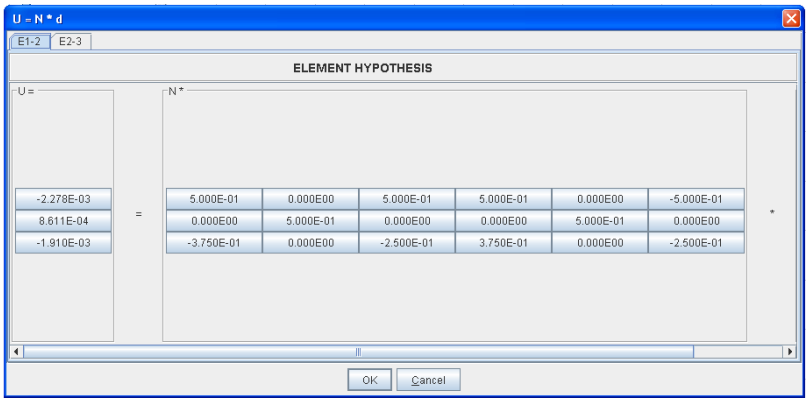


Figura 6.66: Deslocamentos em um ponto qualquer do elemento.

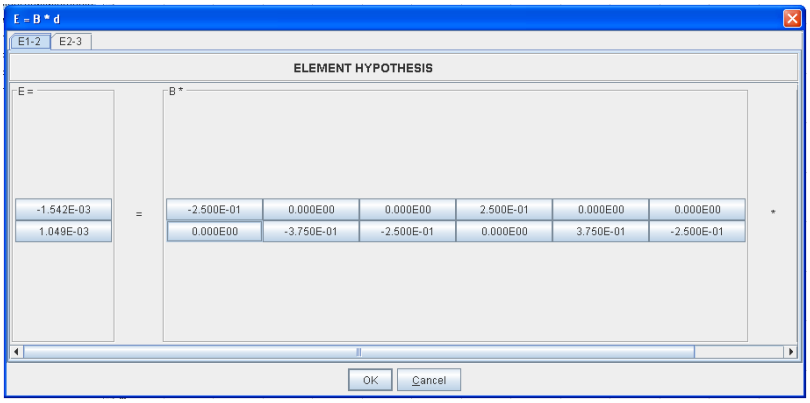


Figura 6.67: Deformações em um ponto qualquer do elemento.

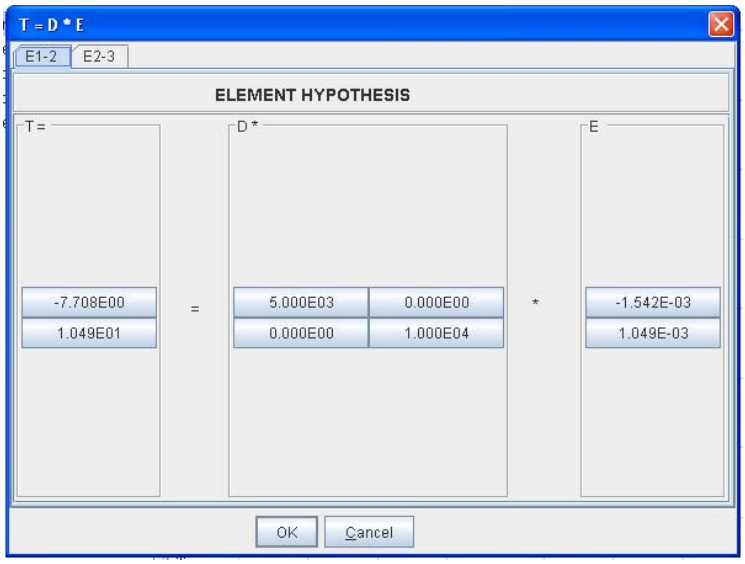


Figura 6.68: Tensões em um ponto qualquer do elemento.

6.6 Estado Plano de Tensões

Este exemplo tem como objetivo mostrar os elementos finitos bidimensionais de estado plano de tensões. O modelo em questão é mostrado na figura 6.69.

Trata-se de uma viga parede submetida a um carregamento distribuído constante. As condições de apoio devem ser tais que os deslocamentos verticais das faces laterais da viga sejam nulos. Considera-se material isotrópico com módulo de elasticidade longitudinal $E = 2 \times 10^5\ MPa$ e coeficiente de Poisson $\nu = 0,3$.

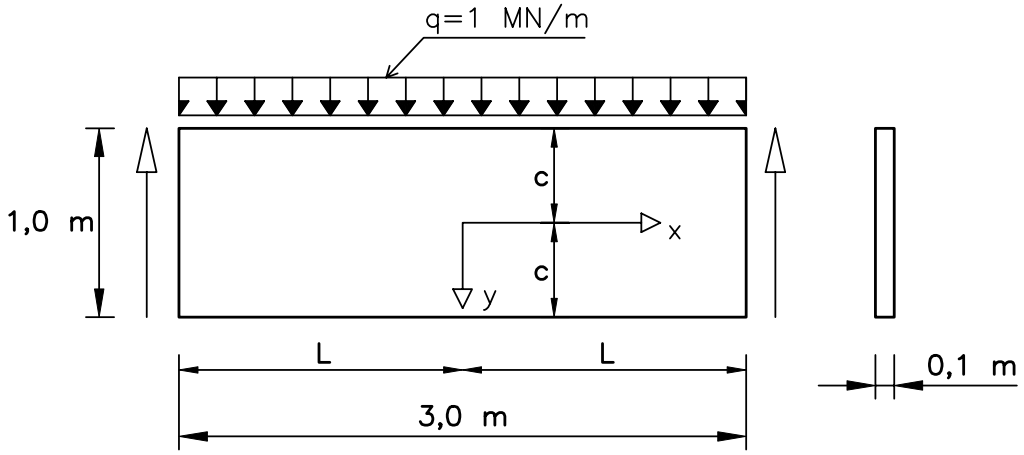


Figura 6.69: Viga parede proposta

Para discretizar o modelo proposto da viga parede com elementos quadrilaterais de quatro nós foi utilizada a malha mostrada na figuras 6.70, com 3 elementos. Para esta malha foram utilizados 2×2 pontos de integração de Gauss em cada elemento finito.

A solução interativa inicia-se pela numeração das equações do modelo conforme mostra a figura 6.71 .

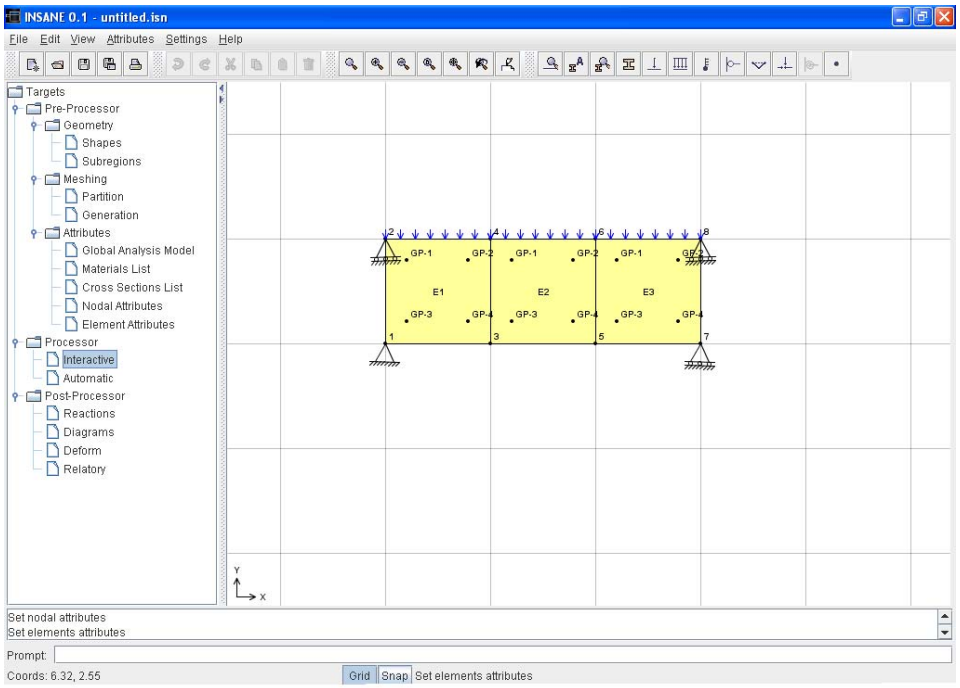


Figura 6.70: Modelo obtido no pré-processador `INSANE`.

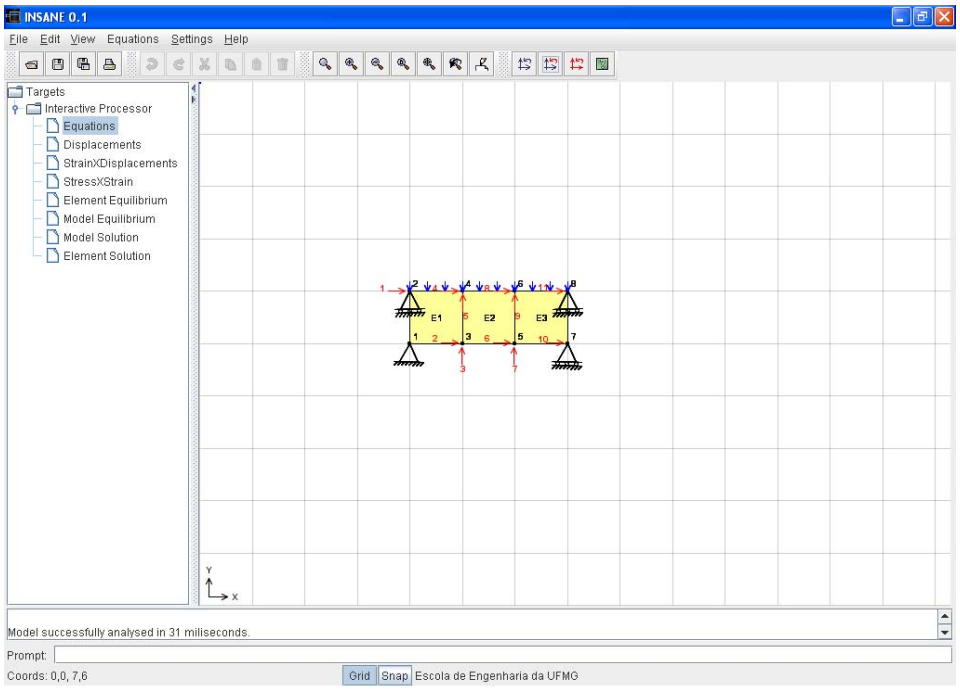


Figura 6.71: Numeração das Equações do modelo.

Nos modelos de estado plano de tensões, a partir da interpolação de deslocamentos, $\{u\} = [N]\ \{d\}^e$, tem-se:

$$\left\{ \begin{array}{c} u \\ v \end{array} \right\} = \left[\begin{array}{cc|c|cc|c|cc} N_1 & 0 & ... & N_i & 0 & ... & N_n & 0 \\ 0 & N_1 & ... & 0 & N_i & ... & 0 & N_n \end{array} \right] \left\{ \begin{array}{c} d_{1x} \\ d_{1y} \\ - - -- \\ \vdots \\ - - -- \\ d_{ix} \\ d_{iy} \\ - - -- \\ \vdots \\ - - -- \\ d_{nx} \\ d_{ny} \end{array} \right\} \tag{6.6.1}$$

onde u e v são, respectivamente, os deslocamentos horizontal e vertical de um ponto do elemento, N_i a função de forma associada ao nó i do mesmo, e d_{ix} e d_{iy} são, respectivamente, os deslocamentos nodais horizontal e vertical, e n, o número de nós.

A matriz das funções de forma $[N]$ para cada ponto de Gauss de cada elemento pode ser consultada, como mostra a figura 6.72.

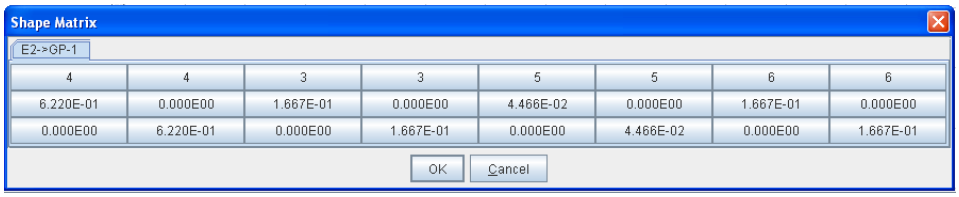


Figura 6.72: Funções de forma para o ponto de Gauss GP-1 do elemento E2.

As deformações podem ser obtidas através de $\{\varepsilon\} = [B]\ \{d\}$ ou:

$$\left\{ \begin{array}{c} \varepsilon_x \\ \varepsilon_y \\ \gamma_{xy} \end{array} \right\} = \left[\begin{array}{cc|c|cc|c|cc} N_{1,x} & 0 & ... & N_{i,x} & 0 & ... & N_{n,x} & 0 \\ 0 & N_{1,y} & ... & 0 & N_{i,y} & ... & 0 & N_{n,y} \\ N_{1,y} & N_{1,x} & ... & N_{i,y} & N_{i,x} & ... & N_{n,y} & N_{n,x} \end{array} \right] \left\{ d \right\} \tag{6.6.2}$$

sendo $[B]$ a matriz das derivadas primeiras das funções de forma em relação a x, e $\hat{d}$ o vetor dos deslocamentos nodais.

A matriz $[B]$, que contém as derivadas das funções de forma dos elementos, pode ser consultada conforme mostra a figura 6.73.

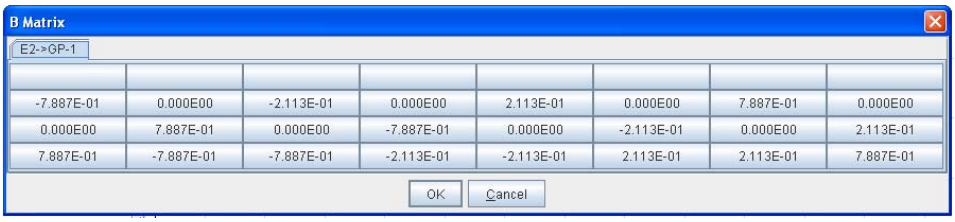


Figura 6.73: Matriz $[B]$ em cada ponto de Gauss do elemento.

As tensões podem ser calculadas através de $\{\sigma\} = [D]\,\{\varepsilon\}$ ou:

$$\begin{Bmatrix} \sigma_x \\ \sigma_y \\ \tau_{xy} \end{Bmatrix} = \frac{E}{1-\nu^2} \begin{bmatrix} 1 & \nu & 0 \\ \nu & 1 & 0 \\ 0 & 0 & \frac{1-\nu}{2} \end{bmatrix} \begin{Bmatrix} \varepsilon_x \\ \varepsilon_y \\ \gamma_{xy} \end{Bmatrix} \tag{6.6.3}$$

As hipóteses relativas ao comportamento do material podem ser visualizadas, através da consulta à matriz $[D]$ (figura 6.74).

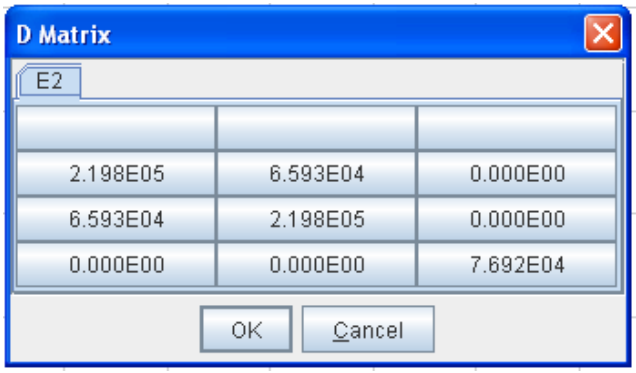
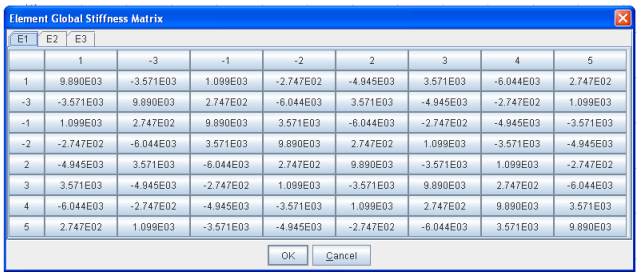


Figura 6.74: Matriz constitutiva do elemento.

Definidas as hipóteses dos elementos, a matriz de rigidez de cada elemento pode ser visualizada (figura 6.75), assim como o vetor de carregamento nodal equivalente (figura 6.76).



	1	-3	-1	-2	2	3	4	5
1	9.890E03	-3.571E03	1.099E03	-2.747E02	-4.945E03	3.571E03	-6.044E03	2.747E02
-3	-3.571E03	9.890E03	2.747E02	-6.044E03	3.571E03	-4.945E03	-2.747E02	1.099E03
-1	1.099E03	2.747E02	9.890E03	3.571E03	-6.044E03	-2.747E02	-4.945E03	-3.571E03
-2	-2.747E02	-6.044E03	3.571E03	9.890E03	2.747E02	1.099E03	-3.571E03	-4.945E03
2	-4.945E03	3.571E03	-6.044E03	2.747E02	9.890E03	-3.571E03	1.099E03	-2.747E02
3	3.571E03	-4.945E03	-2.747E02	1.099E03	-3.571E03	9.890E03	2.747E02	-6.044E03
4	-6.044E03	-2.747E02	-4.945E03	-3.571E03	1.099E03	2.747E02	9.890E03	3.571E03
5	2.747E02	1.099E03	-3.571E03	-4.945E03	-2.747E02	-6.044E03	3.571E03	9.890E03

Figura 6.75: Matriz de rigidez completa de cada elemento do modelo.

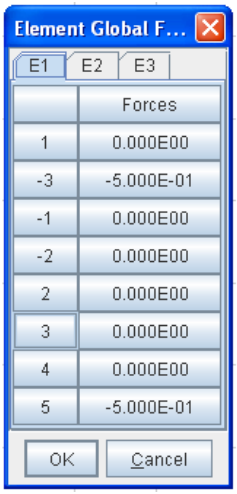
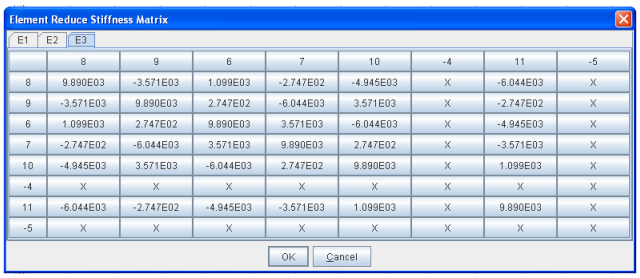


Figura 6.76: Vetor de carregamento nodal equivalente de cada elemento do modelo.

Aplicando as condições de contorno, a matriz de rigidez reduzida de cada elemento é apresentada (figura 6.77), assim como o vetor de carregamento nodal equivalente reduzido (figura 6.78).



	8	9	6	7	10	-4	11	-5
8	9.890E03	-3.571E03	1.099E03	-2.747E02	-4.945E03	X	-6.044E03	X
9	-3.571E03	9.890E03	2.747E02	-6.044E03	3.571E03	X	-2.747E02	X
6	1.099E03	2.747E02	9.890E03	3.571E03	-6.044E03	X	-4.945E03	X
7	-2.747E02	-6.044E03	3.571E03	9.890E03	2.747E02	X	-3.571E03	X
10	-4.945E03	3.571E03	-6.044E03	2.747E02	9.890E03	X	1.099E03	X
-4	X	X	X	X	X	X	X	X
11	-6.044E03	-2.747E02	-4.945E03	-3.571E03	1.099E03	X	9.890E03	X
-5	X	X	X	X	X	X	X	X

Figura 6.77: Matriz de rigidez reduzida de cada elemento do modelo.

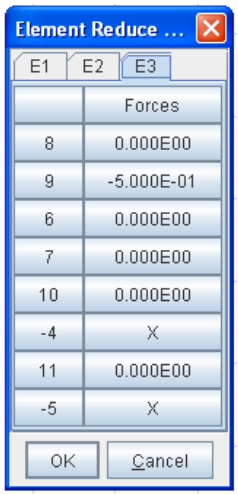


Figura 6.78: Vetor de carregamento nodal equivalente reduzido de cada elemento do modelo.

A partir do equilíbrio de cada elemento, é gerado o equilíbrio do modelo. Este também pode ser consultado, sendo mostrada a contribuição de cada elemento do problema à rigidez total do modelo. A figura 6.79 mostra a matriz de rigidez completa do modelo e a visualização da contribuição de cada elemento para formar um dos termos da mesma. A matriz de rigidez reduzida também pode ser consultada. Para esta opção também é possível visualizar as contribuições de cada elemento. O mesmo pode ser aplicado à consulta dos vetores de força completa e reduzida.

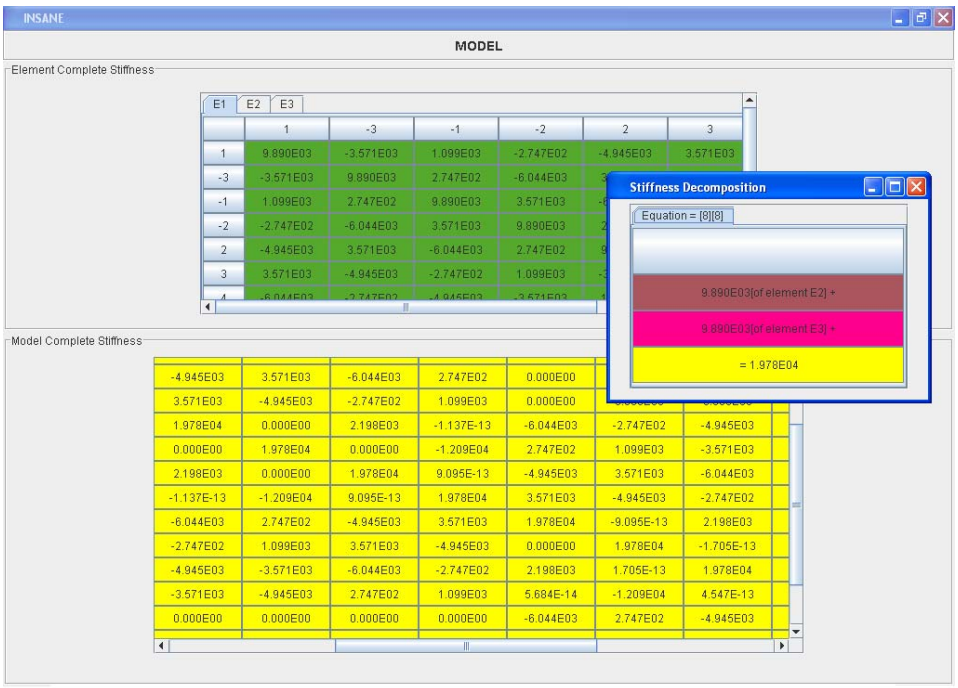
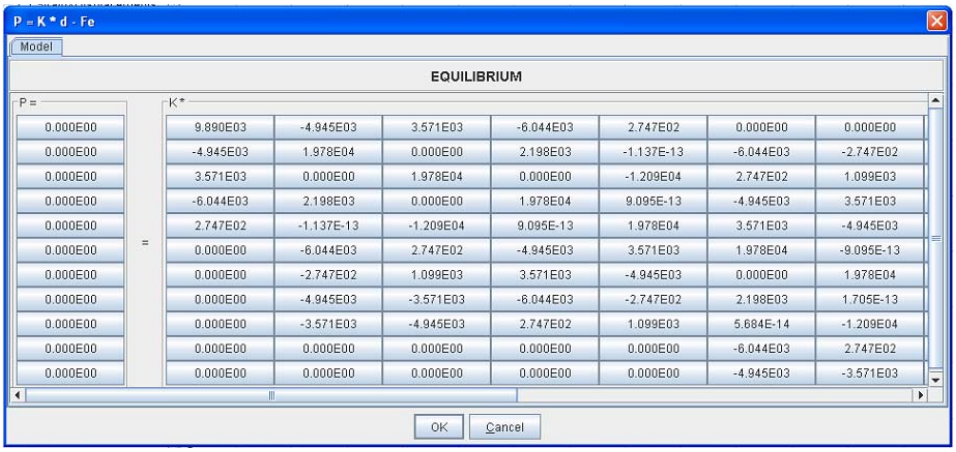


Figura 6.79: Matriz de rigidez completa do modelo.

Também é possível visualizar as equações de equilíbrio do modelo (figura 6.80). Após a montagem destas equações, pode-se solucionar o sistema, para os deslocamentos nodais incógnitos, que podem ser consultados, através da seleção de um nó do modelo (figura 6.81).



P =	K*						
0.000E00	9.890E03	-4.945E03	3.571E03	-6.044E03	2.747E02	0.000E00	0.000E00
0.000E00	-4.945E03	1.978E04	0.000E00	2.198E03	-1.137E-13	-6.044E03	-2.747E02
0.000E00	3.571E03	0.000E00	1.978E04	0.000E00	-1.209E04	2.747E02	1.099E03
0.000E00	-6.044E03	2.198E03	0.000E00	1.978E04	9.095E-13	-4.945E03	3.571E03
0.000E00	2.747E02	-1.137E-13	-1.209E04	9.095E-13	1.978E04	3.571E03	-4.945E03
0.000E00	0.000E00	-6.044E03	2.747E02	-4.945E03	3.571E03	1.978E04	-9.095E-13
0.000E00	0.000E00	-2.747E02	1.099E03	3.571E03	-4.945E03	0.000E00	1.978E04
0.000E00	0.000E00	-4.945E03	-3.571E03	-6.044E03	-2.747E02	2.198E03	1.705E-13
0.000E00	0.000E00	-3.571E03	-4.945E03	2.747E02	1.099E03	5.684E-14	-1.209E04
0.000E00	0.000E00	0.000E00	0.000E00	0.000E00	0.000E00	-6.044E03	2.747E02
0.000E00	0.000E00	0.000E00	0.000E00	0.000E00	0.000E00	-4.945E03	-3.571E03

Figura 6.80: Equações de equilíbrio do modelo.

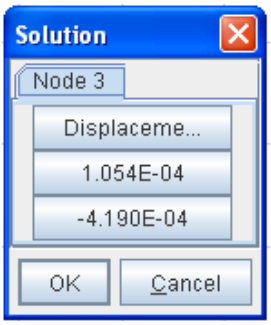


Figura 6.81: Deslocamentos nodais incógnitos.

Obtidos os deslocamentos nodais incógnitos, pode-se voltar às hipóteses dos elementos para conhecer as grandezas internas de cada elemento. Para um ponto de integração qualquer do elemento pode-se obter os deslocamentos (figura 6.82), as deformações (figura 6.83) e as tensões (figura 6.84). Também são apresentadas as forças nos nós de cada elemento a partir dos deslocamentos nodais (figura 6.85).

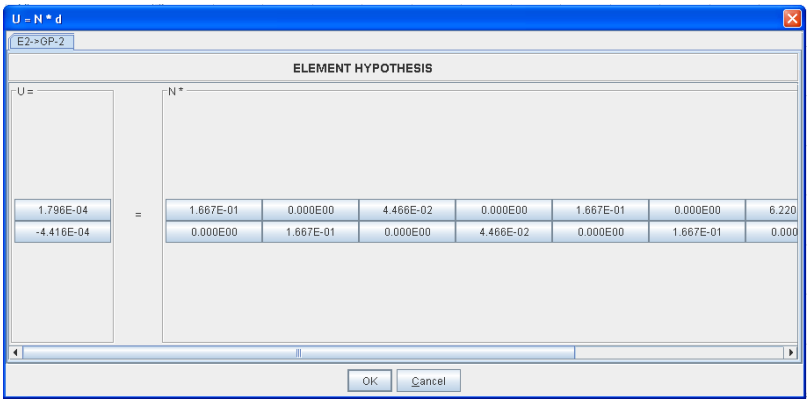


Figura 6.82: Deslocamentos em um ponto de Gauss qualquer do elemento.

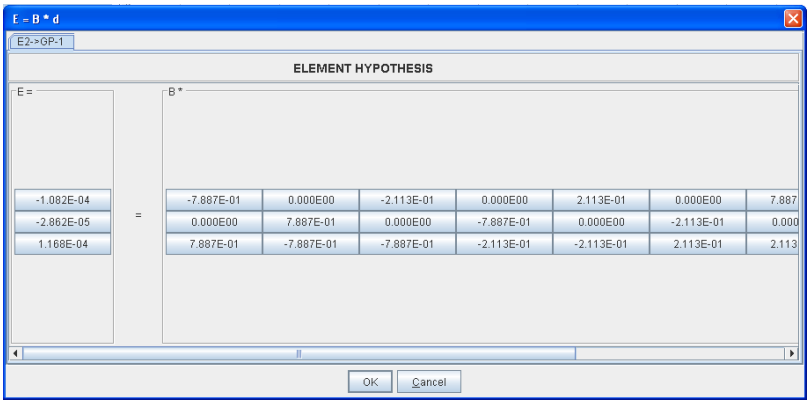


Figura 6.83: Deformações em um ponto de Gauss qualquer do elemento.

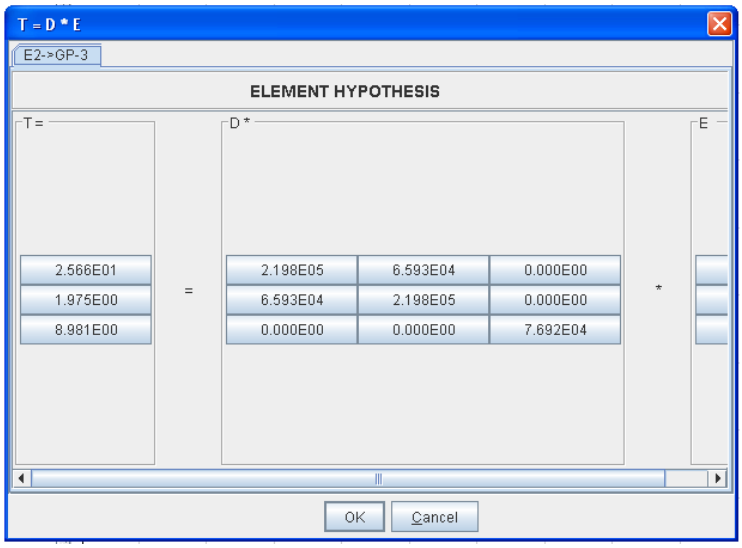


Figura 6.84: Tensões em um ponto de Gauss qualquer do elemento.

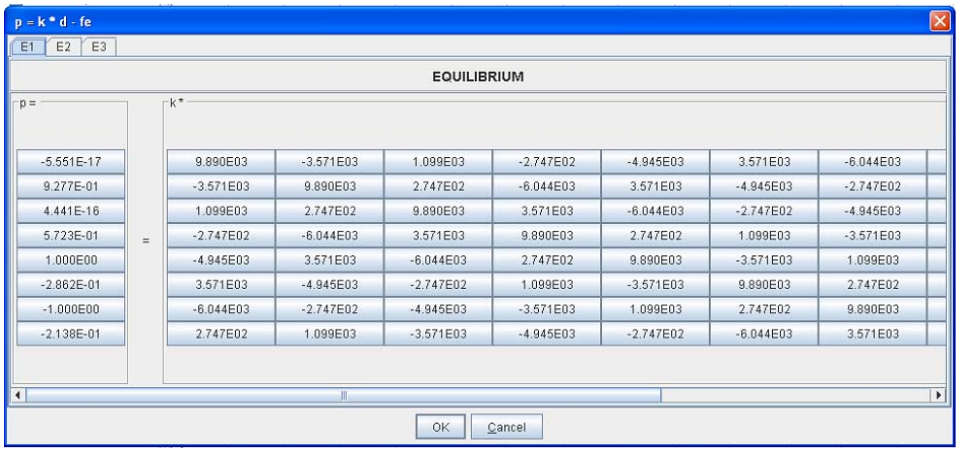


Figura 6.85: Forças nos nós do elemento.

6.7 Estado Plano de Deformações

Este exemplo tem como objetivo mostrar os elementos finitos bidimensionais de estado plano de deformações. Trata-se de uma barragem, obtida na referência (Soriano e Lima 1999), e reproduzida na figura 6.86 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.

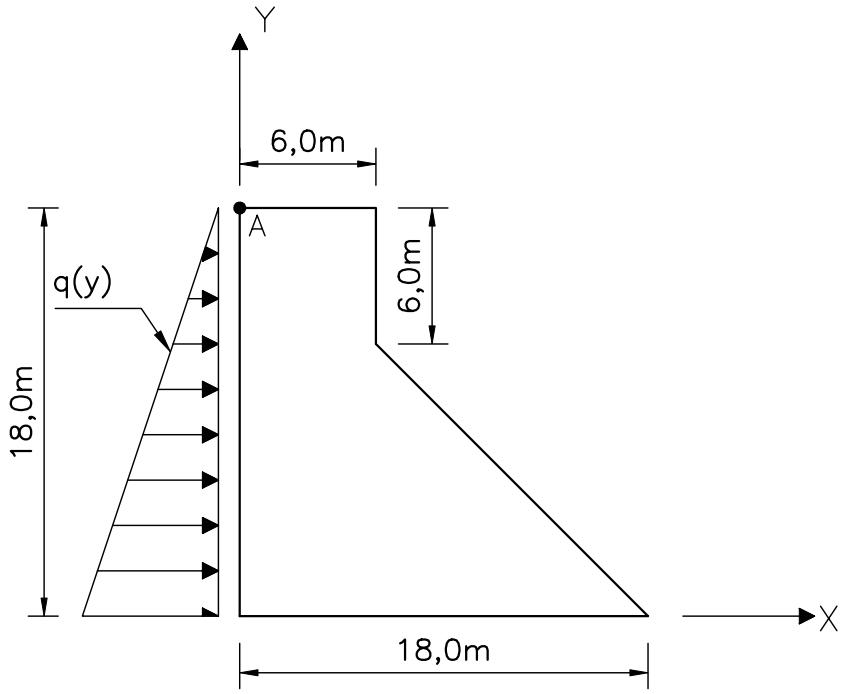


Figura 6.86: Seção transversal da barragem proposta

Para discretizar o modelo proposto da barragem com elementos triangulares de três nós foi utilizada a malha mostrada na figuras 6.87, com 16 elementos. Para esta malha foram utilizados 3×3 pontos de integração de Gauss em cada elemento finito.

A solução interativa inicia-se pela numeração das equações do modelo conforme mostra a figura 6.88.

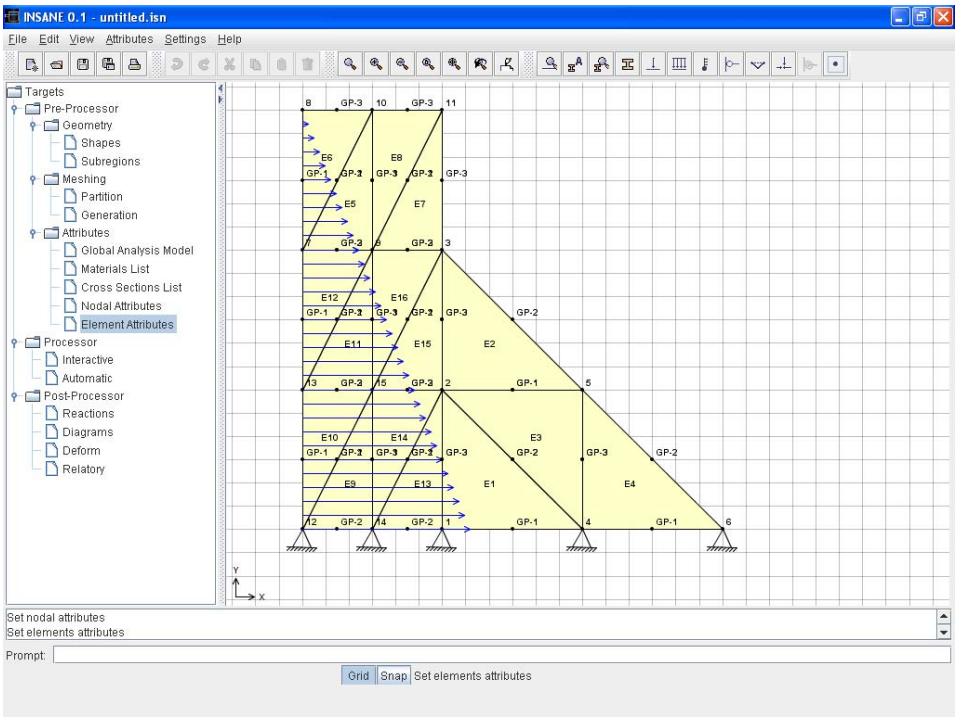


Figura 6.87: Modelo obtido no pré-processador `INSANE`.

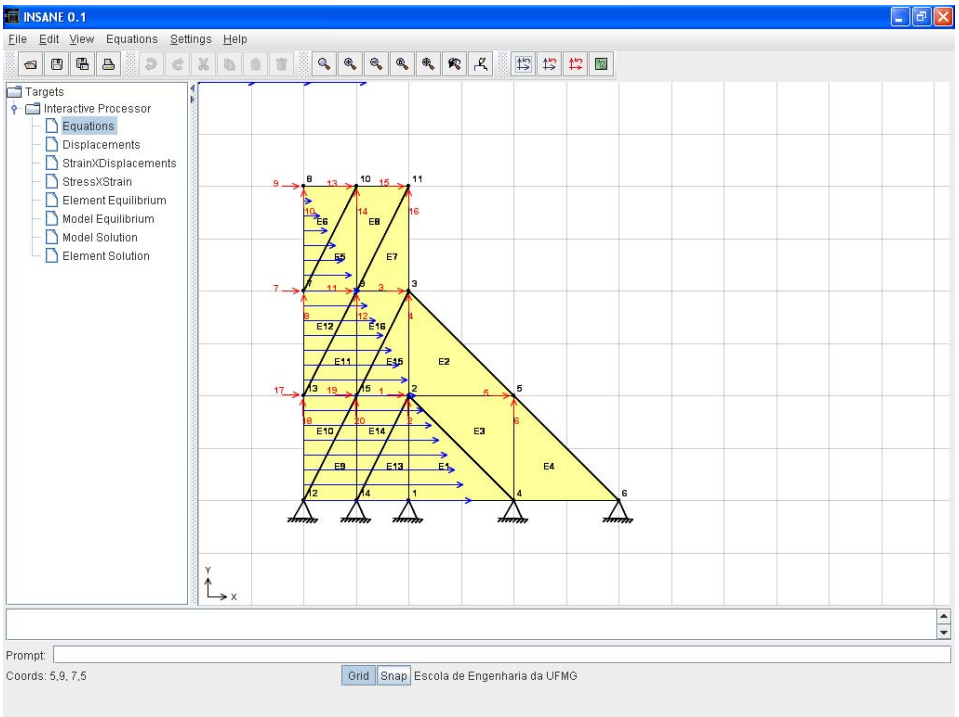


Figura 6.88: Numeração das equações do modelo.

Nos modelos de estado plano de deformações, a partir da interpolação de deslocamentos, $\{u\} = [N]\ \{\hat{d}\}^e$, conforme descrito anteriormente na equação 6.6.1.

A matriz das funções de forma $[N]$ para cada ponto de Gauss de cada elemento pode ser consultada, como mostra a figura 6.89.

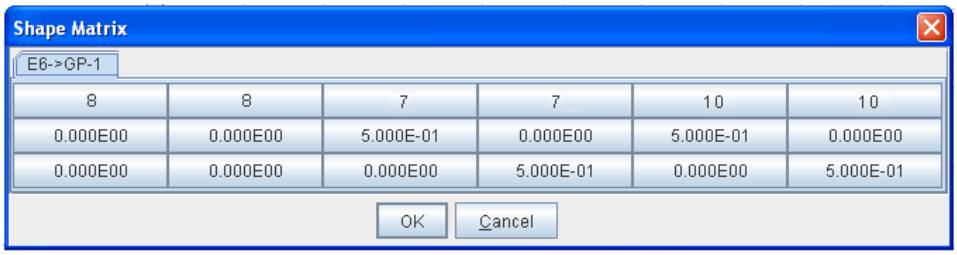


Figura 6.89: Funções de forma para o ponto de Gauss GP-1 do elemento E6.

As deformações podem ser obtidas através de $\{\varepsilon\} = [B]\ \{\hat{d}\}$, conforme descrito na equação 6.6.2.

A matriz $[B]$, que contém as derivadas das funções de forma dos elementos, pode ser consultada conforme mostra a figura 6.90.

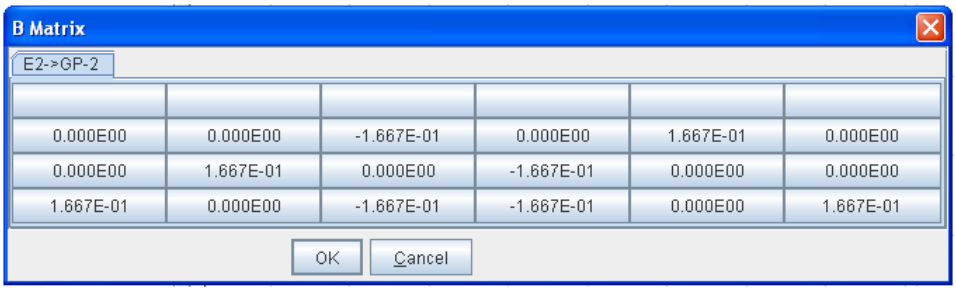


Figura 6.90: Matriz $[B]$ em cada ponto de Gauss do elemento.

As tensões podem ser calculadas através de $\{\sigma\} = [D]\ \{\varepsilon\}$ ou:

$$\begin{Bmatrix} \sigma_x \\ \sigma_y \\ \tau_{xy} \end{Bmatrix} = \frac{E(1-\nu)}{(1+\nu)(1-2\nu)} \begin{bmatrix} 1 & \frac{\nu}{1-\nu} & 0 \\ \frac{\nu}{1-\nu} & 1 & 0 \\ 0 & 0 & \frac{(1-2\nu)}{2(1-\nu)} \end{bmatrix} \begin{Bmatrix} \varepsilon_x \\ \varepsilon_y \\ \gamma_{xy} \end{Bmatrix} \tag{6.7.1}$$

As hipóteses relativas ao comportamento do material podem ser visualizadas, através da consulta à matriz $[D]$ (figura 6.91).

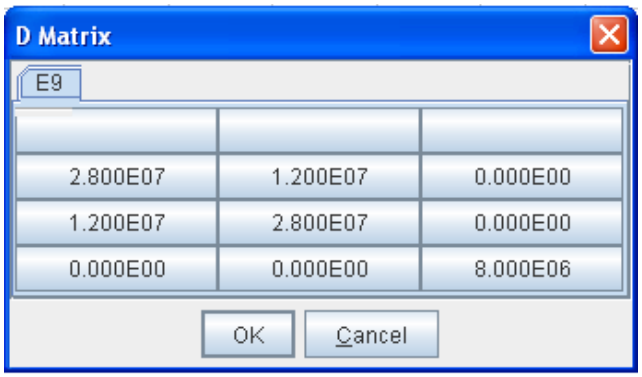


Figura 6.91: Matriz constitutiva do elemento.

Definidas as hipóteses dos elementos, a matriz de rigidez de cada elemento pode ser visualizada (figura 6.92), assim como o vetor de carregamento nodal equivalente (figura 6.93).

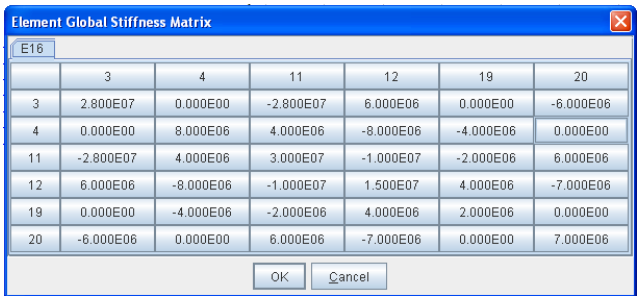


Figura 6.92: Matriz de rigidez completa de cada elemento do modelo.

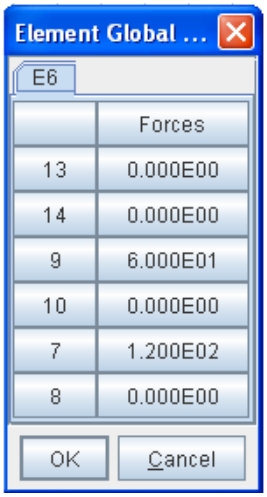


Figura 6.93: Vetor de carregamento nodal equivalente de cada elemento do modelo.

Aplicando as condições de contorno, a matriz de rigidez reduzida de cada elemento é apresentada (figura 6.94), assim como o vetor de carregamento nodal equivalente reduzido (figura 6.95).

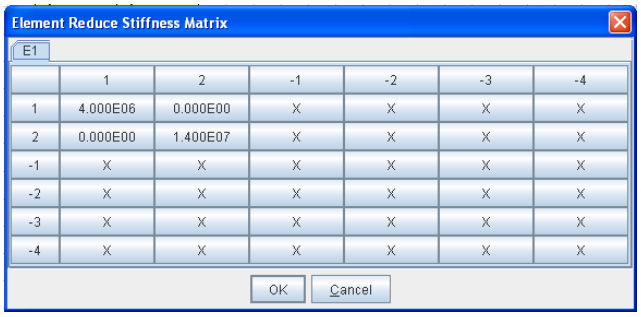


Figura 6.94: Matriz de rigidez reduzida de cada elemento do modelo.

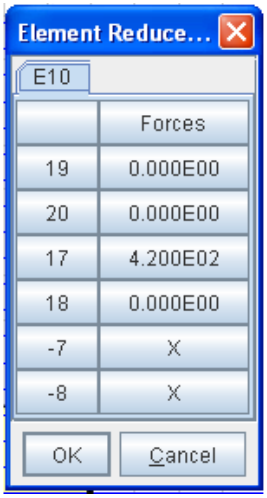


Figura 6.95: Vetor de carregamento nodal equivalente reduzido de cada elemento do modelo.

A partir do equilíbrio de cada elemento, é gerado o equilíbrio do modelo. Este também pode ser consultado, sendo mostrada a contribuição de cada elemento do problema à rigidez total do modelo. A figura 6.96 mostra a matriz de rigidez completa do modelo e a visualização da contribuição de cada elemento para formar um dos termos da mesma. A matriz de rigidez reduzida também pode ser consultada. Para esta opção também é possível visualizar as contribuições de cada elemento. O mesmo pode ser aplicado à consulta dos vetores de força completa e reduzida.

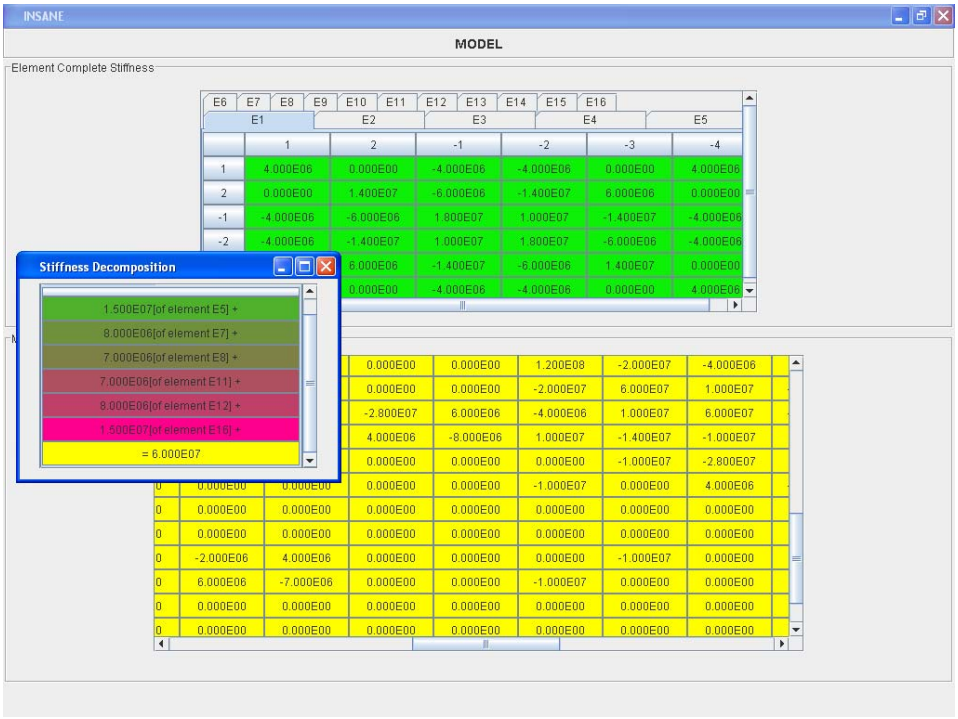


Figura 6.96: Matriz de rigidez completa do modelo.

Também é possível visualizar as equações de equilíbrio do modelo (figura 6.97). Após a montagem destas equações, pode-se solucionar o sistema, para os deslocamentos nodais incógnitos, que podem ser consultados, através da seleção de um nó do modelo (figura 6.98).

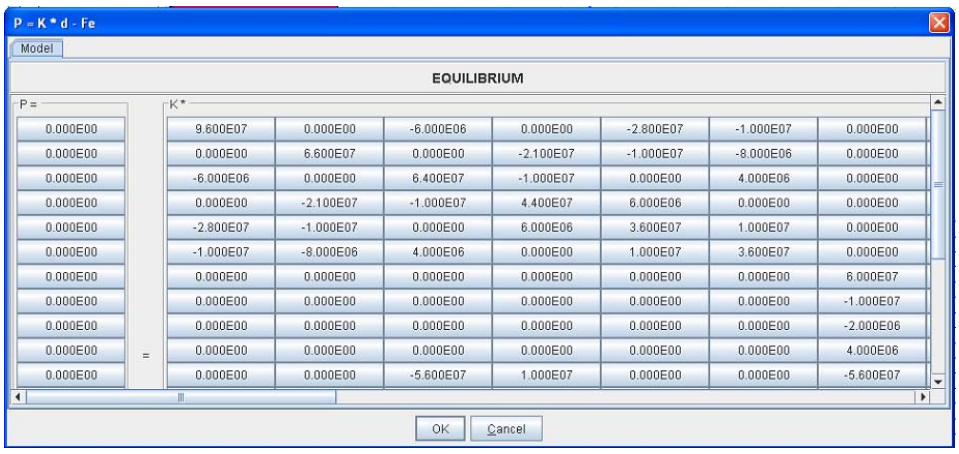


Figura 6.97: Equações de equilíbrio do modelo.

Obtidos os deslocamentos nodais incógnitos, pode-se voltar às hipóteses dos elementos para conhecer as grandezas internas de cada elemento. Para um ponto de integração qualquer do elemento pode-se obter os deslocamentos (figura 6.99), as deformações (figura

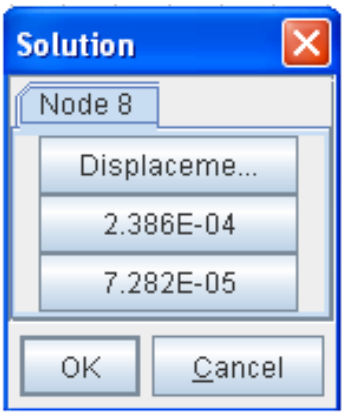


Figura 6.98: Deslocamentos nodais incógnitos.

6.100) e as tensões (figura 6.101). Também são apresentadas as forças nos nós de cada elemento a partir dos deslocamentos nodais (figura 6.102).

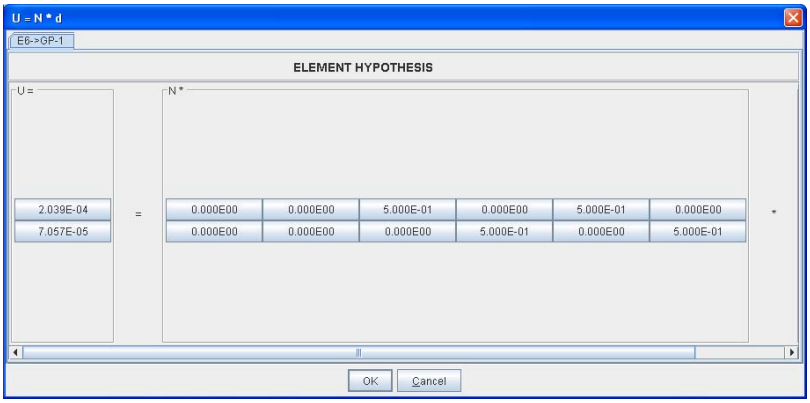


Figura 6.99: Deslocamentos em um ponto de Gauss qualquer do elemento.

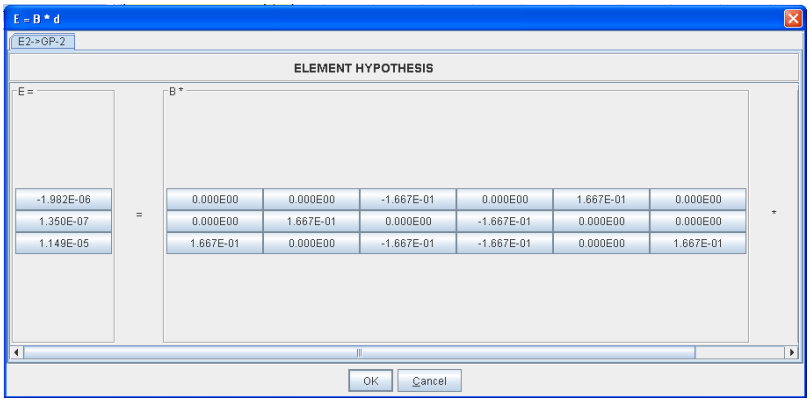


Figura 6.100: Deformações em um ponto de Gauss qualquer do elemento.

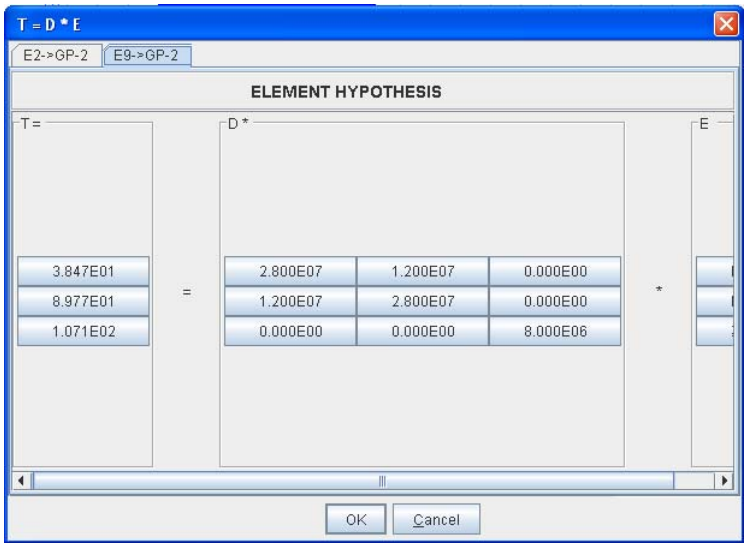


Figura 6.101: Tensões em um ponto de Gauss qualquer do elemento.

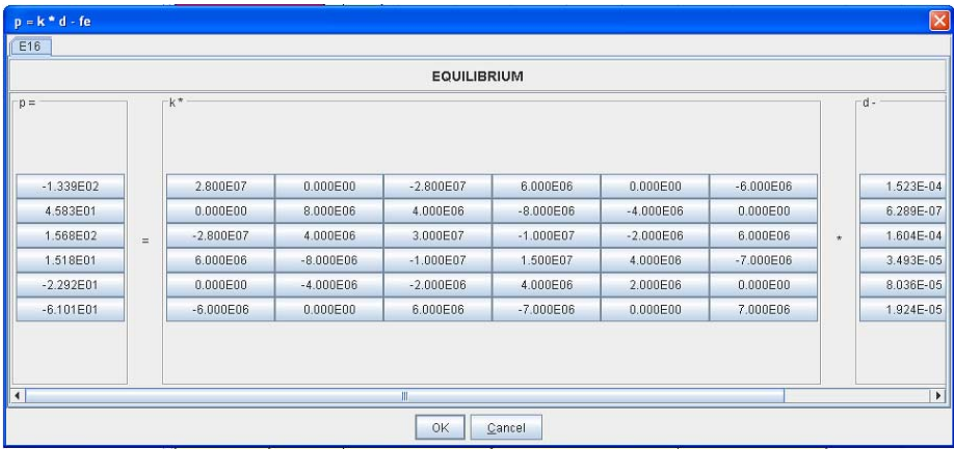


Figura 6.102: Forças nos nós do elemento.

Capítulo 7

Considerações Finais

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, distribuição e manutenção difíceis, desenvolvidos por equipes fechadas, com documentação deficiente, entre outras limitações. Tais dificuldades podem ser creditados à falta de disposição da comunidade em se apropriar das tecnologias emergentes ou mesmo à inexistência das mesmas.

Uma das propostas do projeto `INSANE` é trazer para a comunidade acadêmica soluções tecnológicas para o desenvolvimento de aplicações que auxiliem as pesquisas e o ensino 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 e do ensino de modelos discretos de análise estrutural.

Espera-se que, de posse desta tecnologia, a aplicação aqui apresentada diminua as barreiras existentes entre professor e aluno, facilitando o aprendizado dos diversos conceitos do Método dos Elementos Finitos, evitando que os complicadores matemáticos inerentes ao método interfiram negativamente no processo.

A verificação da real influência desta aplicação no processo de aprendizagem do MEF se dará a partir da utilização do software nos cursos de graduação e pós-graduação em engenharia estrutural. Assim, professor e aluno poderão avaliar a eficiência da ferramenta no auxílio ao ensino presencial do MEF.

Deseja-se também que o INSANE, base do programa aqui apresentado, seja fomentador do desenvolvimento de novos modelos discretos, evitando o recomeço do processo de implementação e permitindo maior agilidade e criatividade da pesquisa na área.

Os resultados obtidos neste trabalho, só foram possíveis graças às soluções tecnológicas empregadas (consideradas no item 7.1) e ao desenvolvimento colaborativo de vários subprojetos (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.

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++.

Obteve-se um produto ainda em fase de aprimoramento, mas bastante superior aos obtidos com a utilização das tecnologias tradicionais no que diz respeito ao reuso, potencial de expansão, modularidade, facilidade de manutenção, uniformidade na estrutura da aplicação, incremento da padronização no desenvolvimento, aplicação imediata por outros desenvolvedores e utilização dos alunos de graduação e pós-graduação.

7.2 Desenvolvimento Colaborativo

O desenvolvimento do projeto `INSANE` é feito de forma colaborativa, envolvendo alunos em diferentes estágios de conhecimento ((Fonseca e Pitangueira 2004), (Fonseca, Pitangueira e Vasconcelos 2004), (Gonçalves e Pitangueira 2004*a*), (Gonçalves e Pitangueira 2004*b*) e (de Almeida 2005)). 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;

- 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.

- Implementação de novas classes que suportem modelos tridimensionais e expansão do código para permitir a visualização de tais modelos através da manipulação das classes do pacote *Java3D*.

Referências Bibliográficas

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

Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P. e Stal, M. (1995), *Pattern-Oriented Software Architecture: A System of Patterns*, Vol. 1, Wiley.

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

de Almeida, M. L. (2005), 'Elementos finitos paramétricos implementados em java, dissertação de mestrado', *Escola de Engenharia da UFMG, Belo Horizonte-MG* .

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

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

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

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

Gamma, E., Helm, R., Johnson, R. e Vlissides, J. (1995*b*), *Design Patterns - Element of Reusable of Object Oriented Software*, Editora Addison-Wesley.

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* .

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

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

Grand, M. (1998), *Patterns in Java: A Catalog of Reusable Design Patterns Illustrated with UML*, Vol. 1, John Wiley & Sons.

Gupta, S. (2004), 'The misteries of business object', URL: `http://javaboutique.internet.com/tutoriais/businessObject`.

Horstmann, C. S. e Cornell, G. (2001*a*), *Core Java 2 - Fundamentos*, Vol. 1, Makron Books.

Horstmann, C. S. e Cornell, G. (2001*b*), *Core Java 2 - Recursos Avançados*, Vol. 2, Makron Books.

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

Logan, D. (2001), *A First Course in the Finite Element Method*, BWS Publishing Company.

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

Lozano, F. (2004), 'Padrões e arquiteturas', *Java Magazine* (Edição 15), 18–19.

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

Pietro, G. A. (2001*a*), Utilização de padrões de projeto de software na reengenharia de sistemas, Master's thesis, Universidade Federal de São Carlos, São Carlos, SP, Brasil.

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

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

Rowe, G. W. (2001), *Computer Graphics with Java*, Palgrave.

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