

MINISTÉRIO DA DEFESA
EXÉRCITO BRASILEIRO
DEPARTAMENTO DE CIÊNCIA E TECNOLOGIA
INSTITUTO MILITAR DE ENGENHARIA
CURSO DE MESTRADO EM ENGENHARIA DE TRANSPORTES

FÁBIO GRISOLIA DE ÁVILA

MODELAGEM NUMÉRICA PARA PAVIMENTAÇÃO FLEXÍVEL PELO
MÉTODO DOS ELEMENTOS FINITOS

Rio de Janeiro
2008

Livros Grátis

<http://www.livrosgratis.com.br>

Milhares de livros grátis para download.

INSTITUTO MILITAR DE ENGENHARIA

FÁBIO GRISOLIA DE ÁVILA

MODELAGEM NUMÉRICA PARA PAVIMENTAÇÃO FLEXÍVEL PELO
MÉTODO DOS ELEMENTOS FINITOS

Dissertação de Mestrado apresentada ao Curso de Mestrado em Engenharia de Transportes do Instituto Militar de Engenharia, como requisito parcial para a obtenção do título de Mestre em Ciências em Engenharia de Transportes.

Orientadores: Marcelo Rodrigues Leão Silva, D. C.
Marco Aurélio Chaves Ferro, D. Sc.

Rio de Janeiro
2008

© 2008

INSTITUTO MILITAR DE ENGENHARIA

Praça General Tibúrcio, 80 – Praia Vermelha

22290-270 Rio de Janeiro, RJ

Este exemplar é de propriedade do Instituto Militar de Engenharia, que poderá incluí-lo em base de dados, armazenar em computador, microfilmar ou adotar qualquer forma de arquivamento.

É permitida a menção, reprodução parcial ou integral e a transmissão entre bibliotecas deste trabalho, sem modificação de seu texto, em qualquer meio que esteja ou venha a ser fixado, para pesquisa acadêmica, comentários e citações, desde que sem finalidade comercial e que seja feita a referência bibliográfica completa.

Os conceitos expressos neste trabalho são de responsabilidade do(s) autor(es) e do(s) orientador(es).

A958m Ávila, Fábio Grisolia de
Modelagem Numérica para Pavimentação Flexível pelo
Método dos Elementos Finitos / Fábio Grisolia de Ávila – Rio
de Janeiro: Instituto Militar de Engenharia, 2008.

196 p.: il., tab.

Dissertação (mestrado) – Instituto Militar de Engenharia –
Rio de Janeiro, 2008.

1. Método dos Elementos Finitos. 2. Cálculo de Tensão e
Deformação. 3. Análise de Pavimentos. I. Título. II. Instituto
Militar de Engenharia.

CDD 515.63

INSTITUTO MILITAR DE ENGENHARIA

FÁBIO GRISOLIA DE ÁVILA

**MODELAGEM NUMÉRICA PARA PAVIMENTAÇÃO
FLEXÍVEL PELO MÉTODO DOS ELEMENTOS FINITOS**

Dissertação de Mestrado apresentada ao Curso de Mestrado em Engenharia de Transportes do Instituto Militar de Engenharia, como requisito parcial para a obtenção do título de Mestre em Ciências em Engenharia de Transportes.

Orientadores: Prof. TC QEM Marcelo Rodrigues Leão Silva, D. C.

Prof. Maj QEM Marco Aurélio Chaves Ferro, D.Sc.

Aprovada em 14 de janeiro de 2008 pela seguinte banca examinadora:

Prof. TC QEM Marcelo Rodrigues Leão Silva, D. C. do IME – Presidente

Prof. Maj QEM Marco Aurélio Chaves Ferro – D. Sc. do IME

Prof. Salomão Pinto – D. Sc. do IME

Prof. Webe João Mansur – Ph. D. da COPPE/ UFRJ

Rio de Janeiro

2008

Dedico este trabalho aos meus pais José Alencar e Maria das Graças, à minha querida irmã Gabriela e à minha amada esposa Paula.

AGRADECIMENTOS

A Deus por ter me concedido a graça de atingir este objetivo.

Aos meus pais, José Alencar de Ávila e Maria das Graças Grisolia de Ávila, pela educação de qualidade, amor, apoio, dedicação e incentivo ao longo da minha existência.

À minha querida irmã, Gabriela Grisolia de Ávila, pela amizade, carinho e “cumplicidade” em todos os momentos da minha vida.

À minha amada esposa, Paula Cristina Dias Ávila, pelo amor, companheirismo e compreensão que partilhamos durante o nosso convívio, principalmente durante a confecção deste trabalho.

Aos meus colegas de mestrado pelos momentos inesquecíveis que passamos juntos durante o curso, inclusive nos finais de semana e feriados, e em especial ao Bruno, ao Cazelli e ao Major Diniz pelas valiosas sugestões e opiniões para a confecção deste trabalho.

Ao André Medeiros e à Cristina Oliveira pela amizade e pelas conversas nos corredores da pós-graduação, que ajudaram a arejar minha cabeça nos momentos de descanso.

Aos demais familiares e amigos pelo apoio e amizade de vocês que sempre farão parte de minha vida.

Aos professores da Seção de Engenharia de Fortificação e Construção do Instituto Militar de Engenharia pelo ensino de excelência que me foi proporcionado durante a minha graduação.

Ao Exército Brasileiro pela oportunidade que me foi dada de realizar este curso e pela experiência profissional adquirida ao longo da minha carreira.

Aos professores da Pós-graduação em Engenharia de Transportes do IME e à professora D. Sc. Laura Maria Goretti da Motta da COPPE/UFRJ pelos ensinamentos transmitidos nas cadeiras do mestrado e pelas orientações fornecidas dentro e fora da sala de aula.

Ao professor D. C. Tenente-Coronel Marcelo Rodrigues Leão Silva, meu orientador, pelas críticas, sugestões e orientação realizadas durante a confecção deste trabalho e pelo fornecimento do código-fonte de parte do programa utilizado em sua tese de doutorado, que serviu de base para a implementação do programa aqui desenvolvido.

Ao professor D. Sc. Major Marco Aurélio Chaves Ferro, meu outro orientador, pelas correções, sugestões e orientação durante a realização desta dissertação e pelas bibliografias fornecidas.

Ao professor D. Sc. Salomão Pinto pelo seu apoio e incentivo durante o desenvolvimento deste trabalho, pelos seus constantes ensinamentos e pela apreciação desta dissertação.

Ao professor Ph. D. Webe João Mansur pelo tempo dedicado à apreciação deste trabalho e pela aceitação de compor a minha banca examinadora, mesmo durante seu período de férias.

“Algo só é impossível até que alguém duvide e acabe provando o contrário.”

ALBERT EINSTEIN

SUMÁRIO

LISTA DE ILUSTRAÇÕES	10
LISTA DE TABELAS	12
LISTA DE ABREVIATURAS E SÍMBOLOS.....	13
LISTA DE SIGLAS	16
1 INTRODUÇÃO	19
2 REVISÃO BIBLIOGRÁFICA.....	23
2.1 A Evolução dos Cálculos de Tensões e Deformações.....	23
2.2 O Método dos Elementos Finitos	29
2.2.1 Introdução.....	29
2.2.2 Características do Método dos Elementos Finitos	31
2.2.3 Elementos Finitos Bidimensionais	32
2.2.4 O Método dos Elementos Finitos Tridimensionais	35
3 MODELAGEM NUMÉRICA DO PAVIMENTO.....	37
3.1 O Pavimento.....	37
3.1.1 Classificação dos Pavimentos	39
3.1.2 A Resiliência em Pavimentos.....	39
3.1.3 O Coeficiente de Poisson nas Camadas do Pavimento	47
3.2 A Modelagem Axissimétrica do Pavimento.....	50
3.2.1 Método Incremental	57
4 IMPLEMENTAÇÃO COMPUTACIONAL	60
4.1 A Linguagem de Programação C++ e a Orientação a Objetos	60
4.2 Programa de Análise de Pavimentos – Características da Implementação	64
4.2.1 Classe Matriz.....	65
4.2.2 Classe Malha	66
4.2.3 Classe Material	67
4.2.4 Classe Pavimento	69
4.2.5 Classe No.....	70
4.2.6 Classe PontoDeGauss.....	70

4.2.7	Classe Elemento	73
4.2.8	Classe Estrutura.....	77
4.3	Peculiaridades do Programa de Análise de Pavimentos.....	82
5	VALIDAÇÃO DO MODELO	93
5.1	Pavimento Analisado por Preussler (1983).....	93
5.2	Pavimento Analisado por Pinto (1991).....	96
6	CONCLUSÕES E RECOMENDAÇÕES	113
6.1	Conclusões	113
6.2	Recomendações e Sugestões	115
7	REFERÊNCIAS BIBLIOGRÁFICAS	116
8	APÊNDICES.....	123
8.1	APÊNDICE A – Manual do Usuário	124
8.2	APÊNDICE B – Código-Fonte do Modelo.....	129
8.2.1	Classe Matriz.....	129
8.2.2	Classe Malha	141
8.2.3	Classe Material	146
8.2.4	Classe Pavimento	148
8.2.5	Classe No.....	154
8.2.6	Classe PontoDeGauss.....	154
8.2.7	Classe Elemento	165
8.2.8	Classe Estrutura.....	175
8.2.9	Classe Main	193

LISTA DE ILUSTRAÇÕES

FIG. 1 – Estrutura esquemática do pavimento (Vieira, 2006)	19
FIG. 2 – Esquema do método de dimensionamento proposto por Motta (1991).....	20
FIG. 3 – Sistema de Boussinesq.....	23
FIG. 4 – Componentes de tensão sob carregamento axissimétrico (Aedo, 1997)	25
FIG. 5 – Sistema de camadas múltiplas linearmente elásticas (Aedo, 1997).....	27
FIG. 6 – Sistema de camadas de comportamento tensão-deformação não lineares.....	27
FIG. 7 – Modelagem de um problema com o MEF (Evangelista Jr. et al., 2003)	30
FIG. 8 – Representação esquemática do estado plano de deformações para um pavimento ...	34
FIG. 9 – Tensões atuantes no corpo de prova durante o ensaio triaxial dinâmico (Vieira, 2006)	41
FIG. 10 – Equipamento de ensaios triaxiais de carga repetida (Vieira, 2006).....	41
FIG. 11 – Forças aplicadas no corpo de prova durante o ensaio de compressão diametral de cargas repetidas (Vieira, 2006).....	42
FIG. 12 – Equipamento de ensaios de compressão diametral de cargas repetidas (Vieira, 2006)	43
FIG. 13 – Deslocamentos de um corpo de prova sujeito a carregamento vertical (University of Washington, 2007)	48
FIG. 14 – Exemplo de discretização axissimétrica do pavimento (Medina e Motta, 2005) ...	51
FIG. 15 – Tensões atuantes em um ponto de um domínio axissimétrico (Soriano, 2003)	55
FIG. 17 – Fluxograma do método incremental	59
FIG. 18 – Eixos de coordenadas locais e nós dos elementos utilizados no programa	87
FIG. 19 – Processo de geração automática de malha.....	88
FIG. 20 – Representação do pavimento analisado por Preussler (1983)	94
FIG. 21 – Representação esquemática do FWD e sua bacia de deformação (Vieira, 2006)....	97
FIG. 22 – Foto de um aparelho do tipo FWD (Vieira, 2006).....	97
FIG. 23 – Representação esquemática da viga Benkelman (Vieira, 2006).....	98
FIG. 24 – Viga Benkelman em utilização (Vieira, 2006)	99
FIG. 25 – Representação esquemática da Seção A	100
FIG. 26 – Representação esquemática da Seção B	100
FIG. 27 – Representação esquemática da Seção C	100

FIG. 28 – Representação esquemática da Seção D	101
FIG. 29 – Representação esquemática da Seção E.....	102
FIG. 30 – Deflexões da seção A.....	106
FIG. 31 – Deflexões da seção B.....	107
FIG. 32 – Deflexões da Seção C	108
FIG. 33 – Deflexões da Seção D	109
FIG. 34 – Deflexões da Seção E	110
FIG. 35 – Exemplo de arquivo de entrada Dados.txt.....	124
FIG. 36 – Processo de geração automática de malha.....	128

LISTA DE TABELAS

TAB. 1 – Modelos matemáticos de módulo de resiliência de solos e materiais de pavimentação em função do estado de tensões.	47
TAB. 2 – Valores de coeficiente de Poisson nos solos (Barata, 1984).	49
TAB. 3 – Valores do coeficiente de Poisson (Pinto e Preussler, 2002).	49
TAB. 4 – Tipos da classe material	68
TAB. 5 – Coordenadas e pesos dos pontos de Gauss do elemento	90
TAB. 6 – Propriedades das camadas do pavimento analisado por Preussler (1983) no primeiro caso	94
TAB. 7 – Propriedades das camadas do pavimento analisado por Preussler (1983) no segundo caso	94
TAB. 8 – Resultados obtidos por Preussler (1983)	95
TAB. 9 – Módulos de resiliência das camadas do pavimento analisado por Preussler (1983), convertidos.	95
TAB. 10 – Resultados obtidos na análise com o programa desta dissertação	95
TAB. 11 – Propriedades das camadas dos pavimentos da BR-101/RJ analisados por Pinto (1991)	101
TAB. 12 – Propriedades das camadas dos pavimentos da BR-040/RJ analisados por Pinto (1991)	102
TAB. 13 – Resultados obtidos nos trechos experimentais analisados por Pinto (1991).....	103
TAB. 14 – Resultados obtidos na análise da Seção A	105
TAB. 15 – Resultados obtidos na análise da Seção B.....	106
TAB. 16 – Resultados obtidos na análise da Seção C.....	107
TAB. 17 – Resultados obtidos na análise da Seção D	108
TAB. 18 – Resultados obtidos na análise da Seção E.....	110
TAB. 19 – Deslocamentos obtidos na análise de sensibilidade	111
TAB. 20 – Variação dos deslocamentos na análise de sensibilidade.....	111
TAB. 21 – Tipos de material	126

LISTA DE ABREVIATURAS E SÍMBOLOS

SÍMBOLOS

σ_z	-	Tensão normal vertical
K_B	-	Constante adimensional
P	-	Carga pontual aplicada na superfície
z	-	Profundidade
r	-	Distância radial
a	-	Raio do carregamento circular (referente ao pneu de uma roda)
q	-	Intensidade uniforme do carregamento circular
E	-	Módulo de elasticidade
ν	-	Coefficiente de Poisson
σ_r	-	Tensão normal radial
σ_t	-	Tensão normal tangencial
τ_{rz}	-	Tensão de cisalhamento no plano r na direção z
τ_{zr}	-	Tensão de cisalhamento no plano z na direção r
$\{F\}$	-	Matriz-coluna dos carregamentos conhecidos aplicados nos nós da estrutura (cargas nodais)
$[K]$	-	Matriz de rigidez da estrutura ou sistema
$\{U\}$	-	Matriz-coluna dos deslocamentos realizados pelos nós da estrutura (deslocamentos nodais), quando submetidos às cargas nodais
K^e	-	Matriz de rigidez do elemento e
B	-	Matriz gradiente de deformação
D	-	Matriz de elasticidade, constitutiva ou de coeficientes elásticos
$\{\varepsilon\}$	-	Vetor das deformações
$\{u\}^e$	-	Vetor dos deslocamentos nodais de elemento
$\{\sigma\}$	-	Vetor das tensões
$W_{interno}$	-	Trabalho interno
M_R	-	Módulo de resiliência

σ_d	-	Tensão-desvio
σ_3	-	Tensão confinante
ε_R	-	Deformação específica vertical recuperável ou resiliente
σ_T	-	Tensão de tração no plano diametral vertical de uma amostra cilíndrica de mistura betuminosa
ε_T	-	Deformação específica recuperável correspondente à tensão σ_T
d	-	Diâmetro do corpo de prova de mistura betuminosa
t	-	Altura do corpo de prova de mistura betuminosa
K_1, K_2, K_3, K_4, K_5	-	Constantes obtidas experimentalmente
Θ	-	Primeiro invariante de tensões ($\sigma_1 + \sigma_2 + \sigma_3$), no caso do ensaio de compressão triaxial $\Theta = \sigma_1 + 2\sigma_3 = \sigma_d + 3\sigma_3$
R^2	-	Coefficiente de determinação
y_i	-	Valor conhecido, referente ao ponto i
\bar{y}	-	Média aritmética dos pontos conhecidos
\hat{y}_i	-	Valor da função de regressão no ponto i
$\varepsilon_D = \frac{\Delta D}{D}$	-	Deformação transversal ou radial
$\varepsilon_L = \frac{\Delta L}{L}$	-	Deformação longitudinal ou axial
b_1, b_2, b_3, b_4	-	Constantes obtidas experimentalmente
u	-	Deslocamento na direção radial (horizontal)
w	-	Deslocamento na direção axial (vertical)
$[L]$	-	Matriz de operadores diferenciais
N	-	Função de interpolação
$\{f_i\}_q^e$	-	Vetor de forças nodais equivalentes às forças de superfície
$[I]$	-	Matriz identidade
$\{q\}$	-	Vetor de forças de superfície de componentes q_x, q_y e q_z
α	-	Ângulo que o eixo r das coordenadas cilíndricas faz com o eixo x das coordenadas cartesianas
σ_x	-	Tensão normal na direção do eixo x das coordenadas cartesianas

σ_y	-	Tensão normal na direção do eixo y das coordenadas cartesianas
τ_{xy}	-	Tensão de cisalhamento no plano x na direção y
τ_{yz}	-	Tensão de cisalhamento no plano y na direção z
τ_{xz}	-	Tensão de cisalhamento no plano x na direção z
$[J]$	-	Jacobiano ou matriz jacobiana
$r(\xi)$	-	Coordenada global r em função da coordenada local ξ
$W\xi_i$ e $W\eta_i$	-	Pesos referentes às coordenadas locais ξ e η do ponto de Gauss i , respectivamente
J_r	-	Matriz jacobiana reduzida
R e Z	-	Eixos radial e vertical de coordenadas globais, respectivamente
$F_z(i)$	-	Carga nodal na direção vertical referente ao nó i
$r(i)$	-	Coordenada global radial do nó i
$\sigma_{d \min}$	-	Tensão-desvio mínima
$\sigma_{3 \min}$	-	Tensão confinante mínima
$D_0, D_{20}, D_{30}, D_{45}$	-	Deflexões dos pontos localizados a 0, 20, 30 e 45 centímetros de distância do centro da carga circular, respectivamente
D_{60}, D_{90} e D_{120}	-	Deflexões dos pontos localizados a 60, 90 e 120 centímetros de distância do centro da carga circular, respectivamente
F_p	-	Força de pico
M	-	Massa do peso que cai
g	-	Aceleração da gravidade
h	-	Altura de queda
k	-	Constante de mola do sistema de amortecedores
$[M]$	-	Matriz de massa
$[C]$	-	Matriz de amortecimento
$[K]$	-	Matriz de rigidez
$\{\dot{U}\}$	-	Derivada primeira do vetor deslocamento em relação ao tempo
$\{\ddot{U}\}$	-	Derivada segunda do vetor deslocamento em relação ao tempo
$\{F(t)\}$	-	Vetor de forças em função do tempo

LISTA DE SIGLAS

CBR	California Bearing Ratio (Índice de Suporte Califórnia)
DNER	Departamento Nacional de Estradas de Rodagem
DNIT	Departamento Nacional de Infra-Estrutura de Transportes
MEF	Método dos Elementos Finitos
2D	Bidimensional
3D	Tridimensional
ABNT	Associação Brasileira de Normas Técnicas
ANSI	American National Standards Institute
ISO	International Standards Organization
POO	Programação Orientada a Objetos
FWD	<i>“Falling Weight Deflectometer”</i>

RESUMO

Considerando a substituição entre os métodos de dimensionamento de pavimentos, este trabalho tem por objetivo propor uma modelagem para o mesmo através do método dos elementos finitos e compará-la com resultados experimentais obtidos em campo.

Com esse intuito, é apresentada a evolução dos métodos de cálculo de tensões e deformações em pavimentos. Em seguida são descritos, para o método dos elementos finitos, sua formulação, suas características e os tipos que podem ser utilizados para pavimentos. Posteriormente, são apresentadas as definições e as propriedades desta estrutura estratificada, sendo realizada sua modelagem através de elementos axissimétricos do método dos elementos finitos.

Através do uso da linguagem de programação C++, implementa-se o modelo proposto em um software, intitulado de *AXIPAV*, o primeiro programa de elementos finitos axissimétricos de oito nós que utiliza módulo de resiliência. Esse modelo é validado através da comparação com pavimentos analisados por Preussler (1983) *apud* Silva (1995) e por Pinto (1991).

Pode-se concluir que os valores da deflexão obtidos para o pavimento analisado por Preussler (1983) se aproximam dos medidos em campo. Para os analisados por Pinto (1991) os resultados em três seções se aproximam dos medidos em campo e, nas outras duas seções, houve grandes diferenças, porém estas deflexões são frutos de ensaios dinâmicos e as do programa são de cargas estáticas. Então estes resultados devem ser comparados qualitativamente.

Concomitantemente, esta diferença pode ser justificada pela adoção de valores não representativos para o módulo de resiliência de cada camada da estrutura durante a análise, pois o mesmo é fortemente dependente do grau de compactação e de sua umidade, entre outros fatores.

ABSTRACT

Considering the changing over the methods of designing pavements, this work aims to propose a modeling of pavements by the finite element method and to compare it with experimental results on the field.

With this intention, it is shown the evolution of the methods to calculate pavement stresses and strains. Then the finite element method formulation, characteristics and types used in pavements are described. Later, the pavements' definitions and properties are presented, and their modeling is accomplished through axisymmetric finite elements.

The proposed model is implemented with the C++ programming language and the software is called as AXIPAV, which is the first axisymmetric finite elements program with eight nodes and resilient modulus consideration. This model is validated by comparison with pavements analyzed by Preussler (1983) *apud* Silva (1995) and by Pinto (1991).

It can be concluded that the deflections values obtained for the pavement analyzed by Preussler (1983) approach the ones measured in field. For the pavements analyzed by Pinto (1991) the results in three sections approach the ones measured in field and, in the other two sections, there were great differences; however, these deflections were obtained from dynamic testing and the ones in program are due to static loads. Then these results should be compared qualitatively.

Accordingly, this difference can be justified by adoption of non-representative resilient modulus of each structure layer during the analysis, because it is strongly dependent on its compaction degree and its humidity, among other factors.

1 INTRODUÇÃO

O método de dimensionamento de pavimentos mais utilizado no Brasil é o método empírico do DNER – Método do CBR, proposto pelo engenheiro Murillo Lopes de Souza em 1969. Entretanto, este método só garante a proteção do pavimento quanto ao afundamento de trilha de roda, não considerando o problema de fadiga das camadas asfálticas (pavimentos flexíveis) e das camadas cimentadas (pavimentos semi-rígidos).

Este método determina a espessura de cada uma das camadas que compõem o pavimento, através do valor do Índice de Suporte Califórnia ou CBR (*California Bearing Ratio*) que cada uma delas possui. A figura 1 mostra esquematicamente a estrutura de um pavimento.

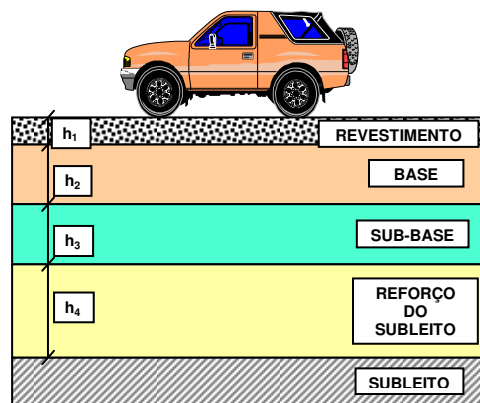


FIG. 1 – Estrutura esquemática do pavimento (Vieira, 2006)

O valor do CBR de cada camada é determinado através do método de ensaio preconizado pelo DNER (atual DNIT), em corpos-de-prova indeformados ou moldados em laboratório, para as condições de massa específica aparente e umidade especificada para o serviço no campo, e submersos durante quatro dias.

Segundo Medina e Motta (2005), a umidade de equilíbrio, nas camadas dos pavimentos de rodovias bem projetadas e construídas com dispositivos de drenagem eficientes, pode ser considerada igual, no máximo, ao teor de umidade ótimo do ensaio Proctor normal de compactação. Afirmam ainda que este é o caso das rodovias federais e estaduais brasileiras em que a drenagem é bem cuidada.

Então, para o caso destas rodovias, o método do CBR superdimensiona a estrutura destes pavimentos para as condições reais de umidade e resistência do subleito. Ele também implica em grandes espessuras de camadas granulares de sub-base e base, que se não forem corretamente compactadas tornam-se sujeitas a deformações excessivas.

Visando sanar estes problemas, foram desenvolvidos métodos de dimensionamento, denominados de mecânicos ou racionais, que consideram os principais mecanismos de degradação do pavimento: fadiga das camadas de maior rigidez (revestimento de concreto asfáltico e camadas cimentantes), afundamento de trilha de roda (deformação permanente) e ruptura plástica (a poucas repetições).

Motta (1991) apresentou um método mecânico para o dimensionamento de pavimentos flexíveis baseado nas propriedades dos materiais, nas condições de tráfego e ambientais, nas tensões e deformações do pavimento, e nas condições de serventia, de forma que o desempenho seja satisfatório ao longo da vida útil. Um esquema que engloba estes fatores é mostrado no fluxograma da figura 2.

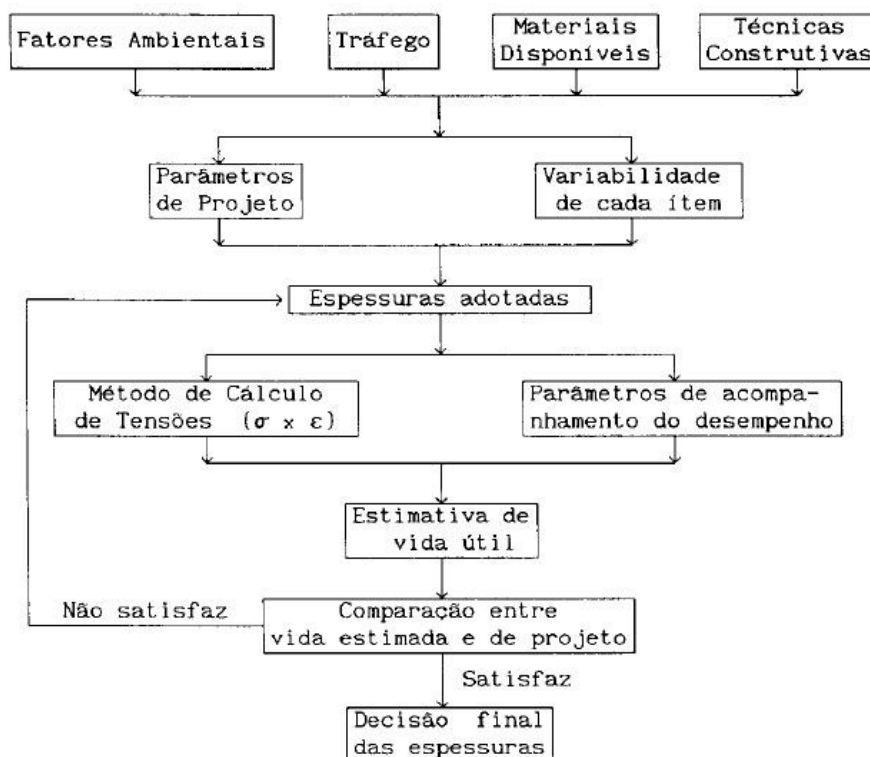


FIG. 2 – Esquema do método de dimensionamento proposto por Motta (1991)

De maneira geral, os principais parâmetros necessários para o dimensionamento racional (ou mecanístico) de uma estrutura são: a tensão vertical no subleito, considerada a camada mais fraca em termos de deformações permanentes; e a deformação específica de tração na camada inferior do revestimento (maior deformação de tração na fibra inferior do revestimento, por unidade de comprimento), responsável pela fadiga do material asfáltico.

Inicialmente, estima-se a estrutura (espessura de cada camada) do pavimento e calculam-se aqueles dois parâmetros para compará-los com os valores estabelecidos pelos critérios de projeto. Se os valores admissíveis de projeto forem maiores, então as espessuras adotadas são satisfatórias, caso contrário, as espessuras devem ser redimensionadas.

Para a obtenção das tensões e deformações utilizam-se programas computacionais. Com esta finalidade, vários programas foram desenvolvidos nos últimos anos, entre eles soluções da teoria de camadas elásticas resolvidas pelo método dos elementos finitos, entre outros.

Deste modo, necessita-se de uma ferramenta confiável e de fácil utilização para a determinação das tensões e deformações atuantes na estrutura do pavimento.

Este trabalho tem por objetivo propor uma modelagem de pavimentos através do método dos elementos finitos e compará-lo com resultados obtidos em campo.

A presente dissertação está estruturada em seis capítulos e dois apêndices, conforme apresentados a seguir:

- Capítulo 1 – Introdução.
- Capítulo 2 – Revisão Bibliográfica. Neste capítulo é apresentado um histórico da evolução dos cálculos de tensões e deformações em pavimentos e é descrita a formulação do Método dos Elementos Finitos.
- Capítulo 3 – Modelagem Numérica do Pavimento. Neste capítulo são apresentadas as definições, a classificação e as principais características de um pavimento, e é realizada a sua modelagem em elementos axissimétricos do método dos elementos finitos.

- Capítulo 4 – Implementação Computacional. Neste capítulo são apresentadas as principais características da linguagem de programação C++, e é realizada a implementação do programa de análise de pavimentos por elementos finitos.
- Capítulo 5 – Validação do Modelo. Neste capítulo é realizada a análise de algumas estruturas de pavimento que foram objeto de análise de outros trabalhos e, mediante comparação de seus resultados, o programa desenvolvido nesta dissertação é validado.
- Capítulo 6 – Conclusões e Recomendações. Neste capítulo são apresentadas as conclusões deste trabalho e as recomendações e sugestões para trabalhos futuros.
- Apêndice A – Manual do Usuário.
- Apêndice B – Código-Fonte do Modelo.

2 REVISÃO BIBLIOGRÁFICA

Este capítulo tem por finalidade apresentar um histórico da evolução dos cálculos de tensões e deformações em pavimentos e descrever a formulação do Método dos Elementos Finitos.

2.1 A EVOLUÇÃO DOS CÁLCULOS DE TENSÕES E DEFORMAÇÕES EM PAVIMENTOS

Em 1885 Boussinesq desenvolveu uma solução que permite calcular a tensão em qualquer ponto de um semi-espço homogêneo, isotrópico e elástico linear, bem como o deslocamento na superfície provocado por uma carga atuante em um ponto qualquer da superfície do meio elástico. A figura 3 ilustra este sistema, no qual o semi-espço possui módulo de elasticidade longitudinal ou de Young E , coeficiente de Poisson ν e espessura h infinita e a origem dos eixos de coordenadas polares situa-se no ponto de aplicação da carga.

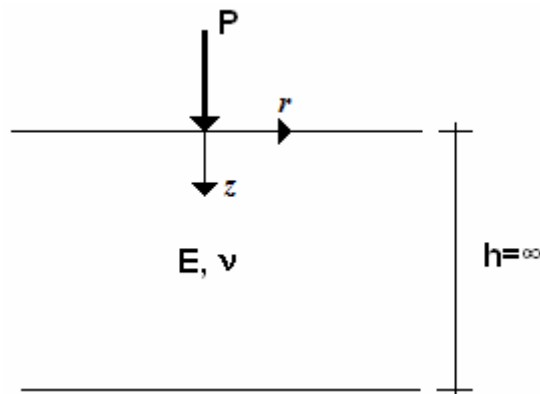


FIG. 3 – Sistema de Boussinesq

A tensão vertical para uma profundidade qualquer é dada por:

$$\sigma_z = K_B \frac{P}{z^2} \quad (2.1)$$

Onde:

σ_z = tensão vertical;

$$K_B = \text{constante adimensional, dada pela expressão } K_B = \frac{3}{2\pi} \frac{1}{\left[1 + \left(\frac{r}{z}\right)^2\right]^{\frac{5}{2}}};$$

P = carga pontual aplicada na superfície;

z = profundidade;

r = distância radial.

Observa-se que a tensão vertical, neste caso, é dependente da profundidade e da distância radial e independente das propriedades físicas do meio. A solução de Boussinesq foi usada para os primeiros estudos das distribuições dos campos de tensão, deformação e deslocamentos em pavimentos flexíveis.

Pesquisas posteriores expandem as equações de Boussinesq a soluções com carregamento circular uniforme, pois no estudo de pavimentos flexíveis, a carga na superfície é distribuída sobre uma área aproximadamente circular, formada pelo contato entre o pneu e o pavimento. Segundo Huang (1993), o erro cometido por esta suposição de carregamento circular é pequeno.

A figura 4 mostra o semi-espço homogêneo sob um carregamento circular de raio a e intensidade uniforme q . Devido à axissimetria, têm-se três componentes de tensão normal σ_z , σ_r , σ_t e uma componente de tensão cisalhante τ_{rz} (igual a τ_{zr}). Estas tensões são função do carregamento (q e a) e de sua posição (r e z), como mostram as equações a seguir.

$$\sigma_z = \frac{3qz^3}{2\pi} \int_0^a \int_0^{2\pi} (r^2 - 2rb \cos \theta + b^2 + z^2)^{\frac{5}{2}} db d\theta \quad (2.2)$$

$$\sigma_r = \frac{q}{2\pi} (1 - 2\nu) \int_0^a \int_0^{2\pi} \left\{ \left[-\frac{1}{(r^2 - 2rb \cos \theta + b^2)} - \frac{z(r^2 - 2rb \cos \theta + b^2 + z^2)^{\frac{1}{2}}}{(r^2 - 2rb \cos \theta + b^2)} \right] - \right. \\ \left. - \left[3(r^2 - 2rb \cos \theta + b^2)z(r^2 - 2rb \cos \theta + b^2 + z^2)^{\frac{5}{2}} \right] \right\} db d\theta \quad (2.3)$$

$$\sigma_r = \frac{q}{2\pi}(1-2\nu) \int_0^a \int_0^{2\pi} \left\{ \left[-\frac{1}{(r^2 - 2rb \cos \theta + b^2)} + \frac{z(r^2 - 2rb \cos \theta + b^2 + z^2)^{\frac{1}{2}}}{(r^2 - 2rb \cos \theta + b^2)} \right] + \left[z(r^2 - 2rb \cos \theta + b^2 + z^2)^{\frac{3}{2}} \right] \right\} db d\theta \quad (2.4)$$

$$\tau_{rz} = \frac{3qz^2}{2\pi} \int_0^a \int_0^{2\pi} \sqrt{r^2 - 2rb \cos \theta + b^2} (r^2 - 2rb \cos \theta + b^2 + z^2)^{\frac{5}{2}} db d\theta \quad (2.5)$$

Devido à grande complexidade destas equações, foram confeccionados diversos ábacos que possibilitam a determinação de cada uma destas tensões para um coeficiente de Poisson dado.

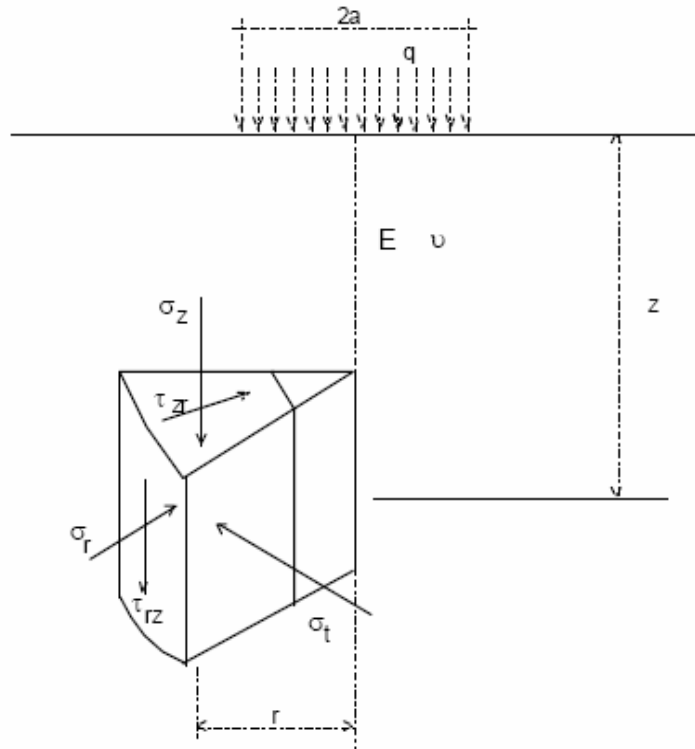


FIG. 4 – Componentes de tensão sob carregamento axissimétrico (Aedo, 1997)

Em 1943, Burmister desenvolveu soluções para um sistema constituído por duas camadas sobrepostas. Em seguida, o mesmo as estendeu para um sistema formado por três camadas, em 1945. Com auxílio de microcomputadores estas soluções têm sido bastante utilizadas nos estudos envolvendo camadas múltiplas.

Na solução dos problemas de sistema de camadas elásticas foram consideradas algumas hipóteses:

- cada camada é homogênea, isotrópica e elástica linear, com módulo de elasticidade longitudinal E e coeficiente de Poisson ν ;
- as camadas são admitidas infinitas horizontalmente e finitas verticalmente;
- a camada final, geralmente o subleito, é considerada infinita nas direções horizontal e vertical;
- as camadas são perfeitamente aderentes entre si;
- a superfície da camada superior é livre de tensão normal e cisalhante fora da área circular uniformemente carregada.

Os valores da tensão vertical no topo do subleito, o deslocamento vertical (recalque) na superfície, o deslocamento vertical na interface entre camadas e a deformação horizontal de tração crítica são, em geral, usados como critérios no projeto de pavimentos.

Vários programas de computador baseados nesta teoria foram desenvolvidos para análises de pavimentos, tais como o CHEV em 1963, pela Chevron Research Company; o BISAR em 1972, pela Shell, considerando carregamentos horizontais e verticais; o ELSYM5 em 1972, pela Universidade da Califórnia, em Berkeley, para sistemas de cinco camadas.

Em todos estes casos foram considerados que os materiais do pavimento tinham comportamento elástico linear, sendo empregado o princípio da superposição das tensões e deformações para determinação dos efeitos de rodas múltiplas a partir dos resultados calculados para uma única roda.

A figura 5 mostra o sistema de camadas múltiplas com carregamento circular.

A principal desvantagem deste modelo é que os solos e materiais granulares quase nunca apresentam este comportamento sob cargas repetidas. Então, torna-se necessário outro modelo para o sistema, sendo, o mais apropriado para estruturas de pavimentos, a análise de sistemas de camadas de comportamento tensão-deformação não-lineares (módulo de elasticidade dependente das tensões atuantes).

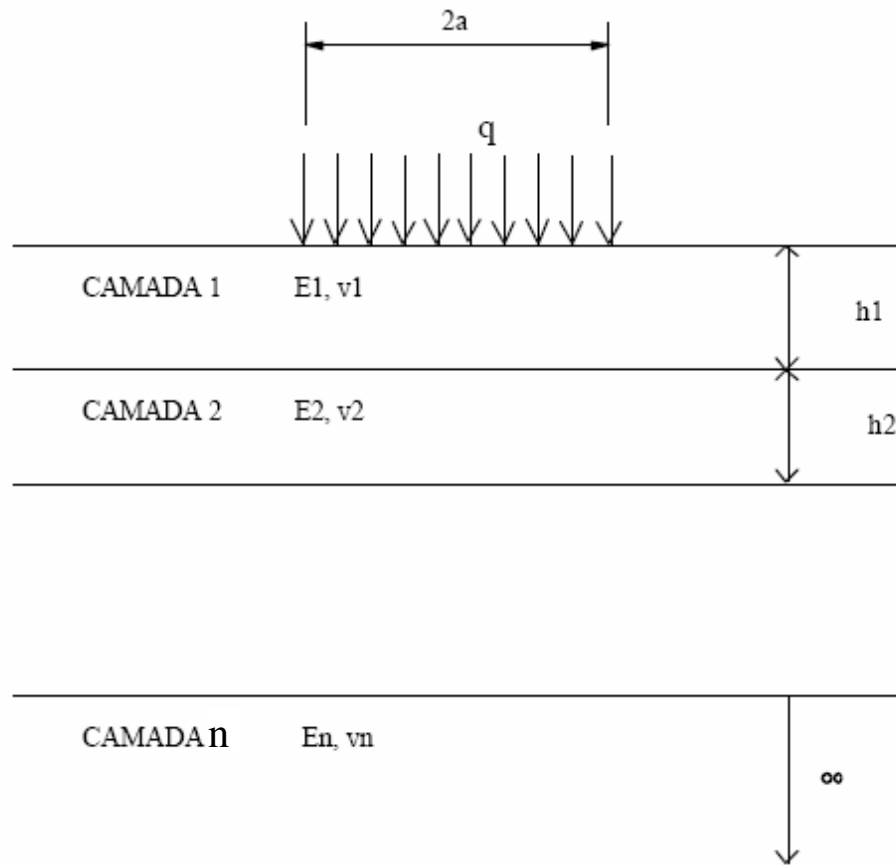


FIG. 5 – Sistema de camadas múltiplas linearmente elásticas (Aedo, 1997)

Neste modelo, as propriedades mecânicas são dependentes do estado de tensões e, portanto, mudam de valor conforme a situação do ponto analisado no interior de cada camada, em relação à posição do carregamento. A figura 6 ilustra este sistema.

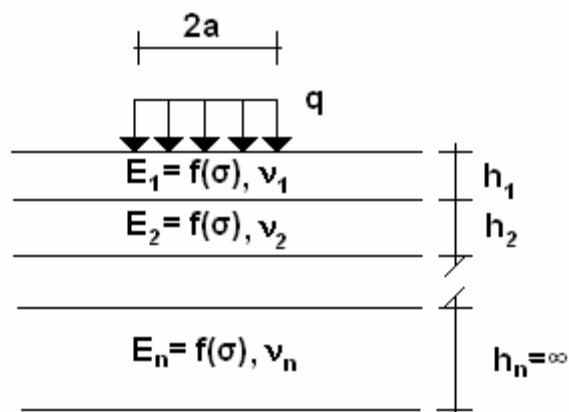


FIG. 6 – Sistema de camadas de comportamento tensão-deformação não lineares

Este modelo foi ampliado no programa TECNAPAV em 1987, dando um tratamento tridimensional às cargas de rodas múltiplas em estruturas de pavimentos flexíveis.

Huang em 1993 desenvolveu o programa KENLAYER baseado na solução de camadas elásticas sujeitas às rodas múltiplas. Algumas simplificações foram incorporadas para a análise de camadas não-lineares e viscoelásticas. Por exemplo, para calcular o módulo de elasticidade de uma camada com comportamento não-linear, o programa considera apenas o ponto no centro e imediatamente abaixo de uma roda única ou, no caso de duas rodas, o ponto situado na distância média entre elas; quando o pavimento é solicitado com carregamento de eixos “tandem” duplo ou triplo, utiliza-se somente um eixo para determinação dos módulos das camadas não-lineares.

Estas dificuldades e simplificações puderam ser superadas usando métodos numéricos mais poderosos, tais como o método dos elementos finitos. O procedimento básico do método consiste em: dividir o meio contínuo através de elementos finitos (malhamento); estabelecer as propriedades de cada elemento; reunir as equações dos elementos para obter o modelo discretizado da estrutura; aplicar os carregamentos conhecidos e as condições de contorno, especificando como o modelo é vinculado; resolver o sistema de equações algébricas resultante, calculando todos os deslocamentos desconhecidos bem como os valores de tensão e deformação desejados (Cook, 1989 *apud* Aedo, 1997).

Nas diversas aplicações do Método dos Elementos Finitos em estudos envolvendo o comportamento de pavimentos quase sempre se considera o caso bidimensional axissimétrico, limitando-se a modelar o caso de uma roda única.

Nos últimos anos foram desenvolvidos vários programas utilizando este método para análises não-lineares, tais como o FEPAVE em 1965, desenvolvido na Universidade da Califórnia, em Berkeley; o ILLIPAVE em 1980, da Universidade de Illinois; o MICHPAVE em 1990, da Universidade de Michigan, sendo o FEPAVE o mais utilizado no Brasil.

Uma cópia do programa FEPAVE, desenvolvido na Universidade da Califórnia, em Berkeley, em 1965 por E.L. Wilson, foi doada para a Universidade Federal de Rio de Janeiro (UFRJ) em 1973, sendo utilizada e aperfeiçoada em várias teses de mestrado e doutorado. As

últimas modificações foram implementadas por Motta em 1991, aplicando o critério de confiabilidade, e por Silva em 1995, criando rotinas para uma utilização interativa e amigável do programa em microcomputador.

Entretanto, o comportamento real da estrutura do pavimento é de natureza tridimensional, submetido a carregamentos múltiplos e com materiais de comportamento tensão deformação não linear. Atualmente, a formulação tridimensional dos elementos finitos está sendo utilizada para auxiliar em uma melhor compreensão da resposta de pavimentos reais.

2.2 O MÉTODO DOS ELEMENTOS FINITOS

A modelagem numérica do Método dos Elementos Finitos (MEF) baseia-se na transformação de equações diferenciais que regem um problema específico em equações algébricas de mais fácil resolução (Rede Asfalto, 2006).

2.2.1 INTRODUÇÃO

Neste processo, o domínio do problema é dividido em uma série de regiões, os elementos, de topologia simples como triângulos, quadriláteros, tetraedros e hexaedros e cuja geometria é definida pelas coordenadas de um conjunto de pontos, os nós. O conjunto de nós e elementos é chamado de malha (Rede Asfalto, 2006).

Segundo Bathe (1996), a modelagem e análise de um problema desta natureza consistem de três etapas: (i) pré-processamento; (ii) processamento e (iii) pós-processamento. Como ilustra a figura 7, cada uma destas etapas requer o emprego de técnicas específicas de modelagem de tal forma que, na grande maioria das vezes há um programa específico para cada etapa.

O pré-processamento é responsável pelo modelo geométrico, o domínio, e sua divisão em sub-regiões específicas denominadas elementos finitos obedecendo a regras de divisão de acordo com a forma geométrica do elemento escolhido (triangular ou quadrática, por

exemplo). Este processo recebe o nome de discretização e o conjunto de elementos subdivididos unidos por pontos nodais é denominado de malha.

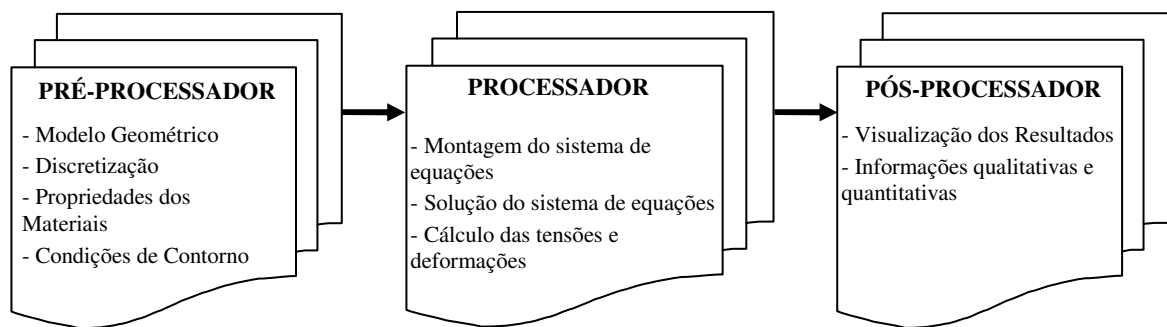


FIG. 7 – Modelagem de um problema com o MEF (Evangelista Jr. et al., 2003)

O pré-processamento também é responsável pela aplicação dos atributos: os carregamentos e as restrições de deslocamentos impostos à estrutura (condições de contorno) e as propriedades dos materiais. Estas tarefas iniciais despendem uma grande quantidade de tempo tanto em sua concepção, como em sua realização.

Em muitas aplicações, o pré-processamento deve disponibilizar a aplicação de novas técnicas que possam criar modelos mais realísticos, discretizações mais eficientes, assim como também reduzir o tempo gasto em tais tarefas.

Segundo Foley *et al.* (1996) *apud* Evangelista Jr. *et al.* (2003), a forma geométrica dos objetos é um dado do problema que requer um processo de transformação de suas formas reais para modelos geométricos representáveis computacionalmente e que podem ser armazenados e manipulados convenientemente.

A etapa de processamento corresponde à fase de análise propriamente dita. É no processamento que a rigidez da estrutura é definida, montando-se a matriz de rigidez global, baseada nos modelos matemáticos que descrevem o comportamento físico dos materiais constituintes da estrutura analisada, ou seja, os modelos constitutivos que relacionam tensão e deformação no material.

A partir desta avaliação, pode-se proceder à montagem e à resolução de equações algébricas que resultarão nas soluções de um campo requerido, como por exemplo,

deslocamentos. O MEF, que além de facilitar a solução, diminui o esforço computacional, tem encontrado ampla utilização em vários campos da engenharia, inclusive na análise estrutural de pavimentos (Cho *et al.*, 1996 *apud* Evangelista Jr. *et al.*, 2003).

Segundo Guimarães (1992) *apud* Evangelista Jr. *et al.* (2003), o pós-processamento recebe como entrada de dados a saída do programa da fase de processamento e objetiva fornecer informações quantitativas e qualitativas das grandezas processadas utilizando técnicas da computação gráfica.

A representação dos resultados de uma análise por elementos finitos possui algumas características que fazem com que determinadas informações sejam fundamentais, como por exemplo, visualizações de deslocamentos, tensões e deformações.

2.2.2 CARACTERÍSTICAS DO MÉTODO DOS ELEMENTOS FINITOS

A apresentação formal do método dos elementos finitos é atribuída a Turner *et al.* (1956) *apud* Wang (2001). O termo “elemento finito” foi introduzido por Clough em 1960. Desde seu início, a literatura sobre a análise de elementos finitos tem crescido rapidamente e existem muitas pesquisas e publicações que são adequadas à teoria e aplicação do método (Zienkiewicz e Taylor, 1988 *apud* Wang, 2001, Zienkiewicz e Taylor, 2005 e Zienkiewicz *et al.*, 2005).

Os modelos bidimensionais foram os primeiros exemplos de sucesso da aplicação do método dos elementos finitos e são utilizados desde 1960 (Zienkiewicz e Taylor, 1988 *apud* Wang, 2001). A aplicação do MEF na análise de modelos estruturais consiste na discretização de um domínio em subdomínios, chamados elementos, responsáveis por capturar variações de tensão, deformação e deslocamento sobre sua área ou volume.

A relação entre os deslocamentos e as cargas de uma estrutura é dada pela seguinte expressão:

$$\{F\} = [K]\{U\} \quad (2.6)$$

Onde:

$\{F\}$ = matriz-coluna dos carregamentos conhecidos aplicados aos nós da estrutura (cargas nodais);

$[K]$ = matriz de rigidez da estrutura;

$\{U\}$ = matriz-coluna dos deslocamentos dos nós da estrutura (deslocamentos nodais), quando submetidos às cargas nodais.

A matriz de rigidez da estrutura é calculada através da combinação das matrizes de rigidez dos elementos em coordenadas globais de cada elemento do sistema. As matrizes de rigidez elementar são calculadas usando a expressão a seguir:

$$K^e = \int_{V^e} B^T D B dV^e \quad (2.7)$$

Onde:

K^e = matriz de rigidez do elemento e ;

B = matriz que relaciona deformações com deslocamentos nodais do elemento (matriz gradiente de deformação);

D = matriz de elasticidade, constitutiva ou de coeficientes elásticos.

Após a montagem da matriz de rigidez do sistema (matriz de rigidez global), os deslocamentos nodais são determinados através da solução do sistema de equações lineares (2.6). As tensões e deformações podem ser determinadas a partir dos deslocamentos nodais, utilizando as expressões (2.8) e (2.9).

$$\{\varepsilon\} = [B]\{u\}^e \quad (2.8)$$

Onde:

$\{\varepsilon\}$ = vetor das deformações em um ponto qualquer do elemento;

$\{u\}^e$ = vetor dos deslocamentos nodais do elemento.

$$\{\sigma\} = [D]\{\varepsilon\} \quad (2.9)$$

Onde:

$\{\sigma\}$ = vetor das tensões em um ponto qualquer do elemento.

2.2.3 ELEMENTOS FINITOS BIDIMENSIONAIS

Um pavimento flexível pode ser representado como um sistema de múltiplas camadas semi-infinitas com diferentes propriedades materiais para cada camada. Basicamente, existem

dois tipos de modelos bidimensionais (2D) de elementos finitos que podem ser desenvolvidos para tal sistema: axissimétrico bidimensional e estado plano de deformações.

A abordagem da modelagem axissimétrica assume que o sistema pavimento possui simetria axial em relação ao seu eixo vertical tanto para as propriedades materiais e geométricas, como para o carregamento de tráfego (uma carga circular agindo no centro do pavimento).

Este tipo de modelo é incapaz de considerar a interface de cisalhamento entre as camadas ou estabelecer condições de contorno ou descontinuidades nas estruturas do pavimento, tais como juntas ou trincas. Este modelo também possui outra limitação: ele só pode ser utilizado para obter respostas do sistema pavimento se o local do carregamento não for perto de sua borda ou de uma trinca.

No estado plano de deformações, a tensão na direção perpendicular ao plano XY não é zero, mas a deformação nesta direção é zero e nenhuma contribuição para o trabalho interno é feita por esta tensão, que pode ser calculado explicitamente a partir das componentes das três tensões principais.

Em um sistema conservativo, o trabalho realizado pelas forças externas (peso próprio, cargas distribuídas e/ou concentradas na superfície) nos deslocamentos associados é igual ao trabalho das forças internas. As forças internas de um corpo em equilíbrio podem ser dadas pelo estado de tensões presente em cada ponto do seu volume e seu trabalho (trabalho interno) é dado pela seguinte expressão:

$$W_{interno} = \int_V \{\epsilon\}^T \{\sigma\} dv \quad (2.10)$$

Onde:

$$W_{interno} = \text{trabalho interno.}$$

A suposição do estado plano de deformações é realista para corpos longos com área de seção transversal constante sujeita a cargas somente na direção X e/ou Y sem variação na direção Z. A carga agirá ao longo do comprimento exterior do corpo.

A figura 8 mostra a representação esquemática do estado plano de deformações para um pavimento.

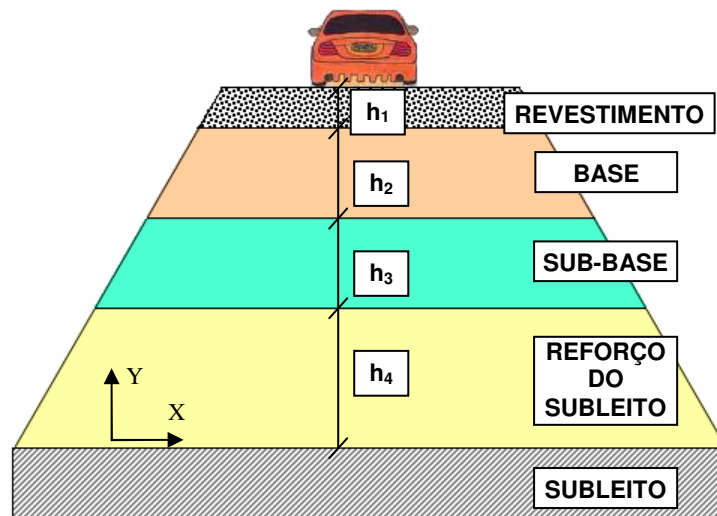


FIG. 8 – Representação esquemática do estado plano de deformações para um pavimento

O método bidimensional de elementos finitos possui algumas vantagens, como:

- requer um tempo computacional relativamente pequeno em relação ao tridimensional;
- o pré-processamento e o pós-processamento são de implementação mais simples;
- pode superar algumas deficiências da análise de camadas elásticas (conforme item 2.2.1).

Entretanto, o método de elementos finitos 2D apresenta algumas limitações, como:

- o modelo de estado plano de deformações é limitado para o uso de cargas lineares, considerando carregamento contínuo ao longo da rodovia, o que levará a superdimensionamento da mesma no estudo da fadiga por exemplo;
- o modelo axissimétrico supõe os modelos de cargas de roda como sendo circulares.

Pesquisas mostram que a área de contato entre o pavimento e a roda é essencialmente retangular (Weissman, 1997 *apud* Wang, 2001 e Cunagin *et al.*, 1991 *apud* Wang, 2001), e que a tensão de contato pneu-pavimento pode variar significativamente com sua área de contato (Tielking *et al.*, 1995 *apud* Wang, 2001).

2.2.4 O MÉTODO DOS ELEMENTOS FINITOS TRIDIMENSIONAIS

Com a finalidade de superar as limitações da análise de camadas elásticas e dos métodos de elementos finitos 2D, métodos de elementos finitos tridimensionais (3D) estão sendo cada vez mais usados para modelar as respostas de pavimentos flexíveis. A modelagem em elementos finitos 3D é vista cada vez mais como a melhor abordagem para compreender o desempenho do pavimento.

Um sistema pavimento é modelado, normalmente, como uma estrutura de múltiplas camadas com diferentes propriedades materiais. Os elementos da última fileira de uma camada podem transferir esforços de cisalhamento para os elementos da primeira fileira da camada inferior, simulando o cisalhamento entre as camadas, e as condições de contorno bem controladas são fundamentais para analisar o comportamento do sistema pavimento como um todo (Blab *et al.*, 2000 *apud* Wang, 2001).

Apesar do método de modelagem 3D poder gerar resultados mais realistas do que os resultados da modelagem 2D, geralmente requer rotinas mais intensas de pré-processamento. Além disso, se o número de elementos aumentar, o tempo computacional total e a quantidade de memória aumentarão demasiadamente.

Felizmente, com o desenvolvimento da indústria computacional e com o surgimento de algoritmos de alto desempenho, mais elementos podem ser usados, tornando os resultados cada vez mais precisos.

Concomitantemente a isso, um grupo de pesquisadores tem mostrado que se a pressão de contato entre o pneu e o pavimento variar espacialmente, a resposta pode ser afetada significativamente (Tielking *et al.*, 1995 *apud* Wang, 2001 e Weissman, 1997 *apud* Wang, 2001), e nos modelos 2D não se obtém este efeito.

Cho *et al.* (1996) *apud* Wang (2001) compararam as soluções dos métodos de elementos finitos (modelagem axissimétrica bidimensional e modelagem tridimensional) com a teoria elástica e concluíram que uma boa aproximação foi obtida no modelo 3D quando o tamanho

do modelo e as condições de contorno foram bem controlados e, também, que os elementos quadráticos da modelagem tridimensional resultaram em tensões compatíveis com a teoria da elasticidade.

O capítulo seguinte utilizará a modelagem axissimétrica do método dos elementos finitos de forma a modelar o pavimento, pois, embora este modelo apresente algumas simplificações, ele é de fácil implementação e seus resultados são satisfatórios.

3 MODELAGEM NUMÉRICA DO PAVIMENTO

Este capítulo tem como objetivo apresentar as definições, a classificação e as principais características de um pavimento, bem como realizar a sua modelagem em elementos axissimétricos pelo método dos elementos finitos.

3.1 O PAVIMENTO

Pinto e Preussler (2002) definem o pavimento rodoviário sob dois pontos de vista: o do usuário e o do engenheiro. Para o usuário, o pavimento é uma superfície capaz de suportar o tráfego em condições de conforto e segurança. Para o engenheiro, o pavimento é uma estrutura constituída por uma ou mais camadas, com características que permitem ao pavimento receber as cargas aplicadas na superfície e distribuí-las, de modo que as tensões resultantes fiquem abaixo das tensões admissíveis dos materiais que constituem a estrutura.

O pavimento é a estrutura que se executa após a terraplenagem e que deve possuir as seguintes características:

- ter resistência para suportar os esforços verticais oriundos do tráfego para os quais foi projetado e distribuí-los ao terreno sobre o qual assenta;
- resistir, sem desgaste excessivo, aos esforços horizontais produzidos pelo tráfego;
- melhorar as condições de rolamento, permitindo uma circulação fácil, cômoda e segura;
- permitir que se realizem operações de reforço ou recapeamento compatíveis com o crescimento do volume de tráfego;
- conservar suas qualidades sob a ação dos agentes intempéricos.

O terreno sobre o qual assenta o pavimento que lhe serve de fundação é denominado de subleito e é considerado como sendo todo o maciço terroso limitado superiormente pelo leito da estrada. Geralmente, o pavimento é composto pelas seguintes camadas: reforço do subleito, sub-base, base e revestimento.

De acordo com a NBR 7207/82 – Terminologia e Classificação de Pavimentação – da Associação Brasileira de Normas Técnicas (ABNT):

- o subleito é o terreno de fundação do pavimento ou do revestimento;
- o reforço do subleito é uma camada situada entre o subleito e a sub-base que tem como objetivo melhorar o solo de fundação do pavimento. Esta camada também é conhecida como a camada final de terraplenagem;
- a sub-base é a camada corretiva do subleito, ou complementar à base, quando por qualquer circunstância não seja aconselhável construir o pavimento sobre o leito obtido pela terraplenagem;
- a base é uma camada destinada a resistir os esforços verticais oriundos dos veículos sobre a qual se constrói um revestimento;
- o revestimento é a camada, tanto quanto possível impermeável, que recebe diretamente a ação do rolamento dos veículos, que se destina, econômica e simultaneamente:
 - a) a melhorar as condições do rolamento quanto à comodidade e segurança;
 - b) a resistir aos esforços horizontais que nele atuam, tornando mais durável a superfície de rolamento.

Segundo Pinto e Preussler (2002):

- o reforço do subleito geralmente é constituído de um solo argiloso selecionado, de boas a excelentes características físicas e elevada resistência. Essas qualidades devem dar ao reforço melhores condições de suporte do que as do subleito, e resistência que permita a absorção e distribuição das cargas que se transmitem através das camadas superiores do pavimento;
- a sub-base pode consistir de uma ou mais camadas de material apropriadamente compactadas e deve ter estabilidade e capacidade de suporte, ótima capacidade de drenar água acumulada e reduzida suscetibilidade às variações volumétricas;
- a base deve reduzir as tensões de compressão no subleito e na sub-base a níveis aceitáveis, ou seja, deve distribuir as cargas aplicadas na superfície do pavimento de modo a minimizar ou eliminar as deformações de adensamento e cisalhamento no subleito e/ou sub-base;
- o revestimento é a camada do pavimento que deve resistir às forças abrasivas do tráfego, reduzir a penetração de água superficial no pavimento, proporcionar uma

superfície resistente ao deslizamento dos veículos e proporcionar um rolamento suave e uniforme ao tráfego.

3.1.1 CLASSIFICAÇÃO DOS PAVIMENTOS

Os pavimentos são classificados em rígido, flexível e semi-rígido:

- o pavimento rígido é aquele em que o revestimento tem uma elevada rigidez em relação às camadas inferiores e, portanto, o carregamento aplicado é distribuído para estas camadas em grandes áreas, proporcionando pequenas tensões na fundação do pavimento. Um exemplo deste tipo de pavimento é o constituído por lajes de concreto de cimento Portland;
- o pavimento flexível é aquele em que todas as camadas sofrem uma deformação elástica significativa sob o carregamento aplicado e, portanto, a carga se distribui em parcelas aproximadamente equivalentes entre as camadas. Um exemplo deste tipo de pavimento é o constituído por uma base de brita (brita graduada, macadame) ou por uma base de solo pedregulhoso, revestida por uma camada asfáltica;
- o pavimento semi-rígido se caracteriza por uma base cimentada quimicamente, como por exemplo, por uma camada de solo-cimento revestida por uma camada asfáltica.

3.1.2 A RESILIÊNCIA EM PAVIMENTOS

A evolução mais recente no que tange ao projeto de pavimentos e avaliação estrutural está calcada nos conceitos da Mecânica dos Pavimentos. Ela baseia-se nas teorias da Mecânica do Contínuo, da Mecânica dos Solos e da Mecânica da Fratura para interpretar o comportamento de sistemas estratificados e, assim, estabelecendo bases mais racionais para o projeto de pavimentos novos e suas restaurações.

Portanto, o objetivo da Mecânica dos Pavimentos é de projetar um pavimento considerando o estado de tensões e de deformações atuantes, compatibilizando-os com as admissíveis ou resistentes, para um período de projeto e condição de serventia.

Segundo Pinto e Preussler (2002), com a adoção dos métodos elásticos através de programas computacionais, convencionou-se chamar em Mecânica dos Pavimentos, de deformação resiliente, a deformação elástica ou recuperável de solos e de estruturas de pavimentos sob a ação de cargas transientes.

De acordo com Miceli Jr. (2006), o primeiro estudo sistemático da deformabilidade dos pavimentos deve-se ao engenheiro norte-americano Francis Hveem, em 1955. O órgão rodoviário da Califórnia (California Division of Highways, atualmente California Department of Transportation - Caltrans) havia começado, em 1938, medições de deflexões de pavimentos sujeitos ao tráfego.

Conforme Medina e Motta (2005), Hveem entendia que o trincamento progressivo dos revestimentos asfálticos se devia à deformação resiliente das camadas subjacentes, em especial o subleito. Ele preferiu usar este termo em vez de deformação elástica sob o argumento de que as deformações nos pavimentos são muito maiores do que nos sólidos elásticos com que lida o engenheiro – concreto, aço, entre outros.

Ferreira (2004) (dicionário Aurélio) define resiliência como a propriedade pela qual a energia armazenada em um corpo deformado é devolvida quando cessa a tensão causadora duma deformação elástica. Portanto, o termo deformação resiliente passou a significar a deformação recuperável dos pavimentos quando submetidos a carregamentos repetidos, isto, como forma de distingui-la daquelas que ocorrem em outras estruturas onde as cargas não são repetidas tão aleatoriamente quanto à frequência, duração e intensidade como a do tráfego de veículos (Pinto e Preussler, 2002).

A grandeza física que quantifica a propriedade resiliente de uma determinada camada do pavimento é o módulo de resiliência. O módulo de resiliência para as camadas formadas por solos é determinado através do ensaio triaxial de cargas repetidas, descrito na norma DNER-ME 131/94. Para as camadas formadas por misturas betuminosas, o módulo de resiliência pode ser obtido através do ensaio de compressão diametral de cargas repetidas, descrito na norma DNER-ME 133/94.

No ensaio triaxial dinâmico, o corpo de prova cilíndrico é submetido a uma pressão uniforme constante em toda sua superfície, chamada de tensão confinante (σ_3), e a uma tensão vertical variável, chamada de tensão-desvio (σ_d). A tensão-desvio é aplicada através de pulsos de carga com duração de 0,1 s e frequência de 1 Hz. A figura 9 mostra a representação das tensões atuantes no corpo de prova durante o ensaio e a figura 10 ilustra esquematicamente o equipamento deste ensaio.

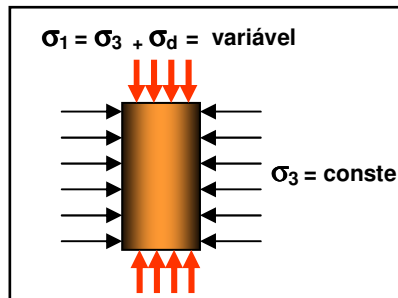


FIG. 9 – Tensões atuantes no corpo de prova durante o ensaio triaxial dinâmico (Vieira, 2006)

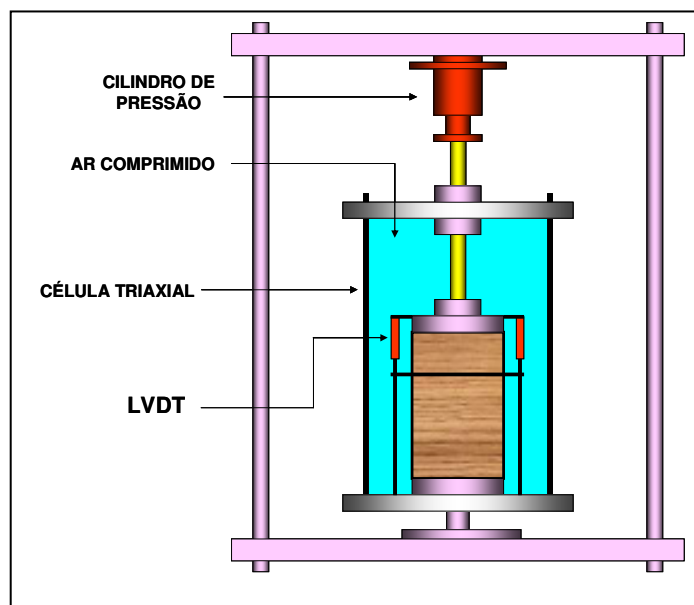


FIG. 10 – Equipamento de ensaios triaxiais de carga repetida (Vieira, 2006)

De acordo com a norma DNER-ME 131/94, o módulo de resiliência (M_R) de solos é a relação entre a tensão-desvio (σ_d), aplicada repetidamente em uma amostra de solo e a correspondente deformação específica vertical recuperável ou resiliente (ϵ_R), conforme a equação (3.1).

$$M_R = \frac{\sigma_d}{\epsilon_R} \quad (3.1)$$

No ensaio de compressão diametral de cargas repetidas, o corpo de provas cilíndrico, posicionado com sua geratriz em um plano horizontal, é submetido a uma força vertical de compressão diametral, sob a forma de pulsos de 0,1 segundo de duração e frequência de 60 ciclos por minuto. Esta força de compressão provoca uma tensão de tração (σ_T) horizontal no centro do corpo de prova e sua respectiva deformação de tração (ϵ_T). A figura 11 ilustra as forças aplicadas no corpo de prova durante o ensaio, onde d é o seu diâmetro e t é a sua altura, e a figura 12 mostra a representação esquemática dos equipamentos deste ensaio.

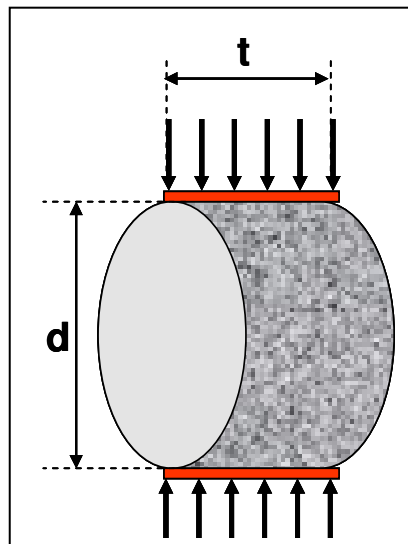


FIG. 11 – Forças aplicadas no corpo de prova durante o ensaio de compressão diametral de cargas repetidas (Vieira, 2006)

A norma DNER-ME 133/94 define o módulo de resiliência de misturas betuminosas como sendo a relação entre a tensão de tração (σ_T), aplicada repetidamente no plano diametral vertical de uma amostra cilíndrica de mistura betuminosa e a deformação específica recuperável (ϵ_T) correspondente à tensão aplicada, numa dada temperatura ($Temp$), como mostra a equação (3.2).

$$M_R = \left(\frac{\sigma_T}{\epsilon_T} \right)_{Temp} \quad (3.2)$$

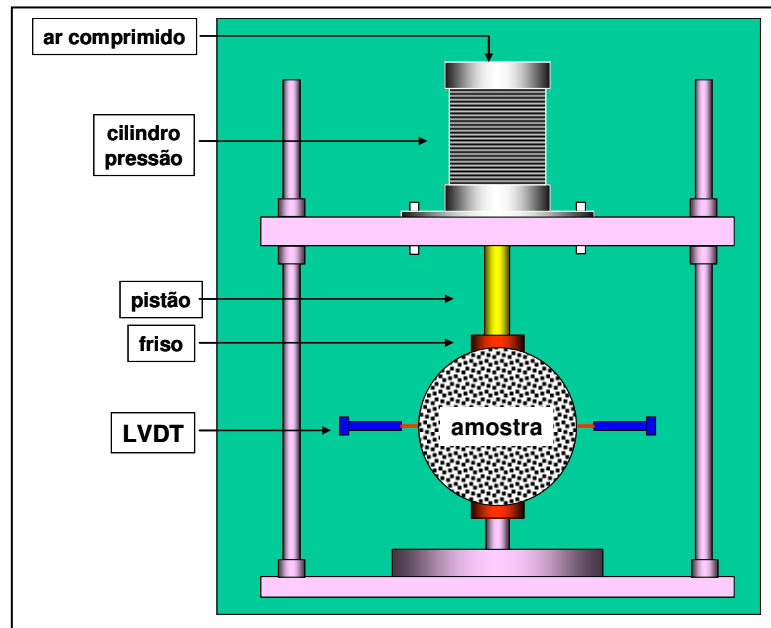


FIG. 12 – Equipamento de ensaios de compressão diametral de cargas repetidas (Vieira, 2006)

Com a utilização de corpos de prova de 10 cm de diâmetro e frisos de carga de 1,27 cm de largura, a equação (3.2) pode ser reescrita como:

$$M_R = \frac{F}{t \cdot \Delta} (0,9976\nu + 0,2692) \quad (3.3)$$

Onde:

F = força vertical aplicada através do friso à geratriz do cilindro;

t = altura do corpo de prova cilíndrico;

Δ = deslocamento total na horizontal;

ν = coeficiente de Poisson.

Entretanto, o módulo resiliente não é um parâmetro constante ou fixo para a maioria dos materiais de pavimentação (Franco, 2000). Ele depende do estado de tensões atuantes – as decorrentes do peso próprio mais as causadas pelas cargas dos veículos (Medina e Motta, 2005).

Segundo Medina e Motta (2005), o que se procura determinar nos ensaios triaxiais é uma relação experimental do módulo de resiliência como função da tensão confinante (σ_3) e da tensão-desvio (σ_d) para as condições de densidade, umidade e grau de saturação que o solo apresenta *in situ*.

Silva (2003) afirma que em geral, solos e britas não possuem um comportamento elástico linear sob carregamentos repetidos, tendo seus módulos resilientes dependentes dos estados de tensão atuantes. Desta forma, diversos modelos matemáticos foram criados na tentativa de equacionar esta relação entre o módulo resiliente e as tensões atuantes. A seguir são apresentados os modelos mais utilizados no Brasil.

De acordo com Silva (2003), no modelo granular ou arenoso proposto por Hicks em 1970 em sua tese de doutorado, o qual tem sido observado em solos com menos de 50 % passando na peneira 0,074 mm, o módulo de resiliência é expresso por:

$$M_R = K_1 \sigma_3^{K_2} \quad (3.4)$$

Onde:

K_1 e K_2 = constantes obtidas experimentalmente.

Aplica-se também aos solos granulares o modelo K - Θ , representado pela equação (3.5).

$$M_R = K_1 \Theta^{K_2} \quad (3.5)$$

Onde:

Θ = primeiro invariante de tensões ($\sigma_1 + \sigma_2 + \sigma_3$). No caso do ensaio de compressão triaxial $\Theta = \sigma_1 + 2\sigma_3 = \sigma_d + 3\sigma_3$.

Considera-se que o modelo K - Θ representa melhor o comportamento de materiais granulares, pois contempla a influência conjunta de σ_1 e σ_3 . Além disso, a utilização dos invariante de tensões contorna, de certa forma, a simplificação adotada no ensaio triaxial axissimétrico ($\sigma_2 = \sigma_3$) para simular a situação real que ocorre no campo (Macedo, 1996).

O modelo argiloso ou bi-linear ocorre nos solos finos ou com mais de 50 % passando na peneira nº 200. São, geralmente, solos lateríticos de subleitos ou camadas de reforço de subleito, comuns nas rodovias principais. A natureza da fração fina determina o comportamento à resiliência (Medina e Motta, 2005). As equações (3.6) e (3.7) representam este modelo.

$$M_R = K_2 + K_3(K_1 - \sigma_d) \text{ para } \sigma_d < K_1 \quad (3.6)$$

$$M_R = K_2 + K_4(\sigma_d - K_1) \text{ para } \sigma_d > K_1 \quad (3.7)$$

Onde:

K_1, K_2, K_3 e K_4 = constantes obtidas experimentalmente.

Ferreira (2002) afirma que tendo em vista as dificuldades experimentais para determinação das constantes do modelo bi-linear, Svenson (1980) propôs para cálculo do módulo de resiliência de solos coesivos o modelo representado pela equação (3.8), em escala bi-logarítmica, com K_2 negativo.

$$M_R = K_1 \sigma_d^{K_2} \quad (3.8)$$

Segundo Motta (1991), Aranovich (1985) ensaiando solos arenosos com bastante argila e solos argilosos lateríticos da região do Paraná propôs um modelo combinado, em que o módulo é função tanto de σ_3 quanto de σ_d , expresso por:

$$M_R = K_2 + K_3(K_1 - \sigma_d)\sigma_3^{K_5}, \text{ para } \sigma_d < K_1 \quad (3.9)$$

$$M_R = K_2 + K_4(\sigma_d - K_1)\sigma_3^{K_5}, \text{ para } \sigma_d > K_1 \quad (3.10)$$

Onde:

K_1, K_2, K_3, K_4 e K_5 = constantes obtidas experimentalmente.

Também é observada em materiais de pavimentação a situação onde o módulo de resiliência não varia de acordo com o estado de tensões, resultando num modelo elástico-linear. Este comportamento, conforme se encontra na equação (3.11), aparece em materiais betuminosos, siltes com baixo módulo e solos estabilizados com módulo elevado. (FERREIRA, 2002; MEDINA e MOTTA, 2005).

$$M_R = K = \text{constante} \quad (3.11)$$

De acordo com Ferreira (2002), as deficiências de cada um dos modelos aqui apresentados dizem respeito principalmente às seguintes limitações:

- dificuldades experimentais para definição das constantes, especialmente no modelo bi-linear e no modelo combinado ou misto;

- necessidade de definição prévia do comportamento resiliente de determinado material, se arenoso ou coesivo, como nos modelos de Hicks e de Svenson;
- simplificações quanto ao real estado de tensões atuante, considerando-se na maioria dos casos apenas influência de σ_3 ou de σ_d , isoladamente.

Macedo (1996) propôs um modelo aparentemente capaz de eliminar as dificuldades quanto à necessidade de definição prévia do comportamento resiliente dos solos. Este modelo, chamado de modelo composto, está representado na equação (3.12), e leva em consideração a influência conjunta das tensões confinante e desvio para qualquer material.

$$M_R = K_1 \sigma_3^{K_2} \sigma_d^{K_3} \quad (3.12)$$

Segundo Ferreira (2002), para este modelo, os coeficientes de determinação R^2 obtidos por Macedo (1996) através do método dos mínimos quadrados atingem valores normalmente acima de 0,90, bastante superiores àqueles auferidos para os demais modelos considerados. Este fato pode indicar que a consideração conjunta dos reflexos das tensões confinante (σ_3) e desvio (σ_d) é de fundamental importância na determinação dos valores de módulo de resiliência no ensaio triaxial dinâmico, não sendo aconselhável a determinação deste módulo apenas em função de uma das tensões impostas ao corpo de prova.

O coeficiente de determinação R^2 trata-se de um indicador de qualidade do ajustamento da função aos pontos conhecidos. Ele expressa a proporção da variação total que é explicada pela função de regressão e é calculado através da equação (3.13).

$$R^2 = \frac{\sum_{i=1}^n (y_i - \bar{y})^2 - \sum_{i=1}^n (y_i - \hat{y}_i)^2}{\sum_{i=1}^n (y_i - \bar{y})^2} \quad (3.13)$$

Onde:

y_i = valor conhecido, referente ao ponto i ;

\bar{y} = média aritmética dos pontos conhecidos;

\hat{y}_i = valor da função de regressão no ponto i .

A tabela 1 mostra os principais modelos para o módulo de resiliência das camadas do pavimento supracitados.

TAB. 1 – Modelos matemáticos de módulo de resiliência de solos e materiais de pavimentação em função do estado de tensões.

Modelo	Equação	Material
Granular ou arenoso	$M_R = K_1 \sigma_3^{K_2}$	Granular ou arenoso
K - Θ	$M_R = K_1 \Theta^{K_2}$	Granular (dependente da soma das tensões principais)
Bi-linear	$M_R = K_2 + K_3(K_1 - \sigma_d)$, para $\sigma_d < K_1$ $M_R = K_2 + K_4(\sigma_d - K_1)$, para $\sigma_d > K_1$	Solos finos (mais de 50 % passando na peneira nº 200)
Argiloso	$M_R = K_1 \sigma_d^{K_2}$	Solos coesivos
Combinado	$M_R = K_2 + K_3(K_1 - \sigma_d)\sigma_3^{K_5}$, para $\sigma_d < K_1$ $M_R = K_2 + K_4(\sigma_d - K_1)\sigma_3^{K_5}$, para $\sigma_d > K_1$	Solos arenosos com bastante argila ou solos argilosos lateríticos
Elástico-linear	MR = K = constante	Misturas asfálticas, siltes com baixo módulo ou solos estabilizados com módulo elevado
Composto	$M_R = K_1 \sigma_3^{K_2} \sigma_d^{K_3}$	Todos os solos e britas em geral

3.1.3 O COEFICIENTE DE POISSON NAS CAMADAS DO PAVIMENTO

Outra grandeza importante dos materiais constituintes do pavimento, usada na análise elástica de pavimentos, é o coeficiente de Poisson. A Resistência dos Materiais define o coeficiente de Poisson como sendo a razão da deformação transversal ou radial pela deformação longitudinal ou axial recuperável e o considera constante. A equação (3.14) representa este relacionamento e a figura 13 mostra esquematicamente os deslocamentos de um corpo de prova submetido a um carregamento vertical.

$$\nu = -\frac{\varepsilon_D}{\varepsilon_L} \quad (3.14)$$

Onde:

ν = coeficiente de Poisson;

$\varepsilon_D = \frac{\Delta D}{D}$ = deformação transversal ou radial;

$\varepsilon_L = \frac{\Delta L}{L}$ = deformação longitudinal ou axial.

O coeficiente de Poisson pode variar inicialmente de 0 (corpo rígido) a aproximadamente 0,5 (corpo incompressível). Coeficientes de Poisson maiores que 0,5 revelam que o material já foi submetido à ruptura ou ocorreu um erro experimental.

Segundo Medina e Motta (2005), o coeficiente de Poisson em pavimentos foi estudado por Trichês (1985) através de ensaios triaxiais com medição das deformações transversal e axial e concluiu-se que este coeficiente é influenciado pelo grau de saturação, energia de compactação e tipo de material e pode ser calculado através da equação (3.15).

$$\nu = b_1 + b_2 \left(\frac{\sigma_1}{\sigma_3} \right) + b_3 \left(\frac{\sigma_1}{\sigma_3} \right)^2 + b_4 \left(\frac{\sigma_1}{\sigma_3} \right)^3 \quad (3.15)$$

Onde:

b_1, b_2, b_3 e b_4 = constantes experimentais.

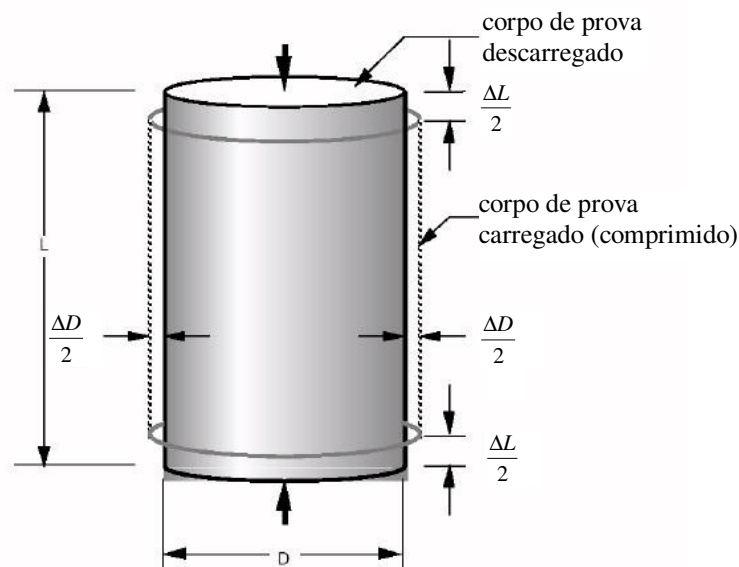


FIG. 13 – Deslocamentos de um corpo de prova sujeito a carregamento vertical (University of Washington, 2007)

Medina e Motta (2005) afirmam que, no entanto, o mais usual é se utilizar valor constante para o coeficiente de Poisson para cada material e sugerem o uso dos seguintes valores:

- $\nu = 0,15$, para concreto de cimento portland;
- $\nu = 0,25$, para misturas asfálticas a 25 °C;
- $\nu = 0,35$, para materiais granulares;
- $\nu = 0,45$, para solos argilosos.

Barata (1984) sugere o uso da tabela 2, para determinação do coeficiente de Poisson nos solos.

TAB. 2 – Valores de coeficiente de Poisson nos solos (Barata, 1984).

SOLO	COEFICIENTE DE POISSON
Solos Arenosos	0,15 – 0,25
Argila (com pouca areia e silte)	0,30 – 0,35
Argila	0,35 – 0,40

Pinto e Preussler (2002) sugerem os valores do coeficiente de Poisson para vários tipos de materiais mostrados na tabela 3.

TAB. 3 – Valores do coeficiente de Poisson (Pinto e Preussler, 2002).

Material	Coefficiente de Poisson
Concreto	0,15 – 0,20
Concreto Asfáltico	0,25 – 0,30
Base Granular	0,30 – 0,40
Areia Densa	0,30 – 0,35
Argila	0,40 – 0,45

3.2 A MODELAGEM AXISSIMÉTRICA DO PAVIMENTO

O pavimento, como visto anteriormente, é um conjunto de camadas de diferentes materiais sobrepostas umas às outras imediatamente acima do terreno natural, chamado de subleito. Cada camada possui uma relação tensão-deformação não-linear, dependente de seu estado de tensões e expressa pelo módulo de resiliência, e um coeficiente de Poisson considerado constante, cujos valores são diferentes das demais camadas.

Como o módulo de resiliência de cada material é dependente do seu estado de tensões atuantes, o cálculo das tensões e deformações do pavimento deve ser feito de maneira iterativa. Para cada nova iteração, deve-se recalcular o módulo de resiliência utilizando os valores das tensões calculadas na iteração anterior e, em seguida, recalcular as tensões e deformações que ocorrem no pavimento. Este processo iterativo é realizado enquanto a diferença da iteração anterior com a atual, para cada um dos módulos de resiliência do pavimento, for maior do que uma tolerância específica.

Na modelagem axissimétrica, considera-se que a roda de um veículo provoca um carregamento circular uniformemente distribuído e que o pavimento é analisado apenas para uma carga de roda. Devido à sua simetria axial, este modelo utiliza o sistema de coordenadas cilíndricas para representar o pavimento e analisa, apenas, o semi-plano definido pela parte positiva dos eixos vertical e radial deste sistema, pois considera-se que o comportamento obtido neste semi-plano seja o mesmo para outros com coordenadas angulares diferentes

Este semi-plano, embora bem definido, ainda é infinito e, em uma análise pelo método dos elementos finitos, necessita-se delimitações verticais e horizontais. Medina e Motta (2005) recomendam a adoção de fronteira lateral de pelo menos 20 vezes o raio da área carregada, e a fronteira do fundo no mínimo 50 vezes este raio. Silva (1995) considerou o subleito com espessura de 40 vezes este raio, independentemente das espessuras das camadas superiores.

Com a definição do domínio, deve-se discretizá-lo em uma malha de elementos finitos de tal forma que o final de uma camada coincida com o final de um elemento e que nos limites da área carregada existam pontos nodais. Nesta malha, não há necessidade de que o

espaçamento de suas linhas de pontos e o de as suas colunas sejam constantes e, também, iguais entre eles.

Geralmente, a utilização de uma quantidade maior de elementos na modelagem de problema propicia a obtenção de resultados mais precisos, mas necessita de maiores recursos computacionais para sua análise. Deve-se, portanto, otimizar esta quantidade ao se discretizar o domínio.

Para o pavimento, Cunagin *et al.* (1991), *apud* Wang (2001), ilustram que as tensões e deflexões críticas (valores de pico) ocorrem sob a carga de roda e diminuem à medida que se afasta desta carga. Então, a malha deste sistema deve ser mais refinada (as colunas de pontos nodais devem ser mais próximas umas das outras) na região do entorno da carga circular e quanto mais longe do centro do carregamento, maior poderá ser o tamanho do elemento.

Segundo Duncan *et al.* (1968), *apud* Silva (1995), para se obter uma maior precisão no cálculo das tensões do pavimento, a razão comprimento/largura de elementos quadriláteros não deve exceder 5/1. A figura 14 ilustra a discretização axissimétrica de um pavimento.

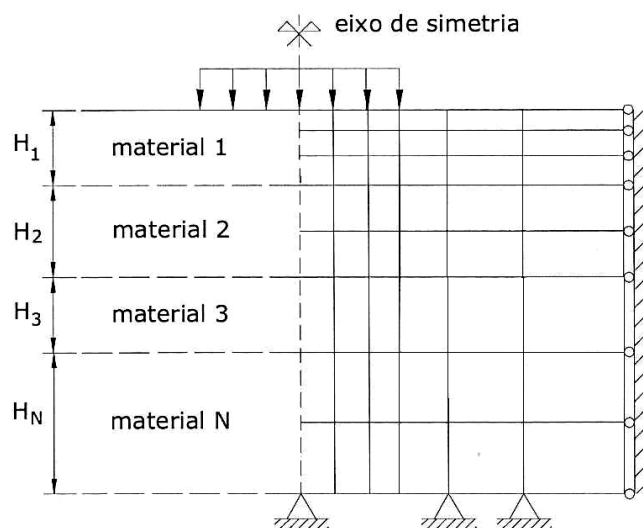


FIG. 14 – Exemplo de discretização axissimétrica do pavimento (Medina e Motta, 2005)

Com a criação da malha, as coordenadas dos nós do domínio são determinadas, bem como o número de elementos do sistema. Após esta criação, devem ser definidas as condições de contorno da estrutura: as forças externas e as restrições de deslocamento aplicadas em seus

nós; as propriedades de cada elemento: seu módulo de resiliência e coeficiente de Poisson; e as funções de interpolação de seus elementos.

As funções de interpolação têm a finalidade de garantir a continuidade de todo meio discretizado. Elas dependem do tipo de elemento (linear, bilinear, quadrático, cúbico, entre outros) e, cada elemento possui, para cada um de seus graus de liberdade, uma destas funções. Segundo Soriano (2003), cada função de interpolação tem a propriedade de ter valor unitário em um ponto nodal e ter valores nulos nos demais.

Para cada elemento, a partir de suas propriedades e suas funções de interpolação, determina-se sua matriz de rigidez. Acumulando sua contribuição (“somatório” de todas as matrizes de rigidez elementar do sistema), calcula-se a matriz de rigidez global da estrutura, que relaciona as forças externas com os deslocamentos dos nós.

Através de um sistema de equações constituído por essa matriz pode-se determinar, de maneira aproximada, todos os deslocamentos nodais da estrutura e, posteriormente, as tensões e deformações de cada elemento. Vieira (2004) afirma que a qualidade desta solução aproximada depende do grau de refinamento das funções de interpolação e da malha de elementos finitos.

Segundo Soriano (2003), no modelo matemático axissimétrico, os deslocamentos são independentes do ângulo θ do sistema de coordenadas cilíndricas, sendo relevantes apenas o deslocamento u na direção radial r e o deslocamento w na direção axial z . Então, as deformações são apenas em plano axial e esse modelo pode ser discretizado em elementos com os mesmos deslocamentos nodais e funções de interpolação dos elementos de estado plano de deformações.

Os deslocamentos deste modelo são calculados através da equação (2.6), onde o vetor de cargas nodais é a representação matricial das forças externas que atuam na estrutura. Essas forças, geralmente, não são aplicadas diretamente nos nós do sistema, pois elas podem ser forças concentradas, de superfície e/ou de volume. Então, para a construção desse vetor, deve-se transformá-las, primeiramente, em forças nodais que provoquem efeitos equivalentes.

Essas forças são conhecidas como cargas nodais equivalentes. Em uma estrutura simples (composta por elementos unidimensionais), elas podem ser determinadas, para cada elemento, invertendo-se o sentido das reações de fixação. Essas reações são as reações de apoio que se obteriam ao engastar todos os nós do elemento.

Para o modelo axissimétrico do pavimento, desprezando o peso próprio dos materiais (forças de volume), as forças nodais equivalentes de cada elemento podem ser calculadas através da equação (3.16). Ao se acumular essas forças, de forma matricial, monta-se o vetor de cargas nodais do sistema.

$$\{f_i\}_q^e = \int_{S_e} N_i [I] \{q\} dS_e \quad (3.16)$$

Onde:

$\{f_i\}_q^e$ = vetor de forças nodais equivalentes às forças de superfície;

N_i = função de interpolação;

$[I]$ = matriz identidade;

$\{q\}$ = vetor de forças de superfície de componentes q_x , q_y e q_z (para o caso do pavimento, apenas o esforço vertical é considerado, desprezando-se os esforços horizontais, como frenagem, ou seja, $q_x = q_y = 0$).

A matriz de rigidez global do sistema é calculada através da contribuição acumulada da matriz de rigidez de cada elemento. Essa matriz para um elemento axissimétrico é dada pela equação (3.17).

$$K_{ij}^e = 2\pi \int_{A_e} B_i^T D B_j r dr dz \quad (3.17)$$

Onde:

B = matriz gradiente de deformação, expressa pela equação (3.18);

D = matriz constitutiva, que para coeficientes de Poisson (ν) menores que 0,5, pode ser calculada pela equação (3.19).

$$B_i = \begin{bmatrix} \frac{\partial N_i}{\partial r} & 0 \\ 0 & \frac{\partial N_i}{\partial z} \\ \frac{N_i}{r} & 0 \\ \frac{\partial N_i}{\partial z} & \frac{\partial N_i}{\partial r} \end{bmatrix} \quad (3.18)$$

Onde:

N_i = função de interpolação.

$$[D] = \frac{M_R}{(1+\nu)(1-2\nu)} \begin{bmatrix} 1-\nu & \nu & \nu & 0 \\ \nu & 1-\nu & \nu & 0 \\ \nu & \nu & 1-\nu & 0 \\ 0 & 0 & 0 & \frac{1-2\nu}{2} \end{bmatrix} \quad (3.19)$$

Onde:

M_R = módulo de resiliência.

Segundo Zienkiewicz e Taylor (2000), como a matriz B é dependente das coordenadas do elemento, a resolução da equação (3.17) deve ser feita através da integração numérica ou da multiplicação matricial seguida da integração termo a termo.

As deformações do modelo axissimétrico são dependentes dos deslocamentos nodais de cada elemento e seus respectivos gradientes variacionais. Estas deformações são representadas pela equação (3.20).

$$\{\varepsilon\} = \begin{Bmatrix} \varepsilon_r \\ \varepsilon_z \\ \varepsilon_\theta \\ \gamma_{rz} \end{Bmatrix} = \begin{Bmatrix} \frac{\partial u}{\partial r} \\ \frac{\partial w}{\partial z} \\ \frac{u}{r} \\ \frac{\partial u}{\partial z} + \frac{\partial w}{\partial r} \end{Bmatrix} = [L] \begin{Bmatrix} u \\ w \end{Bmatrix} \quad (3.20)$$

Onde:

u = deslocamento radial de um elemento;

w = deslocamento axial de um elemento;

$[L]$ = matriz de operadores diferenciais, expressa pela equação (3.21).

$$[L] = \begin{bmatrix} \frac{\partial}{\partial r} & 0 \\ 0 & \frac{\partial}{\partial z} \\ \frac{1}{r} & 0 \\ \frac{r}{\partial} & \frac{\partial}{\partial} \\ \frac{\partial}{\partial z} & \frac{\partial}{\partial r} \end{bmatrix} \quad (3.21)$$

As tensões para cada elemento deste modelo são calculadas através da equação (3.22), sendo a mesma uma simplificação da lei de Hooke generalizada.

$$\{\sigma\} = \begin{Bmatrix} \sigma_r \\ \sigma_z \\ \sigma_\theta \\ \tau_{rz} \end{Bmatrix} = [D]\{\epsilon\} \quad (3.22)$$

A figura 15 ilustra as tensões atuantes em um ponto do sólido obtido pela revolução do trapézio abcd em torno do eixo z, onde r , θ e z são coordenadas cilíndricas.

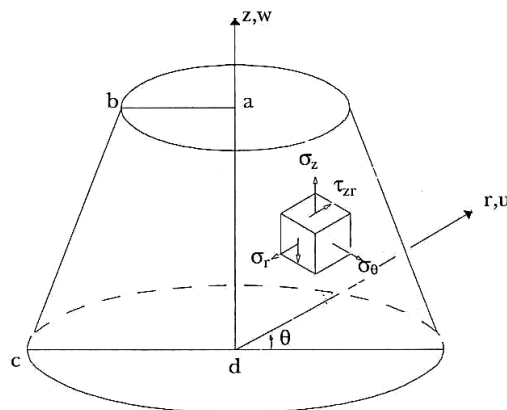


FIG. 15 – Tensões atuantes em um ponto de um domínio axissimétrico (Soriano, 2003)

Como o módulo de resiliência é dependente das tensões principais ou suas correlações (σ_1 , σ_3 , $\sigma_d = \sigma_1 - \sigma_3$, $\theta = \sigma_1 + \sigma_2 + \sigma_3$), é necessário o cálculo das tensões principais que ocorrem nos elementos axissimétricos do sistema. Para efetuar este cálculo deve-se, primeiramente, transformar em coordenadas cartesianas as tensões obtidas nas coordenadas cilíndricas do modelo. Para essa transformação, utilizam-se as equações (3.23).

$$\sigma_x = \sigma_r \cos^2 \alpha + \sigma_\theta \sin^2 \alpha \quad (3.23a)$$

$$\sigma_y = \sigma_r \sin^2 \alpha + \sigma_\theta \cos^2 \alpha \quad (3.23b)$$

$$\tau_{xy} = (\sigma_r - \sigma_\theta) \sin \alpha \cos \alpha \quad (3.23c)$$

$$\tau_{yz} = \tau_{rz} \sin \alpha \quad (3.23d)$$

$$\tau_{xz} = \tau_{rz} \cos \alpha \quad (3.23e)$$

Onde:

α = ângulo que o eixo r das coordenadas cilíndricas faz com o eixo x das coordenadas cartesianas;

σ_x = tensão normal na direção do eixo x das coordenadas cartesianas;

σ_y = tensão normal na direção do eixo y das coordenadas cartesianas;

τ_{xy} = Tensão de cisalhamento no plano x na direção y ;

τ_{yz} = Tensão de cisalhamento no plano y na direção z ;

τ_{xz} = Tensão de cisalhamento no plano x na direção z ;

Após a determinação dessas tensões, as tensões principais (σ_1, σ_2 e σ_3) são determinadas através da solução da equação cúbica (3.24).

$$\begin{aligned} \sigma^3 - (\sigma_x + \sigma_y + \sigma_z) \sigma^2 + (\sigma_x \sigma_y + \sigma_y \sigma_z + \sigma_x \sigma_z - \tau_{yz}^2 - \tau_{xz}^2 - \tau_{xy}^2) \sigma - \\ - (\sigma_x \sigma_y \sigma_z + 2 \tau_{yz} \tau_{xz} \tau_{xy} - \sigma_x \tau_{yz}^2 - \sigma_y \tau_{xz}^2 - \sigma_z \tau_{xy}^2) = 0 \end{aligned} \quad (3.24)$$

Substituindo as equações (3.23) na equação (3.24) e considerando o ângulo α igual a 0, obtém-se a equação (3.25).

$$\begin{aligned} \sigma^3 - (\sigma_r + \sigma_\theta + \sigma_z) \sigma^2 + (\sigma_r \sigma_\theta + \sigma_\theta \sigma_z + \sigma_r \sigma_z - \tau_{rz}^2) \sigma - \\ - (\sigma_r \sigma_\theta \sigma_z - \sigma_\theta \tau_{rz}^2) = 0 \end{aligned} \quad (3.25)$$

Com a determinação das tensões principais, deve-se recalculer os módulos de resiliência do sistema e compará-los com os utilizados para esta determinação. Caso a diferença entre os módulos de resiliência para cada elemento for maior que uma tolerância preestabelecida, deve-se proceder novamente os cálculos de deslocamentos, deformações e tensões. A figura 16 mostra esse processo iterativo.

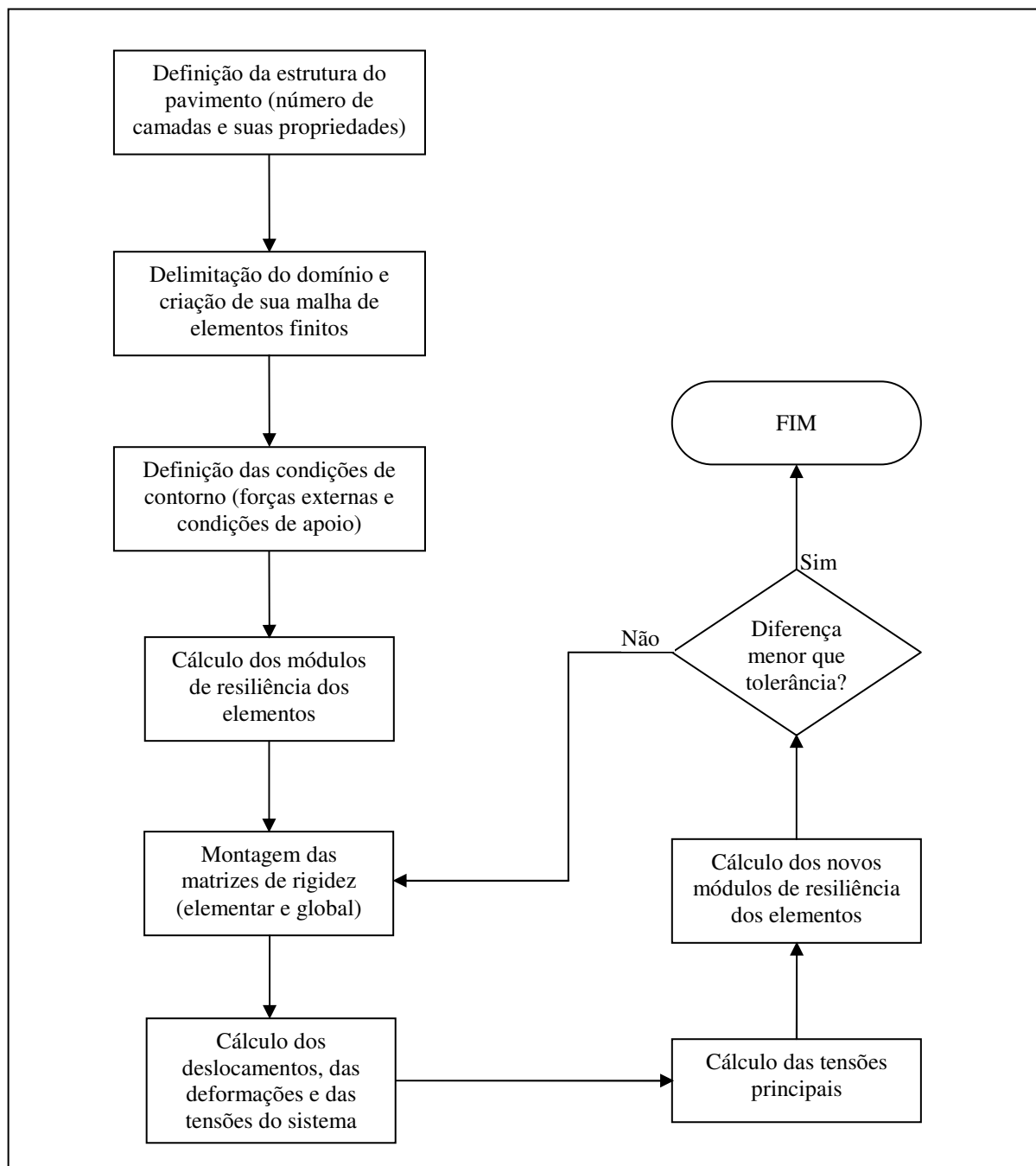


FIG. 16 – Fluxograma da modelagem do pavimento

3.2.1 MÉTODO INCREMENTAL

O método incremental determina os deslocamentos, as tensões e as deformações do sistema através de incrementos no carregamento. Neste método, deve-se definir, inicialmente, o número de incrementos que será utilizado para a análise do pavimento. Segundo Pinto e

Preussler (2002), rotineiramente são considerados quatro incrementos iguais do carregamento neste processo iterativo.

Em seguida, a estrutura do pavimento é definida através do número de camadas e suas propriedades (o módulo de resiliência e o coeficiente de Poisson). Posteriormente, delimita-se o domínio e faz-se a sua discretização em uma malha de elementos finitos, conforme descrito anteriormente (item 3.2).

Calcula-se, a seguir, o módulo de resiliência inicial de cada elemento, sendo este determinado através da utilização de valores de tensões, consideradas mínimas, nas equações contidas na tabela 1. Após isso, determina-se as condições de apoio e as forças externas totais (forças atuantes na estrutura).

Posteriormente, é calculado o vetor de cargas nodais da iteração, multiplicando-se o vetor de forças externas totais pela razão entre o número da iteração atual e o número de incrementos. Em seguida, monta-se as matrizes de rigidez do sistema (elementar e global), conforme descrito na seção 3.2.

Após a definição do vetor de cargas nodais da iteração e da matriz de rigidez global, calcula-se os deslocamentos, as deformações e as tensões, utilizando as equações (2.6), (3.20) e (3.22), respectivamente. Posteriormente, são calculadas as tensões principais de cada elemento, através da solução da equação cúbica (3.25).

A partir das tensões principais, calcula-se o novo módulo de resiliência de cada elemento, de acordo com a tabela 1. Em seguida, compara-se o número da iteração com o número de incrementos. Enquanto esses valores não forem iguais, realiza-se este processo iterativo a partir do cálculo do vetor de cargas nodais da iteração.

A figura 17 mostra este processo iterativo e, juntamente com a figura 16, possibilitará a implementação do modelo no capítulo seguinte.

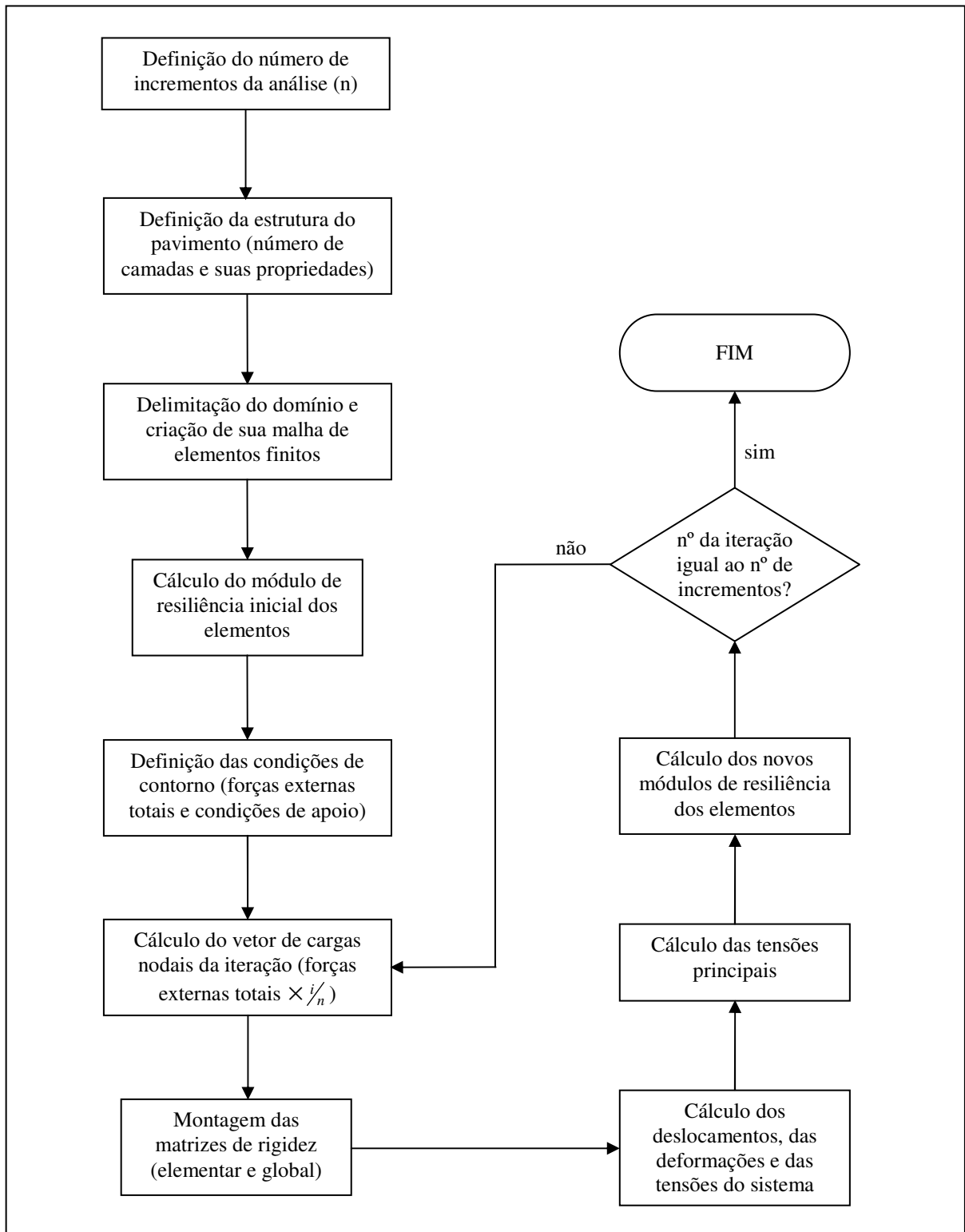


FIG. 17 – Fluxograma do método incremental

4 IMPLEMENTAÇÃO COMPUTACIONAL

Este capítulo tem como objetivo apresentar as principais características da linguagem de programação C++, bem como realizar a implementação do programa de análise de pavimentos em elementos finitos.

4.1 A LINGUAGEM DE PROGRAMAÇÃO C++ E A ORIENTAÇÃO A OBJETOS

Segundo Johann (2004), a linguagem C++ foi desenvolvida inicialmente por Bjarne Stroustrup na AT&T, de 1979 a 1983, a partir da linguagem C, tendo como idéia principal a de agregar o conceito de classes e de orientação a objetos, àquela linguagem. De acordo com Barreto (2005), o nome "C++" advém do fato de que esta linguagem é tida como uma extensão da linguagem C, onde o primeiro "+" representa as melhorias realizadas na linguagem C, e o segundo "+" representa o acréscimo de funcionalidades para suportar o paradigma de orientação a objetos.

Entretanto, Johann (2004) afirma que C++ não é um superconjunto de C e deve ser visto como uma outra linguagem de programação, pois a sintaxe e a semântica de algumas construções diferem entre as duas linguagens e o C++ visa à programação orientada a objetos. Segundo Couto Jr. *et al.* (2005), essa linguagem foi padronizada em 1998 pelo American National Standards Institute (ANSI) e pela International Standards Organization (ISO).

De acordo com Arnaut (2007), a Programação Orientada a Objetos (POO) é uma reação a problemas que foram percebidos pela primeira vez em programas muito grandes desenvolvidos na década de 70. Segundo Fonseca (2007), estes programas eram tão complexos que a maioria dos desenvolvedores de softwares não conseguia compreendê-los integralmente.

Mesquita (2007) afirma que, na POO, descreve-se o problema em termos do mesmo e não em termos da solução. De acordo com Fonseca (2007), por meio da orientação a objetos, divide-se o problema em partes, e cada parte contém um objeto estanco que carrega

informações e instruções relacionadas. Essa decomposição propicia o gerenciamento da complexidade de grandes projetos.

Segundo Barreto (2005), este tipo de programação é um modelo de desenvolvimento de software que enfoca os objetos presentes dentro de um sistema. Ferreira (2004) (dicionário Aurélio) define programação orientada a objetos como o paradigma de programação no qual conjuntos de dados e rotinas são tratados como unidades (ditas objetos) relativamente autônomas, que interagem por meio de mensagens.

A idéia deste paradigma, de acordo com Barreto (2005), é modelar um objeto de software de forma mais semelhante aos objetos da vida real. Para isso, define-se um objeto em termos de suas características (atributos) e das ações (métodos) que o mesmo é capaz de executar.

Segundo Mesquita (2007), cada objeto tem a sua própria região de memória, que, também, pode ser composta por outros objetos. Para Barreto (2005), o objeto é uma entidade ativa dentro do programa e representa um conjunto de atributos (informações) que possuem métodos (ações) associados a eles.

Ferreira (2004) (dicionário Aurélio) define objeto, em POO, como qualquer módulo que contém rotinas e estruturas de dados, e é capaz de interagir com outros módulos similares, trocando mensagens. Essas mensagens, segundo Mesquita (2007), são chamadas feitas a funções pertencentes a um determinado objeto.

Segundo Deitel & Deitel (2001), os objetos são componentes de software essencialmente reutilizáveis que modelam coisas no mundo real e são criados a partir de “modelos” chamados de classes.

Barreto (2005) afirma que uma classe representa um modelo para a criação de objetos, agrupando informações sobre um mesmo objeto e definindo as ações válidas sobre estas informações. Para Mesquita (2007), uma classe descreve um conjunto de objetos semelhantes e defini-la significa formalizar um tipo de dados e todas as operações associadas a este tipo.

Ferreira (2004) (dicionário Aurélio) define classe, em POO, como a categoria descritiva geral, que abrange o conjunto de objetos que compartilham uma ou mais características em comum.

Segundo Fonseca (2007), os três recursos mais importantes em POO são o encapsulamento, a herança e o polimorfismo.

Barreto (2005) define encapsulamento, como a propriedade da POO que determina a visibilidade de atributos e métodos dentro de um objeto. Esta propriedade se refere ao fato de um objeto conhecer somente a interface de outro objeto, e não a forma como os seus métodos são implementados e/ou seus atributos são manipulados.

Ainda, de acordo com Barreto (2005), esta propriedade é essencial para o desenvolvimento de sistemas seguros e robustos. E é determinado em função de três tipos:

- público – permite que qualquer objeto dentro do sistema acesse atributos e métodos de um objeto, de forma direta (através do operador “.”);
- privado – determina atributos e métodos que somente a classe pode acessar diretamente. Os outros objetos acessam estes atributos através de funções-membro;
- protegido – determina atributos e métodos que só podem ser acessados pela classe-base e suas classes derivadas.

Segundo Deitel & Deitel (2001), a herança é uma forma de reutilização de software na qual novas classes são criadas a partir de classes existentes, pela absorção de seus atributos e comportamentos e redefinindo-os com recursos que as novas classes requerem.

Para Fonseca (2007), herança é o processo pelo qual um objeto adquire as propriedades de outro objeto. De acordo com Mesquita (2007), a herança permite modelar uma hierarquia entre classes: classes mais especializadas (classes-filhas ou classes derivadas) herdam propriedades da classe mais geral (classe-pai ou classe-base).

Barreto (2005) afirma que através da herança, uma classe (derivada) pode herdar atributos e métodos de outra classe (base), usar estes atributos e métodos da forma como estão na classe-base ou redefini-los.

Ferreira (2004) (dicionário Aurélio) define herança, em POO, como a transmissão automática de determinadas propriedades de uma classe a outra classe dela derivada. Segundo Barreto (2005), existem dois tipos de herança: a herança simples – uma classe é derivada de uma classe-base; e a herança múltipla – uma classe derivada herda de várias classes base (possivelmente não relacionadas).

Deitel & Deitel (2001) afirmam que ao derivar uma classe-base, ela pode ser herdada como pública, protegida ou privada. Ao se derivar uma classe a partir de uma classe-base:

- pública, os membros públicos da classe-base tornam-se membros públicos da classe derivada e os membros protegidos da classe-base se tornam membros protegidos da classe derivada. Os membros privados de uma classe-base nunca são acessáveis diretamente a partir de uma classe derivada;
- protegida, os membros públicos e protegidos da classe-base se tornam membros protegidos da classe derivada;
- privada, os membros públicos e protegidos da classe-base se tornam membros privados da classe derivada.

De acordo com Mesquita (2007), polimorfismo é a propriedade de se usar o mesmo nome para métodos diferentes, implementados em diferentes níveis de hierarquia de classes. Para cada classe, tem-se um comportamento específico para o método.

Deitel & Deitel (2001) definem polimorfismo como a habilidade de objetos de classes diferentes relacionadas por herança responderem diferentemente à mesma mensagem (isto é, uma chamada a uma função membro). A mesma mensagem enviada para muitos tipos diferentes de objetos assume “muitas formas”.

Estes conceitos são concatenados por Johann (2004), afirmando que um programa orientado a objetos é formado por um conjunto de classes que são modelos para criação de objetos. As classes e objetos possuem atributos, que são os dados, e métodos, que são as funções que operam sobre estes dados.

Ainda, segundo o mesmo autor, quando um trecho de código quer fazer uma operação sobre um objeto ele emite uma mensagem para esse objeto requisitando a operação. Em C++, isso pode ser implementado como uma simples chamada de função (chamada de método). A principal característica da orientação a objetos é o encapsulamento, que é o fato de alguns dados não estarem acessíveis diretamente, mas apenas através de métodos públicos.

Johann (2004) complementa o conceito de orientação a objetos afirmando que o mecanismo de herança faz a definição de classes mais especializadas a partir de classes básicas. Nesse caso, pode-se reimplementar métodos de uma classe básica na classe mais especializada e, posteriormente, tratar diversos objetos diferentemente especializados através de métodos com o mesmo nome, o que se chama de polimorfismo.

De acordo com Hubbard (2003), as classes individuais em uma hierarquia de classe são designadas como abstratas ou concretas. Para Johann (2004), as classes abstratas são classes cuja única função é definir conjuntos de métodos sem implementá-los e as classes concretas são aquelas que realmente implementam os métodos.

Deitel & Deitel (2001) afirmam que como as classes abstratas são usadas como classes base em situações de herança, elas podem ser chamadas de classes básicas abstratas e nenhum objeto pode ser criado a partir delas. O único objetivo de uma classe abstrata é fornecer uma classe básica apropriada da qual outras classes podem herdar a interface e/ou a implementação.

4.2 PROGRAMA DE ANÁLISE DE PAVIMENTOS – CARACTERÍSTICAS DA IMPLEMENTAÇÃO

O programa de análise de pavimentos desenvolvido nesta dissertação, intitulado de *AXIPAV*, foi implementado na linguagem de programação C++. Este programa calcula as tensões e deformações que ocorrem no pavimento através do método dos elementos finitos e obedece ao paradigma de orientação a objetos, descrito anteriormente, sendo composto por oito classes básicas:

- matriz – classe responsável pelas operações com matrizes;

- Malha – classe responsável pela criação da malha de elementos finitos;
- Material – classe responsável pelas propriedades físicas de um elemento;
- Pavimento – classe responsável pelas propriedades físicas de todo o pavimento;
- No – classe responsável pelas coordenadas, condições de contorno (restrições e cargas nodais) e deslocamentos de cada nó;
- PontoDeGauss – classe responsável pelas integrações numéricas realizadas pelo programa e pelo cálculo das tensões no ponto de Gauss correspondente;
- Elemento – classe responsável pelo cálculo da matriz de rigidez do elemento;
- Estrutura – classe responsável pela montagem e resolução do sistema de equações.

Estas classes serão detalhadas a seguir.

4.2.1 CLASSE MATRIZ

Esta classe é responsável pelas operações com matrizes, possuindo três atributos e dezesseis métodos, sendo todos do tipo público. Seus atributos são: *num_linhas* e *num_colunas* – ambos inteiros e responsáveis pelas dimensões da matriz; e *elementos* – vetor dinâmico (alocação de memória durante a execução) do tipo “double”, responsável pelo armazenamento dos valores da matriz.

Nesta classe, os operadores de adição, subtração, multiplicação, exponenciação, entre outros foram sobrecarregados, ou seja, mudados de maneira que se realizem as respectivas operações com matrizes. Além disso, seus métodos são:

- *elemento(i, j)* – retorna o valor do elemento da linha *i* e coluna *j* da matriz;
- *Atribui(i, j, valor)* – atribui *valor* ao elemento da linha *i* e coluna *j* da matriz;
- *Atribui(i, valor)* – atribui *valor* ao elemento da linha *i* da matriz coluna;
- *Adiciona(i, j, valor)* – adiciona *valor* ao elemento da linha *i* e coluna *j* da matriz;
- *Adiciona(i, valor)* – adiciona *valor* ao elemento da linha *i* da matriz coluna;
- *redim(m, n)* – redimensiona a matriz para *m* linhas e *n* colunas. Os valores armazenados anteriormente são perdidos;
- *Determinante()* – calcula e retorna o determinante da matriz;
- *Singular()* – verifica se a matriz é singular e retorna verdadeiro ou falso;

- `Simetrica()` – verifica se a matriz é simétrica e retorna verdadeiro ou falso;
- `AntiSimetrica()` – verifica se a matriz é anti-simétrica e retorna verdadeiro ou falso;
- `Quadrada()` – verifica se a matriz é quadrada e retorna verdadeiro ou falso;
- `Inversa()` – calcula e retorna a sua matriz inversa;
- `Transposta()` – retorna a sua matriz transposta;
- `ZeraMatriz()` – zera os valores de todos os elementos da matriz;
- `TransFormaEmIdentidade()` – transforma a matriz em uma matriz identidade;
- `Limpa()` – apaga todos os elementos da matriz.

4.2.2 CLASSE MALHA

Esta classe é responsável pela criação da malha de elementos finitos, possuindo nove atributos, sendo três públicos e seis protegidos, e dez métodos públicos. Seus atributos públicos são: nx e ny – ambos inteiros e responsáveis pelo controle do tamanho dos vetores de coordenadas X e Y, respectivamente; $coord$ – atributo do tipo matriz, responsável pelo vetor de coordenadas X ou Y, conforme o caso.

Nesta classe, os atributos protegidos são: x e y – ambos do tipo matriz, responsáveis pela construção do vetor de coordenadas X e Y, respectivamente; $xinf$, $xsup$, $yinf$ e $ysup$ – todos do tipo “double”, responsáveis pelos limites inferior e superior dos atributos x e y , respectivamente. Além disso, seus métodos são:

- `pg(xy, a0, q, n, no)` – divide o eixo X ou Y, de acordo com o valor de xy (1 ou 2, respectivamente), através de uma progressão geométrica com termo inicial $a0$, razão q e n termos, a partir do nó no . Se n for igual a zero, este método dividirá o eixo correspondente através desta progressão até atingir o valor limite para este eixo;
- `pa(xy, a0, r, n, no)` – divide o eixo X ou Y, de acordo com o valor de xy (1 ou 2, respectivamente), através de uma progressão aritmética com termo inicial $a0$, razão r e n termos, a partir do nó no . Se n for igual a zero, este método dividirá o eixo correspondente através desta progressão até atingir o valor limite para este eixo;
- `div(xy, comp, n, no)` – divide o segmento de comprimento $comp$ pertencente ao eixo X ou Y, de acordo com o valor de xy (1 ou 2, respectivamente), em n partes iguais, a partir do nó no . Se $comp$ for igual a zero, este método divide em n partes iguais o

segmento iniciado pela coordenada do nó *no* e terminado no limite superior do eixo em questão;

- *acumula(xy, n, no, aux)* – acrescenta na matriz *x* ou *y*, de acordo com o valor de *xy* (1 ou 2, respectivamente), a partir do nó *no*, os *n* primeiros termos da matriz *aux*, matriz auxiliar que possui os valores calculados das coordenadas X ou Y, conforme o caso, para *n* pontos a partir do nó *no*, desde que estes valores não sejam maiores do que o limite superior do eixo correspondente;
- *razao_max(xy, rm)* – modifica os valores da matriz *x* ou *y*, de acordo com o valor de *xy* (1 ou 2, respectivamente), para os elementos nos quais a razão comprimento/largura é maior do que o valor *rm*. Se *rm* for igual a zero, nenhum valor da matriz correspondente é modificado;
- *nodes()* – mostra na tela de execução as coordenadas de todos os nós da malha;
- *Nos(xy, A)* – acrescenta na matriz *A*, matriz das coordenadas X ou Y, conforme o caso, relativas aos pontos já calculados da malha, os valores da matriz *x* ou *y*, de acordo com o valor de *xy* (1 ou 2, respectivamente);
- *Nos(xy)* – iguala a matriz *coord* com a matriz *x* ou *y*, de acordo com o valor de *xy* (1 ou 2, respectivamente);
- *redim(xy, inf, sup)* – redimensiona a matriz *x* ou *y*, de acordo com o valor de *xy* (1 ou 2, respectivamente), para os novos limites inferior *inf* e superior *sup*;
- *redim(xy, sup)* – redimensiona a matriz *x* ou *y*, de acordo com o valor de *xy* (1 ou 2, respectivamente), considerando como novo limite inferior o limite superior anterior e como superior o valor de *sup*.

4.2.3 CLASSE MATERIAL

Esta classe é responsável pelas propriedades físicas de um elemento, possuindo cinco atributos e quatro métodos, sendo todos do tipo público. Seus atributos são: *POISS*, *MR* e *MRant* – todos do tipo “double”, responsáveis pelo coeficiente de Poisson, módulo de resiliência atual e módulo de resiliência anterior, respectivamente; *ki* – atributo do tipo matriz responsável pelos valores das constantes experimentais de determinação do módulo de resiliência; e *tipo* – atributo do tipo inteiro responsável pelo tipo de material (tipo da equação de determinação do módulo de resiliência).

Nesta classe, o operador de atribuição “=” foi sobrecarregado e seus métodos são:

- calcula_MR() – calcula o módulo de resiliência de maneira independente das tensões atuantes no elemento, atribuindo o valor calculado a *MR*;
- calcula_MR(*tensao*) – calcula o módulo de resiliência de materiais que possuem dependência de apenas um tipo de tensão, adotando para esta tensão o valor de *tensao* e atribuindo o valor calculado a *MR*;
- calcula_MR(*Sd*, *S3*) – calcula o módulo de resiliência de materiais que possuem dependência da tensão-desvio (*Sd*) e da tensão confinante (*S3*), atribuindo o valor calculado a *MR*;
- calcula_MR(*S1*, *S2*, *S3*) – calcula o módulo de resiliência de todos os tipos de materiais, a partir dos valores das tensões principais (*S1*, *S2*, *S3*), atribuindo o valor calculado a *MR*.

A tabela 4 mostra os valores que *tipo* pode assumir de acordo com os modelos de determinação de seu módulo de resiliência de material considerados nesta classe e os principais materiais que se aplicam a cada modelo.

TAB. 4 – Tipos da classe Material

Tipo	Modelo	Equação	Material
1	Granular ou arenoso	$M_R = K_1 \sigma_3^{K_2}$	Granular ou arenoso
2	K - Θ	$M_R = K_1 \Theta^{K_2}$	Granular (dependente da soma das tensões principais)
3	Bi-linear	$M_R = K_2 + K_3(K_1 - \sigma_d)$, para $\sigma_d < K_1$ $M_R = K_2 + K_4(\sigma_d - K_1)$, para $\sigma_d > K_1$	Solos finos (mais de 50 % passando na peneira n° 200)
4	Argiloso	$M_R = K_1 \sigma_d^{K_2}$	Solos coesivos
5	Combinado	$M_R = K_2 + K_3(K_1 - \sigma_d)\sigma_3^{K_5}$, para $\sigma_d < K_1$ $M_R = K_2 + K_4(\sigma_d - K_1)\sigma_3^{K_5}$, para $\sigma_d > K_1$	Solos arenosos com bastante argila ou solos argilosos lateríticos
6	Elástico-linear	$MR = K = \text{constante}$	Misturas asfálticas, siltes com baixo módulo ou solos estabilizados com módulo elevado
7	Composto	$M_R = K_1 \sigma_3^{K_2} \sigma_d^{K_3}$	Todos os solos e britas em geral

4.2.4 CLASSE PAVIMENTO

Esta classe é responsável pelas propriedades físicas de todo o pavimento, possuindo seis atributos e onze métodos, sendo todos do tipo público. Seus atributos são: *num_camadas*, *fileiras_tot* e *num_colunas* – todos inteiros, responsáveis pelo número de camadas do pavimento, número total de fileiras de elementos do pavimento e número de colunas total de elementos do pavimento, respectivamente; *fileiras* e *fileiras_at* – ambos vetores dinâmicos do tipo inteiro, responsáveis pelo armazenamento do número de fileiras desejadas e atual de cada camada, respectivamente; e *elementos* – vetor dinâmico do tipo Material, responsável pelo armazenamento dos elementos do pavimento.

Nesta classe, o operador de atribuição “=” e o operador de chamada de função “()” foram sobrecarregados. Além disso, seus métodos são:

- *elemento(f)* – retorna o material da fileira *f* de elementos do pavimento;
- *elemento(c, f)* – retorna o material da fileira *f* de elementos referentes à camada *c* do pavimento;
- *elemento(c, f, col)* – retorna o material da fileira *f* de elementos referentes à camada *c* do pavimento situado na coluna *col*;
- *redim(n)* – redimensiona o vetor de materiais para *n* camadas com apenas uma fileira cada. Os valores armazenados anteriormente no vetor *elementos* são perdidos;
- *redim(n, A)* – redimensiona o vetor de materiais para *n* camadas e cada uma delas com número de fileiras de acordo com a matriz *A*, matriz que contém o número de fileiras de cada camada. Os valores armazenados anteriormente no vetor *elementos* são perdidos;
- *redim()* – redimensiona o vetor de materiais para *num_camadas* camadas com número de fileiras de acordo com a matriz *fileiras*, valores estes, previamente armazenados. Neste método, os valores armazenados anteriormente no vetor *elementos* não são perdidos;
- *redim_col(col)* – redimensiona o vetor de materiais, o transformando em uma matriz com *col* colunas. Neste método, os valores armazenados na primeira coluna são repetidos para as demais;
- *acrescenta(l, c, f)* – acrescenta *l* linhas na camada *c* logo após a fileira *f* da referida camada;

- *inicializa()* – calcula o módulo de resiliência inicial, desconsiderando a influência do estado de tensões, da primeira fileira de cada camada, repetindo, em seguida, o material da primeira fileira de cada camada para as demais fileiras da respectiva camada;
- *inicializa(Sd, S3)* – calcula o módulo de resiliência inicial, a partir dos valores da tensão-desvio *Sd* e da tensão confinante *S3*, da primeira fileira de cada camada, repetindo, em seguida, o material da primeira fileira de cada camada para as demais fileiras da respectiva camada;
- *tolerancia(tol)* – verifica se o valor absoluto da diferença percentual entre os módulos de resiliência *MR* e *MRant* $\left(\frac{MR - MRant}{MRant} \right)$ de cada elemento é menor do que o valor de *tol*, retornando verdadeiro ou falso, conforme o caso.

4.2.5 CLASSE NO

Esta classe é responsável pelas coordenadas, pelas condições de contorno (restrições e cargas nodais) e pelos deslocamentos de um nó, possuindo quinze atributos e um método, sendo todos do tipo público. Seus atributos são: *x*, *y* e *z* – todos do tipo “double”, responsáveis pelas coordenadas do nó; *Fx*, *Fy* e *Fz* – todos do tipo “double”, responsáveis pelos valores das cargas nodais; *restx*, *resty* e *restz* – todos inteiros, responsáveis pelas restrições, ou condições de apoio; *prescx*, *prescy* e *prescz* – todos do tipo “double”, responsáveis pelos deslocamentos prescritos (deslocamentos iniciais ou recalque inicial); e *u*, *v* e *w* – todos do tipo “double”, responsáveis pelo valor dos deslocamentos do nó nas direções X, Y e Z, respectivamente.

Seu método é *GeraResultados(arquivo_saida, arquivo_view, NSD)* – escreve no arquivo *arquivo_saida* os resultados dos deslocamentos *u*, *v* e, se *NSD* (número de dimensões espaciais do problema) for igual a três, *w*.

4.2.6 CLASSE PONTODEGAUSS

Esta classe é uma classe abstrata, responsável pelas integrações numéricas realizadas pelo programa e pelo cálculo das tensões no ponto de Gauss correspondente, possuindo duas

subclasses do tipo público. Suas subclasses são: PontoDeGauss_EPT – responsável pelos cálculos de problemas bidimensionais de Estado Plano de Tensões e, através de sua classe derivada pública, PontoDeGauss_AXISSIMETRICO, para problemas axissimétricos (objetivo desta dissertação); e PontoDeGauss_3D – responsável pelos cálculos de problemas tridimensionais e que será motivo de futuras atualizações no programa.

A classe PontoDeGauss_AXISSIMETRICO possui dezenove atributos públicos, sendo dezesseis herdados indiretamente da classe PontoDeGauss (ou seja, atributos herdados da classe PontoDeGauss_EPT que foram herdados de sua classe básica) e três correspondentes a sua própria classe. Ela possui, também, quinze métodos públicos, sendo treze herdados de sua classe-base (PontoDeGauss_EPT), dos quais sete foram reimplementados através do polimorfismo, e dois relativos à própria classe.

Os atributos herdados desta classe são: *coordenada* e *pesos* – ambos do tipo matriz, responsáveis pelos valores das coordenadas e dos pesos de qualquer ponto de Gauss do sistema (elemento); *XI*, *ETA*, *FI*, *WXI*, *WETA* e *WFI* – todos do tipo “double”, responsáveis pelas coordenadas locais do ponto de Gauss correspondente e seus respectivos pesos, respectivamente; *D*, *K* e *Fint* – todos do tipo matriz, responsáveis pela matriz constitutiva, pela matriz de rigidez e pelo vetor de forças internas, respectivamente; *LN* – atributo do tipo matriz, responsável pela matriz resultante da multiplicação da matriz de operadores diferenciais (*[L]*) pela matriz de funções de interpolação (*[N]*); e *B*, *J*, *u* e *sigma* – todos do tipo matriz, responsáveis pela matriz gradiente de deformação, pela matriz jacobiana (as características desta matriz serão explicadas no item 4.3), pelo vetor de deslocamentos dos pontos nodais e pelo vetor de tensões no ponto de Gauss, respectivamente.

Adicionalmente a estes atributos, os seus próprios são: *S1*, *S2* e *S3* – todos do tipo “double”, responsáveis pelos valores das tensões principais no ponto de Gauss. Além disso, seus métodos são:

- *CoordenadasDoPontoDegauss(IGAUSS)* – método herdado de sua classe-base (PontoDeGauss_EPT), responsável por atribuir os valores das coordenadas *XI* e *ETA* e respectivos pesos *WXI* e *WETA* conforme o número do ponto de Gauss correspondente (*IGAUSS*);

- COORDENADAXI(i) – método herdado de sua classe-base (PontoDeGauss_EPT), responsável por retornar o valor da coordenada local XI correspondente ao nó local i do elemento;
- COORDENADAETA(i) – método herdado de sua classe-base (PontoDeGauss_EPT), responsável por retornar o valor da coordenada local ETA correspondente ao nó local i do elemento;
- Calcula_D(Mat) – método herdado de sua classe-base (PontoDeGauss_EPT) e reimplementado nesta classe, responsável por calcular a matriz constitutiva do material Mat , armazenando na matriz D ;
- Calcula_K($IGAUSS$, $NumeroDePontosNodais$, Nos , Mat) – método herdado de sua classe-base (PontoDeGauss_EPT) e reimplementado nesta classe, responsável por calcular a parcela correspondente ao ponto de Gauss $IGAUSS$ da matriz de rigidez elementar, armazenando na matriz K ;
- Inicializa($NumeroDePontosNodais$) – método herdado de sua classe-base (PontoDeGauss_EPT) e reimplementado nesta classe, responsável por redimensionar as matrizes $Fint$, K , LN e B em função de $NumeroDePontosNodais$;
- Calcula_Tensoes($IGAUSS$, $NumeroDePontosNodais$, Nos , Mat) – método herdado de sua classe-base (PontoDeGauss_EPT) e reimplementado nesta classe, responsável pelo cálculo das tensões do ponto de Gauss $IGAUSS$, armazenando os valores na matriz $sigma$, onde $sigma = [\sigma_r \quad \sigma_z \quad \tau_{rz} \quad \sigma_\theta]$;
- Calcula_J($IGAUSS$, $NumeroDePontosNodais$, Nos) – método herdado de sua classe-base (PontoDeGauss_EPT) e reimplementado nesta classe, responsável pelo cálculo da matriz jacobiana correspondente ao ponto de Gauss $IGAUSS$, armazenando na matriz J ;
- Calcula_B($IGAUSS$, $NumeroDePontosNodais$, Nos) – método herdado de sua classe-base (PontoDeGauss_EPT) e reimplementado nesta classe, responsável pelo cálculo da matriz gradiente de deformação correspondente ao ponto de Gauss $IGAUSS$, armazenando na matriz B ;
- CalculaForcasInternas($IGAUSS$, $NumeroDePontosNodais$, Nos , Mat) – método herdado de sua classe-base (PontoDeGauss_EPT) e reimplementado nesta classe, responsável pelo cálculo da parcela correspondente ao ponto de Gauss $IGAUSS$ do vetor de forças internas do elemento, armazenando na matriz $Fint$;

- $\text{SHAPE}(i, \text{NumeroDePontosNodais}, XI, ETA)$ – método herdado de sua classe-base (PontoDeGauss_EPT), responsável por calcular o valor da função de interpolação referente ao nó local i do elemento, no ponto de coordenadas locais XI e ETA , retornando o valor calculado;
- $\text{LSHAPEXI}(i, \text{NumeroDePontosNodais}, XI, ETA)$ – método herdado de sua classe-base (PontoDeGauss_EPT), responsável por calcular o valor da derivada parcial da função de interpolação referente ao nó local i do elemento em relação à coordenada local XI , no ponto de coordenadas locais XI e ETA , retornando o valor calculado;
- $\text{LSHAPEETA}(i, \text{NumeroDePontosNodais}, XI, ETA)$ – método herdado de sua classe-base (PontoDeGauss_EPT), responsável por calcular o valor da derivada parcial da função de interpolação referente ao nó local i do elemento em relação à coordenada local ETA , no ponto de coordenadas locais XI e ETA , retornando o valor calculado;
- $\text{tensoes_princ}()$ – método exclusivo desta classe, responsável pela transformação das tensões axissimétricas $(\sigma_r, \sigma_z, \tau_{rz}, \sigma_\theta)$, armazenadas na matriz sigma , em tensões principais $(\sigma_1, \sigma_2, \sigma_3)$, armazenando-os em $S1$, $S2$ e $S3$, respectivamente;
- $\text{tensões_princ}(Sr, Sz, Trz, Steta)$ – método exclusivo desta classe, responsável pela transformação das tensões axissimétricas Sr , Sz , Trz e $Steta$, referentes a $\sigma_r, \sigma_z, \tau_{rz}$ e σ_θ , respectivamente, em tensões principais $(\sigma_1, \sigma_2, \sigma_3)$, retornando estes valores em uma matriz coluna 3x1.

4.2.7 CLASSE ELEMENTO

Esta classe é uma classe abstrata, responsável pelo cálculo da matriz de rigidez, das tensões e forças internas do elemento, possuindo uma subclasse do tipo público – Elemento_EPT. Esta subclasse é responsável pelos cálculos de problemas bidimensionais de Estado Plano de Tensões e, através de suas classes derivadas públicas: Elemento_AXISSIMETRICO, para problemas axissimétricos (objetivo desta dissertação); e Elemento_3D, para problemas tridimensionais (sendo motivo de futuras atualizações no programa).

A classe *Elemento_AXISSIMETRICO* possui onze atributos, sendo dez herdados indiretamente da classe *Elemento* (ou seja, atributos herdados da classe *Elemento_EPT* que foram herdados de sua classe básica), com sete públicos e três protegidos, e um relativo à própria classe. Ela possui, também, dezessete métodos públicos, sendo cinco herdados de sua classe-base (*Elemento_EPT*), dos quais três foram reimplementados através do polimorfismo, e doze do tipo público relativo à própria classe.

Os atributos públicos herdados desta classe são: *NumeroDePontosNodais* e *NumeroDePontosDeGauss* – ambos inteiros, responsáveis pelo número de nós e de pontos de Gauss que o elemento possui, respectivamente; *camada*, *fileira* e *coluna* – todos inteiros, responsáveis pela localização do elemento na malha de elementos finitos, ou seja, a camada, a fileira dentro desta camada e a coluna a que o elemento pertence; *NP* – vetor dinâmico do tipo inteiro, responsável pelo vetor de incidência dos pontos nodais (associação entre os nós locais e globais); e *NoLocal* – vetor dinâmico do tipo *No*, responsável pelas informações globais dos nós do elemento.

Adicionalmente a estes atributos, esta classe possui, ainda, os herdados do tipo protegido, que são *IND*, *K* e *Fint* – todos do tipo matriz, responsáveis pelo vetor de incidência dos graus de liberdade do elemento (associação entre os graus de liberdade locais e globais), pela matriz de rigidez e pelo vetor de forças internas do elemento, e o seu próprio atributo, *PontosDeGauss* – vetor dinâmico do tipo *PontoDeGauss_AXISSIMETRICO*, responsável pelo armazenamento dos pontos de Gauss do elemento. Além disso, seus métodos são:

- *Incidencia(Nos)* – método herdado de sua classe-base (*Elemento_EPT*), responsável pela atribuição dos dados globais dos nós do elemento no vetor *NoLocal* e por atribuir os valores da matriz *IND*;
- *redim(n)* – método herdado de sua classe-base (*Elemento_EPT*), responsável pelo redimensionamento das matrizes protegidas *IND*, *K* e *Fint*, em função do número de nós *n*;
- *CalculaRigidez(KGLOBAL, Nos, Mat)* – método herdado de sua classe-base (*Elemento_EPT*) e reimplementado nesta classe, responsável pelo cálculo da matriz de rigidez do elemento e acumulação destes valores na matriz *KGLOBAL*;

- CalculaTensoes(*FintGLOBAL*, *Nos*, *Mat*) – método herdado de sua classe-base (Elemento_EPT) e reimplementado nesta classe, responsável pelo cálculo das tensões axissimétricas ($\sigma_r, \sigma_z, \tau_{rz}, \sigma_\theta$) em cada ponto de Gauss do elemento;
- CalculaForcasInternas(*FintGLOBAL*, *Nos*, *Mat*) – método herdado de sua classe-base (Elemento_EPT) e reimplementado nesta classe, responsável pelo cálculo do vetor de forças internas do elemento e acumulação destes valores na matriz *FintGlobal*;
- Calcula_MR(*Sdmin*, *S3min*, *Mat*) – método exclusivo desta classe, responsável pelo cálculo do módulo de resiliência do elemento, considerando os valores *Sdmin* e *S3min*, como valores mínimos de tensão-desvio e tensão confinante, respectivamente;
- Tensoes(*x*, *y*) – método exclusivo desta classe, responsável pelo cálculo das tensões axissimétricas no ponto de coordenadas globais *x* e *y*, através da interpolação/extrapolação das tensões obtidas nos pontos de Gauss do elemento, retornando estes valores na forma de uma matriz;
- TensoesCoordLocal(*XI*, *ETA*) – método exclusivo desta classe, responsável pelo cálculo das tensões axissimétricas no ponto de coordenadas locais *XI* e *ETA*, através da interpolação/extrapolação das tensões obtidas nos pontos de Gauss do elemento, retornando estes valores na forma de uma matriz;
- TensaoMax(*SupInf*, *sigma*) – método exclusivo desta classe, responsável pelo cálculo de um dos tipos de tensão máxima, conforme o valor de *sigma* (1 para σ_r , 2 para σ_z , 3 para τ_{rz} , 4 para σ_θ , 5 para σ_1 , 6 para σ_2 , 7 para σ_3 e 8 para $\Delta\sigma = \sigma_1 - \sigma_3$), no topo ou no fundo do elemento, de acordo com o valor de *SupInf* (1 ou 2, respectivamente);
- DeformacoesPrinc(*x*, *y*, *Mat*) – método exclusivo desta classe, responsável pelo cálculo das deformações principais ($\varepsilon_1, \varepsilon_2, \varepsilon_3$) no ponto de coordenadas globais *x* e *y*, através das propriedades do material *Mat* (coeficiente de Poisson e módulo de resiliência) e das tensões principais ($\sigma_1, \sigma_2, \sigma_3$) deste ponto, obtidas a partir da transformação de suas tensões axissimétricas ($\sigma_r, \sigma_z, \tau_{rz}, \sigma_\theta$), retornando os valores destas deformações na forma de uma matriz;

- DefPrincLocal(XI , ETA , Mat) – método exclusivo desta classe, responsável pelo cálculo das deformações principais ($\varepsilon_1, \varepsilon_2, \varepsilon_3$) no ponto de coordenadas locais XI e ETA , através das propriedades do material Mat (coeficiente de Poisson e módulo de resiliência) e das tensões principais ($\sigma_1, \sigma_2, \sigma_3$) deste ponto, obtidas a partir da transformação de suas tensões axissimétricas ($\sigma_r, \sigma_z, \tau_{rz}, \sigma_\theta$), retornando os valores destas deformações na forma de uma matriz;
- DeformacaoMax($SupInf$, def , Mat) – método exclusivo desta classe, responsável pelo cálculo de um dos tipos de deformação máxima principal, conforme o valor de def (1 para ε_1 , 2 para ε_2 e 3 para ε_3), no topo ou no fundo do elemento, de acordo com o valor de $SupInf$ (1 ou 2, respectivamente), a partir das propriedades do material Mat (coeficiente de Poisson e módulo de resiliência);
- Deflexao(x , y) – método exclusivo desta classe, responsável pelo cálculo da deflexão (deslocamento vertical) no ponto de coordenadas globais x e y , através da interpolação dos deslocamentos obtidos nos nós do elemento;
- DeflexaoCoordLocal(XI , ETA) – método exclusivo desta classe, responsável pelo cálculo da deflexão (deslocamento vertical) no ponto de coordenadas locais XI e ETA , através da interpolação dos deslocamentos obtidos nos nós do elemento;
- DeflaxaoMax($SupInf$) – método exclusivo desta classe, responsável pelo cálculo da deflexão (deslocamento vertical) máxima no topo ou no fundo do elemento, de acordo com o valor de $SupInf$ (1 ou 2, respectivamente). Se $SupInf$ não for informado, o método realiza o cálculo no topo do elemento;
- MaxPositivo($X1$, $X2$, $X3$) – método exclusivo desta classe, responsável pelo cálculo do maior valor positivo, em uma linha do elemento, da variável, cujos valores nos pontos notáveis (pontos que possuem coordenadas locais XI iguais a -1, 0 e 1, respectivamente) são $X1$, $X2$ e $X3$, respectivamente. Se o elemento possuir apenas quatro nós, o valor de $X3$ é considerado igual a zero e os valores de $X1$ e $X2$ correspondem aos pontos de coordenadas locais XI iguais a -1 e 1, respectivamente;
- MaxNegativo($X1$, $X2$, $X3$) – método exclusivo desta classe, responsável pelo cálculo do maior valor negativo (menor valor), em uma linha do elemento, da variável, cujos valores nos pontos notáveis (pontos que possuem coordenadas locais XI iguais a -1, 0 e 1, respectivamente) são $X1$, $X2$ e $X3$, respectivamente. Se o elemento possuir

apenas quatro nós, o valor de $X3$ é considerado igual a zero e os valores de $X1$ e $X2$ correspondem aos pontos de coordenadas locais XI iguais a -1 e 1, respectivamente;

4.2.8 CLASSE ESTRUTURA

Esta classe é responsável pela montagem e resolução do sistema de equações, possuindo vinte e sete atributos e vinte e nove métodos, sendo todos do tipo público. De seus vinte e nove atributos, oito são inteiros, sete são do tipo matriz, seis são do tipo “double”, dois são vetores de alocação estática do tipo caractere, dois são do tipo saída de dados, um é do tipo entrada de dados, um é vetor dinâmico do tipo No, um é vetor dinâmico do tipo Elemento_AXISSIMETRICO e um é do tipo Pavimento.

Os atributos inteiros desta classe são: *MSD* – responsável pelo número de dimensões espaciais do problema; *NumeroDeElementos* – responsável pelo número total de elementos do sistema; *NumeroDePontosNodais* – responsável pelo número total de pontos nodais do sistema; *NumeroDeNosComRestricao* – responsável pelo número total de nós com deslocamentos restritos; *NumeroDeRestricoes* – responsável pelo número total de deslocamentos restritos do sistema; *NumeroDeCamadas* – responsável pelo número total de camadas do pavimento; *Metodo* – responsável pelo tipo de resolução (1, para o método do fluxograma da figura 16 e 2, para o método incremental, descrito no item 3.2.1 e mostrado na figura 17); e *Iteracao* – responsável pelo número de iterações máximas, para o método 1, e pelo número total de iterações, para o método 2.

Seus atributos do tipo matriz são: *d* – responsável pelo vetor de deslocamentos nodais do sistema; *K* – responsável pela matriz de rigidez global do sistema; *F* – responsável pelo vetor de cargas nodais; *Fint* – responsável pelo vetor de forças internas do sistema; *Espessuras* – responsável pelo vetor das espessuras de cada uma das camadas do pavimento; *rest* – responsável pela parcela da matriz de rigidez referente aos deslocamentos restritos; e *FApoio* – responsável pelo vetor de reações de apoio.

Nesta classe, os atributos do tipo “double” são: *PENALIDADE* – responsável pelo coeficiente de penalidade, utilizado em um dos métodos de aplicação de restrições da

estrutura; *Sdmin* – responsável pelo valor mínimo da tensão-desvio (σ_d) utilizada para os cálculos de módulo de resiliência dos elementos do sistema; *S3min* – responsável pelo valor mínimo da tensão confinante (σ_3) utilizada para os cálculos de módulo de resiliência dos elementos do sistema; *a* – responsável pelo raio da área de contato da carga de roda do sistema; *q* – responsável pela pressão do pneu ou pressão de contato roda pavimento; e *tolerancia* – responsável pela definição da tolerância entre os módulos de resiliência de cada elemento.

Seus atributos do tipo vetor estático de caractere são: *Nome* – responsável pelo armazenamento do nome do problema; e *TipoDeProblema* – responsável pelo armazenamento do tipo de problema a ser resolvido (atualmente, apenas problemas axissimétricos). Nesta classe, o atributo do tipo entrada de dados é *arquivo_entrada* – responsável pela ligação do arquivo de entrada de dados com o programa, a fim de realizar a leitura dos dados da análise; e os atributos do tipo saída de dados são: *arquivo_saida* e *arquivo_view* – responsáveis pela ligação do programa com os arquivos de saída de dados, a fim de realizar a saída de dados do programa e gerar o arquivo de visualização, respectivamente.

Finalmente, os outros atributos que esta classe possui são: *Nos* – vetor dinâmico do tipo No, responsável pelas informações sobre os nós da estrutura; *Elementos* – vetor dinâmico do tipo Elemento_AXISSIMETRICO, responsável pela realização dos cálculos em todos os elementos do sistema; e *Pav* – atributo do tipo Pavimento, responsável pelas propriedades físicas de todos os elementos do pavimento. Além disso, seus métodos são:

- *LeDados(Dados, Resultados, Visualizacao)* – realiza a leitura do arquivo de entrada de dados, fazendo a correlação dos arquivos cujos nomes sejam iguais ao conteúdo de *Dados*, *Resultados* e *Visualização*, vetores dinâmicos de caracteres, com os atributos *arquivo_entrada*, *arquivo_saida* e *arquivo_view*, respectivamente;
- *LeDados(Dados, Resultados, Visualizacao)* – realiza a leitura do arquivo de entrada de dados, fazendo a correlação dos arquivos cujos nomes sejam iguais ao conteúdo de *Dados*, *Resultados* e *Visualização*, parâmetros do tipo “string”, com os atributos *arquivo_entrada*, *arquivo_saida* e *arquivo_view*, respectivamente;

- GeraMalha() – gera a malha de elementos retangulares de oito nós, armazenando as informações relativas aos nós no vetor *Nos* e as relativas aos elementos no vetor *Elementos*;
- EscreveNos() – escreve nos arquivos correspondentes aos atributos *arquivo_saida* e *arquivo_view*, as informações relativas aos nós da estrutura;
- LeMateriais() – realiza a leitura das informações relativas aos materiais do pavimento, no arquivo de entrada de dados correspondente ao atributo *arquivo_entrada*, armazenando estas informações no atributo *Pav*, e, em seguida, as escreve nos arquivos de saída de dados correspondentes aos atributos *arquivo_saida* e *arquivo_view*;
- EscreveElementos() – escreve nos arquivos correspondentes aos atributos *arquivo_saida* e *arquivo_view*, as informações relativas aos elementos da estrutura;
- CalculaRigidez() – calcula a matriz de rigidez global do sistema, calculando a contribuição de cada elemento e armazenando o resultado na matriz *K*;
- CalculaDeslocamentos() – calcula os deslocamentos nodais da estrutura através do método Gauss-Seidel, armazenando-os na matriz *d* e no vetor *Nos*;
- Calcula() – resolve a estrutura (calcula os deslocamentos), a partir da montagem da matriz de rigidez global e aplicação das condições de apoio, em seguida, calcula-se os deslocamentos e as tensões;
- CalculaTensoes() – calcula as tensões nos pontos de Gauss de todos os elementos da estrutura, através dos deslocamentos nodais;
- MontaVetorDeForçasInternas() – monta o vetor de forças internas da estrutura, calculando a contribuição de cada elemento e armazenando o resultado na matriz *Fint*;
- AplicaRestricoes() – aplica as condições de apoio na matriz de rigidez, adicionando o valor de *PENALIDADE* aos elementos da diagonal principal da matriz *K* referentes aos deslocamentos restritos;
- AplicaRestricoes2() – aplica as condições de apoio no sistema de equações, criando a matriz *rest*, a partir das linhas referentes aos deslocamentos restritos da matriz de rigidez global, excluindo as linhas e colunas referentes aos deslocamentos restritos na matriz *K* e redimensionando as matrizes *d* e *F*;

- *MontaVetorDeCargas(fator)* – monta o vetor de cargas externas, calculando-se, para cada grau de liberdade, a multiplicação do valor de *fator* com o resultado da adição entre a força nodal e o produto do deslocamento prescrito com o valor de *PENALIDADE*, referentes ao grau de liberdade em questão. Estes valores são armazenados na matriz *F*;
- *GeraResultados()* – escreve os deslocamentos de cada nó do sistema e as reações de apoio de cada nó restrito no arquivo correspondente a *arquivo_saida*, ambos calculados com a matriz *K* modificada em *AplicaRestricoes()*;
- *GeraResultados2()* – escreve os deslocamentos de cada nó do sistema e as reações de apoio de cada nó restrito no arquivo correspondente a *arquivo_saida*, ambos calculados com a matriz *K* modificada em *AplicaRestricoes2()*;
- *ProcuraElemento(x, y)* – retorna o número do primeiro elemento que possui o ponto de coordenadas globais *x* e *y*;
- *ProcuraElemento(x, c, SupInf)* – retorna o número do primeiro elemento do topo ou do fundo da camada *c*, de acordo com o valor de *SupInf* (1 ou 2, respectivamente), que possui a coordenada global *x*;
- *TensoesAxissimetricas(x, y)* – calcula as tensões axissimétricas ($\sigma_r, \sigma_z, \tau_{rz}, \sigma_\theta$) do ponto de coordenadas globais *x* e *y*, retornando estes valores na forma de uma matriz. Caso este ponto pertença a mais de um elemento (ou seja, situado em uma das arestas do elemento), as tensões calculadas se referem às do primeiro elemento encontrado que contenha este ponto;
- *TensoesPrincipais(x, y)* – calcula as tensões principais ($\sigma_1, \sigma_2, \sigma_3$) do ponto de coordenadas globais *x* e *y*, retornando estes valores na forma de uma matriz. Caso este ponto pertença a mais de um elemento (ou seja, situado em uma das arestas do elemento), as tensões calculadas se referem às do primeiro elemento encontrado que contenha este ponto;
- *TensoesMax(c, SupInf, sigma)* – calcula um dos tipos de tensão máxima, conforme o valor de *sigma* (1 para σ_r , 2 para σ_z , 3 para τ_{rz} , 4 para σ_θ , 5 para σ_1 , 6 para σ_2 , 7 para σ_3 e 8 para $\Delta\sigma = \sigma_1 - \sigma_3$), no topo ou no fundo da camada *c*, de acordo com o valor de *SupInf* (1 ou 2, respectivamente);
- *TensoesMax(c, f, SupInf, sigma)* – calcula um dos tipos de tensão máxima, conforme o valor de *sigma* (1 para σ_r , 2 para σ_z , 3 para τ_{rz} , 4 para σ_θ , 5 para σ_1 , 6 para σ_2 ,

- 7 para σ_3 e 8 para $\Delta\sigma = \sigma_1 - \sigma_3$), no topo ou no fundo da fileira f pertencente à camada c , de acordo com o valor de *SupInf* (1 ou 2, respectivamente);
- *DeformacoesPrincipais(x, y)* – calcula as deformações principais ($\varepsilon_1, \varepsilon_2, \varepsilon_3$) do ponto de coordenadas globais x e y , retornando estes valores na forma de uma matriz. Caso este ponto pertença a mais de um elemento (ou seja, situado em uma das arestas do elemento), as deformações calculadas se referem às do primeiro elemento encontrado que contenha este ponto;
 - *DeformacoesMax(c, SupInf, def)* – calcula um dos tipos de deformação máxima principal, conforme o valor de *def* (1 para ε_1 , 2 para ε_2 e 3 para ε_3), no topo ou no fundo da camada c , de acordo com o valor de *SupInf* (1 ou 2, respectivamente);
 - *DeformacoesMax(c, f, SupInf, def)* – calcula um dos tipos de deformação máxima principal, conforme o valor de *def* (1 para ε_1 , 2 para ε_2 e 3 para ε_3), no topo ou no fundo da fileira f pertencente à camada c , de acordo com o valor de *SupInf* (1 ou 2, respectivamente);
 - *Deflexao(x, y)* – calcula a deflexão (deslocamento vertical) do ponto de coordenadas globais x e y . Caso este ponto pertença a mais de um elemento (ou seja, situado em uma das arestas do elemento), a deflexão calculada se refere à do primeiro elemento encontrado que contenha este ponto;
 - *DeflexaoMax(c, SupInf)* – Calcula a deflexão (deslocamento vertical) máxima no topo ou no fundo da camada c , de acordo com o valor de *SupInf* (1 ou 2, respectivamente). Se *SupInf* não for informado, o método realiza o cálculo no topo da camada;
 - *DeflexaoMax(c, f, SupInf)* – Calcula a deflexão (deslocamento vertical) máxima no topo ou no fundo da fileira f pertencente à camada c , de acordo com o valor de *SupInf* (1 ou 2, respectivamente)
 - *convergencia(tol)* – calcula o novo módulo de resiliência de todos os elementos do pavimento e, em seguida, verifica se o valor absoluto da diferença percentual entre os módulos de resiliência MR e MR_{ant} $\left(\frac{MR - MR_{ant}}{MR_{ant}} \right)$ de cada elemento é menor do que o valor de *tol*, retornando verdadeiro ou falso, conforme o caso.

4.3 PECULIARIDADES DO PROGRAMA DE ANÁLISE DE PAVIMENTOS

Este programa baseia-se no método dos elementos finitos e, atualmente, realiza a análise da estrutura do pavimento através da modelagem axissimétrica com formulação isoparamétrica. Nesta formulação, as funções de interpolação de deslocamentos são as mesmas funções de interpolação de forma (utilizadas para aproximar a geometria do elemento).

Com esta formulação, as coordenadas globais (r e z) dos nós de um elemento são transformadas em coordenadas locais (ξ e η), que assumem valores compreendidos no intervalo $[-1, 1]$, através de uma matriz de transformação denominada matriz jacobiana. Esta matriz é representada pela equação (4.1).

$$[J] = \begin{bmatrix} \frac{\partial r}{\partial \xi} & \frac{\partial r}{\partial \eta} \\ \frac{\partial z}{\partial \xi} & \frac{\partial z}{\partial \eta} \end{bmatrix} \quad (4.1)$$

Onde:

$[J]$ = Jacobiano ou matriz jacobiana.

A matriz jacobiana relaciona, também, as derivadas parciais das funções de interpolação no referencial global com as respectivas derivadas no referencial local e vice-versa, conforme as equações (4.2) e (4.3), respectivamente.

$$\begin{Bmatrix} \frac{\partial N_i}{\partial \xi} \\ \frac{\partial N_i}{\partial \eta} \end{Bmatrix} = [J] \begin{Bmatrix} \frac{\partial N_i}{\partial r} \\ \frac{\partial N_i}{\partial z} \end{Bmatrix} \quad (4.2)$$

$$\begin{Bmatrix} \frac{\partial N_i}{\partial r} \\ \frac{\partial N_i}{\partial z} \end{Bmatrix} = [J]^{-1} \begin{Bmatrix} \frac{\partial N_i}{\partial \xi} \\ \frac{\partial N_i}{\partial \eta} \end{Bmatrix} \quad (4.3)$$

Onde:

N_i = função de interpolação referente ao nó i do elemento.

Visando facilitar a implementação desta modelagem (axissimétrica com formulação isoparamétrica), a equação (3.22) sofreu pequenas modificações, sendo reescrita conforme a equação (4.4).

$$\{\sigma\} = [D]\{\varepsilon\} \quad (4.4)$$

Onde:

$\{\sigma\}$ = vetor das tensões, expresso pela equação (4.5);

$[D]$ = matriz constitutiva, calculada pela equação (4.6);

$\{\varepsilon\}$ = vetor das deformações, expresso pela equação (4.7).

$$\{\sigma\} = \begin{Bmatrix} \sigma_r \\ \sigma_z \\ \tau_{rz} \\ \sigma_\theta \end{Bmatrix} \quad (4.5)$$

$$[D] = \frac{M_R}{(1+\nu)(1-2\nu)} \begin{bmatrix} 1-\nu & \nu & 0 & \nu \\ \nu & 1-\nu & 0 & \nu \\ 0 & 0 & \frac{1-2\nu}{2} & 0 \\ \nu & \nu & 0 & 1-\nu \end{bmatrix} \quad (4.6)$$

Onde:

M_R = módulo de resiliência do material;

ν = coeficiente de Poisson.

$$\{\varepsilon\} = \begin{Bmatrix} \varepsilon_r \\ \varepsilon_z \\ \gamma_{rz} \\ \varepsilon_\theta \end{Bmatrix} \quad (4.7)$$

Outras mudanças realizadas com a aplicação desta modelagem foram nas equações de determinação das matrizes de operadores diferenciais e gradiente de deslocamento, conforme equações (4.8) e (4.9), respectivamente.

$$[L] = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} [J]^{-1} \begin{bmatrix} \frac{\partial}{\partial \xi} & 0 \\ \frac{\partial}{\partial \eta} & 0 \\ 0 & \frac{\partial}{\partial \xi} \\ 0 & \frac{\partial}{\partial \eta} \\ 1 & 0 \end{bmatrix} \quad (4.8)$$

Onde:

$[L]$ = matriz de operadores diferenciais;

$[J]$ = matriz jacobiana, calculada pela equação (4.10), a seguir.

$$[B_i] = [L][N_i] = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} [J]^{-1} \begin{bmatrix} \frac{\partial N_i}{\partial \xi} & 0 \\ \frac{\partial N_i}{\partial \eta} & 0 \\ 0 & \frac{\partial N_i}{\partial \xi} \\ 0 & \frac{\partial N_i}{\partial \eta} \\ N_i & 0 \end{bmatrix} \quad (4.9)$$

Onde:

$[B_i]$ = matriz gradiente de deformação referente ao nó i do elemento;

N_i = função de interpolação referente ao nó i do elemento.

$$[J] = \begin{bmatrix} \frac{\partial r}{\partial \xi} & \frac{\partial z}{\partial \xi} & 0 & 0 & 0 \\ \frac{\partial r}{\partial \eta} & \frac{\partial z}{\partial \eta} & 0 & 0 & 0 \\ 0 & 0 & \frac{\partial r}{\partial \xi} & \frac{\partial z}{\partial \xi} & 0 \\ 0 & 0 & \frac{\partial r}{\partial \eta} & \frac{\partial z}{\partial \eta} & 0 \\ 0 & 0 & 0 & 0 & r(\xi) \end{bmatrix} \quad (4.10)$$

Onde:

$r(\xi)$ = coordenada global r em função da coordenada local ξ .

A matriz jacobiana de cada elemento pode ser calculada, também, através do somatório do produto das derivadas das funções de interpolação com relação às respectivas coordenadas globais de cada nó, conforme a equação (4.11).

$$[J] = \sum_{i=1}^n \begin{bmatrix} \frac{\partial N_i}{\partial \xi} r_i & \frac{\partial N_i}{\partial \xi} z_i & 0 & 0 & 0 \\ \frac{\partial N_i}{\partial \eta} r_i & \frac{\partial N_i}{\partial \eta} z_i & 0 & 0 & 0 \\ 0 & 0 & \frac{\partial N_i}{\partial \xi} r_i & \frac{\partial N_i}{\partial \xi} z_i & 0 \\ 0 & 0 & \frac{\partial N_i}{\partial \eta} r_i & \frac{\partial N_i}{\partial \eta} z_i & 0 \\ 0 & 0 & 0 & 0 & N_i r_i \end{bmatrix} \quad (4.11)$$

Onde:

n = número de nós do elemento;

N_i = função de interpolação referente ao nó i do elemento;

r_i e z_i = coordenadas globais do nó i do elemento.

Na implementação desta modelagem, a solução da equação (3.17) é obtida através da integração numérica de Gauss, representada pela equação (4.12).

$$K^e = \sum_{i=1}^n 2\pi \cdot W\xi_i \cdot W\eta_i \cdot B(\xi_i, \eta_i)^T \cdot D \cdot B(\xi_i, \eta_i) \cdot \det(J_r) \quad (4.12)$$

Onde:

K^e = matriz de rigidez do elemento e ;

n = número de pontos de Gauss do elemento;

$W\xi_i$ e $W\eta_i$ = pesos referentes às coordenadas locais ξ e η do ponto de Gauss i , respectivamente;

$B(\xi_i, \eta_i)$ = matriz gradiente de deformação para o ponto de coordenadas locais ξ e η do ponto de Gauss i ;

J_r = matriz jacobiana reduzida, expressa pela equação (4.13).

$$[J_r] = \begin{bmatrix} \frac{\partial r}{\partial \xi} & \frac{\partial z}{\partial \xi} & 0 \\ \frac{\partial r}{\partial \eta} & \frac{\partial z}{\partial \eta} & 0 \\ 0 & 0 & r(\xi) \end{bmatrix} = \sum_{i=1}^n \begin{bmatrix} \frac{\partial N_i}{\partial \xi} r_i & \frac{\partial N_i}{\partial \xi} z_i & 0 \\ \frac{\partial N_i}{\partial \eta} r_i & \frac{\partial N_i}{\partial \eta} z_i & 0 \\ 0 & 0 & N_i r_i \end{bmatrix} \quad (4.13)$$

O programa *AXIPAV* baseia-se no fluxograma da figura 16, apresentado no capítulo anterior, para a análise pelo método 1, e, na análise pelo método 2, é baseado no método incremental (descrito no item 3.2.1). Estes métodos possuem várias semelhanças, que serão descritas de uma forma genérica, e algumas diferenças, que serão descritas para cada caso.

Na fase de leitura de dados, são definidas a estrutura do pavimento, através de seu número de camadas e as propriedades de cada uma delas (espessura, coeficiente de Poisson, tipo de material e coeficientes para o cálculo do módulo de resiliência), e as condições de carregamento, através da pressão e do raio da área de contato da roda com o pavimento.

Após esta fase, o domínio do sistema é delimitado horizontalmente, por uma distância de vinte vezes o raio do carregamento, e verticalmente, pelas espessuras das camadas do pavimento. Em seguida, é criada, automaticamente, a malha de elementos finitos com elementos axissimétricos retangulares de oito nós, a partir das características da estrutura (raio do carregamento circular, número de camadas e espessura de cada camada), conforme descrito a seguir:

- inicialmente, considera-se apenas a primeira camada, para a determinação do tamanho das colunas de elementos da malha;
- esta camada é dividida transversalmente de duas formas diferentes. Na primeira, divide-se o comprimento referente à parte solicitada do pavimento em três colunas iguais. E na segunda, divide-se o restante da camada em colunas cujos tamanhos formam uma progressão geométrica com termo inicial igual ao tamanho das colunas anteriormente calculadas e razão igual a 1,15;
- a referida camada ainda é dividida longitudinalmente, de acordo com sua espessura. Se esta espessura não for maior do que dezoito centímetros, a camada é dividida em três fileiras de tamanhos iguais, senão, ela é dividida em fileiras cujos tamanhos formam uma progressão geométrica com termo inicial igual a seis centímetros e razão igual a 1,15;
- em seguida, modificam-se as colunas da malha que contenham os elementos cuja razão comprimento / largura (tamanho da coluna / tamanho da fileira) seja maior do que 5 (razão máxima proposta por Duncan *et al.* (1968), *apud* Silva (1995)). Em

- seguida, modificam-se as fileiras da malha cujo tamanho seja maior do que cinco vezes o menor tamanho das colunas da malha;
- após a definição das colunas da malha, divide-se longitudinalmente cada camada em três fileiras iguais, para as camadas de espessura menor ou igual a dezoito centímetros, ou em fileiras cujos tamanhos formam uma progressão geométrica com termo inicial igual a seis centímetros e razão igual a 1,15, para as camadas de espessura superior a dezoito centímetros. Modificando-se as fileiras cujo tamanho seja maior do que cinco vezes o menor tamanho das colunas da malha;
 - Com a determinação do tamanho das colunas e fileiras da malha, determinam-se as coordenadas dos nós referentes ao ponto médio de cada aresta dos elementos. E, em seguida, armazenam-se as coordenadas de todos os nós da estrutura, bem como as informações de cada elemento do sistema.

Neste sistema, os eixos radial (R) e vertical (Z) possuem sua origem no primeiro nó (nó mais a esquerda da superfície do pavimento) e seus sentidos são, respectivamente, da esquerda para a direita e de cima para baixo. Os eixos das coordenadas locais de cada elemento possuem os mesmos sentidos dos eixos de coordenadas globais correspondentes e sua origem está localizada no centro do elemento. A figura 18 apresenta a representação esquemática dos eixos das coordenadas locais e os nós locais de um elemento utilizado no programa.

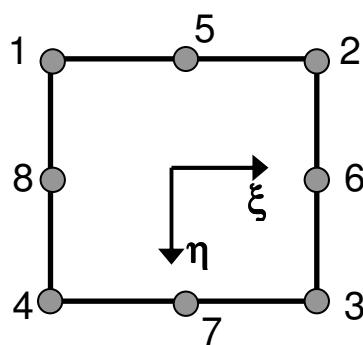


FIG. 18 – Eixos de coordenadas locais e nós dos elementos utilizados no programa

A figura 19 detalha este processo de geração de malha.

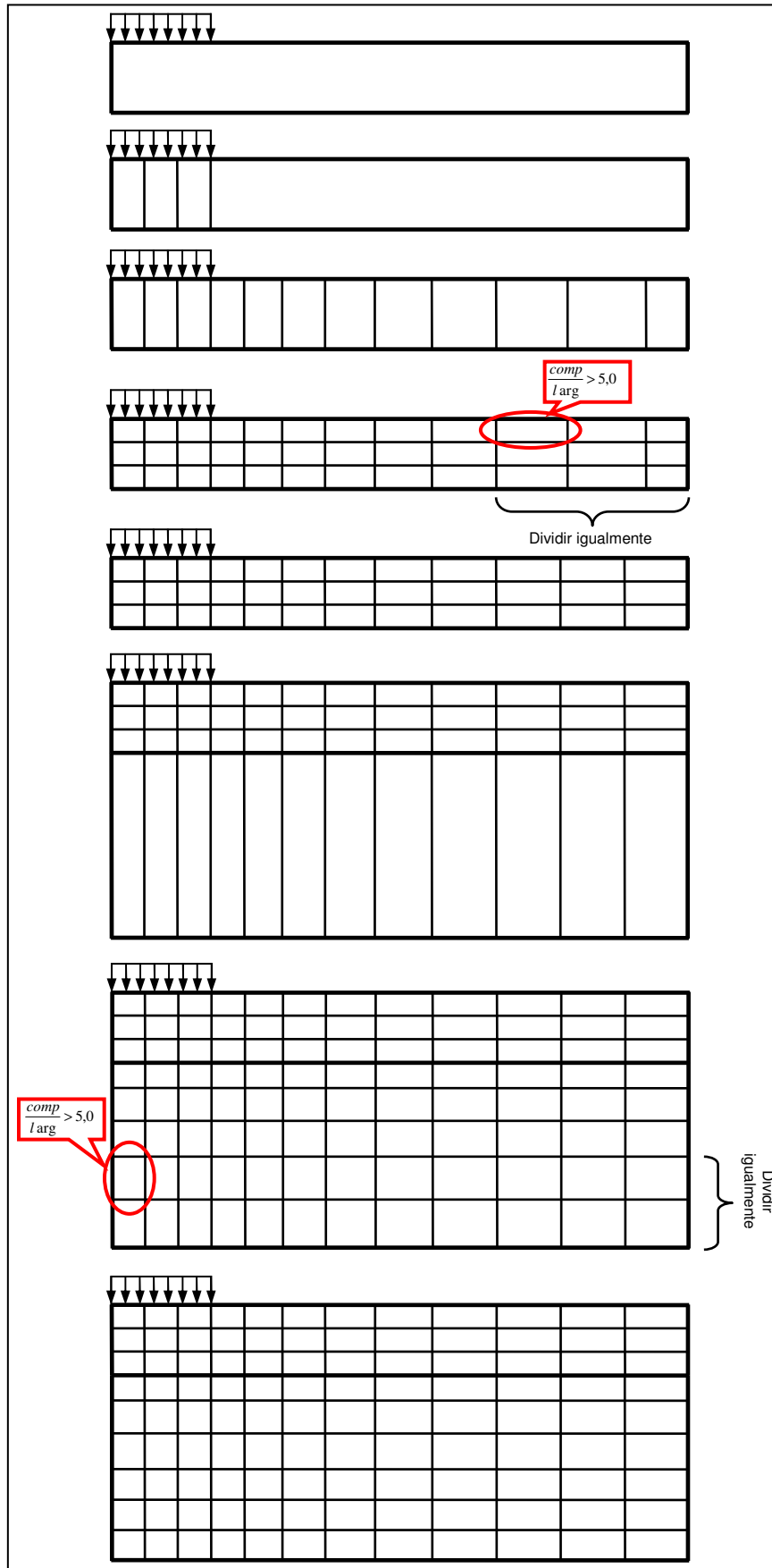


FIG. 19 – Processo de geração automática de malha

Depois da criação da malha, as restrições de deslocamentos, bem como as cargas nodais do sistema são atribuídas automaticamente, ou seja, sem a intervenção do usuário. No programa, as cargas nodais são calculadas conforme as equações (4.14), onde: no método 1 (fluxograma da figura 16), essas cargas são consideradas constantes para todas as iterações; e no método 2 (item 3.2.1), as cargas nodais de cada iteração é a multiplicação dos resultados das equações (4.14) pela razão entre o número da iteração e o número total de iterações, fornecido pelo usuário. Além disso, são consideradas as seguintes restrições no pavimento:

- os nós pertencentes à primeira coluna de nós (coordenada radial igual a zero) não permitem o deslocamento radial;
- os nós pertencentes à última coluna de nós (maior coordenada radial) não permitem o deslocamento radial;
- os nós situados na última fileira (maior coordenada vertical) não permitem o deslocamento vertical e nem o radial.

$$F_z(1) = \pi \cdot q \cdot \left(\frac{r(2) + r(1)}{2} - r(1) \right)^2 \quad (4.14a)$$

$$F_z(i) = \pi \cdot q \cdot \left(\left(\frac{r(i+1) + r(i)}{2} - r(1) \right)^2 - \left(\frac{r(i) + r(i-1)}{2} - r(1) \right)^2 \right), \quad 2 \leq i \leq 6 \quad (4.14b)$$

$$F_z(7) = \pi \cdot q \cdot \left((r(7) - r(1))^2 - \left(\frac{r(7) + r(6)}{2} - r(1) \right)^2 \right) \quad (4.14c)$$

Onde:

$F_z(i)$ = carga nodal na direção vertical referente ao nó i ;

q = pressão de contato pneu pavimento;

$r(i)$ = coordenada global radial do nó i .

Em seguida, são calculados o módulo de resiliência e a matriz de rigidez de cada elemento. O módulo de resiliência do elemento é determinado através das equações contidas na tabela 4, de acordo com o tipo de material, em função dos valores, fornecidos pelo usuário, da tensão-desvio mínima ($\sigma_{d\min}$) e da tensão confinante mínima ($\sigma_{3\min}$). A matriz de rigidez do elemento é calculada através da equação (4.12), utilizando as coordenadas dos oito pontos de Gauss do elemento e seus respectivos pesos, contidos na tabela 5.

Após estes cálculos, monta-se a matriz de rigidez global da estrutura e aplicam-se as condições de apoio do pavimento no sistema. A referida matriz é construída a partir da matriz de rigidez elementar, para cada elemento, e seu respectivo vetor de incidência e é modificada de acordo com as condições de apoio. Esta modificação é realizada através de uma das seguintes maneiras:

- adicionando uma penalidade (valor muito grande) aos elementos da matriz de rigidez global, referentes aos deslocamentos restritos, que pertencem à diagonal principal;
- excluindo as linhas e colunas da matriz de rigidez global que se referem aos deslocamentos restritos da estrutura, bem como as linhas correspondentes do vetor de cargas nodais. Este método é o considerado padrão pelo programa, e, caso se deseje utilizar o primeiro método, deve-se implementar novamente o programa de modo a transformar este método em padrão.

TAB. 5 – Coordenadas e pesos dos pontos de Gauss do elemento

Ponto de Gauss	Coordenada		Peso	
	ξ	η	$W\xi$	$W\eta$
1	$-\sqrt{0,6}$	$-\sqrt{0,6}$	$\frac{5}{9}$	$\frac{5}{9}$
2	$\sqrt{0,6}$	$-\sqrt{0,6}$	$\frac{5}{9}$	$\frac{5}{9}$
3	$\sqrt{0,6}$	$\sqrt{0,6}$	$\frac{5}{9}$	$\frac{5}{9}$
4	$-\sqrt{0,6}$	$\sqrt{0,6}$	$\frac{5}{9}$	$\frac{5}{9}$
5	0	$-\sqrt{0,6}$	$\frac{8}{9}$	$\frac{5}{9}$
6	$\sqrt{0,6}$	0	$\frac{5}{9}$	$\frac{8}{9}$
7	0	$\sqrt{0,6}$	$\frac{8}{9}$	$\frac{5}{9}$
8	$-\sqrt{0,6}$	0	$\frac{5}{9}$	$\frac{8}{9}$

Em seguida, calcula-se o vetor de deslocamentos nodais através da resolução do sistema de equações representado pela equação (2.6), e, para cada ponto de Gauss, são calculadas as tensões axissimétricas ($\sigma_r, \sigma_z, \tau_{rz}, \sigma_\theta$) e transformadas em tensões principais ($\sigma_1, \sigma_2, \sigma_3$). Feito isso, determina-se o novo módulo de resiliência de cada elemento, conforme descrito a seguir:

- calcula-se o módulo de resiliência de cada ponto de Gauss do elemento, a partir das equações contidas na tabela 4, de acordo com o tipo de material, utilizando as tensões principais deste ponto no caso destas serem maiores que as tensões mínimas ($\sigma_{d\min}$ e $\sigma_{3\min}$), caso contrário utilizam-se as tensões mínimas fornecidas pelo usuário;
- o módulo de resiliência do elemento é a média aritmética dos módulos calculados nos oito pontos de Gauss do mesmo.

Para o método 1, após esta determinação, verifica-se, para todos os elementos, se a diferença percentual entre o módulo de resiliência novo e o anterior é menor que o valor da tolerância, fornecida pelo usuário. Se existir algum elemento que não satisfaça esta condição, procede-se novamente a análise da estrutura, calculando as matrizes de rigidez elementar e global, montando o sistema de equações e resolvendo-o, calculando as tensões e o módulo de resiliência de cada elemento e realizando uma nova verificação.

Para o método 2, após esta determinação, verifica-se se o número da iteração é igual ao número máximo de iterações, fornecido pelo usuário. Se estes números não forem iguais, procede-se novamente a análise da estrutura, calculando as matrizes de rigidez elementar e global, montando o sistema de equações e resolvendo-o, calculando as tensões e o módulo de resiliência de cada elemento e realizando uma nova verificação.

Este processo iterativo é realizado enquanto não for satisfeita a verificação supracitada. Ao término desta fase, denominada fase de análise, os resultados são transcritos no arquivo de saída “Resultados.txt”, e as informações necessárias para a realização do dimensionamento mecânico do pavimento serão calculadas na próxima fase do programa.

Nesta última etapa do programa, intitulada de fase de resultados, são calculadas as seguintes informações:

- a deflexão (deslocamento vertical) máxima da superfície do pavimento e o valor relativo ao ponto de coordenada radial igual a uma vez e meia o raio do carregamento;

- a maior deformação específica de tração (ε_3) da fibra inferior do revestimento e o valor relativo ao ponto de coordenada radial igual a uma vez e meia o raio do carregamento;
- a maior tensão de tração (σ_3) na fibra inferior do revestimento e valor relativo ao ponto de coordenada radial igual a uma vez e meia o raio do carregamento;
- a maior diferença de tensão ($\sigma_1 - \sigma_3$) na fibra inferior do revestimento e o valor relativo ao ponto de coordenada radial igual a uma vez e meia o raio do carregamento;
- a maior tensão vertical (σ_z) no topo de cada camada e o valor relativo ao ponto de coordenada radial igual a uma vez e meia o raio do carregamento.

Os valores referentes ao ponto de coordenada radial igual a uma vez e meia o raio do carregamento são calculados para possibilitar a superposição de efeitos do eixo rodoviário padrão. As informações anteriormente citadas são transcritas no arquivo de saída “Resumo.txt”, bem como o módulo de resiliência de cada elemento.

O capítulo seguinte possibilitará a validação do programa descrito neste capítulo.

5 VALIDAÇÃO DO MODELO

Este capítulo tem por objetivo realizar a análise de algumas estruturas de pavimento que foram objeto de análise de outros trabalhos, e, mediante comparação de seus resultados, possibilitar a validação do programa desenvolvido nesta dissertação.

Segundo Rede Asfalto (2006), é prática comum na validação de implementações computacionais a utilização de exemplos com soluções analíticas conhecidas ou exemplos com soluções numéricas já testadas e analisadas, cujos resultados podem ser considerados confiáveis.

Para Pfleeger (2004), *apud* Barreto (2006) a validação assegura que todos os requisitos do sistema foram implementados, de maneira que cada função do sistema possa ser relacionada com um requisito particular na especificação. Segundo Barreto (2006), a norma NBR ISO/IEC 12207/98 define validação como a confirmação, por exame e fornecimento de evidência objetiva, de que os requisitos específicos, para um determinado uso pretendido, são atendidos.

De acordo com Rocha *et al.* (2001), *apud* Barreto (2006), o objetivo da validação é assegurar que o software que está sendo desenvolvido é adequado de acordo com os requisitos do usuário. Para Rede Asfalto (2006), esta etapa requer especial atenção, pois deve-se confirmar a veracidade dos dados obtidos com o novo sistema computacional para que se possa comprovar sua eficiência e confiabilidade.

5.1 PAVIMENTO ANALISADO POR PREUSSLER (1983)

Segundo Silva (1995), Preussler analisou em sua tese de doutorado (Preussler, 1983) o comportamento das deformações e tensões de um pavimento com 5,0 cm de revestimento em concreto asfáltico e 30,0 cm de base granular, assentados diretamente no subleito, sujeitos a uma carga circular uniformemente distribuída de raio igual a 10,8 cm e pressão igual a 5,6 kgf/cm². A figura 20 representa, esquematicamente, a estrutura deste pavimento.

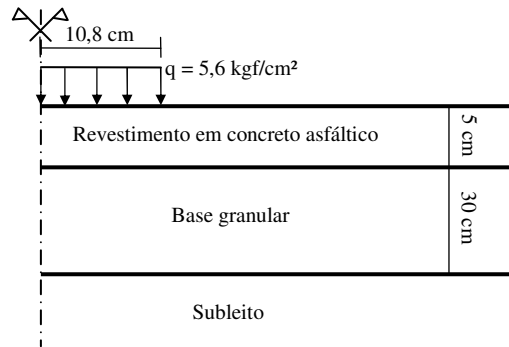


FIG. 20 – Representação do pavimento analisado por Preussler (1983)

Este pavimento foi analisado através de duas abordagens diferentes. Na primeira, os módulos de resiliência das camadas foram considerados constantes (materiais elástico-lineares). Na segunda, adotaram-se materiais de comportamento elástico não-linear. As propriedades das camadas deste pavimento estão contidas na tabela 6, para o primeiro caso, e na tabela 7, para o segundo.

TAB. 6 – Propriedades das camadas do pavimento analisado por Preussler (1983) no primeiro caso

Camada	Módulo de Resiliência	Coefficiente de Poisson
Revestimento	70.000 kgf/cm ²	0,25
Base	5.890 kgf/cm ²	0,35
Subleito	520 kgf/cm ²	0,45

TAB. 7 – Propriedades das camadas do pavimento analisado por Preussler (1983) no segundo caso

Camada	Módulo de Resiliência (kgf/cm ²)	Coefficiente de Poisson
Revestimento	70.000	0,25
Base	$10.800 \sigma_3^{0,99}$	0,35
Subleito	$M_R = 4.100 + 2.860(0,64 - \sigma_d)$, para $\sigma_d < 0,64$ $M_R = 4.100 - 316(\sigma_d - 0,64)$, para $\sigma_d > 0,64$	0,45

Segundo Silva (1995), Preussler utilizou o programa FEPAVE na análise destas estruturas e obteve os resultados contidos na tabela 8.

Com o objetivo de analisar este pavimento, foram necessárias transformações das unidades dos módulos de resiliência de suas camadas, para ambos os casos. A tabela 9 mostra os valores dos módulos de resiliência convertidos.

TAB. 8 – Resultados obtidos por Preussler (1983)

Caso	Deflexão (0,01 mm)	Máxima tensão vertical no subleito	Deformação específica de tração no revestimento
Linear	31	0,018 MPa	$2,44 \times 10^{-4}$
Não-linear	54	0,022 MPa	$3,36 \times 10^{-4}$
Medida de campo	52	-	-

FONTE: Preussler (1983) apud Silva (1995)

TAB. 9 – Módulos de resiliência das camadas do pavimento analisado por Preussler (1983), convertidos.

Camada	Módulo de Resiliência (MPa)	
	Elástico-linear	Elástico não-linear
Revestimento	7.000	7.000
Base	589	$10.554,162 \sigma_3^{0,99}$
Subleito	52	$M_R = 410 + 2.860(0,064 - \sigma_d)$, para $\sigma_d < 0,064$ $M_R = 410 - 316(\sigma_d - 0,064)$, para $\sigma_d > 0,064$

Este pavimento foi analisado pelo programa desenvolvido nesta dissertação, utilizando, inicialmente, o método 1 (baseado no fluxograma da figura 16), para ambos os casos. Porém, para o segundo caso, não foi possível encontrar solução, pois o problema não convergiu até a sétima iteração e, após esta, o programa foi finalizado.

Então, para se realizar a análise deste caso, foi utilizado o método 2 (método incremental, item 3.2.1) com quatro iterações (cada iteração com incremento de um quarto da carga). A tabela 10 mostra os resultados obtidos na análise destas estruturas.

TAB. 10 – Resultados obtidos na análise com o programa desta dissertação

Caso	Deflexão (0,01 mm)	Máxima tensão vertical no subleito	Deformação específica de tração no revestimento
Linear	46,5	0,022 MPa	$1,97 \times 10^{-4}$
Não-linear	57,1	0,152 MPa	$4,07 \times 10^{-4}$

Da análise comparativa dos resultados mostrados nas tabelas 8 e 10, pode-se concluir que os valores da deflexão obtidos com o programa desenvolvido nesta dissertação estão próximos aos medidos em campo. Além disso, os demais valores (tensão vertical no topo do subleito e deformação específica de tração na fibra inferior do revestimento) obtidos com este programa estão coerentes com os obtidos, através do programa FEPAVE, por Preussler, com exceção da máxima tensão vertical no topo do subleito para o segundo caso. Entretanto, não se pode afirmar qual seria o valor correto, pois o mesmo não foi medido no campo.

Concomitantemente, o programa FEPAVE e o método 2 com quatro iterações do programa AXIPAV possuem algumas diferenças entre si, como:

- o FEPAVE utiliza elementos de quatro nós, enquanto que o AXIPAV utiliza elementos de oito nós;
- o FEPAVE subdivide cada elemento quadrilátero em quatro triângulos e utiliza como matriz de rigidez deste elemento, a média das matrizes de rigidez dos triângulos.

5.2 PAVIMENTO ANALISADO POR PINTO (1991)

Pinto, em sua tese de doutorado (Pinto, 1991), analisou o comportamento de trechos experimentais, situados nas rodovias BR-040/RJ (Areal-Moura Brasil) e BR-101/RJ (Niterói-Manilha), com a utilização de um defletômetro de impacto, o “*Falling Weight Deflectometer*” (FWD), e com a viga Benkelman.

De acordo com a norma DNER-PRO 273/96, o FWD é um defletômetro de impacto projetado para simular o efeito da passagem de uma carga de roda em movimento no pavimento. A medida de deflexão é obtida pela queda de um conjunto de massas, a partir de alturas pré-fixadas, sobre um sistema de amortecedores de borracha.

Segundo Borges (2001), este sistema foi especialmente projetado para tornar o pulso de carga recebido pelo pavimento, o mais próximo possível de uma senóide. Segundo Cardoso (1999) *apud* Borges (2001), igualando-se a energia potencial da massa, antes de sua queda, com o trabalho desenvolvido pelos amortecedores, depois da queda, pode-se conhecer a força de pico exercida sobre o pavimento, de acordo com a equação (5.1).

$$F_p = \sqrt{2Mghk} \quad (5.1)$$

Onde:

F_p = força de pico;

M = massa do peso que cai;

g = aceleração da gravidade;

h = altura de queda;

k = constante de mola do sistema de amortecedores.

As deflexões (deslocamentos verticais da superfície do pavimento) são obtidas por até sete transdutores de velocidade, sendo um sob o centro da placa e os outros, em posições variáveis, adaptados a uma barra levadiça de até 4,5 m de comprimento. Estes transdutores armazenam, para os pontos do pavimento em que estiverem instalados, os valores referentes aos seus picos de deflexão.

A figura 21 mostra a representação esquemática deste aparelho e sua bacia de deformação, e na figura 22 é apresentada uma foto de um aparelho deste tipo.

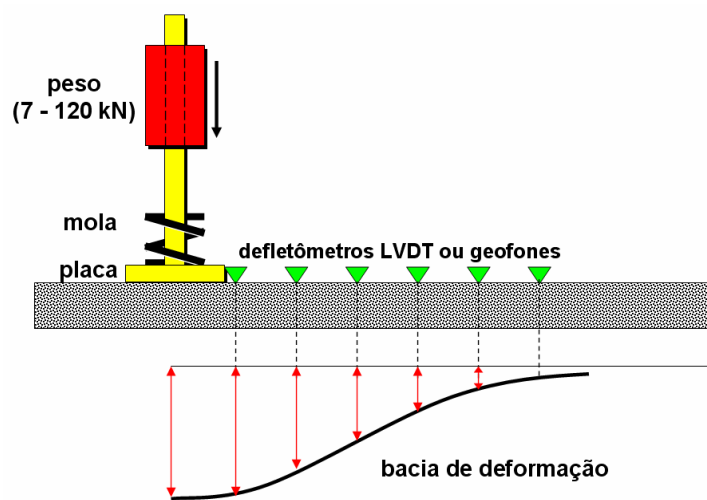


FIG. 21 – Representação esquemática do FWD e sua bacia de deformação (Vieira, 2006)



FIG. 22 – Foto de um aparelho do tipo FWD (Vieira, 2006)

De acordo com a norma DNER-ME 024/94, a viga Benkelman é um aparelho destinado a medir deflexões em pavimentos. Ela é constituída de um conjunto de sustentação em que se articula uma alavanca interfixa, formando dois braços cujos comprimentos a e b obedecem as relações de 2/1, 3/1 ou de 4/1. A extremidade do braço maior contém a ponta de prova da viga e a outra aciona um extensômetro com precisão de 0,01 mm.

Este aparelho possui, ainda, um pequeno vibrador destinado a evitar eventuais inibições do ponteiro do extensômetro e dispõe de uma trava de proteção a ser utilizada por ocasião do transporte. Quando não está em uso, este equipamento deve ser inteiramente revestido com isopor.

Segundo Borges (2001), durante o ensaio, a ponta de prova da viga é colocada a meia-distância das rodas do semi-eixo traseiro simples de roda dupla. O caminhão padronizado para este tipo de levantamento tem suas rodas duplas traseiras com pneus calibrados à pressão de 0,55 MPa (5,6 kgf/cm² ou 80 lb/pol²) e carga em seus eixo traseiro de 80 kN (8,2 tf).

A figura 23 mostra a representação esquemática deste aparelho, e na figura 24 é apresentada uma foto de sua utilização.

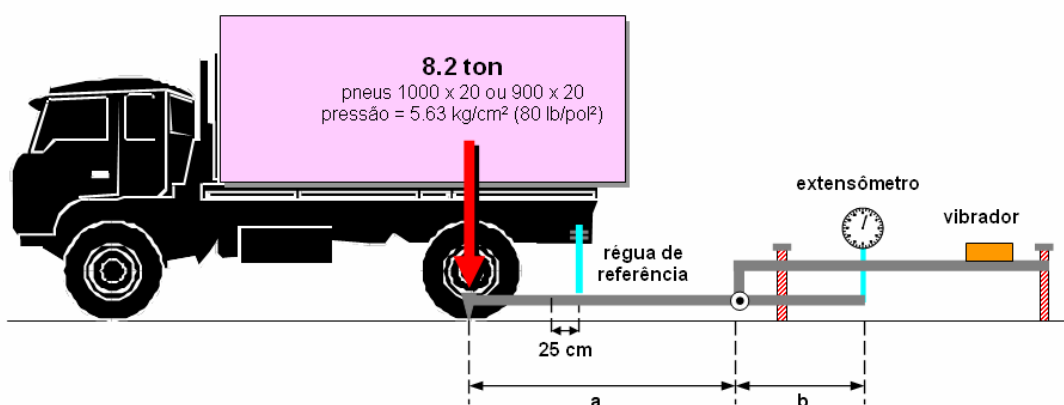


FIG. 23 – Representação esquemática da viga Benkelman (Vieira, 2006)

Na BR-101/RJ, foram analisados três trechos experimentais, com perfis estruturais diferentes e compreendidos, respectivamente, entre as estacas: 162 + 10 m e 165; 165 + 10 m

e 173; e 173 + 10 m e 181 + 10 m. As características de cada um desses trechos são descritas a seguir:

- 1º segmento – formado por um pavimento com 10,0 cm de revestimento em concreto asfáltico, 15,0 cm de base de brita graduada e 19,0 cm de sub-base de brita corrida, assentados diretamente em um subleito de argila amarela de gnaiss. Esta estrutura foi denominada de Seção A e está representada, esquematicamente, na figura 25;
- 2º segmento – formado por um pavimento com 10,0 cm de revestimento em concreto asfáltico, 15,0 cm de base de brita graduada e 25,0 cm de sub-base de saibro (solo arenoso residual de gnaiss), assentados diretamente em um subleito de argila amarela de gnaiss. Esta estrutura foi denominada de Seção B e está representada, esquematicamente, na figura 26;
- 3º segmento – formado por um pavimento com 10,0 cm de revestimento em concreto asfáltico e 25,0 cm de base de brita graduada, assentados diretamente em um subleito de argila amarela de gnaiss. Esta estrutura foi denominada de Seção C e está representada, esquematicamente, na figura 27.



FIG. 24 – Viga Benkelman em utilização (Vieira, 2006)

As propriedades das camadas destes pavimentos estão contidas na tabela 11, onde são apresentados os valores do coeficiente de Poisson e dos módulos de resiliência em kgf/cm^2 e em MPa, para cada camada.

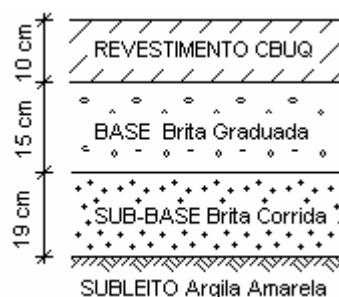


FIG. 25 – Representação esquemática da Seção A

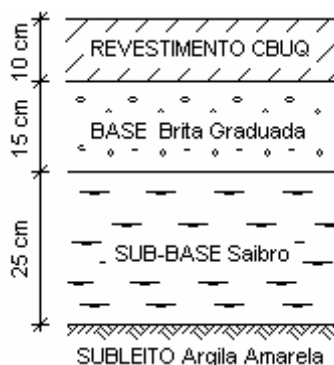


FIG. 26 – Representação esquemática da Seção B

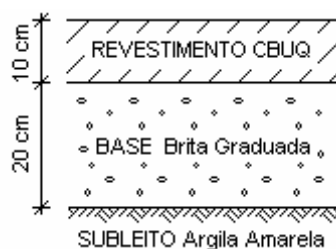


FIG. 27 – Representação esquemática da Seção C

Na BR-040/RJ, foram analisados dois trechos experimentais, com perfis estruturais diferentes e compreendidos, respectivamente, entre as estacas: 12 + 10 m e 17 + 10 m; e 18 e 23. As características de cada um desses trechos são descritas a seguir:

- 1º segmento – formado por um pavimento com 10,0 cm de revestimento em concreto asfáltico, 15,0 cm de base de brita graduada, 20,0 cm de sub-base de areia argilosa de gnaise e 31 cm de sub-base e reforço de areia argilosa de gnaise, assentados diretamente em um subleito de argila vermelha de gnaise. Esta estrutura foi denominada de Seção D e está representada, esquematicamente, na figura 28;
- 2º segmento – formado por um pavimento com 10,0 cm de revestimento em concreto asfáltico, 37,0 cm de base de brita graduada e 26,0 cm de sub-base de argila

amarela de gnaiss, assentados diretamente em um subleito de argila vermelha de gnaiss. Esta estrutura foi denominada de Seção E e está representada, esquematicamente, na figura 29.

TAB. 11 – Propriedades das camadas dos pavimentos da BR-101/RJ analisados por Pinto (1991)

Camada	Módulo de Resiliência		Coefficiente de Poisson
Revestimento	30.000	(kgf/cm ²)	0,25
	3.000	(MPa)	
Base	$5.560 \sigma_3^{1,0}$	(kgf/cm ²)	0,35
	$5.560 \sigma_3^{1,0}$	(MPa)	
Sub-base de Saibro	$2.863 \sigma_3^{0,62}$	(kgf/cm ²)	0,35
	$1.193,497 \sigma_3^{0,62}$	(MPa)	
Sub-base de Brita corrida	$5.560 \sigma_3^{1,0}$	(kgf/cm ²)	0,35
	$5.560 \sigma_3^{1,0}$	(MPa)	
Subleito	$M_R = 2.700 + 4.000(0,58 - \sigma_d)$, para $\sigma_d < 0,58$ $M_R = 2.700 - 1.854(\sigma_d - 0,58)$, para $\sigma_d > 0,58$	(kgf/cm ²)	0,45
	$M_R = 270 + 4.000(0,058 - \sigma_d)$, para $\sigma_d < 0,058$ $M_R = 270 - 1.854(\sigma_d - 0,058)$, para $\sigma_d > 0,058$	(MPa)	

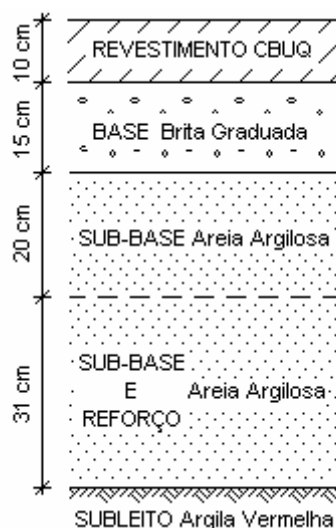


FIG. 28 – Representação esquemática da Seção D

As propriedades das camadas destes pavimentos estão contidas na tabela 12, onde são apresentados os valores do coeficiente de Poisson e dos módulos de resiliência em kgf/cm² e em MPa, para cada camada.

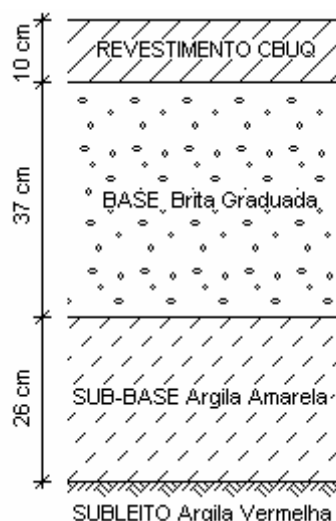


FIG. 29 – Representação esquemática da Seção E

TAB. 12 – Propriedades das camadas dos pavimentos da BR-040/RJ analisados por Pinto (1991)

Camada	Módulo de Resiliência		Coeficiente de Poisson
Revestimento	25.000	(kgf/cm ²)	0,25
	2.500	(MPa)	
Base	$20.000 \sigma_3^{1,0}$	(kgf/cm ²)	0,35
	$20.000 \sigma_3^{1,0}$	(MPa)	
Sub-base de Saibro	$2.863 \sigma_3^{0,62}$	(kgf/cm ²)	0,35
	$1.193,497 \sigma_3^{0,62}$	(MPa)	
Sub-base de Areia argilosa	$4.000 \sigma_3^{0,70}$	(kgf/cm ²)	0,35
	$2.004,749 \sigma_3^{0,70}$	(MPa)	
Sub-base de Argila amarela	$M_R = 5.200 + 50.000(0,64 - \sigma_d)$, para $\sigma_d < 0,64$ $M_R = 5.200 - 1.800(\sigma_d - 0,64)$, para $\sigma_d > 0,64$	(kgf/cm ²)	0,35
	$M_R = 520 + 50.000(0,064 - \sigma_d)$, para $\sigma_d < 0,064$ $M_R = 520 - 1.800(\sigma_d - 0,064)$, para $\sigma_d > 0,064$	(MPa)	
Sub-base e Reforço de Areia argilosa	$4.000 \sigma_3^{0,70}$	(kgf/cm ²)	0,35
	$2.004,749 \sigma_3^{0,70}$	(MPa)	
Subleito	$M_R = 8.400 + 40.000(0,85 - \sigma_d)$, para $\sigma_d < 0,85$ $M_R = 8.400 - 3.000(\sigma_d - 0,85)$, para $\sigma_d > 0,85$	(kgf/cm ²)	0,45
	$M_R = 840 + 40.000(0,085 - \sigma_d)$, para $\sigma_d < 0,085$ $M_R = 840 - 3.000(\sigma_d - 0,085)$, para $\sigma_d > 0,085$	(MPa)	

FONTE: Pinto (1991)

Em todos os cinco trechos experimentais, foram medidas as deflexões (deslocamentos verticais) na superfície através do FWD, utilizando três níveis de carga (25,8 kN, 38,8 kN e

63,1 kN), placa de carga de 30,0 cm de diâmetro e sete geofones (localizados a 0, 20, 30, 45, 60, 90 e 120 centímetros de distância do centro da placa de carga), e através da viga Benkelman, medindo a deflexão que ocorreu entre as rodas duplas do eixo padrão rodoviário (carga de 80 kN, com raio da área de contato de 10,8 cm). Os resultados obtidos estão na tabela 13, onde D_0 , D_{20} , D_{30} , D_{45} , D_{60} , D_{90} e D_{120} são as deflexões dos pontos localizados a 0, 20, 30, 45, 60, 90 e 120 centímetros de distância do centro da carga circular, respectivamente.

TAB. 13 – Resultados obtidos nos trechos experimentais analisados por Pinto (1991)

Seção	Carga (kN)	Deflexões FWD (0,01 mm)							Deflexões viga Benkelman
		D_0	D_{20}	D_{30}	D_{45}	D_{60}	D_{90}	D_{120}	
A	25,8	35,3	22,7	16,1	9,1	5,4	3,6	2,7	52×10^{-2}
	38,8	48,4	32,6	24,1	14,7	9,1	6,0	4,6	
	63,1	71,7	50,6	39,3	25,8	16,8	11,2	8,5	
B	25,8	68,6	41,0	26,4	11,4	4,1	2,0	1,9	107×10^{-2}
	38,8	91,6	57,4	38,8	18,4	7,2	3,4	3,0	
	63,1	128,3	84,1	59,3	30,5	13,4	6,5	5,4	
C	25,8	36,1	21,8	14,7	7,6	4,4	3,1	2,6	55×10^{-2}
	38,8	50,3	31,7	22,2	12,3	7,1	4,9	4,1	
	63,1	75,8	50,0	36,5	21,7	12,9	8,7	7,1	
D	25,8	57,3	39,5	26,8	13,3	5,6	2,9	2,3	127×10^{-2}
	38,8	79,0	57,0	40,7	21,9	9,8	4,7	3,6	
	63,1	114,6	86,0	64,6	37,6	18,2	8,6	6,2	
E	25,8	37,4	24,8	17,9	10,2	5,6	3,3	2,6	79×10^{-2}
	38,8	49,4	34,2	25,5	15,4	9,0	5,4	4,2	
	63,1	69,7	50,3	38,8	25,2	15,7	9,7	7,3	

FONTE: Pinto (1991)

As deflexões obtidas através do deflectômetro de impacto FWD devem ser comparadas com respostas oriundas de análises dinâmicas, pois o carregamento do ensaio varia com o tempo. Em uma análise dinâmica, os deslocamentos do sistema, que num carregamento estático, eram calculados através de um sistema de equações lineares conforme a equação (2.6), são calculados através da equação (5.2), que representa um sistema de equações diferenciais ordinárias.

$$[M]\{\ddot{U}\} + [C]\{\dot{U}\} + [K]\{U\} = \{F(t)\} \quad (5.2)$$

Onde:

$[M]$ = matriz de massa;

$[C]$ = matriz de amortecimento;

$[K]$ = matriz de rigidez;

$\{U\}$ = vetor deslocamento;

$\{\dot{U}\}$ = derivada primeira do vetor deslocamento em relação ao tempo;

$\{\ddot{U}\}$ = derivada segunda do vetor deslocamento em relação ao tempo;

$\{F(t)\}$ = vetor de forças em função do tempo.

Costuma-se determinar, também, nesta análise, o fator de amplificação dinâmica, definido como a razão entre a resposta dinâmica num determinado tempo e o deslocamento estático. Segundo Craig Jr. (1981), o fator de amplificação dinâmica máximo de uma estrutura submetida a uma carga degrau ideal (carregamento constante com o tempo) varia entre 1 e 2, quando o carregamento é aplicado de forma gradativa e quando é aplicado de forma súbita, respectivamente.

Para uma carga retangular (carregamento constante apenas durante um determinado intervalo de tempo), Craig Jr. (1981) afirma que o fator de amplificação dinâmica máximo da estrutura varia entre 0 e 2, de acordo com o tempo de aplicação da carga e a frequência natural da estrutura.

Embora o programa *AXIPAV*, programa desenvolvido nesta dissertação, utilize a metodologia do método dos elementos finitos para carregamentos estáticos, utilizou-se do mesmo para comparar com os resultados obtidos no ensaio dinâmico, do deflectômetro de impacto FWD. Nesta comparação, deve-se analisar os resultados sob o ponto de vista qualitativo, verificando apenas o fator de amplificação dinâmica para cada caso, e realizar uma verificação do comportamento do pavimento para cada caso analisado.

Então, as cinco seções foram analisadas pelo programa *AXIPAV*, utilizando o método 2 (método incremental) com quatro iterações (cada iteração com incremento de um quarto da carga), para os quatro tipos de carga (FWD com três níveis de carga e viga Benkelman) e para cada análise foram consideradas quatro espessuras de subleito (1,00 m, 2,00 m, 3,00 m e 4,00 m). As tabelas 14, 15, 16, 17 e 18 mostram os resultados obtidos na análise da Seção A, Seção B, Seção C, Seção D e Seção E, respectivamente.

TAB. 14 – Resultados obtidos na análise da Seção A

Carga FWD	Espessura Subleito	Deflexões FWD (0,01 mm)						
		D ₀	D ₂₀	D ₃₀	D ₄₅	D ₆₀	D ₉₀	D ₁₂₀
25,8 (kN)	1,00 m	53,5	39,4	30,0	18,6	10,6	2,0	-0,8
	2,00 m	54,6	40,5	31,0	19,5	11,4	2,6	-0,4
	3,00 m	54,9	40,7	31,3	19,8	11,7	2,8	-0,3
	4,00 m	55,0	40,9	31,4	19,9	11,8	2,9	-0,2
	Medido no Campo	35,3	22,7	16,1	9,1	5,4	3,6	2,7
38,8 (kN)	1,00 m	67,5	48,8	36,7	22,4	12,6	2,2	-1,1
	2,00 m	69,0	50,3	38,1	23,9	13,9	3,2	-0,5
	3,00 m	69,4	50,7	38,6	24,3	14,3	3,5	-0,2
	4,00 m	69,6	50,8	38,7	24,4	14,4	3,6	-0,1
	Medido no Campo	48,4	32,6	24,1	14,7	9,1	6,0	4,6
63,1 (kN)	1,00 m	95,4	68,4	51,5	31,7	18,0	3,3	-1,4
	2,00 m	97,6	70,7	53,8	34,0	20,1	4,9	-0,5
	3,00 m	98,1	71,3	54,4	34,6	20,7	5,4	0,0
	4,00 m	98,4	71,6	54,7	34,9	21,0	5,7	0,2
	Medido no Campo	71,7	50,6	39,3	25,8	16,8	11,2	8,5
Deflexões da Viga Benkelman (0,01 mm)								
Subleito de 1,00 m	Subleito de 2,00 m	Subleito de 3,00 m	Subleito de 4,00 m	Medido no Campo				
79,9	81,4	81,7	81,9	52				

Da análise comparativa dos resultados mostrados na tabela 14, pode-se concluir que a espessura do subleito considerada na análise provocou influência nos resultados obtidos pelo programa, em especial, quando foi adotada espessura de 1,00 m. Houve, também, discrepância com os deslocamentos medidos em campo, obtendo-se um fator de amplificação dinâmica médio de 0,70.

Além disso, os valores obtidos para o ponto mais afastado do carregamento (120 cm em relação à origem) não correspondem à realidade do pavimento, podendo ser considerado uma deficiência do modelo axissimétrico para pavimento, que se preocupa com os pontos próximos ao carregamento. A figura 30 mostra o gráfico das deflexões calculadas pelo programa, considerando espessura do subleito de 4,00 m, e medidas em campo.

Da análise comparativa dos resultados mostrados na tabela 15, pode-se concluir que a espessura do subleito considerada na análise provocou influência nos resultados obtidos pelo programa, em especial, quando foi adotada espessura de 1,00 m. Houve, também, convergência com os deslocamentos medidos em campo, exceto para seis valores da tabela (afastamento de 45 cm e 60 cm em relação à origem para as cargas de 25,8 kN e 38,8 kN; e

afastamento de 45 cm e 60 cm em relação à origem, para a carga de 63,1 kN) e obteve-se um fator de amplificação dinâmica médio de 0,85.

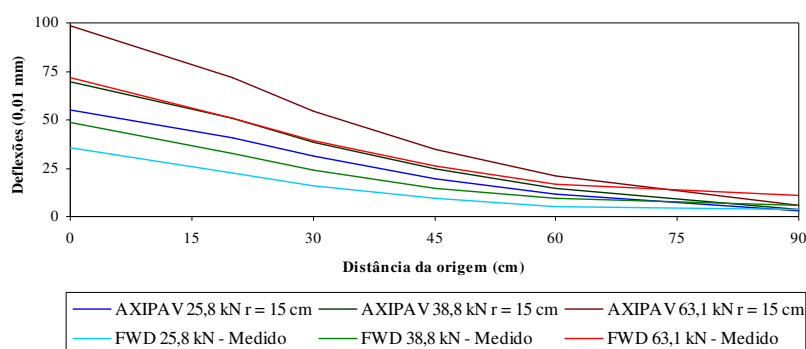


FIG. 30 – Deflexões da seção A

TAB. 15 – Resultados obtidos na análise da Seção B

Carga FWD	Espessura Subleito	Deflexões FWD (0,01 mm)						
		D ₀	D ₂₀	D ₃₀	D ₄₅	D ₆₀	D ₉₀	D ₁₂₀
25,8 (kN)	1,00 m	54,9	40,0	30,0	18,0	9,7	1,3	-1,0
	2,00 m	55,8	40,9	30,8	18,6	10,4	1,9	-0,6
	3,00 m	56,0	41,1	31,1	19,0	10,7	2,1	-0,5
	4,00 m	56,1	41,2	31,2	19,1	10,7	2,2	-0,4
	Medido no Campo	68,6	41,0	26,4	11,4	4,1	2,0	1,9
38,8 (kN)	1,00 m	74,0	52,9	39,1	22,9	12,0	1,3	-1,4
	2,00 m	75,3	54,2	40,3	24,1	13,1	2,2	-0,9
	3,00 m	75,7	54,6	40,7	24,5	13,5	2,5	-0,6
	4,00 m	75,8	54,7	40,9	24,6	13,6	2,6	-0,5
	Medido no Campo	91,6	57,4	38,8	18,4	7,2	3,4	3,0
63,1 (kN)	1,00 m	112,4	78,1	56,5	32,3	16,4	1,3	-2,3
	2,00 m	114,3	80,0	58,5	34,2	18,2	2,7	-1,4
	3,00 m	114,8	80,6	59,1	34,8	18,8	3,2	-1,0
	4,00 m	115,1	80,8	59,3	35,0	19,0	3,5	-0,8
	Medido no Campo	128,3	84,1	59,3	30,5	13,4	6,5	5,4
Deflexões da Viga Benkelman (0,01 mm)								
Subleito de 1,00 m	Subleito de 2,00 m	Subleito de 3,00 m	Subleito de 4,00 m	Medido no Campo				
78,1	79,3	79,6	79,8	107				

Além disso, os valores obtidos para o ponto mais afastado do carregamento (120 cm em relação à origem) não correspondem à realidade de um pavimento, podendo ser considerado uma deficiência do modelo axissimétrico para pavimento, que se preocupa com os pontos próximos ao carregamento. A figura 31 mostra o gráfico das deflexões calculadas pelo programa, considerando espessura do subleito de 4,00 m, e medidas em campo.

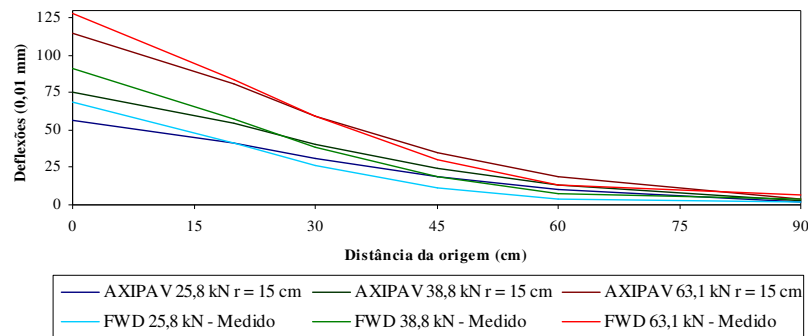


FIG. 31 – Deflexões da seção B

TAB. 16 – Resultados obtidos na análise da Seção C

Carga FWD	Espessura Subleito	Deflexões FWD (0,01 mm)						
		D ₀	D ₂₀	D ₃₀	D ₄₅	D ₆₀	D ₉₀	D ₁₂₀
25,8 (kN)	1,00 m	31,9	21,3	14,9	8,1	3,9	1,4	-0,6
	2,00 m	33,0	22,4	16,1	9,2	4,8	0,8	-0,2
	3,00 m	33,3	22,7	16,4	9,5	5,1	1,0	0,0
	4,00 m	33,4	22,8	16,4	9,6	5,2	1,1	0,0
	Medido no Campo	36,1	21,8	14,7	7,6	4,4	3,1	2,6
38,8 (kN)	1,00 m	43,4	28,8	20,1	10,9	5,2	0,2	-0,9
	2,00 m	45,0	30,4	21,8	12,5	6,6	1,2	-0,3
	3,00 m	45,4	30,8	22,2	12,9	7,1	1,5	0,0
	4,00 m	45,5	31,0	22,4	13,1	7,2	1,7	0,1
	Medido no Campo	50,3	31,7	22,2	12,3	7,1	4,9	4,1
63,1 (kN)	1,00 m	70,8	47,0	32,9	17,7	8,4	0,2	-1,5
	2,00 m	73,0	49,5	35,4	20,3	10,7	1,8	-0,5
	3,00 m	73,6	50,1	36,1	21,0	11,4	2,4	-0,1
	4,00 m	73,8	50,4	36,4	21,2	11,7	2,6	0,2
	Medido no Campo	75,8	50,0	36,5	21,7	12,9	8,7	7,1
Deflexões da Viga Benkelman (0,01 mm)								
Subleito de 1,00 m	Subleito de 2,00 m	Subleito de 3,00 m	Subleito de 4,00 m	Medido no Campo				
43,0	44,6	44,9	45,2	55				

Da análise comparativa dos resultados mostrados na tabela 16, pode-se concluir que a espessura do subleito considerada na análise provocou influência nos resultados obtidos pelo programa, em especial, quando foi adotada espessura de 1,00 m. Houve, também, excelente convergência com os deslocamentos medidos em campo, exceto para três valores da tabela (afastamento de 90 cm em relação à origem para todas as cargas em questão) e obteve-se um fator de amplificação dinâmica médio de 0,98.

Além disso, os valores obtidos para o ponto mais afastado do carregamento (120 cm em relação à origem), em sua grande maioria, não correspondem à realidade de um pavimento,

podendo ser considerado uma deficiência do modelo axissimétrico para pavimento, que se preocupa com os pontos próximos ao carregamento. A figura 32 mostra o gráfico das deflexões calculadas pelo programa, considerando espessura do subleito de 4,00 m, e medidas em campo.

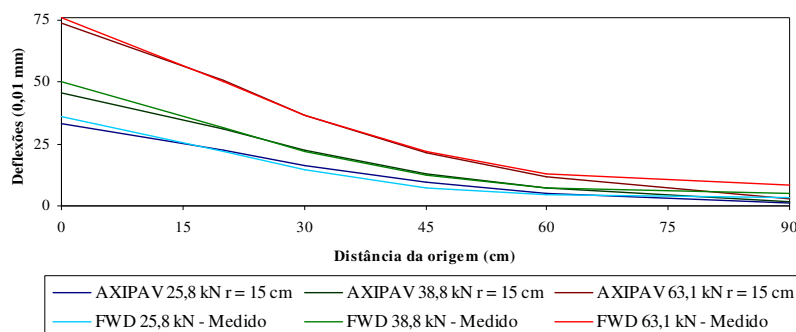


FIG. 32 – Deflexões da Seção C

TAB. 17 – Resultados obtidos na análise da Seção D

Carga FWD	Espessura Subleito	Deflexões FWD (0,01 mm)						
		D ₀	D ₂₀	D ₃₀	D ₄₅	D ₆₀	D ₉₀	D ₁₂₀
25,8 (kN)	1,00 m	68,7	53,6	42,8	28,5	17,2	4,0	-0,6
	2,00 m	68,8	53,7	42,9	28,6	17,3	4,1	-0,6
	3,00 m	68,9	53,7	42,9	28,6	17,3	4,1	-0,5
	4,00 m	68,9	53,7	43,0	28,6	17,3	4,1	-0,5
	Medido no Campo	57,3	39,5	26,8	13,3	5,6	2,9	2,3
38,8 (kN)	1,00 m	75,7	58,6	47,3	32,3	20,1	5,1	-0,6
	2,00 m	75,8	58,8	47,4	32,4	20,2	5,2	-0,5
	3,00 m	75,9	58,8	47,5	32,5	20,3	5,2	-0,5
	4,00 m	75,9	58,8	47,5	32,5	20,3	5,2	-0,5
	Medido no Campo	79,0	57,0	40,7	21,9	9,8	4,7	3,6
63,1 (kN)	1,00 m	87,3	66,7	54,4	38,5	25,0	7,1	-0,3
	2,00 m	87,5	66,9	54,6	38,7	25,2	7,2	-0,2
	3,00 m	87,5	66,9	54,7	38,8	25,3	7,3	-0,2
	4,00 m	87,5	66,9	54,7	38,8	25,3	7,3	-0,1
	Medido no Campo	114,6	86,0	64,6	37,6	18,2	8,6	6,2
Deflexões da Viga Benkelman (0,01 mm)								
Subleito de 1,00 m	Subleito de 2,00 m	Subleito de 3,00 m	Subleito de 4,00 m	Medido no Campo				
104,1	104,2	104,2	104,3	127				

Da análise comparativa dos resultados mostrados na tabela 17, pode-se concluir que a espessura do subleito considerada na análise não provocou influência significativa nos resultados obtidos pelo programa. Houve, também, coerência com os deslocamentos medidos em campo. Entretanto, houve uma grande divergência destes valores, nos pontos afastados do

centro do carregamento (20 cm, 30 cm, 45 cm, 60 cm e 90 cm em relação à origem do sistema), quando submetidos ao primeiro nível de carga (25,8 kN), obtendo-se para este caso um fator de amplificação dinâmica médio de 0,61.

Além disso, houve discrepância em outros três valores da tabela (afastamento de 45 cm e 60 cm em relação à origem para a carga de 38,8 kN; e afastamento de 60 cm em relação à origem, para a carga de 63,1 kN) e obteve-se, para o segundo nível de carga (38,8 kN) e para o terceiro nível de carga (63,1 kN), fatores de amplificação dinâmica médios de 0,89 e de 1,18, respectivamente.

Concomitantemente, os valores obtidos para o ponto mais afastado do carregamento (120 cm em relação à origem) não correspondem à realidade de um pavimento, podendo ser considerado uma deficiência do modelo axissimétrico para pavimento, que se preocupa com os pontos próximos ao carregamento. A figura 33 mostra o gráfico das deflexões calculadas pelo programa, considerando espessura do subleito de 4,00 m, e medidas em campo.

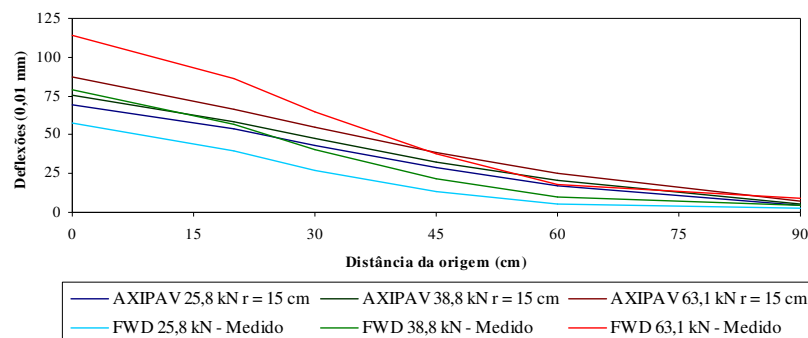


FIG. 33 – Deflexões da Seção D

Da análise comparativa dos resultados mostrados na tabela 18, pode-se concluir que a espessura do subleito considerada na análise não provocou influência nos resultados obtidos pelo programa. Houve, também, significativa discrepância com os deslocamentos medidos em campo, obtendo-se um fator de amplificação dinâmica médio de 1,90.

Além disso, os valores obtidos para os pontos mais afastados do carregamento (90 cm e 120 cm em relação à origem) não correspondem à realidade de um pavimento, podendo ser considerado uma deficiência do modelo axissimétrico para pavimento, que se preocupa com

os pontos próximos ao carregamento. A figura 34 mostra o gráfico das deflexões calculadas pelo programa, considerando espessura do subleito de 4,00 m, e medidas em campo.

TAB. 18 – Resultados obtidos na análise da Seção E

Carga FWD	Espessura Subleito	Deflexões FWD (0,01 mm)						
		D ₀	D ₂₀	D ₃₀	D ₄₅	D ₆₀	D ₉₀	D ₁₂₀
25,8 (kN)	1,00 m	25,5	16,7	11,5	5,8	2,4	-0,3	-0,5
	2,00 m	25,6	16,8	11,6	5,9	2,4	-0,2	-0,4
	3,00 m	25,6	16,8	11,6	5,9	2,5	-0,2	-0,4
	4,00 m	25,6	16,8	11,6	5,9	2,5	-0,2	-0,4
	Medido no Campo	37,4	24,8	17,9	10,2	5,6	3,3	2,6
38,8 (kN)	1,00 m	30,9	20,5	14,4	7,5	3,2	-0,3	-0,6
	2,00 m	31,0	20,5	14,5	7,6	3,3	-0,2	-0,5
	3,00 m	31,0	20,5	14,5	7,6	3,3	-0,1	-0,5
	4,00 m	31,0	20,6	14,5	7,7	3,3	-0,1	-0,5
	Medido no Campo	49,4	34,2	25,5	15,4	9,0	5,4	4,2
63,1 (kN)	1,00 m	41,0	28,2	20,7	11,6	5,3	-0,1	-0,8
	2,00 m	41,1	28,2	20,8	11,7	5,5	0,1	-0,7
	3,00 m	41,1	28,3	20,9	11,8	5,5	0,1	-0,7
	4,00 m	41,1	28,3	20,9	11,8	5,6	0,1	-0,6
	Medido no Campo	69,7	50,3	38,8	25,2	15,7	9,7	7,3
Deflexões da Viga Benkelman (0,01 mm)								
Subleito de 1,00 m	Subleito de 2,00 m	Subleito de 3,00 m	Subleito de 4,00 m	Medido no Campo				
35,4	35,5	35,5	35,5	79				

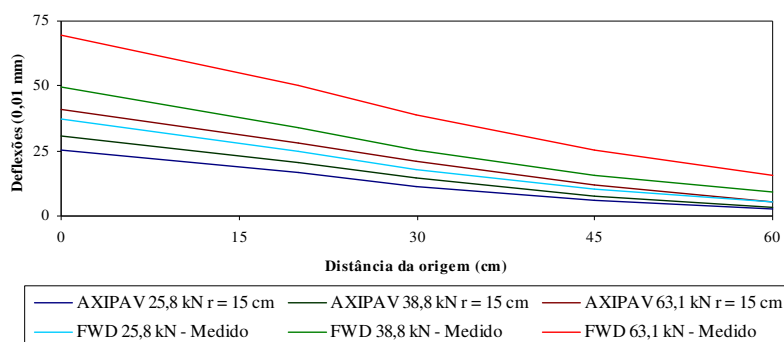


FIG. 34 – Deflexões da Seção E

Após a comparação dos resultados para essas cinco seções, realizou-se a análise de sensibilidade dos coeficientes dos módulos de resiliência das camadas da Seção C (seção que obteve os valores calculados mais próximos dos medidos em campo) adotando-se subleito com 4,00 m de espessura. A tabela 19 apresenta os valores dos deslocamentos obtidos para variações de 10% para cada coeficiente do módulo de resiliência de cada camada e a tabela 20, o acréscimo (ou decréscimo) percentual dos deslocamentos para cada caso.

TAB. 19 – Deslocamentos obtidos na análise de sensibilidade

Variação do Módulo de Resiliência		Deflexão obtida pelo programa AXIPAV (0,01 mm)						
Camada	Coefficiente	D ₀	D ₂₀	D ₃₀	D ₄₅	D ₆₀	D ₉₀	D ₁₂₀
Estrutura original		33,4	22,8	16,4	9,6	5,2	1,1	0,0
Revestimento	K ₁ + 10%	32,4	22,4	16,3	9,6	5,3	1,2	0,1
	K ₁ – 10%	34,6	23,3	16,7	9,6	5,1	1,0	0,0
Base	K ₁ + 10%	32,3	22,0	15,8	9,2	5,0	1,1	0,1
	K ₁ – 10%	34,8	23,9	17,3	10,1	5,5	1,2	0,0
	K ₂ + 10%	39,4	27,7	20,4	12,4	7,1	1,7	0,0
	K ₂ – 10%	29,2	19,5	13,7	7,7	4,0	0,9	0,2
Subleito	K ₁ + 10%	32,8	22,3	16,1	9,3	5,0	1,1	0,0
	K ₁ – 10%	34,1	23,4	16,9	9,9	5,5	1,2	0,1
	K ₂ + 10%	32,7	22,3	16,0	9,2	5,0	1,0	0,0
	K ₂ – 10%	34,2	23,5	17,0	10,0	5,5	1,3	0,1
	K ₃ + 10%	33,1	22,5	16,2	9,4	5,1	1,1	0,0
	K ₃ – 10%	33,8	23,1	16,7	9,8	5,4	1,2	0,1
	K ₄ + 10%	33,4	22,8	16,5	9,6	5,2	1,1	0,0
	K ₄ – 10%	33,4	22,8	16,5	9,6	5,2	1,1	0,0

TAB. 20 – Variação dos deslocamentos na análise de sensibilidade

Variação do Módulo de Resiliência		Variação da deflexão em relação à estrutura original (%)					
Camada	Coefficiente	D ₀	D ₂₀	D ₃₀	D ₄₅	D ₆₀	D ₉₀
Revestimento	K ₁ + 10%	-2,99	-1,75	-0,61	0,00	1,92	9,09
	K ₁ – 10%	3,59	2,19	1,83	0,00	-1,92	-9,09
Base	K ₁ + 10%	-3,29	-3,51	-3,66	-4,17	-3,85	0,00
	K ₁ – 10%	4,19	4,82	5,49	5,21	5,77	9,09
	K ₂ + 10%	17,96	21,49	24,39	29,17	36,54	54,55
	K ₂ – 10%	-12,57	-14,47	-16,46	-19,79	-23,08	-18,18
Subleito	K ₁ + 10%	-1,80	-2,19	-1,83	-3,12	-3,85	0,00
	K ₁ – 10%	2,10	2,63	3,05	3,13	5,77	9,09
	K ₂ + 10%	-2,10	-2,19	-2,44	-4,17	-3,85	-9,09
	K ₂ – 10%	2,40	3,07	3,66	4,17	5,77	18,18
	K ₃ + 10%	-0,90	-1,32	-1,22	-2,08	-1,92	0,00
	K ₃ – 10%	1,20	1,32	1,83	2,08	3,85	9,09
	K ₄ + 10%	0,00	0,00	0,61	0,00	0,00	0,00
	K ₄ – 10%	0,00	0,00	0,61	0,00	0,00	0,00

Da análise dos resultados da tabela 20, pode-se concluir que na Seção C, o módulo de resiliência da base (do tipo $M_R = K_1 \sigma_3^{K_2}$) tem influência significativa na obtenção das deflexões da estrutura, em especial quando se varia o coeficiente K_2 . Pode-se concluir, ainda, que o módulo de resiliência do revestimento (do tipo $M_R = K_1$) e do subleito (do tipo

$M_R = K_2 + K_3(K_1 - \sigma_d)$, para $\sigma_d < K_1$; e $M_R = K_2 + K_4(\sigma_d - K_1)$, para $\sigma_d > K_1$) não influenciam significativamente os resultados obtidos pelo programa, quando se varia algum de seus coeficientes, exceto para os coeficientes K_1 e K_2 do subleito.

O capítulo seguinte abordará as conclusões deste trabalho e sugestões e recomendações para trabalhos futuros.

6 CONCLUSÕES E RECOMENDAÇÕES

Este capítulo tem por objetivo apresentar as conclusões deste trabalho, além de recomendações e sugestões para estudos futuros.

6.1 CONCLUSÕES

Inicialmente, apresentou-se a evolução dos métodos de cálculo de tensões e deformações em pavimentos. Posteriormente, foram descritos, para o método dos elementos finitos, sua formulação, suas características e os tipos que podem ser utilizados para pavimentos.

Em seguida, foram apresentadas as definições, a classificação e as propriedades de um pavimento, para posterior modelagem em elementos axissimétricos do método dos elementos finitos. Após essa modelagem, foi implementado, na linguagem de programação C++, um programa de análise pavimentos, intitulado de *AXIPAV*.

Foi realizada a validação do programa desenvolvido através de comparação com estruturas de pavimento analisadas por Preussler (1983) *apud* Silva (1995) e por Pinto (1991), em suas respectivas teses de doutorado. Entretanto, os problemas não-lineares (módulos de resiliência variáveis com o estado de tensões) não convergiram utilizando a primeira modelagem proposta no Capítulo 3, então utilizou-se o método incremental (análise com incrementos de carga) com quatro iterações (incrementos de um quarto da força).

Para o pavimento analisado por Preussler, pôde-se concluir que:

- os valores da deflexão obtidos com o programa desenvolvido nesta dissertação foram próximos aos medidos em campo (10% a menos para a análise linear e 10% a mais para a não-linear);
- os demais valores (tensão vertical no topo do subleito e deformação específica de tração na fibra inferior do revestimento) obtidos com este programa foram coerentes com os obtidos, através do programa FEPAVE, por Preussler, com exceção da

tensão vertical no topo do subleito para o segundo caso. Entretanto, não se pode afirmar qual seria o valor correto, pois o mesmo não foi medido no campo.

Para os pavimentos analisados por Pinto (Seções A, B, C, D e E) e para cada nível carregamento (no total de quatro carregamentos diferentes), utilizou-se quatro espessuras de subleito (1,00 m, 2,00 m, 3,00 m e 4,00 m) para realizar a análise pelo programa proposto e concluiu-se que:

- para as Seções D e E, a variação da espessura do subleito não ocasionou alteração nos resultados obtidos pelo programa, pois as mesmas possuíam grande espessuras de pavimentos (soma das espessuras do revestimento, da base, da sub-base, e do reforço do subleito) (76 cm e 73 cm, respectivamente);
- para as demais Seções (A, B e C), a espessura do subleito provocou influência nos resultados obtidos pelo programa, em especial, quando foi adotada espessura de 1,00 m, pois as mesmas possuíam pequenas espessuras de pavimento (44 cm, 50 cm e 30 cm, respectivamente);
- para as Seções B e C e para o segundo e terceiro níveis de carga da Seção D, os resultados se aproximaram dos deslocamentos medidos em campo, obtendo-se fatores de amplificação dinâmica (razão entre os deslocamentos dinâmicos e os deslocamentos estáticos) médios de 0,85, 0,98, 0,89 e 1,18, respectivamente;
- para as Seções A e E e para o primeiro nível de carga da Seção D, ocorreu discrepância entre os resultados obtidos e os deslocamentos medidos em campo, obtendo-se fatores de amplificação dinâmica médio de 0,70, 1,90 e 0,61, respectivamente. Entretanto, estes fatores se encontram dentro da faixa prevista segundo Craig Jr. (1981) (de 0 a 2);
- para todas as Seções, os valores obtidos para o ponto mais afastado do carregamento (120 cm em relação à origem) não correspondem à realidade de um pavimento, podendo ser considerado uma deficiência do modelo axissimétrico para estruturas estratificadas, que se preocupa com os pontos próximos ao carregamento.

Além disso, ao se realizar a análise de sensibilidade nos coeficientes dos módulos de resiliência das camadas da Seção C, concluiu-se que as deflexões obtidas pelo programa são influenciadas significativamente pelo módulo de resiliência da base, em especial quando se varia o coeficiente K_2 , e as demais camadas (revestimento e subleito) não alteram

expressivamente os valores destas deflexões, com exceção dos coeficientes K_1 e K_2 do subleito.

6.2 RECOMENDAÇÕES E SUGESTÕES

Na atual versão do programa, recomenda-se que o usuário, para analisar pavimentos com comportamento elástico-linear (materiais com módulos de resiliência constantes), utilize o método 1 (baseado na modelagem proposta no Capítulo 3), e, para analisar pavimentos com comportamento elástico não-linear (materiais com módulos de resiliência dependentes do estado de tensões), utilize o método 2 (método incremental) com quatro iterações (incrementos de um quarto da força), devido à não convergência do método 1 para este caso.

Como sugestões para trabalhos futuros, sugerem-se as descritas a seguir:

- realizar a reestruturação de alguns métodos (rotinas), principalmente o responsável pelo cálculo dos deslocamentos, de modo a propiciar maior rapidez na execução do programa, pois o mesmo demora cerca de quinze minutos para realizar os cálculos necessários para cada iteração;
- estudar a influência do módulo de resiliência inicial de cada camada no processo de convergência da análise, pelo método 1, de pavimentos com materiais de comportamento elástico não-linear, bem como propor valores iniciais de módulo de resiliência para cada tipo de material;
- executar pistas experimentais instrumentadas para medição de tensões e deformações em pontos do pavimento, devido a uma carga conhecida, e realizar ensaios de determinação do módulo de resiliência em amostras indeformadas das camadas do pavimento, de modo a facilitar o processo de validação de novos programas de análise de pavimentos;
- modelar e implementar métodos para a realização de análises dinâmicas do pavimento considerando carregamento senoidal;
- implementar rotinas de interface com o usuário, de modo a facilitar a entrada de dados e a análise dos resultados pelo usuário.

REFERÊNCIAS BIBLIOGRÁFICAS

- AEDO, J. L. C. **Programa para Análise Tridimensional de Pavimentos Asfálticos.** Dissertação de Mestrado. PUC-Rio, Rio de Janeiro, Brasil, 1997.
- ARANOVICH, L. A. S. **Desempenho de Pavimentos de Baixo Custo no Estado do Paraná.** Dissertação de Mestrado. Universidade Federal do Rio de Janeiro, Rio de Janeiro, 1985.
- ARNAUT. **Entendendo C++.** Disponível em <http://www.arnaut.eti.br/op/CPPAI01.htm>. Capturado em 19/06/2007.
- ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS, ABNT. **NBR 7207 – Terminologia e classificação de pavimentação.** Rio de Janeiro, 1982.
- ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS, ABNT. **NBR ISO/IEC 12207 – Tecnologia de informação: processos de ciclo de vida de software.** Rio de Janeiro, 1998.
- BARATA, F. E. **Propriedades Mecânicas dos Solos: Uma Introdução ao Projeto de Fundações.** 1. ed. Rio de Janeiro: Ltc - Livros Técnicos e Científicos S.A., 1984.
- BARRETO, M. E. **Programação Orientada a Objetos em C++.** Curso de Extensão do Centro Universitário La Salle, Canoas-RS, 2004.
- BARRETO, A. O. S. **Apoio à Verificação de Software em Ambientes de Desenvolvimento de Software Orientados à Organização.** Dissertação de Mestrado. Universidade Federal do Rio de Janeiro, Rio de Janeiro, 2006.
- BATHE, K. J. **Finite Element Procedures.** 1. ed. New York, USA: Prentice Hall, 1996.

- BLAB, R. e HARVEY, J. T. **Modeling Measured 3D Tire Contact Stresses in a Viscoelastic FE Pavement Model.** Preproceedings of the Second International Symposium on 3D Finite Element for Pavement Analysis, Design, and Research, pp123-148, 2000.
- BORGES, C. B. S. **Estudo Comparativo entre Medidas de Deflexão com Viga Benkelman e FWD em Pavimentos da Malha Rodoviária Estadual de Santa Catarina.** Dissertação de Mestrado. Universidade Federal de Santa Catarina, Florianópolis, 2001.
- CARDOSO, S. H. **Faixas de Módulos Dinâmicos (Elásticos) Obtidos por Retroanálises durante Sete Anos.** 29ª Reunião Anual de Pavimentação, Cuiabá, 1999.
- CHO, Y. H.; MCCULLOUGH, B. F. e WEISSMAN J. **Considerations on Finite Element Method Application in Pavement Structural Analysis.** TRB 1996 Annual Meeting CD-ROM. p. 96-101, 1996.
- COOK, R. D.; MALKUS, D. S. e PLESHA, M. E. **Concepts and Applications of Finite Element Analysis.** 3. ed. New York, USA: John Wiley & Sons, 1989.
- COUTO JR., M. A., VIRTUOSO, G. H. F. e MARTINS, P. J. **Propriedades Desejáveis a uma Linguagem de Programação: Uma Análise Comparativa entre as Linguagens C, C++ e Java.** I Congresso Sul Catarinense de Computação, Criciúma, 2005.
- CRAIG JR, R. R. **Structural Dynamics – An Introduction to Computer Methods.** 1. ed. New York: John Wiley & Sons, Inc., 1981.
- CUNAGIN, W. D. e GRUBBS, A. B. **Automated Acquisition of Truck Tire Pressure Data.** Transportation Research Record, nº 1322, pp. 112 – 121, 1991.
- DEITEL, H. M. e DEITEL, P. J. **C++: Como Programar.** Carlos Arthur Lang Lisboa e Maria Lúcia Lang Lisboa. 3. ed. Porto Alegre: Bookman, 2001.

- DEPARTAMENTO NACIONAL DE ESTRADAS DE RODAGEM. **DNER-ME 024/94 – Pavimento – determinação das deflexões pela Viga Benkelman.** Rio de Janeiro, 1994.
- DEPARTAMENTO NACIONAL DE ESTRADAS DE RODAGEM. **DNER-ME 131/94 – Solos - determinação do módulo de resiliência.** Rio de Janeiro, 1994.
- DEPARTAMENTO NACIONAL DE ESTRADAS DE RODAGEM. **DNER-ME 133/94 – Misturas betuminosas - determinação do módulo de resiliência.** Rio de Janeiro, 1994.
- DEPARTAMENTO NACIONAL DE ESTRADAS DE RODAGEM. **DNER-PRO 273/96 – Determinação de deflexões utilizando o deflectômetro de impacto tipo “Falling Weight Deflectometer (FWD)”.** Rio de Janeiro, 1996.
- DUNCAN, J. M., MONISMITH, C. L. e WILSON, E. L. **Finite Element Analysis of Pavements.** Highway Research Record - TRB, n. 228, pp. 18-33, 1968.
- EVANGELISTA JR, F.; SOARES, J. B. e SILVA, E. R. A. **Modelagem Computacional de Pavimentos Asfálticos – Integração entre Interface Gráfica e Programas de Análise.** Submetido ao XVII Congresso de Pesquisa e Ensino em Transportes, ANPET. Rio de Janeiro, Brasil, 2003.
- FERREIRA, J. G. H. M. **Elaboração e Análise da Base de Dados de Ensaios Triaxiais Dinâmicos da COPPE/UFRJ.** Dissertação de Mestrado. Universidade Federal do Rio de Janeiro, Rio de Janeiro, 2002.
- FERREIRA, A. B. H. **Aurélio Século XXI, O Dicionário da Língua Portuguesa.** 3. ed. Curitiba: Positivo, 2004.
- FOLEY, J. D.; VAN DAM, A.; FEINER, S. K. e HUGHES, J. F. **Computer Graphics: Principle and Practice.** 2. ed. New York, USA: Addison-Wesley, 1996.

- FONSECA, T. A. **Curso de Programação em C++ - Introdução à Linguagem C++**. Disponível em <http://www.image.unb.br/~tiago/cursoCpp/aulaCpp1.pdf>. Capturado em 18/06/2007.
- FRANCO, F. A. C. P. **Um Sistema para Análise Mecânica de Pavimentos Asfálticos**. Dissertação de Mestrado. Universidade Federal do Rio de Janeiro, Rio de Janeiro, 2000.
- GUIMARÃES, L. S. G. **Disciplina de Programação Orientada a Objetos para Análise e Visualização Bidimensional de Modelos de Elementos Finitos**. Dissertação de Mestrado. PUC-Rio, Rio de Janeiro, Brasil, 1992.
- HUANG, Y. H. **Pavement Analysis and Design**. 1. ed. New Jersey, USA: Prentice Hall, 1993.
- HUBBARD, J. R. **Teoria e Problemas de Programação em C++**. Edson Furmankiewicz. 2. ed. Porto Alegre: Bookman, 2003.
- JOHANN, M. O. **Curso de Introdução à Programação em C++**. Apostila do Curso de Extensão da Universidade Federal do Rio Grande do Sul, Porto Alegre, 2004.
- MACÊDO, J. A. G. **Interpretação de Ensaio Defletométricos para Avaliação Estrutural de Pavimentos Flexíveis**. Tese de Doutorado. Universidade Federal do Rio de Janeiro, Rio de Janeiro, 1996.
- MEDINA, J. e MOTTA, L. M. G. **Mecânica dos Pavimentos**. 2. ed. Rio de Janeiro: COPPE, 2005.
- MESQUITA, R. C. **Programação Orientada a Objetos em C++**. Disponível em: http://www.bax.com.br/Bax/orientacao/alunos/PSI/Programacao_OO1.pdf. Capturado em 19/06/2007.

- MICELI JR., G. **Comportamento de solos do Estado do Rio de Janeiro estabilizados com emulsão asfáltica**. Dissertação de Mestrado. Instituto Militar de Engenharia, Rio de Janeiro, 2006.
- MOTTA, L. M. G. **Método de Dimensionamento de Pavimentos Flexíveis; Critério da Confiabilidade e Ensaio de Cargas Repetidas**. Tese de Doutorado. UFRJ, Rio de Janeiro, Brasil, 1991.
- PFLEEGER, S. L. **Engenharia de Software – Teoria e Prática**. 2. ed. São Paulo: Prentice Hall, 2004.
- PINTO, S. **Estudo do Comportamento à Fadiga de Misturas Betuminosas e Aplicação na Avaliação Estrutural de Pavimentos**. Tese de Doutorado. Universidade Federal do Rio de Janeiro, Rio de Janeiro, 1991.
- PINTO, S. e PREUSSLER, E. **Pavimentação Rodoviária: Conceitos Fundamentais sobre Pavimentos Flexíveis**. 2. ed. Rio de Janeiro: Copiadora e Artes Gráficas Ltda., 2002.
- PREUSSLER, E. S. **Estudo da deformação resiliente de pavimentos flexíveis e aplicação ao projeto de camadas de reforço**. Tese de Doutorado. Universidade Federal do Rio de Janeiro, Rio de Janeiro, 1983.
- REDE ASFALTO. Disponível em www.redeasfalto.org.br/sistema/relatorios/PC03_N_08-05-2006_03_55_UFC.doc. Capturado em 16/10/2006.
- ROCHA, A. R. C., MALDONADO, J. C. e WEBER, K. C. **Qualidade de Software – Teoria e Prática**. 1. ed. São Paulo: Prentice Hall, 2001.
- SILVA, P. D. E. A. **Contribuição para o Aperfeiçoamento do Emprego do Programa FEPAVE2 em Estudos e Projetos de Pavimentos Flexíveis**. Dissertação de Mestrado. Universidade Federal do Rio de Janeiro, Rio de Janeiro, 1995.

- SILVA, B. A. **Aplicação das metodologias MCT e resiliente a solos finos do centro-norte do Mato Grosso**. Dissertação de Mestrado. Instituto Militar de Engenharia, Rio de Janeiro, 2003.
- SORIANO, H. L. **Método de Elementos Finitos em Análise de Estruturas**. 1. ed. São Paulo: Universidade de São Paulo, 2003.
- SVENSON, M. **Ensaio Triaxiais Dinâmicos de Solos Argilosos**. Dissertação de Mestrado. Universidade Federal do Rio de Janeiro, Rio de Janeiro, 1980.
- TIELKING, J. T. e ABRAHAM, M. A. **Measurement of Truck Tire Footprint Pressures**. Transportation Research Record, nº 1435 pp. 92-99, 1995.
- TRICHÊS, G. **Determinação do Coeficiente de Poisson de Solos Compactados no Ensaio Triaxial Dinâmico e o Cálculo da Deflexão de Pavimentos**. Dissertação de Mestrado. Universidade Federal do Rio de Janeiro, Rio de Janeiro, 1985.
- TURNER, M. J.; CLOUGH, R. W.; MARTIN, H. C. e TOPP, L. J. **Stiffness and Deflection Analysis of Complex Structures**. Journal of Aeronautical Science, Vol. 23, pp. 805-823, 1956.
- UNIVERSITY OF WASHINGTON. **Poisson's Ratio**. Disponível em http://training.ce.washington.edu/WSDOT/Modules/06_structural_design/poissons_ratio.htm. Capturado em 31/05/2007.
- VIEIRA, L. C. L. M. **Estudo de Algoritmos de Integração Elemento por Elemento para Análise Dinâmica não Linear de Estruturas**. Dissertação de Mestrado. Universidade Federal de Alagoas, Maceió, 2004.
- VIEIRA, A. **Mecânica dos Pavimentos**. Notas de Aula. Engenharia de Transportes, IME, 2006.

WANG, J. **Three-Dimensional Finite Element Analysis of Flexible Pavements.**
Dissertação de Mestrado. University of Maine, Maine, USA, 2001.

WEISSMAN, S. L. **The Influence of Tire-Pavement Contact Stress Distribution on the Development of Distress Mechanisms in Pavements.** Paper nº 00510, proceeding of the 78th Annual Meeting of the Transportation Research Board, Washington, D.C., USA, 1997.

ZIENKIEWICZ, O. C. e TAYLOR, R. L. **The Finite Element Method.** 4. ed. London, UK: McGraw-Hill, 1988. Vol. 1.

ZIENKIEWICZ, O. C. e TAYLOR, R. L. **The Finite Element Method.** 5. ed. Woburn, MA, EUA: Butterworth-Heinemann, 2000. Vol. 1.

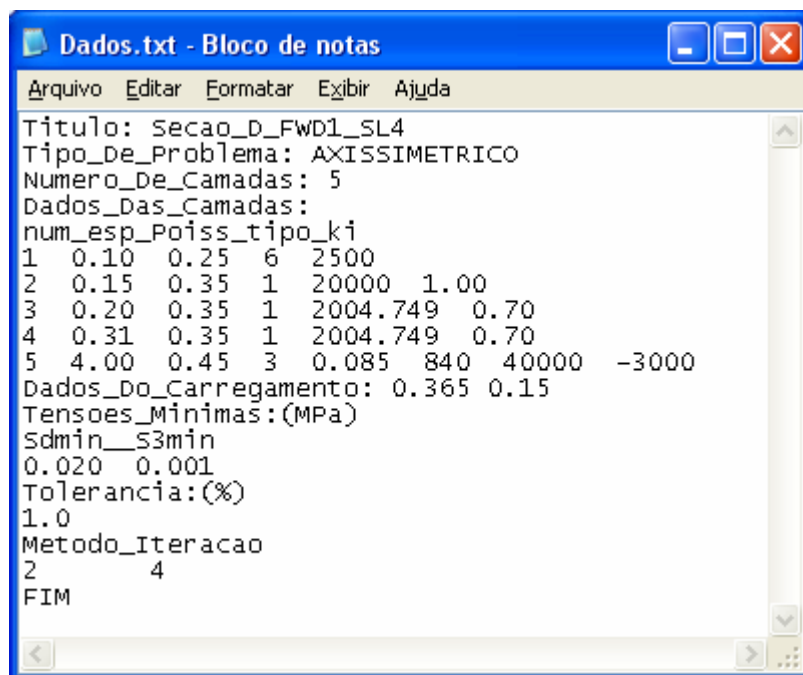
ZIENKIEWICZ, O. C. e TAYLOR, R. L. **The Finite Element Method for Solid and Structural Mechanics.** 6. ed. Oxford, UK: Elsevier Butterworth-Heinemann, 2005.

ZIENKIEWICZ, O. C., TAYLOR, R. L. e ZHU, J. Z. **The Finite Element Method – Its Basis and Fundamentals.** 6. ed. Oxford, UK: Elsevier Butterworth-Heinemann, 2005.

8 APÊNDICES

8.1 APÊNDICE A – Manual do Usuário

Para utilizar o programa *AXIPAV*, deve-se criar um arquivo de entrada de dados intitulado de *Dados.txt*, possuindo as seguintes informações: o título da estrutura a analisar, o tipo de problema, o número de camadas do pavimento, as propriedades de cada camada, os dados do carregamento, as tensões mínimas, a tolerância e o tipo de método de análise e o seu número de iterações. A figura 35 ilustra o formato deste arquivo.



```
Titulo: Secao_D_FWD1_SL4
Tipo_De_Problema: AXISSIMETRICO
Numero_De_Camadas: 5
Dados_Das_Camadas:
num_esp_Poiss_tipo_ki
1 0.10 0.25 6 2500
2 0.15 0.35 1 20000 1.00
3 0.20 0.35 1 2004.749 0.70
4 0.31 0.35 1 2004.749 0.70
5 4.00 0.45 3 0.085 840 40000 -3000
Dados_Do_Carregamento: 0.365 0.15
Tensoes_Minimas:(MPa)
sadmin_s3min
0.020 0.001
Tolerancia:(%)
1.0
Metodo_Iteracao
2 4
FIM
```

FIG. 35 – Exemplo de arquivo de entrada *Dados.txt*

Este arquivo deve estar no mesmo diretório que o programa (arquivo executável) e a entrada de dados se procede da seguinte maneira:

- O título da estrutura a analisar deve ser informado após o cabeçalho “Titulo:” separado por um espaço;
- O tipo de problema deve ser informado após o cabeçalho “Tipo_De_Problema:” separado por um espaço. Atualmente o programa só analisa problemas do tipo axissimétrico (AXISSIMETRICO);
- o número de camadas do pavimento deve ser informada após o cabeçalho “Numero_De_Camadas:” separado por um espaço;

- As propriedades de cada camada devem ser informadas depois do cabeçalho “Dados_Das_Camadas:” e embaixo do cabeçalho “num_esp_Poiss_tipo_ki”, que corresponde ao tipo de dado a ser informado. Os dados das camadas devem ser informados em linhas diferentes, uma linha para cada camada, e para cada camada devem ser informadas, separadas por pelo menos um espaço, as seguintes propriedades: o número da camada, a espessura da camada (em metros), o coeficiente de Poisson, o tipo de material (conforme tabela 21) e os coeficientes do módulo de resiliência;
- Os dados do carregamento devem ser informados após o cabeçalho “Dados_Do_Carregamento:” separado por um espaço, sendo os mesmos, respectivamente, a pressão de contato (em MPa) e o raio do carregamento (em metros), ambos separados por um espaço;
- As tensões mínimas devem ser informadas depois do cabeçalho “Tensoes_Minimas:(MPa)” e embaixo do cabeçalho “Sdmin__S3min”. Estes valores, que devem estar separados por pelo menos um espaço, correspondem, respectivamente, aos valores mínimos, em MPa, da tensão-desvio (σ_d) e da tensão confinante (σ_3), utilizados no cálculo dos módulos de resiliência da estrutura;
- A tolerância deve ser informada embaixo do cabeçalho “Tolerancia:(%)” e corresponde à tolerância admitida da diferença entre os módulos de resiliência de cada elemento (anterior e atual);
- o tipo de método de análise e seu número de iterações devem ser informados embaixo do cabeçalho “Metodo_Iteracao”. O tipo de método de análise pode admitir o valor 1 para o método 1 (método proposto no capítulo 3) ou 2, para o método 2 (método incremental), respectivamente. O segundo valor informado corresponde ao número de iterações que serão realizadas antes da interferência do usuário para o método 1 e o número de iterações que serão realizadas no método 2;
- A última linha do arquivo deve ser o cabeçalho “FIM” de modo a possibilitar o encerramento da fase de leitura de dados do programa.

Após o término da fase de análise, o programa gera três arquivos de saída de resultados: Visualizacao.txt, Resultados.txt e Resumo.txt. Estes arquivos são responsáveis, respectivamente, pelas coordenadas dos nós, possibilitando a visualização da malha gerada;

pelos dados da malha de elementos finitos, pelos deslocamentos nodais e pelas reações de apoio; e pelos resultados importantes no dimensionamento de pavimentos pelo método mecanístico.

TAB. 21 – Tipos de material

Tipo	Modelo	Equação	Material
1	Granular ou arenoso	$M_R = K_1 \sigma_3^{K_2}$	Granular ou arenoso
2	K - Θ	$M_R = K_1 \Theta^{K_2}$	Granular (dependente da soma das tensões principais)
3	Bi-linear	$M_R = K_2 + K_3(K_1 - \sigma_d)$, para $\sigma_d < K_1$ $M_R = K_2 + K_4(\sigma_d - K_1)$, para $\sigma_d > K_1$	Solos finos (mais de 50 % passando na peneira nº 200)
4	Argiloso	$M_R = K_1 \sigma_d^{K_2}$	Solos coesivos
5	Combinado	$M_R = K_2 + K_3(K_1 - \sigma_d)\sigma_3^{K_5}$, para $\sigma_d < K_1$ $M_R = K_2 + K_4(\sigma_d - K_1)\sigma_3^{K_5}$, para $\sigma_d > K_1$	Solos arenosos com bastante argila ou solos argilosos lateríticos
6	Elástico-linear	MR = K = constante	Misturas asfálticas, siltes com baixo módulo ou solos estabilizados com módulo elevado
7	Composto	$M_R = K_1 \sigma_3^{K_2} \sigma_d^{K_3}$	Todos os solos e britas em geral

A malha de elementos finitos é composta de elementos axissimétricos retangulares de oito nós, sendo gerada automaticamente pelo programa a partir das características da estrutura (raio do carregamento circular, número de camadas e espessura de cada camada), conforme descrito a seguir:

- inicialmente, considera-se apenas a primeira camada, para a determinação do tamanho das colunas de elementos da malha;
- esta camada é dividida transversalmente de duas formas diferentes. Na primeira, divide-se o comprimento referente à parte solicitada do pavimento em três colunas iguais. E na segunda, divide-se o restante da camada em colunas cujos tamanhos formam uma progressão geométrica com termo inicial igual ao tamanho das colunas anteriormente calculadas e razão igual a 1,15;
- a referida camada ainda é dividida longitudinalmente, de acordo com sua espessura. Se esta espessura não for maior do que dezoito centímetros, a camada é dividida em três fileiras de tamanhos iguais, senão, ela é dividida em fileiras cujos tamanhos

formam uma progressão geométrica com termo inicial igual a seis centímetros e razão igual a 1,15;

- feito isso, modificam-se as colunas da malha que contenham os elementos cuja razão comprimento / largura (tamanho da coluna / tamanho da fileira) seja maior do que 5 (razão máxima proposta por Duncan et al. (1968), apud Silva (1995)). Em seguida, modificam-se as fileiras da malha cujo tamanho seja maior do que cinco vezes o menor tamanho das colunas da malha;
- após a definição das colunas da malha, divide-se longitudinalmente cada camada em três fileiras iguais, para as camadas de espessura menor ou igual a dezoito centímetros, ou em fileiras cujos tamanhos formam uma progressão geométrica com termo inicial igual a seis centímetros e razão igual a 1,15, para as camadas de espessura superior a dezoito centímetros. Modificando-se as fileiras cujo tamanho seja maior do que cinco vezes o menor tamanho das colunas da malha;
- Com a determinação do tamanho das colunas e fileiras da malha, determinam-se as coordenadas dos nós referentes ao ponto médio de cada aresta dos elementos. E, em seguida, armazenam-se as coordenadas de todos os nós da estrutura, bem como as informações de cada elemento do sistema.

A figura 36 detalha este processo de geração de malha.

No arquivo Resultados.txt pode-se realizar uma conferência dos dados de entrada, pois estes dados são transcritos nas primeiras linhas deste arquivo.

Para analisar um pavimento pelo programa *AXIPAV*, deve-se realizar as seguintes etapas:

- crie ou altere o arquivo Dados.txt, de acordo com as propriedades do pavimento;
- execute o arquivo AXIPAV.exe. Quando o programa terminar a análise deste pavimento aparecerá a mensagem “Pressione qualquer tecla para continuar. . .”;
- abra e analise os arquivos de saída de resultados (Visualizacao.txt, Resultados.txt e Resumo.txt) contidos no diretório do programa.

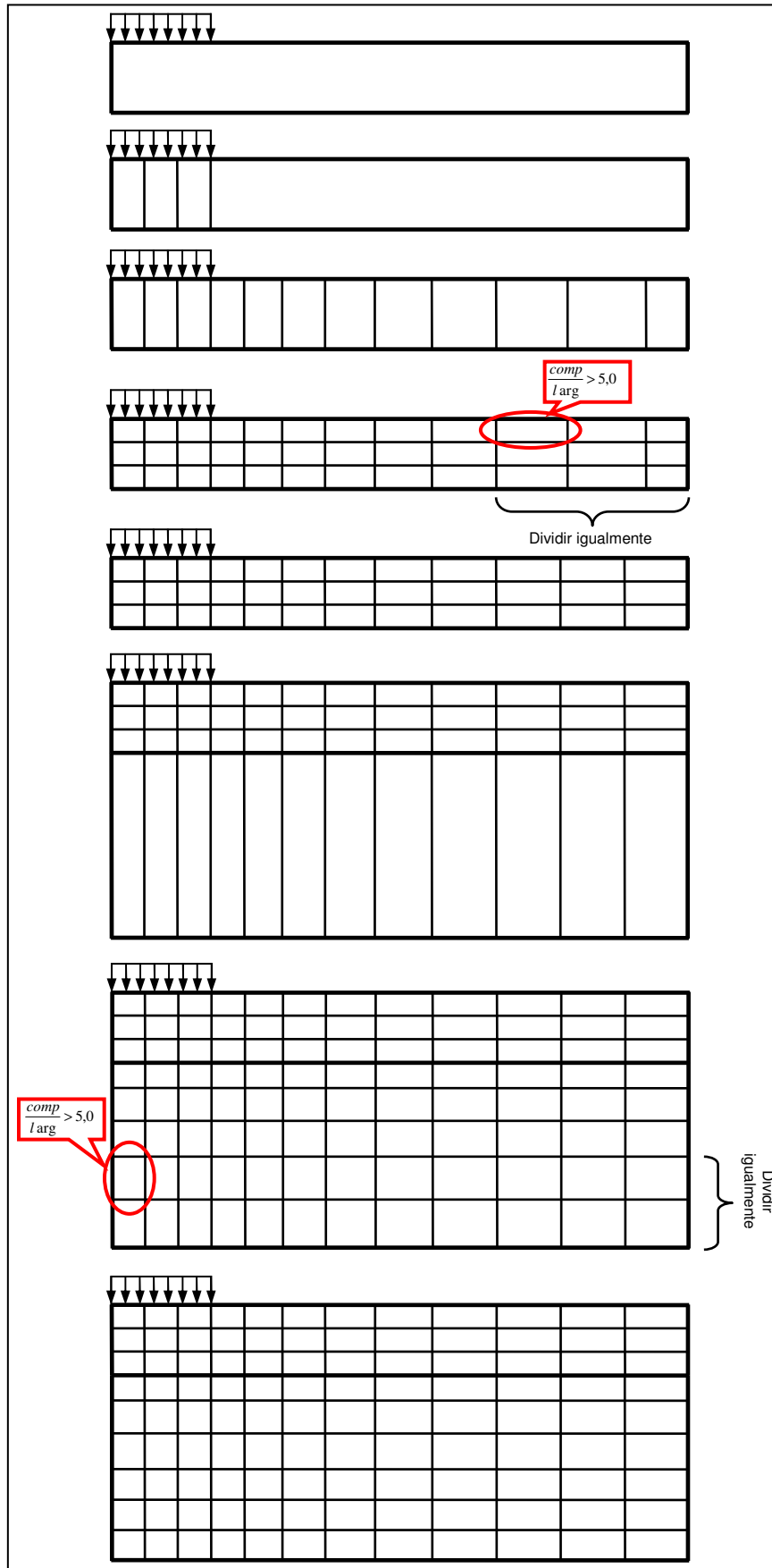


FIG. 36 – Processo de geração automática de malha

8.2 APÊNDICE B – Código-Fonte do Modelo

8.2.1 CLASSE MATRIZ

```
#include<math.h>

class matriz
{
public:
//////////
// Atributos da Classe ///
//////////
int num_linhas, num_colunas; // Número de Linhas e de Colunas da matriz
double *elementos; // Armazena os elementos da matriz
//////////
// Métodos da Classe ///
//////////
matriz(); // Construtor Default
~matriz(); // Destrutor da Classe
matriz(int m, int n = 1); // Construtor com Dimensões da Matriz
double elemento(int i, int j = 1) const; // retorna um elemento da matriz
void Atribui(int i, int j, double valor); // Atribui valor a um elemento da matriz;
void Atribui(int i, double valor); // Atribui valor a um elemento da matriz;
void Adiciona(int i, int j, double valor); // Adiciona valor a um elemento da matriz;
void Adiciona(int i, double valor); // Adiciona valor a um elemento da matriz;
void redim(int m, int n = 1); // Redimensiona a Matriz
double Determinante(); // Retorna o Determinante da Matriz
bool Singular(); // determina se uma matriz é singular
bool Simetrica(); // determina se uma matriz é simétrica
bool AntiSimetrica(); // determina se uma matriz é antissimétrica
bool Quadrada(); // determina se uma matriz é quadrada
matriz Inversa(); // Obtém a matriz Inversa
matriz Transposta(); // Obtém a matriz Transposta
void ZeraMatriz(); // Zera os elementos da Matriz
void TransFormaEmIdentidade(); // Transforma a Matriz em uma matriz Identidade.
void Limpa();
//////////
// Operadores Sobrecarregados ///
//////////
matriz operator = (double valor); // Operador de Atribuição.
matriz operator = (matriz B); // Operador de Atribuição.
matriz operator ^(int valor); // Operador de Exponenciação.
void operator ^=(int valor); // Operador de Exponenciação.
matriz operator +(double valor); // Operador de Adição.
void operator +=(double valor); // Operador de Adição.
matriz operator -(double valor); // Operador de Subtração.
void operator -=(double valor); // Operador de Subtração.
matriz operator +(matriz B);
void operator +=(matriz B);
matriz operator -(matriz B);
void operator -=(matriz B);
matriz operator *(matriz B);
void operator *=(matriz B);
matriz operator *(double fator);
void operator *=(double fator);
matriz operator /(matriz A);
```

```

void operator /=(matriz A);
matriz operator /(double fator);
void operator /=(double fator);
matriz operator -();
matriz operator ++();
matriz operator ++(int);
matriz operator --();
matriz operator --(int);
bool operator ==(matriz B);
bool operator !=(matriz B);
matriz operator !(); // Transposta da matriz
matriz operator ~(); // Inversa da matriz
double& operator()(int i, int j = 1);
double& operator()(double i, double j = 1);
void operator()(int i, double valor);
void operator()(int i, int j, double valor);
////////////////////
// Implementação realizada em 12/09/99
// Faltam Autovalores e Autovetores
////////////////////
void imprime();
};

```

//-----

```

#include "M.h"
#include "global.h"
#include<iostream>

matriz::matriz()
{
    // Construtor default da Classe
    num_linhas = 1;
    num_colunas = 1;
    elementos = new double[2];
}

matriz::~matriz()
{
    //Destrutor da classe
}

matriz::matriz(int m, int n)
{
    // Construtor da Classe
    elementos = new double[m*n+1];
    num_linhas = m;
    num_colunas = n;
    for (int i=1;i<=num_linhas;i++)
    {
        for (int j=1;j<=num_colunas;j++)
        {
            Atribui(i,j,0.0);
        }
    }
}

```

```

double matriz::elemento(int i, int j) const
{
    // Obtem o valor armazenado em um elemento da Matriz
    return elementos[(i-1)*num_colunas+j];
}

void matriz::Limpa()
{
    // Limpa o valor armazenado em um elemento da Matriz
    delete []elementos;
}

void matriz::Atribui(int i, int j, double valor)
{
    // Atribui um valor a um elemento da matriz
    elementos[(i-1)*num_colunas+j]= valor;
}

void matriz::Atribui(int i, double valor)
{
    // Atribui um valor a um elemento da matriz
    num_colunas = 1;
    elementos[(i-1)*num_colunas+1]= valor;
}

void matriz::Adiciona(int i, int j, double valor)
{
    // Adiciona um valor a um elemento da matriz
    elementos[(i-1)*num_colunas+j]+= valor;
}

void matriz::Adiciona(int i, double valor)
{
    // Adiciona um valor a um elemento da matriz
    num_colunas = 1;
    elementos[(i-1)*num_colunas+1]+= valor;
}

void matriz::redim(int m, int n)
{
    // Redimensiona a Matriz
    // Os valores anteriormente armazenados na Matriz são perdidos
    delete [] elementos;
    elementos = new double[m*n+1];
    num_linhas = m;
    num_colunas = n;
    for (int i=1;i<=num_linhas;i++)
    {
        for (int j=1;j<=num_colunas;j++)
        {
            Atribui(i,j,0.0);
        }
    }
}

double matriz::Determinante()
{

```

```

// Retorna o Determinante da matriz
double det, C, X, P;
int L;
matriz A(num_linhas, num_colunas);
for (int i=1;i<=num_linhas;i++)
{
    for (int j=1;j<=num_linhas;j++)
    {
        A.Atribui(i,j,elemento(i,j));
    }
}
det = 1.0;
int i = 1, N = num_linhas;
while((i <= N)&&(det != 0.0))
{
    C = A.elemento(i,i);
    L = i;
    for(int k=i+1;k<=N;k++)
    {
        if (fabs(C) < fabs(A.elemento(k,i)))
        {
            C = A.elemento(k,i);
            L = k;
        }
    }
    if (L != i)
    {
        det *= -1.0;
        for(int j=1;j<=N;j++)
        {
            X = A.elemento(i,j);
            A.Atribui(i,j,A.elemento(L,j));
            A.Atribui(L,j,X);
        }
    }
    P = A.elemento(i,i);
    det *= P;
    if (P != 0.0)
    {
        for(int j=1;j<=N;j++)
        {
            A.Atribui(i,j,A.elemento(i,j)/P);
        }
        for(int k=i+1;k<=N;k++)
        {
            P = A.elemento(k,i);
            for(int j=1;j<=N;j++)
            {
                A.Atribui(k,j,A.elemento(k,j)-P*A.elemento(i,j));
            }
        }
        i++;
    }
}
return det;
}

```

```
bool matriz::Singular()
```

```

{
    // Verifica se a matriz é singular
    return (Determinante() == 0.0);
}

bool matriz::Quadrada()
{
    // Verifica se a matriz é quadrada
    return (num_linhas == num_colunas);
}

bool matriz::Simetrica()
{
    // Verifica se a matriz é simétrica
    if (!Quadrada()) return false;
    bool resultado = true;
    for (int i=1;i<=num_linhas;i++)
    {
        for (int j=1;j<i;j++)
        {
            if (elemento(i,j) != elemento(j,i)) resultado = false;
        }
    }
    return resultado;
}

bool matriz::AntiSimetrica()
{
    // Verifica se a matriz é antissimétrica
    if (!Quadrada()) return false;
    for (int i=1;i<=num_linhas;i++)
    {
        if (elemento(i,i) != 0.0) return false;
        for (int j=1;j<i;j++)
        {
            if (elemento(i,j) != -1*elemento(j,i)) return false;
        }
    }
    return true;
}

matriz matriz::Inversa()
{
    return ~(*this);
}

matriz matriz::Transposta()
{
    return !(*this);
}

void matriz::ZeraMatriz()
{
    redim(num_linhas, num_colunas);
}

void matriz::TransFormaEmIdentidade()
{
    int i,j;

```

```

for (i=1;i<=num_linhas;i++)
{
    for (j=1;j<=num_colunas;j++)
    {
        if (i==j) Atribui(i,j,1); else Atribui(i,j,0);
    }
}
}

```

```

////////////////////////////////////
// Operadores Sobrecarregados ////
////////////////////////////////////

```

```

matriz matriz::operator =(double valor)
{
    for (int i=1;i<=num_linhas*num_colunas;i++)
    {
        elementos[i] = valor;
    }
    return *this;
}

```

```

matriz matriz::operator =(matriz B)
{
    for (int i=1;i<=num_linhas*num_colunas;i++)
    {
        elementos[i] = B.elementos[i];
    }
    return *this;
}

```

```

matriz matriz::operator ^(int valor)
{
    // Soma um valor aos elementos da Matriz
    matriz Temp(num_linhas,num_colunas);
    Temp.TransFormaEmIdentidade();
    for (int i=1;i<=valor;i++)
    {
        Temp *= *this;
    }
    return Temp;
}

```

```

void matriz::operator ^=(int valor)
{
    // Soma um valor aos elementos da Matriz
    *this = *this ^ valor;
}

```

```

matriz matriz::operator +(double valor)
{
    // Soma um valor aos elementos da Matriz
    matriz Temp(num_linhas,num_colunas);
    for (int i=1;i<=num_linhas*num_colunas;i++)
    {
        Temp.elementos[i] = elementos[i]+valor;
    }
    return Temp;
}

```



```

}

void matriz::operator +=(double valor)
{
    // Soma um valor aos elementos da Matriz
    *this = *this + valor;
}

matriz matriz::operator -(double valor)
{
    // Subtrai um valor aos elementos da Matriz
    matriz Temp(num_linhas,num_colunas);
    for (int i=1;i<=num_linhas*num_colunas;i++)
    {
        Temp.elementos[i] = elementos[i]-valor;
    }
    return Temp;
}

void matriz::operator -=(double valor)
{
    // Soma um valor aos elementos da Matriz
    *this = *this - valor;
}

matriz matriz::operator +(matriz B)
{
    // Soma de Matrizes
    matriz Temp(num_linhas,num_colunas);
    for (int i=1;i<=num_linhas*num_colunas;i++)
    {
        Temp.elementos[i] = elementos[i]+B.elementos[i];
    }
    return Temp;
}

void matriz::operator +=(matriz B)
{
    *this = *this + B;
}

matriz matriz::operator -(matriz B)
{
    // Subtração de Matrizes
    matriz Temp(num_linhas,num_colunas);
    for (int i=1;i<=num_linhas*num_colunas;i++)
    {
        Temp.elementos[i] = elementos[i]-B.elementos[i];
    }
    return Temp;
}

void matriz::operator -=(matriz B)
{
    *this = *this - B;
}

matriz matriz::operator *(matriz B)

```

```

{
// Multiplicação de Matrizes
matriz Temp(num_linhas,B.num_colunas);
for (int i=1;i<=num_linhas;i++)
{
for (int j=1;j<=B.num_colunas;j++)
{
for (int k=1;k<=num_colunas;k++)
{
Temp.elementos[(i-1)*Temp.num_colunas+j] +=
elemento(i,k)*B.elemento(k,j);
}
}
}
return Temp;
}

```

```

void matriz::operator *=(matriz B)
{
*this = *this * B;
}

```

```

matriz matriz::operator *(double fator)
{
// Multiplica uma matriz por um número
matriz Temp(num_linhas, num_colunas);
for (int i=1;i<=num_linhas;i++)
{
for (int j=1;j<=num_colunas;j++)
{
Temp.Atribui(i,j,fator*elemento(i,j));
}
}
return Temp;
}

```

```

void matriz::operator *=(double fator)
{
for (int i=1;i<=num_linhas;i++)
{
for (int j=1;j<=num_colunas;j++)
{
Atribui(i,j,fator*elemento(i,j));
}
}
}

```

```

matriz matriz::operator /(matriz A)
{
// Resolve um sistema de equações considerando um
// sistema  $Ax = B$ , onde B é a matriz atual
int i, j, k;
double c, sum;
int N = num_linhas;
matriz b(N,num_colunas);
matriz a(N,N);
for(int coluna=1;coluna<=num_colunas;coluna++)
{

```

```

for(i=1;i<=N;i++)
{
    for(j=1;j<=N;j++)
    {
        a.Atribui(i,j,A.elemento(i,j));
    }
}
for(i=1;i<=N;i++)
{
    b.Atribui(i,coluna,elemento(i,coluna));
}
for(k=1;k<N;k++)
{
    for(i=k+1;i<=N;i++)
    {
        c = a.elemento(i,k)/a.elemento(k,k); //Cálculo do pivo.
        for(j=k+1;j<=N;j++)
        {
            a.Atribui(i,j,a.elemento(i,j)-c*a.elemento(k,j));
        }
        b.Atribui(i,coluna,b.elemento(i,coluna)-c*b.elemento(k,coluna));
    }
}
// Retrossubstituição.
b.Atribui(N,coluna,b.elemento(N,coluna)/a.elemento(N,N));
for (i=N-1;i>=1;i--)
{
    sum = 0.0;
    for(j=i+1;j<=N;j++)
    {
        sum = sum + a.elemento(i,j)*b.elemento(j,coluna);
    }
    b.Atribui(i,coluna,(b.elemento(i,coluna) - sum)/a.elemento(i,i));
}
}
return b;
}

void matriz::operator /=(matriz A)
{
    *this = *this / A;
}

matriz matriz::operator /(double fator)
{
    // Divide os elementos da matriz por um valor real
    matriz Temp(num_linhas, num_colunas);
    for (int i=1;i<=num_linhas;i++)
    {
        for (int j=1;j<=num_colunas;j++)
        {
            Temp.Atribui(i,j,elemento(i,j)/fator);
        }
    }
    return Temp;
};

void matriz::operator /=(double fator)
{

```

```

for (int i=1;i<=num_linhas;i++)
{
    for (int j=1;j<=num_colunas;j++)
    {
        Atribui(i,j,(elemento(i,j)/fator));
    }
}

matriz matriz::operator -()
{
    // Decrementa os elementos da matriz em uma unidade
    matriz Temp(num_linhas, num_colunas);
    for (int i=1;i<=num_linhas;i++)
    {
        for (int j=1;j<=num_colunas;j++)
        {
            Temp.Atribui(i,j,-1*elemento(i,j));
        }
    }
    return Temp;
}

matriz matriz::operator ++()
{
    // Incrementa os elementos da matriz em uma unidade
    matriz Temp(num_linhas, num_colunas);
    for (int i=1;i<=num_linhas;i++)
    {
        for (int j=1;j<=num_colunas;j++)
        {
            Atribui(i,j,elemento(i,j)+1);
            Temp.Atribui(i,j,elemento(i,j));
        }
    }
    return Temp;
}

matriz matriz::operator ++(int)
{
    // Incrementa os elementos da matriz em uma unidade
    matriz Temp(num_linhas, num_colunas);
    for (int i=1;i<=num_linhas;i++)
    {
        for (int j=1;j<=num_colunas;j++)
        {
            Temp.Atribui(i,j,elemento(i,j));
            Atribui(i,j,elemento(i,j)+1);
        }
    }
    return Temp;
}

matriz matriz::operator --()
{
    // Decrementa os elementos da matriz em uma unidade
    matriz Temp(num_linhas, num_colunas);
    for (int i=1;i<=num_linhas;i++)

```

```

    {
        for (int j=1;j<=num_colunas;j++)
        {
            Atribui(i,j,elemento(i,j)-1);
            Temp.Atribui(i,j,elemento(i,j));
        }
    }
    return Temp;
}

matriz matriz::operator --(int)
{
    // Decrementa os elementos da matriz em uma unidade
    matriz Temp(num_linhas, num_colunas);
    for (int i=1;i<=num_linhas;i++)
    {
        for (int j=1;j<=num_colunas;j++)
        {
            Temp.Atribui(i,j,elemento(i,j));
            Atribui(i,j,elemento(i,j)-1);
        }
    }
    return Temp;
}

bool matriz::operator ==(matriz B)
{
    // Verifica a igualdade entre matrizes
    for (int i=1;i<=num_linhas;i++)
    {
        for (int j=1;j<=num_colunas;j++)
        {
            if (elemento(i,j) != B.elemento(i,j)) return false;
        }
    }
    return true;
}

bool matriz::operator !=(matriz B)
{
    // Verifica a desigualdade entre matrizes
    for (int i=1;i<=num_linhas;i++)
    {
        for (int j=1;j<=num_colunas;j++)
        {
            if (elemento(i,j) != B.elemento(i,j)) return true;
        }
    }
    return false;
}

matriz matriz::operator !()
{
    // Obtem a Transposta da Matriz
    matriz Temp(num_colunas,num_linhas);
    for (int i=1;i<=num_colunas;i++)
    {
        for (int j=1;j<=num_linhas;j++)

```

```

    {
        Temp.Atribui(i,j,elemento(j,i));
    }
}
return Temp;
}

matriz matriz::operator ~()
{
    // Retorna a Inversa da Matriz
    double fator;
    matriz Inversa(num_linhas, num_colunas);
    for (int i=1;i<=num_linhas;i++)
    {
        for (int j=1;j<=num_linhas;j++)
        {
            Inversa.Atribui(i,j,elemento(i,j));
        }
    }
    for (int k=1;k<=num_linhas;k++)
    {
        fator = Inversa.elemento(k,k);
        Inversa.Atribui(k,k,1.0);
        for (int j=1;j<=num_linhas;j++)
        {
            Inversa.Atribui(k,j,Inversa.elemento(k,j)/fator);
        }
        for (int i=1;i<=num_linhas;i++)
        {
            if (i != k)
            {
                fator = Inversa.elemento(i,k);
                Inversa.Atribui(i,k,0.0);
                for (int j=1;j<=num_linhas;j++)
                {
                    Inversa.Atribui(i,j,Inversa.elemento(i,j)-Inversa.elemento(k,j)*fator);
                }
            }
        }
    }
}
return Inversa;
}

double& matriz::operator()(int i, int j)
{
    // retorna um elemento da matriz
    //return elemento(i, j);
    return elementos[(i-1)*num_colunas+j];
}

double& matriz::operator()(double i, double j)
{
    // retorna um elemento da matriz
    //return elemento(i, j);
    return elementos[((int)i-1)*num_colunas+(int)j];
}

```

```

void matriz::operator()(int i, int j, double valor)
{
    // Operador de Atribuição
    Atribui(i, j, valor);
}

void matriz::operator()(int i, double valor)
{
    // Operador de Atribuição
    Atribui(i, 1, valor);
}

void matriz::imprime()
{
    for (int i=1;i<=num_linhas;i++)
    {
        printf("\n");
        for (int j=1;j<=num_colunas;j++)
        {
            printf("%f ",elementos[(i-1)*num_colunas+j] );
        }
    }
}

```

8.2.2 CLASSE MALHA

```

#include "Matriz.h"
#include <math.h>
class Malha
{ public:
    int nx, ny; // numero de nós em X e em Y
    matriz coord;//Matriz de coordenadas X ou Y dos nós da malha
    Malha(double xi=0.0, double yi=0.0, double xf=20.0, double yf=20.0): xinf(xi), yinf(yi), xsup(xf),
ysup(yf) {nx=ny=1;}
    ~Malha() {}
    void pg(int xy, double a0, double q, int n, int no=1);//Divide o eixo X ou Y em PG
    void pa(int xy, double a0, double r, int n, int no=1);//Divide o eixo X ou Y em PA
    // xy=1 para divisão na direção x e 2 na direção y, a0 tamanho do 1º elemento
    // q ou r razão da progressão, n nº de termos, no nº do nó na direção x ou y
    // em que se começa esta progressão (nó inicial de a0)
    void div(int xy, double comp, int n, int no=1);//Divide igualmente o segmento
    // de comprimento comp do eixo X ou Y em n partes iguais a partir do nó no
    void acumula(int xy, int n, int no, matriz aux);
    void razao_max(int xy, double rm=5.0);//Verifica a razão comprimento/largura
    // de cada elemento é maior que a razão máxima e se for esses elementos são
    // modificados, se rm=0, não se realiza esta verificação.
    void nodes();
    void Nos(int xy, matriz A);//Acrescenta o vetor de coordenadas X ou Y dos nós na matriz A
    void Nos(int xy);//Retorna o vetor de coordenadas X ou Y dos nós
    void redim(int xy, double inf, double sup);//Redimensiona o vetor de coordenadas
    // X ou Y da malha para os novos limites inferior inf e superior sup
    void redim(int xy, double sup);
protected:
    matriz x, y;
    double xinf, xsup, yinf, ysup;
};

```

```

void Malha::pg(int xy, double a0, double q, int n, int no)
{
    if (n==0)
    { double Sn=0;
      if (xy==1)
      { if (no==1)
        { x(no) = xinf;
          while ((x(no)+Sn)<xsup)
            Sn = a0*(pow(q,++n)-1)/(q-1);
          }
        else
        { if (no==1)
          { y(no) = yinf;
            while ((y(no)+Sn)<ysup)
              Sn = a0*(pow(q,++n)-1)/(q-1);
            }
          }
        matriz aux(n+1);
        for (int i=0;i<n;i++)
            aux(i+2) = aux(i+1) + a0*pow(q,i);
        acumula(xy,n,no,aux);
    }
}

```

```

void Malha::pa(int xy, double a0, double r, int n, int no)
{
    if (n==0)
    { double Sn=0;
      if (xy==1)
      { if (no==1)
        { x(no) = xinf;
          while ((x(no)+Sn)<xsup)
            { n++;
              Sn = (2*a0 + r*(n-1))*n/2;
            }
          }
        else
        { if (no==1)
          { y(no) = yinf;
            while ((y(no)+Sn)<ysup)
              { n++;
                Sn = (2*a0 + r*(n-1))*n/2;
              }
          }
        }
      matriz aux(n+1);
      for (int i=0;i<n;i++)
          aux(i+2) = aux(i+1) + a0 + i*r;
      acumula(xy,n,no,aux);
    }
}

```

```

void Malha::div(int xy, double comp, int n, int no)
{
    if (comp==0)
    { if (no==1)
      { if (xy==1)
        { comp = xsup - xinf;
          }
        else
        { comp = ysup - yinf;
          }
      }
    }
}

```



```

    }
    else
    { if (xy= =1)
        comp = xsup - x(no);
      else
        comp = ysup - y(no);
    }
  }
  matriz aux(n+1);
  double tam;
  tam = comp/n;
  for (int i=0;i<n;i++)
    aux(i+2) = aux(i+1) + tam;
  acumula(xy,n,no,aux);
}

void Malha::acumula(int xy, int n, int no, matriz aux)
{
  int i;
  if (no!=1)
  { if (xy= =1)
    { if (no>nx)
      no = nx;
      if (x(no)= =xsup)
        return;
      for (i=1;i<=n+1;i++)
        aux(i) += x(no);
      matriz B(nx);
      B = x;
      if (aux(n+1)>xsup)
      { int k=1;
        while (aux(k)<xsup)
          k++;
        aux(k)=xsup;
        n=k-1;
      }
      x.redim(no+n);
      for (i=1;i<=no;i++)
        x(i)=B(i);
      for (i=1;i<=n;i++)
        x(no+i)=aux(i+1);
      nx = no + n;
    }
  }
  else
  { if (no>ny)
    no = ny;
    if (y(no)= =ysup)
      return;
    for (i=1;i<=n+1;i++)
      aux(i) += y(no);
    matriz B(no);
    B = y;
    if (aux(n+1)>ysup)
    { int k=1;
      while (aux(k)<ysup)
        k++;
      aux(k)=ysup;
      n=k-1;
    }
  }
}

```

```

    y.redim(no+n);
    for (i=1;i<=no;i++)
        y(i)=B(i);
    for (i=1;i<=n;i++)
        y(no+i)=aux(i+1);
    ny = no + n;
}
}
else
{ if (xy= =1)
  { for (i=1;i<=n+1;i++)
    aux(i) += xinf;
    if (aux(n+1)>xsup)
    { int k=1;
      while (aux(k)<xsup)
        k++;
      aux(k)=xsup;
      n=k-1;
    }
    x.redim(n+1);
    x=aux;
    nx = n + 1;
  }
  else
  { for (i=1;i<=n+1;i++)
    aux(i) += yinf;
    if (aux(n+1)>ysup)
    { int k=1;
      while (aux(k)<ysup)
        k++;
      aux(k)=ysup;
      n=k-1;
    }
    y.redim(n+1);
    y=aux;
    ny = n + 1;
  }
}
}
}

```

```

void Malha::razao_max(int xy, double rm)

```

```

{
  if (rm= =0)
    return;
  double min, max;
  int i;
  if (xy= =1)
  { min = y(2) - y(1);
    for (i=3;i<=ny;i++)
    { if (min>(y(i)-y(i-1)))
      min = y(i) - y(i-1);
    }
    max = min*rm;
    if ((x(nx)-x(nx-1))<=max)
      return;
    min = 0.0;
    i=1;
    while (min<=max)
    { min = x(i+1) - x(i);

```

```

        i++;
    }
    int no=i-1;
    min = x(nx) - x(no);
    i=1;
    while ((min/i)>max)
        i++;
    div(xy,min,i,no);
}
else
{
    min = x(2) - x(1);
    for (i=3;i<=nx;i++){
        { if (min>(x(i)-x(i-1)))
            min = x(i) - x(i-1);
        }
    }
    max = min*rm;
    if ((y(ny)-y(ny-1))<=max)
        return;
    min = 0.0;
    i=1;
    while (min<=max)
    {
        min = y(i+1) - y(i);
        i++;
    }
    int no=i-1;
    min = y(ny) - y(no);
    i=1;
    while ((min/i)>max)
        i++;
    div(xy,min,i,no);
}
}

void Malha::nodes()
{
    int i, j;
    for(i=1 ; i <= ny ; i++){
        for(j=1 ; j <= nx ; j++){
            printf("\nm(%d,%d)=(%2.4e, %2.4e)",i,j,x(j),y(i));
        }
        printf("\n");
    }
}

void Malha::Nos(int xy,matriz A)
{
    int i;
    if (A.num_linhas!=1)
    {
        matriz aux(A.num_linhas);
        aux = A;
        if (xy==1)
        {
            coord.redim(aux.num_linhas+nx-1);
            for (i=1;i<aux.num_linhas;i++)
                coord.Atribui(i,aux(i));
            for (i=0;i<nx;i++)
                coord.Atribui(aux.num_linhas+i,x(i+1));
        }
    }
    else
    {
        coord.redim(aux.num_linhas+ny-1);
    }
}

```

```

        for (i=1;i<aux.num_linhas;i++)
            coord.Atribui(i,aux(i));
        for (i=0;i<ny;i++)
            coord.Atribui(aux.num_linhas+i,y(i+1));
    }
}
else
    Nos(xy);
}

```

```

void Malha::Nos(int xy)
{
    if (xy= =1)
    { coord.redim(nx);
      coord = x;
    }
    else
    { coord.redim(ny);
      coord = y;
    }
}

```

```

void Malha::redim(int xy, double inf, double sup)
{
    if (xy= =1)
    { xinf = inf;
      xsup = sup;
      x.redim(1);
      nx = 1;
    }
    else
    { yinf = inf;
      ysup = sup;
      y.redim(1);
      ny = 1;
    }
}

```

```

void Malha::redim(int xy, double sup)
{
    if (xy= =1)
        redim(1,xsup,sup);
    else
        redim(2,ysup,sup);
}

```

8.2.3 CLASSE MATERIAL

```

class Material
{
public:
    double POISS, MR, MRant; // Coeficiente de Poisson, Módulo de Resiliência e Módulo de Resiliência
anterior
    matriz ki; // Constantes de determinação do módulo de resiliência
    int tipo; // Tipo de material (equação do módulo de resiliência)
    Material(); // Construtor da Classe
    void calcula_MR();
    void calcula_MR(double tensao);
}

```

```

void calcula_MR(double Sd, double S3);
void calcula_MR(double S1, double S2, double S3);
Material operator =(Material Mat);// Operador de Atribuição.
};

```

```

////////////////////////////////////
/// Métodos da Classe Material ////
////////////////////////////////////

```

```

Material::Material()
{
    POISS = MR = MRant = 0.0;
    ki.redim(5);
    tipo = 6;
}

```

```

void Material::calcula_MR()
{
    if (tipo==6)
        MR = ki(1);
    if ((tipo==1)||(tipo==2)||(tipo==4)||(tipo==7))
        MR = ki(1);
    if ((tipo==3)||(tipo==5))
        MR = ki(2);
}

```

```

void Material::calcula_MR(double tensao)
{
    if ((tipo==1)||(tipo==2)||(tipo==4))
        MR = ki(1)*(pow(tensao,ki(2)));
    if (tipo==3)
    { if (tensao<ki(1))
        MR = ki(2) + ki(3)*(ki(1) - tensao);
      else
        MR = ki(2) + ki(4)*(tensao - ki(1));
    }
}

```

```

void Material::calcula_MR(double Sd, double S3)
{
    if (tipo==5)
    { if(Sd<ki(1))
        MR = ki(2) + ki(3)*(ki(1) - Sd)*pow(S3,ki(5));
      else
        MR = ki(2) + ki(4)*(Sd - ki(1))*pow(S3,ki(5));
    }
    if (tipo==7)
        MR = ki(1)*pow(S3,ki(2))*pow(Sd,ki(3));
}

```

```

void Material::calcula_MR(double S1,double S2, double S3)
{
    MRant = MR;
    if (tipo==1) calcula_MR(S3);
    if (tipo==2) calcula_MR(S1+S2+S3);
    if ((tipo==3)||(tipo==4)) calcula_MR(S1-S3);
    if ((tipo==5)||(tipo==7)) calcula_MR(S1-S3,S3);
    if (tipo==6) calcula_MR();
}

```

```

Material Material::operator =(Material Mat)
{
    POISS = Mat.POISS;
    MR = Mat.MR;
    MRant = Mat.MRant;
    ki = Mat.ki;
    tipo = Mat.tipo;
    return *this;
}

```

8.2.4 CLASSE PAVIMENTO

```

class Pavimento // Matriz dos materias constituintes do pavimento (para cada fileira)
{
public:
    int num_camadas, fileiras_tot; // Número de camadas do pavimento e número total de fileiras de
elementos do pavimento
    int num_colunas; // Número de colunas total de elementos do pavimento
    int *fileiras; // Matriz com o número de fileiras de elementos de cada camada
    int *fileiras_at; // Matriz com o número atual de fileiras de elemento de cada camada
    Material *elementos; // Armazena os elementos do pavimento
    Pavimento();
    Pavimento(int n); // Construtor com o número de camadas do pavimento
    Pavimento(int n, int *A); // construtor com o número de fileiras de elementos do pavimento
    Material elemento(int f); // Retorna o material da fileira f de elementos do pavimento
    Material elemento(int c, int f); // Retorna o material da fileira f de elementos refernetes à camada c do
pavimento
    Material elemento(int c, int f, int col); // Retorna o material da fileira f de elementos refernetes à camada
c do pavimento na coluna col
    void redim(int n); // Redimensiona a matriz dos materiais
    void redim(int n, int *A); // Redimensiona a matriz dos materiais
    void redim(); // Redimensiona a matriz dos materiais, inserindo as fileiras necessárias entre cada camada
    void redim_col(int col); // Redimensiona a matriz dos materiais, inserindo colunas
    void acrescenta(int l, int c, int f = 1); // Acrescenta l linhas na camada c depois da fileira f
    void inicializa(); // Repete a primenira fileira para as demais de cada camada e calcula os módulos de
resiliência inicial
    void inicializa(double Sd, double S3); // Os módulos de resiliência inicial sao calculados a partir de Sd e
S3
    bool tolerancia(double tol); // Compara os módulos de resiliência (MR e MRant) de cada elemento e se
satisfatório retorna true
    Pavimento operator =(Pavimento Pav); // Operador de Atribuição.
    Material& operator()(int f);
    Material& operator()(double f);
    Material& operator()(int c, int f);
    Material& operator()(double c, double f);
    Material& operator()(int c, int f, int col);
    Material& operator()(double c, double f, double col);
    Material& operator()(int f, double col);
};

////////////////////////////////////
/// Métodos da Classe Pavimento ////
////////////////////////////////////

Pavimento::Pavimento()
{
    num_camadas = 1;
}

```

```

    num_colunas = 1;
    fileiras_tot = 1;
    fileiras_at = new int[2];
    fileiras = new int[2];
    elementos = new Material[2];
}

```

```

Pavimento::Pavimento(int n)

```

```

{
    num_camadas = n;
    fileiras_tot = n;
    elementos = new Material[n+1];
    delete [] fileiras_at;
    delete [] fileiras;
    fileiras_at = new int[n+1];
    for (int i=1;i<=n;i++)
        fileiras_at[i]=1;
    fileiras = new int[n+1];
}

```

```

Pavimento::Pavimento(int n, int *A)

```

```

{
    num_camadas = n;
    fileiras_tot = 0;
    for (int i=1;i<=n;i++)
        fileiras_tot += A[i];
    elementos = new Material[fileiras_tot+1];
    fileiras_at = new int[n+1];
    fileiras = new int[n+1];
    for (int i=1;i<=n;i++)
    { fileiras_at[i] = A[i];
      fileiras[i] = A[i];
    }
}

```

```

Material Pavimento::elemento(int f)

```

```

{
    // Obtem o material armazenado em um elemento da Matriz
    return elementos[f];
}

```

```

Material Pavimento::elemento(int c, int f)

```

```

{
    int pos = 0; // Inicializa a posição do elemento
    int i = 1;
    while (i<c)
    { pos += int(fileiras_at[i]);
      i++;
    }
    for (i=1;i<=f;i++)
        pos += 1;
    return elementos[pos];
}

```

```

Material Pavimento::elemento(int c, int f, int col)

```

```

{
    int pos = 0; // Inicializa a posição do elemento
    int i = 1;
    while (i<c)

```

```

    { pos += int(fileiras_at[i]);
      i++;
    }
    for (i=1;i<=f;i++)
      pos += 1;
    return elementos[pos+(col-1)*fileiras_tot];
}

void Pavimento::redim(int n)
{
    // Redimensiona a Matriz de material para n camadas
    // Os valores anteriormente armazenados nesta Matriz são perdidos
    delete [] elementos;
    elementos = new Material[n+1];
    num_camadas = n;
    delete [] fileiras_at;
    delete [] fileiras;
    fileiras_at = new int[n+1];
    fileiras_tot = n;
    fileiras = new int[n+1];
    for (int i=1;i<=n;i++)
        fileiras_at[i] = 1;
}

void Pavimento::redim(int n, int *A)
{
    // Redimensiona a Matriz de material de acordo com os elemento da matriz A
    // Os valores anteriormente armazenados nesta Matriz são perdidos
    delete [] elementos;
    num_camadas = n;
    fileiras_tot = 0;
    int *B;
    B = new int[n+1];
    for (int i=1;i<=n;i++)
    { fileiras_tot += A[i];
      B[i] = A[i];
    }
    elementos = new Material[fileiras_tot+1];
    delete [] fileiras_at;
    delete [] fileiras;
    fileiras_at = new int[n+1];
    fileiras = new int[n+1];
    for (int i=1;i<=n;i++)
    { fileiras_at[i] = B[i];
      fileiras[i] = B[i];
    }
}

void Pavimento::redim()
{
    // Redimensiona a Matriz de material com num_camadas elementos
    // A nova dimensão depende de num_camadas e da matriz fileiras
    // Os valores anteriormente armazenados nesta Matriz não são perdidos
    Material *A;
    A = new Material[fileiras_tot+1];
    for (int i=1;i<=fileiras_tot;i++)
        A[i] = elementos[i];
    int *B;
    B = new int[num_camadas+1];
}

```



```

for (int i=1;i<=num_camadas;i++)
    B[i] = fileiras_at[i];
redim(num_camadas,fileiras);
int k = 0;
int l = 1;
for (int i=1;i<=num_camadas;i++)
{ for (int j=1;j<=B[i];j++)
    elementos[k+j] = A[l++];
    k += fileiras[i];
}
delete [] A;
delete [] B;
}

void Pavimento::redim_col(int col)
{
// Redimensiona a Matriz de material com col colunas
// Os valores anteriormente armazenados nesta Matriz não são perdidos e são
// transferidos para as novas colunas
Material *A;
A = new Material[fileiras_tot+1];
for (int i=1;i<=fileiras_tot;i++)
    A[i] = elementos[i];
delete [] elementos;
elementos = new Material[fileiras_tot*col+1];
for (int i=1;i<=fileiras_tot;i++)
{ for (int j=1;j<=col;j++)
    elementos[i+(j-1)*fileiras_tot] = A[i];
}
delete [] A;
num_colunas = col;
}

void Pavimento::acrescenta(int l, int c, int f)
{
// Acrescenta l linhas na camada c depois da fileira f
// Os valores armazenados anteriormente não são perdidos
Material *A;
A = new Material[fileiras_tot+1];
for (int i=1;i<=fileiras_tot;i++)
    A[i] = elementos[i];
int *auxiliar;
auxiliar = new int[num_camadas+1];
for (int i=1;i<=num_camadas;i++)
    auxiliar[i] = fileiras_at[i];
fileiras_at[c] += l;
redim(num_camadas,fileiras_at);
int k = 0;
int p = 1;
for (int i=1;i<=c;i++)
{ for (int j=1;j<=fileiras_at[i];j++)
    elementos[k+j] = A[p++];
    k += fileiras_at[i];
}
if (auxiliar[c]>f)
{ for (int i=1;i<=f;i++)
    elementos[k+i] = A[p++];
    for (int i=f+1;i<=auxiliar[c];i++)
        elementos[k+l+i] = A[p++];
}
}

```

```

    }
    else
    { for (int i=1;i<=auxiliar[c];i++)
        elementos[k+i] = A[p++];
    }
    k += fileiras_at[c];
    for (int i=c+1;i<=num_camadas;i++)
    { for (int j=1;j<=fileiras_at[i];j++)
        elementos[k+j] = A[p++];
        k += fileiras_at[i];
    }
    delete [] A;
}

void Pavimento::inicializa()
{
    int k = 0;
    for (int i=1;i<=num_camadas;i++)
    { elementos[k+1].calcula_MR();
        for (int j=2;j<=fileiras_at[i];j++)
            elementos[k+j] = elementos[k+1];
        k += fileiras_at[i];
    }
}

void Pavimento::inicializa(double Sd, double S3)
{
    int k = 0;
    for (int i=1;i<=num_camadas;i++)
    { elementos[k+1].calcula_MR(Sd+S3,S3,S3);
        for (int j=2;j<=fileiras_at[i];j++)
            elementos[k+j] = elementos[k+1];
        k += fileiras_at[i];
    }
}

bool Pavimento::tolerancia(double tol)
{
    double dif; // diferença dos módulos de resiliência
    for (int i=1;i<=fileiras_tot*num_colunas;i++)
    { if (elementos[i].MRant==0)
        dif = (elementos[i].MR - elementos[i].MRant)/0.00001;
        else
        dif = (elementos[i].MR - elementos[i].MRant)/elementos[i].MRant;
        if (fabs(dif)>tol)return false;
    }
    return true;
}

Pavimento Pavimento::operator =(Pavimento Pav)
{
    redim(Pav.num_camadas,Pav.fileiras_at);
    delete [] fileiras;
    fileiras = new int[num_camadas];
    for (int i=1;i<=num_camadas;i++)
        fileiras[i] = Pav.fileiras[i];
    redim_col(Pav.num_colunas);
    for (int i=1;i<=fileiras_tot*num_colunas;i++)
        elementos[i] = Pav.elementos[i];
}

```

```

    return *this;
}

Material& Pavimento::operator()(int f)
{
    // retorna um material do pavimento
    return elementos[f];
}

Material& Pavimento::operator()(double f)
{
    // retorna um material do pavimento
    return elementos[int(f)];
}

Material& Pavimento::operator()(int c, int f)
{
    // retorna um material do pavimento
    int pos = 0; // Inicializa a posição do elemento
    int i = 1;
    while (i<c)
    { pos += fileiras_at[i];
      i++;
    }
    for (i=1;i<=f;i++)
        pos += 1;
    return elementos[pos];
}

Material& Pavimento::operator()(double c, double f)
{
    // retorna um material do pavimento
    int pos = 0; // Inicializa a posição do elemento
    int i = 1;
    while (i<int(c))
    { pos += fileiras_at[i];
      i++;
    }
    for (i=1;i<=int(f);i++)
        pos += 1;
    return elementos[pos];
}

Material& Pavimento::operator()(int c, int f, int col)
{
    // retorna um material do pavimento
    int pos = 0; // Inicializa a posição do elemento
    int i = 1;
    while (i<c)
    { pos += fileiras_at[i];
      i++;
    }
    for (i=1;i<=f;i++)
        pos += 1;
    return elementos[pos+(col-1)*fileiras_tot];
}

Material& Pavimento::operator()(double c, double f, double col)
{

```

```

// retorna um material do pavimento
int pos = 0; // Inicializa a posição do elemento
int i = 1;
while (i<int(c))
{ pos += fileiras_at[i];
  i++;
}
for (i=1;i<=int(f);i++)
  pos += 1;
return elementos[pos+(int(col)-1)*fileiras_tot];
}

```

```

Material& Pavimento::operator()(int f, double col)
{
// retorna um material do pavimento
return elementos[f+(int(col)-1)*fileiras_tot];
}

```

8.2.5 CLASSE NO

```

class No
{
public:
  double x, y, z; // Coordenadas
  double Fx, Fy, Fz; //Cargas Nodais.
  int restx, resty, restz; // Restrições
  double prescx, prescy, prescz; // Deslocamentos Prescritos
  double u, v, w; //Deslocamentos Nodais;
  No(); // Construtor da classe.
  void GeraResultados(ofstream &arquivo_saida, ofstream &arquivo_view, int NSD);
};

////////////////////////////////////
// Métodos da Classe Nó ////
////////////////////////////////////

No::No()
{
  x = y = z = u = v = w = Fx = Fy = Fz = prescx = prescy = prescz = 0.0;
  restx = resty = restz = 0;
}

void No::GeraResultados(ofstream &arquivo_saida, ofstream &arquivo_view, int NSD)
{
  arquivo_saida << u << "\t" << v;
  if (NSD == 3) arquivo_saida << "\t" << w;
  arquivo_saida << "\n";
}

```

8.2.6 CLASSE PONTODEGAUSS

```

class PontoDeGauss
{
// OBSERVACAO
// O PROGRAMA ESTA CONSIDERANDO AGORA A HIPOTESE DE ATE 8 PONTOS
// DE GAUSS EM CADA ELEMENTO EPT
public:

```

```

matriz coordenada, pesos;
double XI, ETA, FI; // Coordenadas Locais do ponto de gauss.
double WXI, WETA, WFI; // Pesos do ponto de gauss.
matriz D; // Matriz Constitutiva do Material
matriz K; // Matriz de Rigidez
matriz Fint; // Vetor de Forças Internas
matriz LN; // Matriz Resultante da Multiplicação do Operador [L]
    // Pela Matriz de Funções de Interpolação [N]
matriz B; // Matriz que Associa Deformações a Deslocamentos
    // [B] = Inversa(J)*[L]{N}
matriz J; // Matriz Jacobiano Responsável pela Mudança de Coordenadas
matriz u; // Matriz de Deslocamentos Dos Pontos Nodais
matriz sigma; // Vetor de Tensões no Ponto de Gauss
// sigma(1) = sigmax = Sr
// sigma(2) = sigmay = Sz
// sigma(3) = sigmaz = Trz
// sigma(4) = sigmaxy = Steta
// sigma(5) = sigmaxz
// sigma(6) = sigmayz
virtual void Calcula_D(Material &Mat){}; // Calcula a Matriz Constitutiva
virtual void Calcula_B(int i, int NumeroDePontosNodais, No * Nos){}; // Calcula a Matriz [B]
virtual void Calcula_K(int i, int NumeroDePontosNodais, No * Nos, Material &Mat){};
// Calcula a Matriz de Rigidez
virtual void Calcula_J(int i, int NumeroDePontosNodais, No * Nos){};
// calcula o Jacobiano
virtual void Calcula_Tensoes(int i, int NumeroDePontosNodais, No * Nos, Material &Mat){};
// Calcula as Tensões no Ponto de Gauss
virtual void CalculaForcasInternas(int i, int NumeroDePontosNodais, No * Nos, Material &Mat){};
// Calcula as Tensões no Ponto de Gauss
virtual void GeraResultados(){};
virtual void Retorno(Material &Mat){};
};

class PontoDeGauss_EPT : public PontoDeGauss
{
public:
    PontoDeGauss_EPT();
    // Implementação dos Métodos Herdados da Classe-Base
    // double Invariante();// Não foi considerado ()
    void CoordenadasDoPontoDegauss(int IGAUSS);
    double COORDENADAXI(int i);
    double COORDENADAETA(int i);
    void Calcula_D(Material &Mat);
    void Calcula_K(int IGAUSS, int NumeroDePontosNodais, No * Nos, Material &Mat);
    void Inicializa(int NumeroDePontosNodais);
    virtual void Calcula_Tensoes(int IGAUSS, int NumeroDePontosNodais, No * Nos, Material &Mat);
    virtual void Calcula_J(int IGAUSS, int NumeroDePontosNodais, No * Nos);
    virtual void Calcula_B(int IGAUSS, int NumeroDePontosNodais, No * Nos); // Calcula a Matriz [B]
    virtual void CalculaForcasInternas(int IGAUSS, int NumeroDePontosNodais, No * Nos, Material
&Mat);
    virtual double SHAPE(int i, int NumeroDePontosNodais, double XI, double ETA);
    virtual double LSHAPEXI(int i, int NumeroDePontosNodais, double XI, double ETA);
    virtual double LSHAPEETA(int i, int NumeroDePontosNodais, double XI, double ETA);
    // void retorno(Material &Mat, No * Nos); // Não foi considerado regime plástico
};

class PontoDeGauss_AXISSIMETRICO : public PontoDeGauss_EPT
{
public:

```

```

double S1, S2, S3; // Tensões Principais
PontoDeGauss_AXISSIMETRICO();
// Implementação dos Métodos Herdados da Classe-Base
virtual void Inicializa(int NumeroDePontosNodais);
virtual void Calcula_K(int IGAUSS, int NumeroDePontosNodais, No * Nos, Material &Mat);
virtual void Calcula_B(int IGAUSS, int NumeroDePontosNodais, No * Nos); // Calcula a Matriz [B]
virtual void Calcula_J(int IGAUSS, int NumeroDePontosNodais, No * Nos);
virtual void Calcula_D(Material &Mat);
virtual void Calcula_Tensoes(int IGAUSS, int NumeroDePontosNodais, No * Nos, Material &Mat);
virtual void CalculaForcasInternas(int IGAUSS, int NumeroDePontosNodais, No * Nos, Material
&Mat);
void tensoes_princ();
matriz tensoes_princ(double Sr, double Sz, double Trz, double Steta);
// Transforma as tensões axissimétricas em tensões principais
/*
virtual double Invariante();
virtual void Calcula_Tensoes(int IGAUSS, int NumeroDePontosNodais, No * Nos, Material &Mat);
virtual void CalculaForcasInternas(int IGAUSS, int NumeroDePontosNodais, No * Nos, Material
&Mat);
*/
};

////////////////////////////////////
/// Métodos da Classe PontoDeGauss ////
////////////////////////////////////

////////////////////////////////////
/// Métodos da Classe PontoDeGauss_EPT ////
////////////////////////////////////

PontoDeGauss_EPT::PontoDeGauss_EPT()
{
    coordenada.redim(3);
    coordenada(1) = -sqrt(3.0/5.0);
    coordenada(3) = sqrt(3.0/5.0);
    pesos.redim(3);
    pesos(1) = 5.0/9.0;
    pesos(2) = 8.0/9.0;
    pesos(3) = 5.0/9.0;
    sigma.redim(3);
    J.redim(4,4);
    D.redim(3,3);
}

void PontoDeGauss_EPT::Inicializa(int NumeroDePontosNodais)
{
    Fint.redim(2*NumeroDePontosNodais);
    K.redim(2*NumeroDePontosNodais,2*NumeroDePontosNodais);
    LN.redim(4,2*NumeroDePontosNodais);
    B.redim(3,2*NumeroDePontosNodais);
}

void PontoDeGauss_EPT::CoordenadasDoPontoDegauss(int IGAUSS)
{
    if (IGAUSS == 1)
    {
        XI = ETA = coordenada(1);
        WXI = WETA = pesos(1);
    }
}

```

```

if (IGAUSS == 2)
{
    XI = coordenada(3);
    ETA = coordenada(1);
    WXI = pesos(3);
    WETA = pesos(1);
}
if (IGAUSS == 3)
{
    XI = ETA = coordenada(3);
    WXI = WETA = pesos(3);
}
if (IGAUSS == 4)
{
    XI = coordenada(1);
    ETA = coordenada(3);
    WXI = pesos(1);
    WETA = pesos(3);
}
if (IGAUSS == 5)
{
    XI = coordenada(2);
    ETA = coordenada(1);
    WXI = pesos(2);
    WETA = pesos(1);
}
if (IGAUSS == 6)
{
    XI = coordenada(3);
    ETA = coordenada(2);
    WXI = pesos(3);
    WETA = pesos(2);
}
if (IGAUSS == 7)
{
    XI = coordenada(2);
    ETA = coordenada(3);
    WXI = pesos(2);
    WETA = pesos(3);
}
if (IGAUSS == 8)
{
    XI = coordenada(1);
    ETA = coordenada(2);
    WXI = pesos(1);
    WETA = pesos(2);
}
}

void PontoDeGauss_EPT::Calcula_J(int IGAUSS, int NumeroDePontosNodais, No * Nos)
{
    CoordenadasDoPontoDegauss(IGAUSS);
    J.ZeraMatriz();
    for (int i=1;i<=NumeroDePontosNodais;i++)
    {
        J.Adiciona(1,1,LSHAPEXI(i, NumeroDePontosNodais, XI, ETA)*Nos[i].x);
        J.Adiciona(1,2,LSHAPEXI(i, NumeroDePontosNodais, XI, ETA)*Nos[i].y);
        J.Adiciona(2,1,LSHAPEETA(i, NumeroDePontosNodais, XI, ETA)*Nos[i].x);
        J.Adiciona(2,2,LSHAPEETA(i, NumeroDePontosNodais, XI, ETA)*Nos[i].y);
    }
}

```

```

        J.Adiciona(3,3,LSHAPEXI(i, NumeroDePontosNodais, XI, ETA)*Nos[i].x);
        J.Adiciona(3,4,LSHAPEXI(i, NumeroDePontosNodais, XI, ETA)*Nos[i].y);
        J.Adiciona(4,3,LSHAPEETA(i, NumeroDePontosNodais, XI, ETA)*Nos[i].x);
        J.Adiciona(4,4,LSHAPEETA(i, NumeroDePontosNodais, XI, ETA)*Nos[i].y);
    }
}

void PontoDeGauss_EPT::Calcula_D(Material &Mat)
{
    double fator = Mat.MR/(1.0 - Mat.POISS*Mat.POISS);
    D.Atribui(1,1,fator);
    D.Atribui(1,2,fator*Mat.POISS);
    D.Atribui(2,1,D(1,2));
    D.Atribui(2,2,D(1,1));
    D.Atribui(3,3,fator*Mat.MR/(2*(1.0 + Mat.POISS)));
}

void PontoDeGauss_EPT::Calcula_K(int IGAUSS, int NumeroDePontosNodais, No * Nos, Material
&Mat)
{
    Calcula_D(Mat);
    Calcula_B(IGAUSS, NumeroDePontosNodais, Nos);
    K = !B*D*B*J.Determinante()*WXI*WETA;
}

void PontoDeGauss_EPT::Calcula_Tensoes(int IGAUSS, int NumeroDePontosNodais, No * Nos,
Material &Mat)
{
    Calcula_D(Mat);
    Calcula_B(IGAUSS, NumeroDePontosNodais, Nos);
    u.redim(2*NumeroDePontosNodais);
    for (int i=1;i<=NumeroDePontosNodais;i++)
    {
        u.Atribui(2*i-1,Nos[i].u);
        u.Atribui(2*i,Nos[i].v);
    }
    sigma = D*B*u;
}

void PontoDeGauss_EPT::Calcula_B(int IGAUSS, int NumeroDePontosNodais, No * Nos)
{
    matriz Transforma(3,4);
    Calcula_J(IGAUSS, NumeroDePontosNodais, Nos);
    CoordenadasDoPontoDegauss(IGAUSS);
    for(int i=1;i<=NumeroDePontosNodais;i++)
    {
        LN.Atribui(1,2*i-1,LSHAPEXI(i,NumeroDePontosNodais,XI,ETA));
        LN.Atribui(2,2*i-1,LSHAPEETA(i,NumeroDePontosNodais,XI,ETA));
        LN.Atribui(3,2*i,LSHAPEXI(i,NumeroDePontosNodais,XI,ETA));
        LN.Atribui(4,2*i,LSHAPEETA(i,NumeroDePontosNodais,XI,ETA));
    }
    Transforma.Atribui(1,1,1);
    Transforma.Atribui(2,4,1);
    Transforma.Atribui(3,2,1);
    Transforma.Atribui(3,3,1);
    B = Transforma*(~J)*LN;
}

double PontoDeGauss_EPT::COORDENADAXI(int i)

```



```

{
// "i" indica o Numero do Ponto Nodal
if ((i == 1)||i == 4)||i == 8) return -1;
if ((i == 2)||i == 3)||i == 6) return 1;
if ((i == 5)||i == 7) return 0;
}

double PontoDeGauss_EPT::COORDENADAETA(int i)
{
// "i" indica o Numero do Ponto Nodal
if ((i == 1)||i == 2)||i == 5) return -1;
if ((i == 3)||i == 4)||i == 7) return 1;
if ((i == 6)||i == 8) return 0;
}

double PontoDeGauss_EPT::SHAPE(int i, int NumeroDePontosNodais, double XI, double ETA)
{
// "i" indica o Numero do Ponto Nodal
if (NumeroDePontosNodais == 4)
{
return (0.25*(1+XI*COORDENADAXI(i))*(1+ETA*COORDENADAETA(i)));
}
if (NumeroDePontosNodais == 8)
{
if (i==1) return -0.25*(1 - XI)*(1 - ETA)*(1 + XI + ETA);
if (i==2) return -0.25*(1 + XI)*(1 - ETA)*(1 - XI + ETA);
if (i==3) return -0.25*(1 + XI)*(1 + ETA)*(1 - XI - ETA);
if (i==4) return -0.25*(1 - XI)*(1 + ETA)*(1 + XI - ETA);
if (i==5) return 0.5*(1 - XI*XI)*(1 - ETA);
if (i==6) return 0.5*(1 + XI)*(1 - ETA*ETA);
if (i==7) return 0.5*(1 - XI*XI)*(1 + ETA);
if (i==8) return 0.5*(1 - XI)*(1 - ETA*ETA);
}
}

double PontoDeGauss_EPT::LSHAPEXI(int i, int NumeroDePontosNodais, double XI, double ETA)
{
// "i" indica o Numero do Ponto Nodal
if (NumeroDePontosNodais == 4)
{
return (0.25*(COORDENADAXI(i))*(1+ETA*COORDENADAETA(i)));
}
if (NumeroDePontosNodais == 8)
{
if (i==1) return 0.25*(1 - ETA)*(2*XI + ETA);
if (i==2) return 0.25*(1 - ETA)*(2*XI - ETA);
if (i==3) return 0.25*(1 + ETA)*(2*XI + ETA);
if (i==4) return 0.25*(1 + ETA)*(2*XI - ETA);
if (i==5) return -XI*(1 - ETA);
if (i==6) return 0.5*(1 - ETA*ETA);
if (i==7) return -XI*(1 + ETA);
if (i==8) return -0.5*(1 - ETA*ETA);
}
}

double PontoDeGauss_EPT::LSHAPEETA(int i, int NumeroDePontosNodais, double XI, double ETA)
{
// "i" indica o Numero do Ponto Nodal
if (NumeroDePontosNodais == 4)

```

```

    {
        return (0.25*(1+XI*COORDENADAXI(i))*(COORDENADAETA(i)));
    }
if (NumeroDePontosNodais == 8)
{
    if (i==1) return 0.25*(1 - XI)*(XI + 2*ETA);
    if (i==2) return -0.25*(1 + XI)*(XI - 2*ETA);
    if (i==3) return 0.25*(1 + XI)*(XI + 2*ETA);
    if (i==4) return -0.25*(1 - XI)*(XI - 2*ETA);
    if (i==5) return -0.5*(1 - XI*XI);
    if (i==6) return -ETA*(1 + XI);
    if (i==7) return 0.5*(1 - XI*XI);
    if (i==8) return -ETA*(1 - XI);
}
}

void PontoDeGauss_EPT::CalculaForcasInternas(int IGAUSS, int NumeroDePontosNodais, No * Nos,
Material &Mat)
{
    Calcula_Tensoes(IGAUSS, NumeroDePontosNodais, Nos, Mat);
    Fint = !B*sigma*J.Determinante()*WXI*WETA;
}

////////////////////////////////////
/// Métodos da Classe PontoDeGauss_AXISSIMETRICO ////
////////////////////////////////////

PontoDeGauss_AXISSIMETRICO::PontoDeGauss_AXISSIMETRICO()
{
    coordenada.redim(3);
    coordenada(1) = -sqrt(3.0/5.0);
    coordenada(3) = sqrt(3.0/5.0);
    pesos.redim(3);
    pesos(1) = 5.0/9.0;
    pesos(2) = 8.0/9.0;
    pesos(3) = 5.0/9.0;
    sigma.redim(4);
    J.redim(5,5);
    D.redim(4,4);
    S1 = 0;
    S2 = 0;
    S3 = 0;
}

void PontoDeGauss_AXISSIMETRICO::Inicializa(int NumeroDePontosNodais)
{
    Fint.redim(2*NumeroDePontosNodais);
    K.redim(2*NumeroDePontosNodais,2*NumeroDePontosNodais);
    LN.redim(5,2*NumeroDePontosNodais);
    B.redim(4,2*NumeroDePontosNodais);
}

void PontoDeGauss_AXISSIMETRICO::Calcula_K(int IGAUSS, int NumeroDePontosNodais, No *
Nos, Material &Mat)
{
    Calcula_D(Mat);
    Calcula_B(IGAUSS, NumeroDePontosNodais, Nos);
    matriz JI(3,3);
    for(int i=1;i<=3;i++)

```

```

    { for(int j=1;j<=3;j++)
      JI(i,j)=J(i+2,j+2);
    }
    K = !B*D*B*JI.Determinante()*2*PI*WXI*WETA;
  }

void PontoDeGauss_AXISSIMETRICO::Calcula_B(int IGAUSS, int NumeroDePontosNodais, No *
Nos)
{
  matriz Transforma(4,5);
  Calcula_J(IGAUSS, NumeroDePontosNodais, Nos);
  CoordenadasDoPontoDegauss(IGAUSS);
  for(int i=1;i<=NumeroDePontosNodais;i++)
  {
    LN.Atribui(1,2*i-1,LSHAPEXI(i,NumeroDePontosNodais,XI,ETA));
    LN.Atribui(2,2*i-1,LSHAPEETA(i,NumeroDePontosNodais,XI,ETA));
    LN.Atribui(3,2*i,LSHAPEXI(i,NumeroDePontosNodais,XI,ETA));
    LN.Atribui(4,2*i,LSHAPEETA(i,NumeroDePontosNodais,XI,ETA));
    LN.Atribui(5,2*i-1,SHAPE(i,NumeroDePontosNodais,XI,ETA));
  }
  Transforma.Atribui(1,1,1);
  Transforma.Atribui(2,4,1);
  Transforma.Atribui(3,2,1);
  Transforma.Atribui(3,3,1);
  Transforma.Atribui(4,5,1);
  B = Transforma*(~J)*LN;
}

void PontoDeGauss_AXISSIMETRICO::Calcula_J(int IGAUSS, int NumeroDePontosNodais, No * Nos)
{
  CoordenadasDoPontoDegauss(IGAUSS);
  J.ZeraMatriz();
  for (int i=1;i<=NumeroDePontosNodais;i++)
  {
    J.Adiciona(1,1,LSHAPEXI(i, NumeroDePontosNodais, XI, ETA)*Nos[i].x);
    J.Adiciona(1,2,LSHAPEXI(i, NumeroDePontosNodais, XI, ETA)*Nos[i].y);
    J.Adiciona(2,1,LSHAPEETA(i, NumeroDePontosNodais, XI, ETA)*Nos[i].x);
    J.Adiciona(2,2,LSHAPEETA(i, NumeroDePontosNodais, XI, ETA)*Nos[i].y);
    J.Adiciona(3,3,LSHAPEXI(i, NumeroDePontosNodais, XI, ETA)*Nos[i].x);
    J.Adiciona(3,4,LSHAPEXI(i, NumeroDePontosNodais, XI, ETA)*Nos[i].y);
    J.Adiciona(4,3,LSHAPEETA(i, NumeroDePontosNodais, XI, ETA)*Nos[i].x);
    J.Adiciona(4,4,LSHAPEETA(i, NumeroDePontosNodais, XI, ETA)*Nos[i].y);
    J.Adiciona(5,5,SHAPE(i, NumeroDePontosNodais, XI, ETA)*Nos[i].x);
  }
  if (J(5,5) != 0.0)
    J.Adiciona(5,5,0.000001);
}

void PontoDeGauss_AXISSIMETRICO::Calcula_D(Material &Mat)
{
  double fator = Mat.MR*(1.0 - Mat.POISS)/((1.0 + Mat.POISS)*(1.0 - 2*Mat.POISS));
  D.Atribui(1,1,fator);
  D.Atribui(1,2,D(1,1)*Mat.POISS/(1.0 - Mat.POISS));
  D.Atribui(2,1,D(1,2));
  D.Atribui(2,2,D(1,1));
  D.Atribui(3,3,fator*(1.0 - 2*Mat.POISS)/(2.0*(1-Mat.POISS)));
  D.Atribui(1,4,D(1,2));
  D.Atribui(2,4,D(1,2));
  D.Atribui(4,1,D(1,2));
}

```

```

        D.Atribui(4,2,D(1,2));
        D.Atribui(4,4,D(1,1));
    }

void PontoDeGauss_AXISSIMETRICO::Calcula_Tensoes(int IGAUSS, int NumeroDePontosNodais, No
* Nos, Material &Mat)
{
    Calcula_D(Mat);
    Calcula_B(IGAUSS, NumeroDePontosNodais, Nos);
    u.redim(2*NumeroDePontosNodais);
    for (int i=1;i<=NumeroDePontosNodais;i++)
    {
        u.Atribui(2*i-1,Nos[i].u);
        u.Atribui(2*i,Nos[i].v);
    }
    sigma = D*B*u;
}

void PontoDeGauss_AXISSIMETRICO::CalculaForcasInternas(int IGAUSS, int
NumeroDePontosNodais, No * Nos, Material &Mat)
{
    Calcula_Tensoes(IGAUSS, NumeroDePontosNodais, Nos, Mat);
    matriz Jl(3,3);
    for(int i=1;i<=3;i++)
    { for(int j=1;j<=3;j++)
        Jl(i,j)=J(i+2,j+2);
    }
    Fint = !B*sigma*Jl.Determinante()*2*PI*WXI*WETA;
}

void PontoDeGauss_AXISSIMETRICO::tensoes_princ()
{
    double c0, c1, c2, Sr, Sz, Steta, Trz;
    Sr = sigma(1);
    Sz = sigma(2);
    Trz = sigma(3);
    Steta = sigma(4);
    if ((Sr== 0)&&(Sz== 0)&&(Trz== 0)&&(Steta== 0))
    { S1 = 0;
      S2 = 0;
      S3 = 0;
    }
    else
    { c2 = Sr + Sz + Steta;
      c1 = Sr*Steta + Sz*Steta + Sr*Sz - Trz*Trz;
      c0 = Sr*Sz*Steta - Steta*Trz*Trz;
      c2 *= -1;
      c0 *= -1;
      double P, Q, DELTA, X1, X2, X3;
      P = (3*c1 - c2*c2)/9;
      Q = c2*c1/6 - pow(c2,3)/27 - c0/2;
      DELTA = pow(P,3) + Q*Q;
      if (DELTA<0)
      { double TETA = acos(Q/sqrt(pow(-P,3)));
        X1 = 2*sqrt(-P)*cos(TETA/3) - c2/3;
        X2 = 2*sqrt(-P)*cos((TETA+2*PI)/3) - c2/3;
        X3 = 2*sqrt(-P)*cos((TETA+4*PI)/3) - c2/3;
      }
      else

```

```

{ X1 = pow((Q + sqrt(DELTA)),1/3) + pow((Q - sqrt(DELTA)),1/3) - c2/3;
  X2 = (-0.5)*(pow((Q + sqrt(DELTA)),1/3) + pow((Q - sqrt(DELTA)),1/3)) - c2/3;
  X3 = X2;
}
if((X1>=X2))
{ if(X2>X3)
  { S1 = X3;
    S2 = X2;
    S3 = X1;
  }
  else
  { S1 = X2;
    if(X1>X3)
    { S2 = X3;
      S3 = X1;
    }
    else
    { S2 = X1;
      S3 = X3;
    }
  }
}
else
{ if(X1>X3)
  { S1 = X3;
    S2 = X1;
    S3 = X2;
  }
  else
  { S1 = X1;
    if(X2>X3)
    { S2 = X3;
      S3 = X2;
    }
    else
    { S2 = X2;
      S3 = X3;
    }
  }
}
}
}

```

```

matriz PontoDeGauss_AXISSIMETRICO::tensoes_princ(double Sr, double Sz, double Trz, double Steta)
{
  double S1, S2, S3;
  if ((Sr= =0)&&(Sz= =0)&&(Trz= =0)&&(Steta= =0))
  { S1 = 0;
    S2 = 0;
    S3 = 0;
  }
  else
  { double c0, c1, c2;
    c2 = Sr + Sz + Steta;
    c1 = Sr*Steta + Sz*Steta + Sr*Sz - Trz*Trz;
    c0 = Sr*Sz*Steta - Steta*Trz*Trz;
    c2 *= -1;
    c0 *= -1;
    double P, Q, DELTA, X1, X2, X3;

```

```

P = (3*c1 - c2*c2)/9;
Q = c2*c1/6 - pow(c2,3)/27 - c0/2;
DELTA = pow(P,3) + Q*Q;
if (DELTA<0)
{ double TETA = acos(Q/sqrt(pow(-P,3)));
  X1 = 2*sqrt(-P)*cos(TETA/3) - c2/3;
  X2 = 2*sqrt(-P)*cos((TETA+2*PI)/3) - c2/3;
  X3 = 2*sqrt(-P)*cos((TETA+4*PI)/3) - c2/3;
}
else
{ X1 = pow((Q + sqrt(DELTA)),1/3) + pow((Q - sqrt(DELTA)),1/3) - c2/3;
  X2 = (-0.5)*(pow((Q + sqrt(DELTA)),1/3) + pow((Q - sqrt(DELTA)),1/3)) - c2/3;
  X3 = X2;
}
if((X1>=X2))
{ if(X2>X3)
  { S1 = X3;
    S2 = X2;
    S3 = X1;
  }
  else
  { S1 = X2;
    if(X1>X3)
    { S2 = X3;
      S3 = X1;
    }
    else
    { S2 = X1;
      S3 = X3;
    }
  }
}
else
{ if(X1>X3)
  { S1 = X3;
    S2 = X1;
    S3 = X2;
  }
  else
  { S1 = X1;
    if(X2>X3)
    { S2 = X3;
      S3 = X2;
    }
    else
    { S2 = X2;
      S3 = X3;
    }
  }
}
}
matriz tensoes(3);
tensoes(1) = S1;
tensoes(2) = S2;
tensoes(3) = S3;
return tensoes;
}

```

8.2.7 CLASSE ELEMENTO

```
class Elemento
{
public:
    int NumeroDePontosNodais;// Número de Nós Que Definem o Elemento
    int NumeroDePontosDeGauss;// Número de Pontos de Gauss do Elemento
    int camada; // Número da Camada na qual o Elemento pertence
    int fileira; // Número da Fileira de Elementos onde se encontra o Elemento dentro de sua Camada
    int coluna; // Número da Coluna na qual o Elemento pertence
    int* NP;// Incidência dos Pontos Nodais
    No * NoLocal; // Armazena Informações Globais Sobre os Nós do Elemento
    PontoDeGauss *PontosDeGauss;// Armazena os Pontos de Gauss
    virtual void CalculaRigidez(matriz &KGLOBAL, No * Nos, Material &Mat){};
    // Método Para Cálculo da Rigidez
    virtual void Incidencia(No * Nos){}; // Determina a Incidência Nodal do Elemento
    virtual void CalculaForçasInternas(matriz &FintGLOBAL, No * Nos, Material &Mat){};
    virtual void CalculaTensoes(matriz &FintGLOBAL, No * Nos, Material &Mat){};
    // Calcula as Tensões no Elemento
    virtual void LeDados(ifstream &arquivo_entrada, ofstream &arquivo_saida, ofstream
&arquivo_view){};
    virtual void GeraResultados(){};
    // Le os Dados do Elemento
    virtual void redim(int n){}; // Redimensiona as matrizes protected (IND, K, Fint)
protected:
    matriz IND; // Vetor de Incidências
    matriz K; // Matriz de Rigidez
    matriz Fint; // Vetor de Forças Internas
};

class Elemento_EPT : public Elemento
{
public:
    PontoDeGauss_EPT *PontosDeGauss;// Armazena os Pontos de Gauss
    Elemento_EPT();
    // Implementação dos Métodos Herdados da Classe-Base
    virtual void Incidencia(No * Nos);
    virtual void CalculaRigidez(matriz &KGLOBAL, No * Nos, Material &Mat);
    virtual void LeDados(ifstream &arquivo_entrada, ofstream &arquivo_saida, ofstream
&arquivo_view){};
    virtual void CalculaTensoes(matriz &FintGLOBAL, No * Nos, Material &Mat);
    virtual void CalculaForçasInternas(matriz &FintGLOBAL, No * Nos, Material &Mat);
    virtual void redim(int n);
};

class Elemento_AXISSIMETRICO : public Elemento_EPT
{
public:
    PontoDeGauss_AXISSIMETRICO *PontosDeGauss;// Armazena os Pontos de Gauss
    Elemento_AXISSIMETRICO();
    void calcula_MR(double Sdmin, double S3min, Material &Mat);
    virtual void CalculaRigidez(matriz &KGLOBAL, No * Nos, Material &Mat);
    virtual void CalculaTensoes(matriz &FintGLOBAL, No * Nos, Material &Mat);
    virtual void CalculaForçasInternas(matriz &FintGLOBAL, No * Nos, Material &Mat);
    matriz Tensoes(double x, double y); // Calcula as tensões axissimétricas em (x,y) global
    matriz TensoesCoordLocal(double XI, double ETA);//Calcula as tensões axissimétricas em (XI,ETA)
local
    double TensaoMax(int SupInf, int sigma);//Calcula a tensão máxima do tipo sigma no topo ou no fundo
do elemento
```

```

    matriz DeformacoesPrinc(double x, double y, Material &Mat);//Calcula as deformações principais em
(x,y) global
    matriz DefPrincLocal(double XI, double ETA, Material &Mat);//Calcula as deformações principais em
(XI,ETA) local
    double DeformacaoMax(int SupInf, int def, Material &Mat);//Calcula a deformação máxima do tipo def
no topo ou no fundo do elemento
    double Deflexao(double x, double y);//Calcula a deflexão ou deslocamento vertical em (x,y) global
    double DeflexaoCoordLocal(double XI, double ETA);//Calcula a deflexão ou deslocamento vertical em
(XI,ETA) local
    double DeflexaoMax(int SupInf = 1);//Calcula a deflexão máxima no topo ou no fundo do elemento
    double MaxPositivo(double X1, double X2, double X3 = 0);//Calcula o valor máximo positivo de X,
onde X1, X2 e X3 são os valores de X
    // nos nós do elemento
    double MaxNegativo(double X1, double X2, double X3 = 0);//Calcula o valor máximo negativo de X,
onde X1, X2 e X3 são os valores de X
    // nos nós do elemento
};

////////////////////////////////////
/// Métodos da Classe Elemento ///
////////////////////////////////////

////////////////////////////////////
/// Métodos da Classe Elemento_EPT ///
////////////////////////////////////

Elemento_EPT::Elemento_EPT()
{
    NumeroDePontosDeGauss = 8;// Interpolação quadrática
    PontosDeGauss = new PontoDeGauss_EPT[NumeroDePontosDeGauss+1];
}

void Elemento_EPT::Incidencia(No * Nos)
{
    // Incidencia para elemento quadrilatero plano
    for(int i=1;i<=NumeroDePontosNodais;i++)
    {
        NoLocal[i] = Nos[NP[i]];
        IND.Atribui(2*i,2*NP[i]);
        IND.Atribui(2*i-1,2*NP[i]-1);
    }
}

void Elemento_EPT::CalculaRigidez(matriz &KGLOBAL, No * Nos, Material &Mat)
{
    Incidencia(Nos);
    K.ZeraMatriz();
    for(int IGAUSS=1;IGAUSS<=NumeroDePontosDeGauss;IGAUSS++)
    {
        PontosDeGauss[IGAUSS].Calcula_K(IGAUSS, NumeroDePontosNodais, NoLocal, Mat);
        K += PontosDeGauss[IGAUSS].K;
    }
    // Transfere a Sua Contribuição Para a Matriz Global da Estrutura
    for (int i=1;i<=2*NumeroDePontosNodais;i++)
    {
        for (int j=1;j<=2*NumeroDePontosNodais;j++)
        {
            KGLOBAL.Adiciona(int(IND(i)),int(IND(j)),K(i,j));
        }
    }
}

```



```

    }
}

void Elemento_EPT::CalculaTensoes(matriz &FintGLOBAL, No * Nos, Material &Mat)
{
    Incidencia(Nos);
    for(int IGAUSS=1;IGAUSS<=NumeroDePontosDeGauss;IGAUSS++)
    {
        PontosDeGauss[IGAUSS].Calcula_Tensoes(IGAUSS, NumeroDePontosNodais, NoLocal, Mat);
    }
}

void Elemento_EPT::CalculaForcasInternas(matriz &FintGLOBAL, No * Nos, Material &Mat)
{
    Fint.ZeraMatriz();
    for(int IGAUSS=1;IGAUSS<=NumeroDePontosDeGauss;IGAUSS++)
    {
        PontosDeGauss[IGAUSS].CalculaForcasInternas(IGAUSS, NumeroDePontosNodais, NoLocal,
Mat);
        Fint += PontosDeGauss[IGAUSS].Fint;
    }
    // Transfere a Sua Contribuição Para a Matriz Global da Estrutura
    for (int i=1;i<=2*NumeroDePontosNodais;i++)
    {
        FintGLOBAL.Adiciona(int(IND(i)),Fint(i));
    }
}

void Elemento_EPT::redim(int n)
{
    K.redim(2*n,2*n);
    Fint.redim(2*n);
    IND.redim(2*n);
}

////////////////////////////////////
// Métodos da Classe Elemento_AXISSIMETRICO ////
////////////////////////////////////

Elemento_AXISSIMETRICO::Elemento_AXISSIMETRICO()
{
    NumeroDePontosDeGauss = 8; // Interpolação quadrática
    PontosDeGauss = new PontoDeGauss_AXISSIMETRICO[NumeroDePontosDeGauss+1];
}

void Elemento_AXISSIMETRICO::calcula_MR(double Sdmin, double S3min, Material &Mat)
{
    double MR = 0;
    double MRant = Mat.MR;
    double s1, s2, s3;
    for (int i=1;i<=NumeroDePontosDeGauss;i++)
    {
        PontosDeGauss[i].tensoes_princ();
        s1 = -PontosDeGauss[i].S1;
        s2 = -PontosDeGauss[i].S2;
        s3 = -PontosDeGauss[i].S3;
        if (s3<S3min) s3=S3min;
        if (s2<S3min) s2=S3min;
        if (s1-s3<Sdmin) s1=Sdmin+s3;
        Mat.calcula_MR(s1,s2,s3);
    }
}

```

```

    MR += Mat.MR/NumeroDePontosDeGauss;
}
Mat.MR = MR;
Mat.MRant = MRant;
}

void Elemento_AXISSIMETRICO::CalculaRigidez(matriz &KGLOBAL, No * Nos, Material &Mat)
{
    Incidencia(Nos);
    K.ZeraMatriz();
    for(int IGAUSS=1;IGAUSS<=NumeroDePontosDeGauss;IGAUSS++)
    {
        PontosDeGauss[IGAUSS].Calcula_K(IGAUSS, NumeroDePontosNodais, NoLocal, Mat);
        K += PontosDeGauss[IGAUSS].K;
    }
    // Transfere a Sua Contribuição Para a Matriz Global da Estrutura
    for (int i=1;i<=2*NumeroDePontosNodais;i++)
    {
        for (int j=1;j<=2*NumeroDePontosNodais;j++)
        {
            KGLOBAL.Adiciona(int(IND(i)),int(IND(j)),K(i,j));
        }
    }
}

void Elemento_AXISSIMETRICO::CalculaTensoes(matriz &FintGLOBAL, No * Nos, Material &Mat)
{
    Incidencia(Nos);
    for(int IGAUSS=1;IGAUSS<=NumeroDePontosDeGauss;IGAUSS++)
    {
        PontosDeGauss[IGAUSS].Calcula_Tensoes(IGAUSS, NumeroDePontosNodais, NoLocal, Mat);
    }
}

void Elemento_AXISSIMETRICO::CalculaForcasInternas(matriz &FintGLOBAL, No * Nos, Material
&Mat)
{
    Fint.ZeraMatriz();
    for(int IGAUSS=1;IGAUSS<=NumeroDePontosDeGauss;IGAUSS++)
    {
        PontosDeGauss[IGAUSS].CalculaForcasInternas(IGAUSS, NumeroDePontosNodais, NoLocal,
Mat);
        Fint += PontosDeGauss[IGAUSS].Fint;
    }
    // Transfere a Sua Contribuição Para a Matriz Global da Estrutura
    for (int i=1;i<=2*NumeroDePontosNodais;i++)
    {
        FintGLOBAL.Adiciona(int(IND(i)),Fint(i));
    }
}

matriz Elemento_AXISSIMETRICO::Tensoes(double x, double y)
{
    double XI, ETA;
    XI = (x - (NoLocal[3].x + NoLocal[1].x)/2)/((NoLocal[3].x - NoLocal[1].x)/2);
    ETA = (y - (NoLocal[3].y + NoLocal[1].y)/2)/((NoLocal[3].y - NoLocal[1].y)/2);
    matriz S(4);
    S = TensoesCoordLocal(XI,ETA);
    return S;
}

```

```

}

matriz Elemento_AXISSIMETRICO::TensoesCoordLocal(double XI, double ETA)
{
    matriz N, S, sigma;
    if (NumeroDePontosDeGauss== =4)
    {
        N.redim(1,4);
        S.redim(1,4);
        sigma.redim(4,4);
        XI *= sqrt(3);
        ETA *= sqrt(3);
        N(1,1) = 0.25*(1 - XI)*(1 - ETA);
        N(1,2) = 0.25*(1 + XI)*(1 - ETA);
        N(1,3) = 0.25*(1 + XI)*(1 + ETA);
        N(1,4) = 0.25*(1 - XI)*(1 + ETA);
        for(int i=1;i<=4;i++)
        {
            for(int j=1;j<=4;j++)
                sigma(i,j) = PontosDeGauss[i].sigma(j);
        }
        S = N * sigma;
        return !(S);
    }
    if (NumeroDePontosDeGauss== =8)
    {
        N.redim(1,8);
        S.redim(1,4);
        sigma.redim(8,4);
        XI *= sqrt(5/3);
        ETA *= sqrt(5/3);
        N(1,1) = -0.25*(1 - XI)*(1 - ETA)*(1 + XI + ETA);
        N(1,2) = -0.25*(1 + XI)*(1 - ETA)*(1 - XI + ETA);
        N(1,3) = -0.25*(1 + XI)*(1 + ETA)*(1 - XI - ETA);
        N(1,4) = -0.25*(1 - XI)*(1 + ETA)*(1 + XI - ETA);
        N(1,5) = 0.5*(1 - XI*XI)*(1 - ETA);
        N(1,6) = 0.5*(1 + XI)*(1 - ETA*ETA);
        N(1,7) = 0.5*(1 - XI*XI)*(1 + ETA);
        N(1,8) = 0.5*(1 - XI)*(1 - ETA*ETA);
        for(int i=1;i<=8;i++)
        {
            for(int j=1;j<=4;j++)
                sigma(i,j) = PontosDeGauss[i].sigma(j);
        }
        S = N * sigma;
        return !(S);
    }
}

double Elemento_AXISSIMETRICO::TensaoMax(int SupInf, int sigma)
{//Calcula a tensão referente a sigma (1=Sr,2=Sz,3=Trz,4=Steta,5=S1,6=S2,7=S3,8=S1-S3)
    double tensao, ETA;
    if (SupInf== =1)
        ETA = -1;
    else
        ETA = 1;
    if (sigma<=4)
    {
        if (NumeroDePontosDeGauss== =4)
        {
            double X1, X2;
            matriz S(4);
            S = TensoesCoordLocal(-1,ETA);
            X1 = S(sigma);
            S = TensoesCoordLocal(1,ETA);
        }
    }
}

```

```

X2 = S(sigma);
tensao = MaxNegativo(X1,X2);
if (sigma!=2)
{ double X;
  X = MaxPositivo(X1,X2);
  if (fabs(X)>fabs(tensao))
    tensao = X;
}
}
else
{ matriz S(4);
  double X1, X2, X3;
  S = TensoesCoordLocal(-1,ETA);
  X1 = S(sigma);
  S = TensoesCoordLocal(0,ETA);
  X2 = S(sigma);
  S = TensoesCoordLocal(1,ETA);
  X3 = S(sigma);
  tensao = MaxNegativo(X1,X2,X3);
  if (sigma!=2)
  { double X;
    X = MaxPositivo(X1,X2,X3);
    if (fabs(X)>fabs(tensao))
      tensao = X;
  }
}
}
else
{ if (NumeroDePontosDeGauss== 4)
  { matriz S(4);
    matriz ten(3);
    double X1, X2;
    S = TensoesCoordLocal(-1,ETA);
    ten = PontosDeGauss[1].tensoes_princ(S(1),S(2),S(3),S(4));
    if (sigma== 8)
      X1 = ten(1) - ten(3);
    else
      X1 = ten(sigma-4);
    S = TensoesCoordLocal(1,ETA);
    ten = PontosDeGauss[1].tensoes_princ(S(1),S(2),S(3),S(4));
    if (sigma== 8)
      X2 = ten(1) - ten(3);
    else
      X2 = ten(sigma-4);
    tensao = MaxNegativo(X1,X2);
    if (sigma== 6)
    { double X;
      X = MaxPositivo(X1,X2);
      if (fabs(X)>fabs(tensao))
        tensao = X;
    }
    if (sigma== 7)
      tensao = MaxPositivo(X1,X2);
  }
}
else
{ matriz S(4);
  matriz ten(3);
  double X1, X2, X3;
  S = TensoesCoordLocal(-1,ETA);

```

```

ten = PontosDeGauss[1].tensoes_princ(S(1),S(2),S(3),S(4));
if (sigma==8)
    X1 = ten(1) - ten(3);
else
    X1 = ten(sigma-4);
S = TensoesCoordLocal(0,ETA);
ten = PontosDeGauss[1].tensoes_princ(S(1),S(2),S(3),S(4));
if (sigma==8)
    X2 = ten(1) - ten(3);
else
    X2 = ten(sigma-4);
S = TensoesCoordLocal(1,ETA);
ten = PontosDeGauss[1].tensoes_princ(S(1),S(2),S(3),S(4));
if (sigma==8)
    X3 = ten(1) - ten(3);
else
    X3 = ten(sigma-4);
tensao = MaxNegativo(X1,X2,X3);
if (sigma==6)
{ double X;
  X = MaxPositivo(X1,X2,X3);
  if (fabs(X)>fabs(tensao))
    tensao = X;
}
if (sigma==7)
    tensao = MaxPositivo(X1,X2,X3);
}
}
return tensao;
}

```

```

matriz Elemento_AXISSIMETRICO::DeformacoesPrinc(double x, double y, Material &Mat)
{
    double XI, ETA;
    XI = (x - (NoLocal[3].x + NoLocal[1].x)/2)/((NoLocal[3].x - NoLocal[1].x)/2);
    ETA = (y - (NoLocal[3].y + NoLocal[1].y)/2)/((NoLocal[3].y - NoLocal[1].y)/2);
    matriz Def(3);
    Def = DefPrincLocal(XI,ETA,Mat);
    return Def;
}

```

```

matriz Elemento_AXISSIMETRICO::DefPrincLocal(double XI, double ETA, Material &Mat)
{
    matriz TenAxi(4);
    matriz TenPrinc(3);
    matriz Def(3);
    matriz E(3,3);
    TenAxi = TensoesCoordLocal(XI,ETA);
    TenPrinc = PontosDeGauss[1].tensoes_princ(TenAxi(1),TenAxi(2),TenAxi(3),TenAxi(4));
    for(int i=1;i<=3;i++)
    { for(int j=1;j<=3;j++)
      { if (i==j)
        E(i,j) = 1/Mat.MR;
        else
        E(i,j) = -1*Mat.POISS/Mat.MR;
      }
    }
    Def = E*TenPrinc;
    return Def;
}

```

```

}

double Elemento_AXISSIMETRICO::DeformacaoMax(int SupInf, int def, Material &Mat)
{
    double deformacao, ETA;
    matriz D(3);
    if (SupInf== =1)
        ETA = -1;
    else
        ETA = 1;
    if (NumeroDePontosDeGauss== =4)
    { double X1, X2;
      D = DefPrincLocal(-1,ETA,Mat);
      X1 = D(def);
      D = DefPrincLocal(1,ETA,Mat);
      X2 = D(def);
      deformacao = MaxNegativo(X1,X2);
      if (def== =2)
      { double X;
        X = MaxPositivo(X1,X2);
        if (fabs(X)>fabs(deformacao))
            deformacao = X;
        }
      if (def== =3)
          deformacao = MaxPositivo(X1,X2);
    }
    else
    { double X1, X2, X3;
      D = DefPrincLocal(-1,ETA,Mat);
      X1 = D(def);
      D = DefPrincLocal(0,ETA,Mat);
      X2 = D(def);
      D = DefPrincLocal(1,ETA,Mat);
      X3 = D(def);
      deformacao = MaxNegativo(X1,X2,X3);
      if (def== =2)
      { double X;
        X = MaxPositivo(X1,X2,X3);
        if (fabs(X)>fabs(deformacao))
            deformacao = X;
        }
      if (def== =3)
          deformacao = MaxPositivo(X1,X2,X3);
    }
    return deformacao;
}

double Elemento_AXISSIMETRICO::Deflexao(double x, double y)
{
    double XI, ETA, Def;
    XI = (x - (NoLocal[3].x + NoLocal[1].x)/2)/((NoLocal[3].x - NoLocal[1].x)/2);
    ETA = (y - (NoLocal[3].y + NoLocal[1].y)/2)/((NoLocal[3].y - NoLocal[1].y)/2);
    Def = DeflexaoCoordLocal(XI,ETA);
    return Def;
}

double Elemento_AXISSIMETRICO::DeflexaoCoordLocal(double XI, double ETA)
{
    matriz N, DesVert, Def;

```

```

if (NumeroDePontosDeGauss==4)
{
  N.redim(1,4);
  DesVert.redim(4);
  N(1,1) = 0.25*(1 - XI)*(1 - ETA);
  N(1,2) = 0.25*(1 + XI)*(1 - ETA);
  N(1,3) = 0.25*(1 + XI)*(1 + ETA);
  N(1,4) = 0.25*(1 - XI)*(1 + ETA);
  for(int i=1;i<=4;i++)
    DesVert(i) = NoLocal[i].v;
  Def = N * DesVert;
  return Def(1);
}
if (NumeroDePontosDeGauss==8)
{
  N.redim(1,8);
  DesVert.redim(8);
  N(1,1) = -0.25*(1 - XI)*(1 - ETA)*(1 + XI + ETA);
  N(1,2) = -0.25*(1 + XI)*(1 - ETA)*(1 - XI + ETA);
  N(1,3) = -0.25*(1 + XI)*(1 + ETA)*(1 - XI - ETA);
  N(1,4) = -0.25*(1 - XI)*(1 + ETA)*(1 + XI - ETA);
  N(1,5) = 0.5*(1 - XI*XI)*(1 - ETA);
  N(1,6) = 0.5*(1 + XI)*(1 - ETA*ETA);
  N(1,7) = 0.5*(1 - XI*XI)*(1 + ETA);
  N(1,8) = 0.5*(1 - XI)*(1 - ETA*ETA);
  for(int i=1;i<=8;i++)
    DesVert(i) = NoLocal[i].v;
  Def = N * DesVert;
  return Def(1);
}
}

double Elemento_AXISSIMETRICO::DeflexaoMax(int SupInf)
{
  double Def, ETA;
  if (SupInf== 1)
    ETA = -1;
  else
    ETA = 1;
  if (NumeroDePontosDeGauss==4)
  {
    double X1, X2;
    X1 = DeflexaoCoordLocal(-1,ETA);
    X2 = DeflexaoCoordLocal(1,ETA);
    Def = MaxPositivo(X1,X2);
  }
  else
  {
    double X1, X2, X3;
    X1 = DeflexaoCoordLocal(-1,ETA);
    X2 = DeflexaoCoordLocal(0,ETA);
    X3 = DeflexaoCoordLocal(1,ETA);
    Def = MaxPositivo(X1,X2,X3);
  }
  return Def;
}

double Elemento_AXISSIMETRICO::MaxPositivo(double X1, double X2, double X3)
{
  if (NumeroDePontosDeGauss==4)
  {
    if (X1>X2)
      return X1;
    else

```

```

    return X2;
}
else
{ double a;
  a = (X1 + X3)/2 - X2;
  if (a==0)
  { if (X1>X2)
    return X1;
    else
    return X2;
  }
  else
  { double max, x;
    x = (X1 - X3)/(4*a);
    if ((x<-1)|| (x>1))
      max = X2;
    else
      max = a*x*x + 0.5*(X3 - X1)*x + X2;
    if (max<X1)
      max = X1;
    if (max<X3)
      max = X3;
    return max;
  }
}
}
}

```

```

double Elemento_AXISSIMETRICO::MaxNegativo(double X1, double X2, double X3)
{
  if (NumeroDePontosDeGauss==4)
  { if (X1<X2)
    return X1;
    else
    return X2;
  }
  else
  { double a;
    a = (X1 + X3)/2 - X2;
    if (a==0)
    { if (X1<X2)
      return X1;
      else
      return X2;
    }
    else
    { double max, x;
      x = (X1 - X3)/(4*a);
      if ((x<-1)|| (x>1))
        max = X2;
      else
        max = a*x*x + 0.5*(X3 - X1)*x + X2;
      if (max>X1)
        max = X1;
      if (max>X3)
        max = X3;
      return max;
    }
  }
}
}

```


8.2.8 CLASSE ESTRUTURA

```
class Estrutura
{
public:
    int NSD; // Número de Dimensões Espaciais do Problems
    int NumeroDeElementos; // Número de Elementos da Estrutura
    int NumeroDePontosNodais; // Número de Nós da Estrutura
    int NumeroDeNosComRestricao; // Número de Nós com Restrição da Estrutura
    int NumeroDeRestricoes; // Número de restrições de deslocamento
    int NumeroDeCamadas; // Número de Camadas da Estrutura
    int Metodo; // Tipo de resolução
    int Iteracao; // Número de iterações ou número máximo de iterações
    double PENALIDADE; // Coeficiente de Penalidade
    double Sdmin, S3min; // Valores mínimos de Sd e S3 para o cálculo do Módulo de Resiliência
    char Nome[81]; // Título do Problema
    char TipoDeProblema[81]; // Tipo de Problema
    No* Nos; // Vetor de Nos da estrutura
    matriz d; // Vetor de Deslocamentos Nodais;
    matriz K; // Matriz de Rigidez
    matriz F; // Matriz de Cargas Nodais Externas Aplicadas
    matriz Fint; // Matriz de Forças Internas
    matriz Espessuras; // Matriz das espessuras de cada camada
    matriz rest; // Vetor dos graus de liberdade restritos
    matriz FApoio; // Matriz para o cálculo das reações de apoio
    //Elemento * Elementos; // Vetor de Elementos
    //Elemento_TRUSS2D * Elementos; // Vetor de Elementos;
    Elemento_AXISSIMETRICO * Elementos; // Vetor de Elementos;
    //Elemento_3D * Elementos; // Vetor de Elementos;
    Pavimento Pav;
    double a, q; // Raio da área de contato da roda e Pressão do pneu;
    double tolerancia; // Tolerância para a diferença entre os módulos novo e anterior
    ifstream arquivo_entrada;
    ofstream arquivo_saida;
    ofstream arquivo_view;
    Estrutura(); // Construtor da Classe
    ~Estrutura(); // Destrutor da Classe
    void LeDados(char * Dados, char * Resultados, char * Visualizacao);
    void LeDados(string Dados, string Resultados, string Visualizacao);
    void GeraMalha(); // Gera a malha de elementos com 8 nós por elemento
    void EscreveNos(); // Retorna os dados dos Pontos Nodais da Estrutura
    void LeMateriais(); // Lê os dados dos Materiais da Estrutura
    void EscreveElementos(); // Lê os dados dos Elementos da Estrutura
    void CalculaRigidez(); // Calcula a Matriz de Rigidez da estrutura
    void CalculaDeslocamentos(); // Calcula os Deslocamentos da Estrutura
    void Calcula(); // Resolve a Estrutura
    void CalculaTensoes(); // calcula as Tensões na Estrutura
    void MontaVetorDeForçasInternas(); // Monta o Vetor de Forças internas
    void AplicaRestricoes(); // Aplica as Restrições à estrutura
    void AplicaRestricoes2(); // Aplica as Restrições à estrutura eliminando as linhas e colunas da matriz de
rigidez
    void MontaVetorDeCargas(double fator = 1.0); // Monta Vetor de Cargas Externas
    void GeraResultados();
    void GeraResultados2();
    int ProcuraElemento(double x, double y); // Procura o elemento de coordenadas x e y
    int ProcuraElemento(double x, int c, int SupInf); // Procura o elemento com coordenada x pertencente à
primeira ou última fileira da camada c
```

```

    matriz TensoesAxissimetricas(double x, double y);// Calcula as tensões axissimétricas do ponto (x,y)
    matriz TensoesPrincipais(double x, double y);// Calcula as tensões principais do ponto (x,y)
    double TensoesMax(int c, int SupInf, int sigma);// Calcula a tensão máxima do tipo sigma no topo ou no
fundo da camada c
    double TensoesMax(int c, int f, int SupInf, int sigma);// Calcula a tensão máxima do tipo sigma no topo
ou no fundo da fileira f pertencente à camada c
    matriz DeformacoesPrincipais(double x, double y);// Calcula as deformações principais do ponto (x,y)
    double DeformacoesMax(int c, int SupInf, int def);// Calcula a deformação máxima do tipo def no topo
ou no fundo da camada c
    double DeformacoesMax(int c, int f, int SupInf, int def);// Calcula a deformação máxima do tipo def no
topo ou no fundo da fileira f pertencente à camada c
    double Deflexao(double x, double y);//Calcula a deflexão do ponto (x,y)
    double DeflexaoMax(int c, int SupInf = 1);//Calcula a deflexão máxima no topo ou no fundo da camada
c
    double DeflexaoMax(int c, int f, int SupInf);//Calcula a deflexão máxima no topo ou no fundo da fileira
f pertencente à camada c
    bool convergencia(double tol);
};

////////////////////////////////////
/// Métodos da Classe Estrutura ///
////////////////////////////////////

Estrutura::Estrutura()
{
    PENALIDADE = 1.0e20; // Define o Valor do Coeficiente de Penalidade
    Sdmin = 0.020;
    S3min = 0.001;
    Metodo = 1;
    Iteracao = 5;
    tolerancia = 0.01;
}

Estrutura::~Estrutura()
{
    arquivo_entrada.close();
    arquivo_saida.close();
    arquivo_view.close();
}

void Estrutura::CalculaRigidez()
{
    K.redim(NSD*NumeroDePontosNodais,NSD*NumeroDePontosNodais);
    for (int i=1;i<=NumeroDeElementos;i++)
    {
        Elementos[i].CalculaRigidez(K,      Nos,      Pav(Elementos[i].camada,Elementos[i].fileira,
Elementos[i].coluna));
    }
}

void Estrutura::GeraResultados()
{
    arquivo_saida << "\n" << "/Deslocamentos Nodais/" << endl;
    for (int i=1;i<=NumeroDePontosNodais;i++)
    {
        arquivo_saida << i << "\t";
        Nos[i].GeraResultados(arquivo_saida, arquivo_view, NSD);
    }
    MontaVetorDeForcasInternas();
}

```

```

arquivo_saida << "\n" << "/Reacoes de Apoio/" << endl;
for (int i=1;i<=NumeroDePontosNodais;i++)
{
    if ((Nos[i].restx == 1)||(Nos[i].resty == 1)||(Nos[i].restz == 1))
    {
        arquivo_saida << "Reações_do_Nó " << i;
        if (Nos[i].restx == 1)
        {
            arquivo_saida << "\t" << "Reacao_X : ";
            if (NSD == 2) arquivo_saida << Fint(2*i-1); else arquivo_saida << Fint(3*i-2);
        }
        if (Nos[i].resty == 1)
        {
            arquivo_saida << "\t" << "Reacao_Y : ";
            if (NSD == 2) arquivo_saida << Fint(2*i); else arquivo_saida << Fint(3*i-1);
        }
        if (Nos[i].restz == 1)
        {
            arquivo_saida << "\t" << "Reacao_Z : " << Fint(3*i);
        }
        arquivo_saida << endl;
    }
}
for (int i=1;i<=NumeroDeElementos;i++)
{
    Elementos[i].GeraResultados();
}
}

void Estrutura::GeraResultados2()
{
    arquivo_saida << "\n" << "/Deslocamentos Nodais/" << endl;
    for (int i=1;i<=NumeroDePontosNodais;i++)
    {
        arquivo_saida << i << "\t";
        Nos[i].GeraResultados(arquivo_saida, arquivo_view, NSD);
    }
    matriz RAp(NumeroDeRestricoes);
    RAp = FApoio*d;
    int l=1;
    arquivo_saida << "\n" << "/Reacoes de Apoio/" << endl;
    for (int i=1;i<=NumeroDePontosNodais;i++)
    {
        if ((Nos[i].restx == 1)||(Nos[i].resty == 1)||(Nos[i].restz == 1))
        {
            arquivo_saida << "Reações_do_Nó " << i;
            if (Nos[i].restx == 1)
            {
                arquivo_saida << "\t" << "Reacao_X : " << RAp(l++);
            }
            if (Nos[i].resty == 1)
            {
                arquivo_saida << "\t" << "Reacao_Y : " << RAp(l++);
            }
            if (Nos[i].restz == 1)
            {
                arquivo_saida << "\t" << "Reacao_Z : " << RAp(l++);
            }
            arquivo_saida << endl;
        }
    }
}

```

```

    }
  }
  for (int i=1;i<=NumeroDeElementos;i++)
  {
    Elementos[i].GeraResultados();
  }
}

void Estrutura::CalculaDeslocamentos()
{
  if (d.num_linhas==NSD*NumeroDePontosNodais)//Método da Penalidade
  // Resolve o Sistema F = Kd pelo método de Gauss-Seidel;
  { bool para = false;
    int k = 0;
    while (!para)
    { para = true;
      for (int i=1;i<=K.num_linhas;i++)
      { double XV = d(i);
        double soma = F(i);
        for (int j=1;j<=K.num_colunas;j++)
        { if (j!=i)
          soma -= K(i,j)*d(j);
        }
        d(i) = soma/K(i,i);
        if (XV==0)
        { if (fabs((d(i)-XV)/1e-5)>tolerancia)
          para = false;
        }
        else
        { if (fabs((d(i)-XV)/XV)>tolerancia)
          para = false;
        }
      }
      k++;
    }
    cout << "\n\t Foi encontrada a solucao na " << ++k << "a. iteracao do Metodo de Gauss-Seidel.";
    // Transfere os Valores dos Deslocamentos para os Nos

    for (int i=1;i<=NumeroDePontosNodais;i++)
    { if (NSD == 2)
      { Nos[i].u = d(2*i-1);
        Nos[i].v = d(2*i);
      }
      else
      { Nos[i].u = d(3*i-2);
        Nos[i].v = d(3*i-1);
        Nos[i].w = d(3*i);
      }
    }
  }
  else
  { //Resolve o Sistema F = Kd
    d = F / K;
    // Transfere os Valores dos Deslocamentos para os Nos

    if (NSD==2)
    { int l=0;
      for (int i=1;i<=NumeroDePontosNodais;i++)
      { if (l<NumeroDeRestricoes)

```

```

    { if ((2*i-1) == rest(l+1))
      { Nos[i].u = 0;
        l++;
      }
      else
        Nos[i].u = d(2*i-1-1);
      if (2*i == rest(l+1))
        { Nos[i].v = 0;
          l++;
        }
        else
          Nos[i].v = d(2*i-1);
    }
    else
    { Nos[i].u = d(2*i-1-1);
      Nos[i].v = d(2*i-1);
    }
  }
}
else
{ int l=0;
  for (int i=1; i<=NumeroDePontosNodais; i++)
  { if (l<NumeroDeRestricoes)
    { if ((3*i-2) == rest(l+1))
      { Nos[i].u = 0;
        l++;
      }
      else
        Nos[i].u = d(3*i-2-1);
      if ((3*i-1) == rest(l+1))
        { Nos[i].v = 0;
          l++;
        }
        else
          Nos[i].v = d(3*i-1-1);
      if ((3*i) == rest(l+1))
        { Nos[i].w = 0;
          l++;
        }
        else
          Nos[i].w = d(3*i-1);
    }
    else
    { Nos[i].u = d(3*i-2-1);
      Nos[i].v = d(3*i-1-1);
      Nos[i].w = d(3*i-1);
    }
  }
}
}
}
cout << "\n\t O deslocamento do primeiro no e " << Nos[1].u << " e " << Nos[1].v;
}

void Estrutura::AplicaRestricoes()
{
  //Condições de contorno.
  for(int i=1; i<=NumeroDePontosNodais; i++)
  {
    if (NSD == 2)

```

```

    {
        if (Nos[i].restx != 0)
            K.Adiciona(2*i-1,2*i-1,PENALIDADE);
        if (Nos[i].resty != 0)
            K.Adiciona(2*i,2*i,PENALIDADE);
    }
    else
    {
        if (Nos[i].restx != 0)
            K.Adiciona(3*i-2,3*i-2,PENALIDADE);
        if (Nos[i].resty != 0)
            K.Adiciona(3*i-1,3*i-1,PENALIDADE);
    }
    if (Nos[i].restz != 0)
        K.Adiciona(3*i,3*i,PENALIDADE);
    /* AVALIAR DEPOIS
    //////////////////////////////////////
    ///Incluido para deslocamentos prescritos
    if ((nos[i].prescy != 0)&&(primeiropasso==FALSE)) Fint[2*i] = F[2*i];
    if ((nos[i].prescx != 0)&&(primeiropasso==FALSE)) Fint[2*i-1] = F[2*i-1];
    //////////////////////////////////////
    */
    }
}

void Estrutura::AplicaRestricoes2()
{
    // Monta uma nova matriz de rigidez onde as linhas e colunas referentes
    // aos graus de liberdade restritos são excluídos
    // Só serve para NSD = 2
    matriz          Kl(2*NumeroDePontosNodais-NumeroDeRestricoes,2*NumeroDePontosNodais-
NumeroDeRestricoes);
    FApoio.redim(NumeroDeRestricoes,2*NumeroDePontosNodais-NumeroDeRestricoes);
    int l = 0;
    int c = 0;
    for(int i=1;i<=2*NumeroDePontosNodais;i++)
    { for(int j=1;j<=2*NumeroDePontosNodais;j++)
      { if (l<NumeroDeRestricoes)
        { if (c<NumeroDeRestricoes)
          { if (i==rest(l+1))
            { if (j==rest(c+1))
              c++;
            }
            else
              FApoio(l+1,j-c)=K(i,j);
          }
          else
            { if (j==rest(c+1))
              c++;
            }
            else
              Kl(i-1,j-c)=K(i,j);
          }
        }
      }
    }
    else
    { if (i==rest(l+1))
      FApoio(l+1,j-c)=K(i,j);
    }
    else
      Kl(i-1,j-c)=K(i,j);
    }
}
}

```

```

else
{ if (c<NumeroDeRestricoes)
{ if (j= =rest(c+1))
c++;
else
Kl(i-1,j-c)=K(i,j);
}
else
Kl(i-1,j-c)=K(i,j);
}
}
c = 0;
if (l<NumeroDeRestricoes)
{ if (i= =rest(l+1))
l++;
}
}
K.redim(2*NumeroDePontosNodais-NumeroDeRestricoes,2*NumeroDePontosNodais-
NumeroDeRestricoes);
K=Kl;
if (F.num_linhas= =2*NumeroDePontosNodais)
{ d.redim(2*NumeroDePontosNodais-NumeroDeRestricoes);
matriz Fl(2*NumeroDePontosNodais-NumeroDeRestricoes);
l = 0;
for(int i=1;i<=2*NumeroDePontosNodais;i++)
{ if (l<NumeroDeRestricoes)
{ if (i= =rest(l+1))
l++;
else
Fl(i-l) = F(i);
}
else
Fl(i-l) = F(i);
}
F.redim(2*NumeroDePontosNodais-NumeroDeRestricoes);
F = Fl;
}
}

void Estrutura::MontaVetorDeCargas(double fator)
{ F.redim(NSD*NumeroDePontosNodais);
for(int i=1;i<=NumeroDePontosNodais;i++)
{
if (NSD == 2)
{
F.Atribui(2*i,fator*(Nos[i].Fy+PENALIDADE*Nos[i].prescy)-Fint(2*i));
F.Atribui(2*i-1,fator*(Nos[i].Fx+PENALIDADE*Nos[i].prescx)-Fint(2*i-1));
}
else
{
F.Atribui(3*i,fator*(Nos[i].Fz+PENALIDADE*Nos[i].prescz)-Fint(3*i));
F.Atribui(3*i-1,fator*(Nos[i].Fy+PENALIDADE*Nos[i].prescy)-Fint(3*i-1));
F.Atribui(3*i-2,fator*(Nos[i].Fx+PENALIDADE*Nos[i].prescx)-Fint(3*i-2));
}
}
}

void Estrutura::CalculaTensoes()
{

```

```

        for(int i=1;i<=NumeroDeElementos;i++)
        {
            Elementos[i].CalculaTensoes(Fint, Nos, Pav(Elementos[i].camada,Elementos[i].fileira,
Elementos[i].coluna));
        }
    }

void Estrutura::MontaVetorDeForcasInternas()
{
    Fint.ZeraMatriz();
    for(int i=1;i<=NumeroDeElementos;i++)
    {
        Elementos[i].CalculaForcasInternas(Fint, Nos, Pav(Elementos[i].camada,Elementos[i].fileira,
Elementos[i].coluna));
    }
}

void Estrutura::Calcula()
{
    cout << "\n\tCalculando a Matriz de Rigidez...";
    CalculaRigidez();
    /* cout<< "\n\tMontando Vetor De Forcas Internas...";
    MontaVetorDeForcasInternas();*/
    cout << "\n\tAplicando as Restricoes...";
    AplicaRestricoes2();
    cout << "\n\tCalculando os Deslocamentos...";
    CalculaDeslocamentos();
    cout << "\n\tCalculando as Tensoes...";
    CalculaTensoes();
}

void Estrutura::LeDados(char * Dados, char * Resultados, char * Visualizacao)
{
    // Lê os dados do Problema
    char cabecalho[81];
    arquivo_entrada.open(Dados);
    arquivo_saida.open(Resultados);
    arquivo_view.open(Visualizacao);
    while (!arquivo_entrada.eof())
    {
        arquivo_entrada >> cabecalho;
        string s = cabecalho;
        if (s.find("Titulo") < string::npos)
        {
            // Lê o título do problema no arquivo de entrada e grava no arquivo de saída
            arquivo_entrada >> Nome;
            arquivo_saida << "/Titulo/" << endl;
            arquivo_saida << Nome << endl;
            // Gera Arquivo do MVIEW
            arquivo_view << "%HEADER" << endl;
            arquivo_view << "Saida Gerada pelo programa Dissertação de Mestrado" << endl;
            arquivo_view << "\n" << "%HEADER.AUTHOR" << endl;
            arquivo_view << "Fabio Grisolia de Avila" << endl;
            arquivo_view << "Marcelo Rodrigues Leao Silva" << endl;
            arquivo_view << "\n" << "%HEADER.ANALYSIS" << endl;
            arquivo_view << "A Definir" << endl;
        }
        if (s.find("Tipo_De_Problema") < string::npos)
        {

```



```

// Lê o Tipo de Problema.
arquivo_entrada >> TipoDeProblema;
arquivo_saida << "\n" << "/Tipo_De_Problema/" << endl;
s = TipoDeProblema;
arquivo_saida << s << endl;
if (s.find("3D") < string::npos) NSD = 3;
else NSD = 2;
}
/*if (AnsiStrPos(cabecalho, "/Passos_De_Carga/") != NULL)
{
// Lê o número de passos de carga.
arquivo_entrada >> NumeroDePassosDeCarga;
arquivo_saida << "\n" << "/Passos_De_Carga/" << endl;
arquivo_saida << NumeroDePassosDeCarga << endl;
}
if (AnsiStrPos(cabecalho, "/Numero_De_Nos/") != NULL)
{
// Lê o número de Nós.
arquivo_entrada >> NumeroDePontosNodais;
arquivo_saida << "\n" << "/Numero_De_Nós/" << endl;
arquivo_saida << NumeroDePontosNodais << endl;
Nos = new No[NumeroDePontosNodais+1];
K.redim(NSD*NumeroDePontosNodais, NSD*NumeroDePontosNodais);
d.redim(NSD*NumeroDePontosNodais);
F.redim(NSD*NumeroDePontosNodais);
Fint.redim(NSD*NumeroDePontosNodais);
}
if (AnsiStrPos(cabecalho, "/Numero_De_Nos_Com_Restricao/") != NULL)
{
// Lê o número de Nós Com Restrição.
arquivo_entrada >> NumeroDeNosComRestricao;
arquivo_saida << "\n" << "/Numero_De_Nós_Com_Restrição/" << endl;
arquivo_saida << NumeroDeNosComRestricao << endl;
}*/
if (s.find("Numero_De_Camadas") < string::npos)
{
// Lê o número de Tipos de Material.
arquivo_entrada >> NumeroDeCamadas;
arquivo_saida << "\n" << "/Numero_De_Camadas/" << endl;
arquivo_saida << NumeroDeCamadas << endl;
Pav.redim(NumeroDeCamadas);
Espessuras.redim(NumeroDeCamadas);
}
/*if (AnsiStrPos(cabecalho, "/Numero_De_Elementos/") != NULL)
{
// Lê o número de Elementos.
arquivo_entrada >> NumeroDeElementos;
arquivo_saida << "\n" << "/Numero_De_Elementos/" << endl;
arquivo_saida << NumeroDeElementos << endl;
//if (AnsiStrPos(TipoDeProblema, "TRUSS_2D") != NULL)
// Elementos = new Elemento_TRUSS2D[NumeroDeElementos+1];
//if (AnsiStrPos(TipoDeProblema, "ESTADO_PLANO_DE_TENSOES") != NULL)
//Elementos = new Elemento_EPT[NumeroDeElementos+1];
if (AnsiStrPos(TipoDeProblema, "AXISSIMETRICO") != NULL)
Elementos = new Elemento_AXISSIMETRICO[NumeroDeElementos+1];
//if (AnsiStrPos(TipoDeProblema, "3D") != NULL)
// Elementos = new Elemento_3D[NumeroDeElementos+1];
}
if (AnsiStrPos(cabecalho, "/Dados_Dos_Nos/") != NULL)

```

```

{
    // Lê os Dados dos Nós.
    LeNos();
}
if (AnsiStrPos(cabecalho, "/Dados_Dos_Elementos/") != NULL)
{
    // Lê os Dados dos Elementos.
    LeElementos();
}*/
if (s.find("Dados_Das_Camadas") < string::npos)
{
    // Lê os Dados dos Materiais.
    arquivo_entrada >> s;
    LeMateriais();
}
/*if (AnsiStrPos(cabecalho, "/Dados_Das_Restricoes/") != NULL)
{
    // Lê os Dados das Condições de Contorno.
    LeBounds();
}*/
if (s.find("Dados_Do_Carregamento") < string::npos)
{
    // Lê os dados do carregamento
    arquivo_entrada >> q;
    arquivo_saida << "\n" << "/Dados_Do_Carregamento/" << endl;
    arquivo_saida << "/Presao_De_Contato/" << "\t" << q << endl;
    arquivo_entrada >> a;
    arquivo_saida << "/Raio_Da_Area_De_Contato/" << "\t" << a << endl;
}
if (s.find("Tensoes_Minimas") < string::npos)
{
    // Lê os valores das Tensões Mínimas.
    arquivo_entrada >> s;
    arquivo_entrada >> Sdmin;
    arquivo_entrada >> S3min;
    arquivo_saida << "\n/Tensoes Minimas para o calculo do Modulo de Resiliencia/" << endl;
    arquivo_saida << "Sd <= " << Sdmin << "\tS3 <= " << S3min << endl;
}
if (s.find("Tolerancia") < string::npos)
{
    // Lê o valor da tolerância.
    arquivo_entrada >> tolerancia;
    tolerancia /= 100;
}
if (s.find("Metodo") < string::npos)
{
    // Lê o tipo do método a ser executado e o número de iterações.
    arquivo_entrada >> Metodo;
    arquivo_entrada >> Iteracao;
}
if (s.find("FIM") < string::npos)
{
    // Finaliza a Entrada de Dados.
    break;
}
}
}

```

```
void Estrutura::LeDados(string Dados, string Resultados, string Visualizacao)
```

```

{
  LeDados(Dados.c_str(), Resultados.c_str(), Visualizacao.c_str());
}

void Estrutura::GeraMalha()
{
  Malha pontos(0,0,20*a,Espessuras(1));
  pontos.div(1,a,3);
  pontos.pg(1,a/3,1.15,0,4);
  if (Espessuras(1)>0.18)
    pontos.pg(2,0.06,1.15,0);
  else
    pontos.div(2,0,3);
  pontos.razao_max(1);
  pontos.razao_max(2);
  pontos.Nos(1);
  matriz X(pontos.coord.num_linhas);
  X = pontos.coord;
  pontos.Nos(2);
  matriz Y(pontos.coord.num_linhas);
  Y = pontos.coord;
  Pav.num_colunas = pontos.nx - 1;
  Pav.fileiras[1] = pontos.ny - 1;
  if (NumeroDeCamadas>1)
  { double esp = Espessuras(1);
    for (int i=2;i<=NumeroDeCamadas;i++)
    { esp += Espessuras(i);
      pontos.redim(2,esp);
      if (Espessuras(i)>0.18)
        pontos.pg(2,0.06,1.15,0);
      else
        pontos.div(2,0,3);
      pontos.razao_max(2);
      pontos.Nos(2,Y);
      Y.redim(pontos.coord.num_linhas);
      Y = pontos.coord;
      Pav.fileiras[i] = pontos.ny - 1;
    }
  }
  Pav.redim();
  if (Metodo==1)
    Pav.inicializa();
  if (Metodo==2)
    Pav.inicializa(Sdmin,S3min);
  Pav.redim_col(Pav.num_colunas);
  NumeroDePontosNodais = (2*X.num_linhas - 1)*Y.num_linhas + (Y.num_linhas - 1)*X.num_linhas;
  Nos = new No[NumeroDePontosNodais+1];
  for (int i=1;i<=Y.num_linhas;i++)
  { for (int j=1;j<=X.num_linhas;j++)
    { Nos[(i-1)*(3*X.num_linhas-1)+(2*j-1)].x = X(j);
      Nos[(i-1)*(3*X.num_linhas-1)+(2*j-1)].y = Y(i);
      if (j<X.num_linhas)
        { Nos[(i-1)*(3*X.num_linhas-1)+(2*j)].x = (X(j) + X(j+1))/2;
          Nos[(i-1)*(3*X.num_linhas-1)+(2*j)].y = Y(i);
        }
      if (i<Y.num_linhas)
        { Nos[i*(3*X.num_linhas-1)-X.num_linhas+j].x = X(j);
          Nos[i*(3*X.num_linhas-1)-X.num_linhas+j].y = (Y(i) + Y(i+1))/2;
        }
    }
  }
}

```

```

if ((j= =1)||j= =X.num_linhas)
{ if (i<Y.num_linhas)
{ Nos[(i-1)*(3*X.num_linhas-1)+(2*j-1)].restx = 1;
Nos[i*(3*X.num_linhas-1)-X.num_linhas+j].restx = 1;
}
}
if (i= =Y.num_linhas)
{ Nos[(i-1)*(3*X.num_linhas-1)+(2*j-1)].restx = 1;
Nos[(i-1)*(3*X.num_linhas-1)+(2*j-1)].resty = 1;
if (j<X.num_linhas)
{ Nos[(i-1)*(3*X.num_linhas-1)+(2*j)].restx = 1;
Nos[(i-1)*(3*X.num_linhas-1)+(2*j)].resty = 1;
}
}
}
}
NumeroDeNosComRestricao = (2*X.num_linhas - 1) + 4*(Y.num_linhas - 1);
NumeroDeRestricoes = 2*(2*X.num_linhas - 1) + 4*(Y.num_linhas - 1);
rest.redim(NumeroDeRestricoes);
for (int i=1;i<=(Y.num_linhas-1);i++)
{ rest(4*i-3) = 2*((i-1)*(3*X.num_linhas-1)+1)-1;
rest(4*i-2) = 2*(i*(3*X.num_linhas-1)-X.num_linhas)-1;
rest(4*i-1) = 2*(i*(3*X.num_linhas-1)-X.num_linhas+1)-1;
rest(4*i) = 2*(i*(3*X.num_linhas-1))-1;
}
for (int i=4*(Y.num_linhas-1)+1;i<=NumeroDeRestricoes;i++)
rest(i) = 2*(Y.num_linhas-1)*(3*X.num_linhas-3)+i;
int k = 2;
while ((Nos[k].x-Nos[1].x)<a)
k++;
if ((Nos[k].x-Nos[1].x)>a)
k--;
Nos[1].Fy = PI*q*pow(((Nos[2].x+Nos[1].x)/2-Nos[1].x),2);
for (int i=2;i<k;i++)
Nos[i].Fy = PI*q*(pow(((Nos[i+1].x+Nos[i].x)/2-Nos[1].x),2)-pow(((Nos[i].x+Nos[i-1].x)/2-
Nos[1].x),2));
Nos[k].Fy = PI*q*(pow((Nos[k].x-Nos[1].x),2)-pow(((Nos[k].x+Nos[k-1].x)/2-Nos[1].x),2));
K.redim(NSD*NumeroDePontosNodais,NSD*NumeroDePontosNodais);
d.redim(NSD*NumeroDePontosNodais);
F.redim(NSD*NumeroDePontosNodais);
Fint.redim(NSD*NumeroDePontosNodais);
NumeroDeElementos = (X.num_linhas - 1)*(Y.num_linhas - 1);
Elementos = new Elemento_AXISSIMETRICO[NumeroDeElementos+1];
int num_nodais=8;//Numero de pontos nodais de cada elemento, para este caso 8
k=1;
int f=Pav.fileiras[1];
int fant=0;
for (int i=1;i<Y.num_linhas;i++)
{ for (int j=1;j<X.num_linhas;j++)
{ Elementos[(i-1)*(X.num_linhas-1)+j].NumeroDePontosNodais = num_nodais;
Elementos[(i-1)*(X.num_linhas-1)+j].NP = new int[num_nodais+1];
Elementos[(i-1)*(X.num_linhas-1)+j].NP[1] = (i-1)*(3*X.num_linhas-1)+(2*j-1);
Elementos[(i-1)*(X.num_linhas-1)+j].NP[2] = (i-1)*(3*X.num_linhas-1)+(2*j+1);
Elementos[(i-1)*(X.num_linhas-1)+j].NP[3] = (i)*(3*X.num_linhas-1)+(2*j+1);
Elementos[(i-1)*(X.num_linhas-1)+j].NP[4] = (i)*(3*X.num_linhas-1)+(2*j-1);
Elementos[(i-1)*(X.num_linhas-1)+j].NP[5] = (i-1)*(3*X.num_linhas-1)+(2*j);
Elementos[(i-1)*(X.num_linhas-1)+j].NP[6] = i*(3*X.num_linhas-1)-X.num_linhas+j+1;
Elementos[(i-1)*(X.num_linhas-1)+j].NP[7] = (i)*(3*X.num_linhas-1)+(2*j);
Elementos[(i-1)*(X.num_linhas-1)+j].NP[8] = i*(3*X.num_linhas-1)-X.num_linhas+j;
}
}
}

```

```

    if (i>f)
    { fant = f;
      f += Pav.fileiras[++k];
    }
    Elementos[(i-1)*(X.num_linhas-1)+j].camada = k;
    Elementos[(i-1)*(X.num_linhas-1)+j].fileira = i - fant;
    Elementos[(i-1)*(X.num_linhas-1)+j].coluna = j;
    Elementos[(i-1)*(X.num_linhas-1)+j].NoLocal = new No[num_nodais+1];
    Elementos[(i-1)*(X.num_linhas-1)+j].redim(num_nodais);
    for (int l=1;l<=Elementos[(i-1)*(X.num_linhas-1)+j].NumeroDePontosDeGauss;l++)
      Elementos[(i-1)*(X.num_linhas-1)+j].PontosDeGauss[l].Inicializa(num_nodais);
  }
}
}

```

void Estrutura::EscreveNos()

```

{
// Este Método Precisar  Ser Alterado Para Problemas 3D
arquivo_saida << "\n" << "/Dados_Dos_Nos/" << endl;
arquivo_view << "\n" << "%NODE" << endl;
arquivo_view << NumeroDePontosNodais << endl;
arquivo_view << "\n" << "%NODE.COORD" << endl;
arquivo_view << NumeroDePontosNodais << endl;
for (int i=1;i<=NumeroDePontosNodais;i++)
{
arquivo_saida << i << "\t" << Nos[i].x << "\t" << Nos[i].y;
if (NSD= =3) arquivo_saida << "\t" << Nos[i].z;
arquivo_saida << "\t" << Nos[i].Fx << "\t" << Nos[i].Fy;
if (NSD= =3) arquivo_saida << "\t" << Nos[i].Fz;
arquivo_saida << endl ;
arquivo_view << i << "\t" << Nos[i].x << "\t" << Nos[i].y << "\t" << 0.0 << endl;
}
arquivo_saida << endl;
arquivo_saida << "\n" << "/Dados_Das_Restri es/" << endl;
arquivo_view << "\n" << "%NODE.SUPPORT" << endl;
arquivo_view << NumeroDeNosComRestricao << endl;
for (int i=1;i<=NumeroDePontosNodais;i++)
{ if ((Nos[i].restx!=0)||((Nos[i].resty!=0)||((Nos[i].restz!=0)))
{ arquivo_saida << i << "\t" << Nos[i].restx << "\t" << Nos[i].resty;
if (NSD = = 3) arquivo_saida << "\t" << Nos[i].restz;
arquivo_saida << "\t" << Nos[i].prescx << "\t" << Nos[i].prescy;
if (NSD = = 3) arquivo_saida << "\t" << Nos[i].prescz;
arquivo_saida << endl;
arquivo_view << i << "\t" << Nos[i].restx << "\t" << Nos[i].resty
<< "\t" << 0 << "\t" << 0 << "\t" << 0 << "\t" << 0 << endl;
}
}
}
}
}

```

void Estrutura::EscreveElementos()

```

{
arquivo_saida << "\n/Dados_Dos_Elementos/" << endl;
arquivo_view << "\n" << "%ELEMENT" << endl;
arquivo_view << NumeroDeElementos << endl;
for (int i=1;i<=NumeroDeElementos;i++)
{ arquivo_saida << "\nElemento: " << i << "\tCamada: " << Elementos[i].camada
<< "\tFileira: " << Elementos[i].fileira << "\tColuna: "
<< Elementos[i].coluna << endl;
for(int j=1;j<=Elementos[i].NumeroDePontosNodais;j++)

```

```

        arquivo_saida << Elementos[i].NP[j] << "\t";
    }
    arquivo_saida << endl;
    //////////////////////////////////////
    /* Preparar Para MVIEW //////////////////////////////////////
    //////////////////////////////////////
    // arquivo_view << "\n" << "%INTEGRATION.ORDER" << endl;
    // arquivo_view << 1 << endl;
    // arquivo_view << 1 << "\t" << 2 << "\t" << 2 << "\t" << 1 << "\t" << 2 << "\t" << 2 << "\t" << 1 <<
"\t" << endl;
    // arquivo_view << "\n" << "%ELEMENT.Q8" << endl;
    // arquivo_view << NUMEL << endl;
    */
}

void Estrutura::LeMateriais()
{
    int i, k;
    arquivo_saida << "/Dados_Dos_Materiais/" << endl;
    arquivo_view << "\n" << "%MATERIAL" << endl;
    arquivo_view << NumeroDeCamadas << endl;
    /* arquivo_view << "\n" << "%MATERIAL.LABEL" << endl;
    arquivo_view << NumeroDeCamadas << endl;
    for (k=1;k<=NumeroDeCamadas;k++)
    {
        arquivo_view << k << "\t" << "camada" << endl;
    }*/
    arquivo_view << "\n" << "%MATERIAL.ISOTROPIC" << endl;
    arquivo_view << NumeroDeCamadas << endl;
    for (k=1;k<=NumeroDeCamadas;k++)
    {
        arquivo_entrada >> i;
        arquivo_entrada >> Espessuras(i);
        arquivo_entrada >> Pav(i).POISS >> Pav(i).tipo;
        arquivo_saida << i << "\t" << Espessuras(i) << "\t" << Pav(i).POISS
            << "\t" << Pav(i).tipo;
        arquivo_view << i << "\t" << Espessuras(i) << "\t" << Pav(i).POISS
            << "\t" << Pav(i).tipo;
        if ((Pav(i).tipo= =1)||Pav(i).tipo= =2)||Pav(i).tipo= =4)
        { arquivo_entrada >> Pav(i).ki(1) >> Pav(i).ki(2);
          arquivo_saida << "\t" << Pav(i).ki(1) << "\t" << Pav(i).ki(2) << endl;
          arquivo_view << "\t" << Pav(i).ki(1) << "\t" << Pav(i).ki(2) << endl;
        }
        if (Pav(i).tipo= =3)
        { arquivo_entrada >> Pav(i).ki(1) >> Pav(i).ki(2)
          >> Pav(i).ki(3) >> Pav(i).ki(4);
          arquivo_saida << "\t" << Pav(i).ki(1) << "\t" << Pav(i).ki(2)
            << "\t" << Pav(i).ki(3) << "\t" << Pav(i).ki(4) << endl;
          arquivo_view << "\t" << Pav(i).ki(1) << "\t" << Pav(i).ki(2)
            << "\t" << Pav(i).ki(3) << "\t" << Pav(i).ki(4) << endl;
        }
        if (Pav(i).tipo= =5)
        { arquivo_entrada >> Pav(i).ki(1) >> Pav(i).ki(2)
          >> Pav(i).ki(3) >> Pav(i).ki(4) >> Pav(i).ki(5);
          arquivo_saida << "\t" << Pav(i).ki(1) << "\t" << Pav(i).ki(2) << "\t" << Pav(i).ki(3)
            << "\t" << Pav(i).ki(4) << "\t" << Pav(i).ki(5) << endl;
          arquivo_view << "\t" << Pav(i).ki(1) << "\t" << Pav(i).ki(2) << "\t" << Pav(i).ki(3)
            << "\t" << Pav(i).ki(4) << "\t" << Pav(i).ki(5) << endl;
        }
    }
}

```

```

    if (Pav(i).tipo= =6)
    { arquivo_entrada >> Pav(i).ki(1);
      arquivo_saida << "\t" << Pav(i).ki(1) << endl;
      arquivo_view << "\t" << Pav(i).ki(1) << endl;
    }
    if (Pav(i).tipo= =7)
    { arquivo_entrada >> Pav(i).ki(1) >> Pav(i).ki(2)
      >> Pav(i).ki(3);
      arquivo_saida << "\t" << Pav(i).ki(1) << "\t" << Pav(i).ki(2)
        << "\t" << Pav(i).ki(3) << endl;
      arquivo_view << "\t" << Pav(i).ki(1) << "\t" << Pav(i).ki(2)
        << "\t" << Pav(i).ki(3) << endl;
    }
  }
  arquivo_saida << endl;
//  arquivo_view << "\n" << "%THICKNESS" << endl;
//  arquivo_view << 1 << endl;
//  arquivo_view << 1 << "\t" << 1 << endl;
}

```

```

int Estrutura::ProcuraElemento(double x, double y)
{
  int n = 0;
  int i = 1;
  while (n= =0)
  { if (Nos[i].y<y)
    { i++;
      else
      { if (Nos[i].x<x)
        { i++;
          else
          { n = i;
            }
          }
        }
    }
  }
  int elm = 0;
  i = 1;
  while (elm= =0)
  { for(int j=1;j<=Elementos[i].NumeroDePontosNodais;j++)
    { if (Elementos[i].NP[j]= =n)
      { elm = i;
        }
      }
    }
  }
  return elm;
}

```

```

int Estrutura::ProcuraElemento(double x, int c, int SupInf)
{
  int n, f, col, i;
  n = ProcuraElemento(x,Nos[1].y);
  col = Elementos[n].coluna;
  n = 0;
  i = 1;
  if (SupInf= =1)
  { f = 1;
    else
    { f = Pav.fileiras[c];
      while (n= =0)
      { if ((Elementos[i].camada= =c)&&(Elementos[i].fileira= =f)&&(Elementos[i].coluna= =col))

```

```

        n = i;
    else
        i++;
    }
    return n;
}

matriz Estrutura::TensoesAxissimetricas(double x, double y)
{
    int n;
    matriz ten(4);
    n = ProcuraElemento(x,y);
    ten = Elementos[n].Tensoes(x,y);
    return ten;
}

matriz Estrutura::TensoesPrincipais(double x, double y)
{
    int n;
    matriz TenAxi(4);
    matriz TenPrinc(3);
    n = ProcuraElemento(x,y);
    TenAxi = Elementos[n].Tensoes(x,y);
    TenPrinc = Elementos[n].PontosDeGauss[1].tensoes_princ(TenAxi(1),TenAxi(2),TenAxi(3),
TenAxi(4));
    return TenPrinc;
}

double Estrutura::TensoesMax(int c, int SupInf, int sigma)
{
    int f, n;
    double tensao, TenAtual;
    if (SupInf== 1)
        f = 1;
    else
        f = Pav.fileiras[c];
    n = 1;
    while (!(Elementos[n].camada== c)&&(Elementos[n].fileira== f))
        n++;
    tensao = Elementos[n].TensaoMax(SupInf,sigma);
    while ((Elementos[n].camada== c)&&(Elementos[n].fileira== f))
    { TenAtual = Elementos[n].TensaoMax(SupInf,sigma);
    if ((sigma== 2)||sigma== 5)||sigma== 8))
        { if (tensao>TenAtual)
            tensao = TenAtual;
        }
    if (sigma== 8)
        { if (tensao<TenAtual)
            tensao = TenAtual;
        }
    if ((sigma== 1)||sigma== 3)||sigma== 4)||sigma== 6))
        { if (fabs(tensao)<fabs(TenAtual))
            tensao = TenAtual;
        }
    }
    return tensao;
}

double Estrutura::TensoesMax(int c, int f, int SupInf, int sigma)

```



```

{
  int n = 1;
  double tensao, TenAtual;
  if (f>Pav.fileiras[c])
    f = Pav.fileiras[c];
  while (!(Elementos[n].camada= =c)&&(Elementos[n].fileira= =f))
    n++;
  tensao = Elementos[n++].TensaoMax(SupInf,sigma);
  while ((Elementos[n].camada= =c)&&(Elementos[n].fileira= =f))
  { TenAtual = Elementos[n++].TensaoMax(SupInf,sigma);
    if ((sigma= =2)||sigma= =5)||sigma= =8)
    { if (tensao>TenAtual)
      tensao = TenAtual;
    }
    if (sigma= =8)
    { if (tensao<TenAtual)
      tensao = TenAtual;
    }
    if ((sigma= =1)||sigma= =3)||sigma= =4)||sigma= =6)
    { if (fabs(tensao)<fabs(TenAtual))
      tensao = TenAtual;
    }
  }
  return tensao;
}

matriz Estrutura::DeformacoesPrincipais(double x, double y)
{
  int n;
  matriz def(3);
  n = ProcuraElemento(x,y);
  def = Elementos[n].DeformacoesPrinc(x,y,Pav(Elementos[n].camada,Elementos[n].fileira,
Elementos[n].coluna));
  return def;
}

double Estrutura::DeformacoesMax(int c, int SupInf, int def)
{
  int f, n;
  double deformacao, DefAtual;
  if (SupInf= =1)
    f = 1;
  else
    f = Pav.fileiras[c];
  n = 1;
  while (!(Elementos[n].camada= =c)&&(Elementos[n].fileira= =f))
    n++;
  deformacao = Elementos[n].DeformacaoMax(SupInf,def,Pav(c,f,Elementos[n].coluna));
  n++;
  while ((Elementos[n].camada= =c)&&(Elementos[n].fileira= =f))
  { DefAtual = Elementos[n].DeformacaoMax(SupInf,def,Pav(c,f,Elementos[n].coluna));
    n++;
    if (def= =1)
    { if (deformacao>DefAtual)
      deformacao = DefAtual;
    }
    if (def= =3)
    { if (deformacao<DefAtual)
      deformacao = DefAtual;
    }
  }
}

```

```

    }
    if (def==2)
    { if (fabs(deformacao)<fabs(DefAtual))
      deformacao = DefAtual;
    }
  }
  return deformacao;
}

double Estrutura::DeformacoesMax(int c, int f, int SupInf, int def)
{
  int n = 1;
  double deformacao, DefAtual;
  if (f>Pav.fileiras[c])
    f = Pav.fileiras[c];
  while (!(Elementos[n].camada==c)&&(Elementos[n].fileira==f))
    n++;
  deformacao = Elementos[n].DeformacaoMax(SupInf,def,Pav(Elementos[n].camada,Elementos[n].
fileira,Elementos[n].coluna));
  n++;
  while ((Elementos[n].camada==c)&&(Elementos[n].fileira==f))
  { DefAtual = Elementos[n].DeformacaoMax(SupInf,def,Pav(Elementos[n].camada,Elementos[n].
fileira,Elementos[n].coluna));
    n++;
    if (def==1)
    { if (deformacao>DefAtual)
      deformacao = DefAtual;
    }
    if (def==3)
    { if (deformacao<DefAtual)
      deformacao = DefAtual;
    }
    if (def==2)
    { if (fabs(deformacao)<fabs(DefAtual))
      deformacao = DefAtual;
    }
  }
  return deformacao;
}

double Estrutura::Deflexao(double x, double y)
{
  int n;
  double Def;
  n = ProcuraElemento(x,y);
  Def = Elementos[n].Deflexao(x,y);
  return Def;
}

double Estrutura::DeflexaoMax(int c, int SupInf)
{
  int f, n;
  double Def, deslocamento;
  if (SupInf==1)
    f = 1;
  else
    f = Pav.fileiras[c];
  n = 1;
  while (!(Elementos[n].camada==c)&&(Elementos[n].fileira==f))

```

```

    n++;
    Def = Elementos[n++].DeflexaoMax(SupInf);
    while ((Elementos[n].camada== =c)&&(Elementos[n].fileira== =f))
    { deslocamento = Elementos[n++].DeflexaoMax(SupInf);
      if (Def<deslocamento)
        Def = deslocamento;
    }
    return Def;
}

double Estrutura::DeflexaoMax(int c, int f, int SupInf)
{
    int n = 1;
    double Def, deslocamento;
    if (f>Pav.fileiras[c])
        f = Pav.fileiras[c];
    while (!(Elementos[n].camada== =c)&&(Elementos[n].fileira== =f))
        n++;
    Def = Elementos[n++].DeflexaoMax(SupInf);
    while ((Elementos[n].camada== =c)&&(Elementos[n].fileira== =f))
    { deslocamento = Elementos[n++].DeflexaoMax(SupInf);
      if (Def<deslocamento)
        Def = deslocamento;
    }
    return Def;
}

bool Estrutura::convergencia(double tol)
{
    // arquivo_saida << "\n/Tensoes nos pontos de Gauss dos elementos/" << endl;
    int k=1;
    int f=1;
    bool menor;
    for (int i=1;i<=Pav.fileiras_tot;i++)
    { for (int j=1;j<=Pav.num_colunas;j++)
      { if (f>Pav.fileiras_at[k])
        f -= Pav.fileiras_at[k++];
        Elementos[(i-1)*Pav.num_colunas+j].calcula_MR(Sdmin,S3min,Pav(k,f,j));
      }
      f++;
    }
    menor = Pav.tolerancia(tol);
    return menor;
}

```

8.2.9 CLASSE MAIN

```

#include <cstdlib>
#include <iostream>
#include "Elemento_Axissimetrico1.h"
using namespace std;

int main(int argc, char *argv[])
{
    Estrutura est;
    est.LeDados("Dados.txt","Resultados.txt","Visualizacao.txt");
    cout << "Gerando a malha...";
    est.GeraMalha();
}

```

```

bool para = false;
bool outro = false;
if (est.Metodo==1)
{
    int i = 1;
    est.MontaVetorDeCargas();
    while ((!para)&&(!outro))
    {
        cout << "\nCalculando a " << i << "a. iteracao...";
        est.Calcula();
        cout << "\nTestando a tolerancia...";
        para = est.convergencia(est.tolerancia);
        if ((i>=est.Iteracao)&&(para==false))
        {
            char c;
            cout << "O problema ate " << i << "a. iteracao nao convergiu!" << endl;
            cout << "Deseja continuar? (s/n) ";
            cin >> c;
            cout << endl;
            if (toupper(c)!='N')
                para = true;
            else
            {
                cout << "Deseja mudar de metodo? (s/n) ";
                cin >> c;
                cout << endl;
                if (toupper(c)=='S')
                {
                    outro = true;
                    cout << "Favor alterar entrada de dados!" << endl;
                }
            }
        }
        i++;
    }
}
if (est.Metodo==2)
{
    matriz d (2*est.NumeroDePontosNodais-est.NumeroDeRestricoes);
    for(int i=1;i<=est.Iteracao;i++)
    {
        est.MontaVetorDeCargas(1.0*i/est.Iteracao);
        cout << "\nCalculando a estrutura para " << i << "/" << est.Iteracao << " da forca...";
        cout << "\n\tCalculando a Matriz de Rigidez...";
        est.CalculaRigidez();
        cout << "\n\tAplicando as Restricoes...";
        est.AplicaRestricoes2();
        cout << "\n\tCalculando os Deslocamentos...";
        est.CalculaDeslocamentos();
        cout << "\n\tCalculando as Tensoes...";
        est.CalculaTensoes();
        cout << "\nRecalculando os Modulos de resiliencia...";
        est.convergencia(est.tolerancia);
        cout << endl;
    }
}
cout << "\nEscrevendo os resultados nos arquivos de saida...";
est.EscreveNos();
est.EscreveElementos();
est.GeraResultados2();
cout << endl;
ofstream resumo;
resumo.open("Resumo.txt");
double valor;
resumo << "Título: " << est.Nome << endl;
resumo << "Tipo de Problema: " << est.TipoDeProblema << endl;

```

```

resumo << "\nValores Máximos para:" << endl;
resumo << "\tDeflexão: ";
valor = est.DeflexaoMax(1);
resumo << valor << " m" << endl;
resumo << "\tDeformação Específica de Tração no revestimento: ";
valor = est.DeformacoesMax(1,2,3);
resumo << valor << endl;
resumo << "\tTensão de Tração no revestimento: ";
valor = est.TensoesMax(1,2,7);
if (valor>0.0)
    resumo << valor << " MPa (tração)" << endl;
else
    resumo << "A tensão de tração não é significativa!" << endl;
resumo << "\tDiferença de tensão no revestimento: ";
valor = est.TensoesMax(1,2,8);
resumo << fabs(valor) << " MPa" << endl;
for(int i=1;i<=est.NumeroDeCamadas;i++)
{ resumo << "\tA Tensão vertical no topo da camada " << i << " é: ";
  valor = est.TensoesMax(i,1,2);
  if (valor<0.0)
    resumo << fabs(valor) << " MPa (compressão)" << endl;
  else
  { resumo << valor << " MPa";
    if (valor= =0)
      resumo << endl;
    else
      resumo << " (tração)" << endl;
  }
}
}
resumo << "\nValores, referentes a 3/2 do raio do carregamento, para:" << endl;
resumo << "\tDeflexão: ";
valor = est.Deflexao((1.5*est.a),0.0);
resumo << valor << " m" << endl;
resumo << "\tDeformação Específica de Tração no revestimento: ";
matriz A(3);
A = est.DeformacoesPrincipais((1.5*est.a),est.Espessuras(1));
resumo << A(3) << endl;
resumo << "\tTensão de Tração no revestimento: ";
A = est.TensoesPrincipais((1.5*est.a),est.Espessuras(1));
valor = A(3);
if (valor>0.0)
    resumo << valor << " MPa (tração)" << endl;
else
    resumo << "A tensão de tração não é significativa!" << endl;
resumo << "\tDiferença de tensão no revestimento: ";
valor = A(1) - A(3);
resumo << fabs(valor) << " MPa" << endl;
int n;
A.redim(4);
for(int i=1;i<=est.NumeroDeCamadas;i++)
{ resumo << "\tA Tensão vertical no topo da camada " << i << " é: ";
  n = est.ProcuraElemento((1.5*est.a),i,1);
  A = est.Elementos[n].Tensoes((1.5*est.a),est.Elementos[n].NoLocal[1].y);
  valor = A(2);
  if (valor<0.0)
    resumo << fabs(valor) << " MPa (compressão)" << endl;
  else
  { resumo << valor << " MPa";
    if (valor= =0)

```

```

        resumo << endl;
    else
        resumo << " (tração)" << endl;
    }
}
resumo << "\nAs deflexões na superfície são:" << endl;
resumo << "\tD(0) = " << est.Deflexao(0.0,0.0) << endl;
resumo << "\tD(20) = " << est.Deflexao(0.20,0.0) << endl;
resumo << "\tD(30) = " << est.Deflexao(0.30,0.0) << endl;
resumo << "\tD(45) = " << est.Deflexao(0.45,0.0) << endl;
resumo << "\tD(60) = " << est.Deflexao(0.60,0.0) << endl;
resumo << "\tD(90) = " << est.Deflexao(0.90,0.0) << endl;
resumo << "\tD(120) = " << est.Deflexao(1.20,0.0) << endl;
resumo << "\n\nOs módulos de resiliência dos elementos são:" << endl;
for(int i=1;i<=est.NumeroDeCamadas;i++)
{ resumo << "Camada " << i << ":" << endl;
  for(int j=1;j<=est.Pav.fileiras[i];j++)
  { for(int k=1;k<=est.Pav.num_colunas;k++)
    resumo << "\t" << est.Pav(i,j,k).MR;
    resumo << endl;
  }
}
resumo.close();
cout << endl;
system("PAUSE");
return EXIT_SUCCESS;
}

```

Livros Grátis

(<http://www.livrosgratis.com.br>)

Milhares de Livros para Download:

[Baixar livros de Administração](#)

[Baixar livros de Agronomia](#)

[Baixar livros de Arquitetura](#)

[Baixar livros de Artes](#)

[Baixar livros de Astronomia](#)

[Baixar livros de Biologia Geral](#)

[Baixar livros de Ciência da Computação](#)

[Baixar livros de Ciência da Informação](#)

[Baixar livros de Ciência Política](#)

[Baixar livros de Ciências da Saúde](#)

[Baixar livros de Comunicação](#)

[Baixar livros do Conselho Nacional de Educação - CNE](#)

[Baixar livros de Defesa civil](#)

[Baixar livros de Direito](#)

[Baixar livros de Direitos humanos](#)

[Baixar livros de Economia](#)

[Baixar livros de Economia Doméstica](#)

[Baixar livros de Educação](#)

[Baixar livros de Educação - Trânsito](#)

[Baixar livros de Educação Física](#)

[Baixar livros de Engenharia Aeroespacial](#)

[Baixar livros de Farmácia](#)

[Baixar livros de Filosofia](#)

[Baixar livros de Física](#)

[Baixar livros de Geociências](#)

[Baixar livros de Geografia](#)

[Baixar livros de História](#)

[Baixar livros de Línguas](#)

[Baixar livros de Literatura](#)
[Baixar livros de Literatura de Cordel](#)
[Baixar livros de Literatura Infantil](#)
[Baixar livros de Matemática](#)
[Baixar livros de Medicina](#)
[Baixar livros de Medicina Veterinária](#)
[Baixar livros de Meio Ambiente](#)
[Baixar livros de Meteorologia](#)
[Baixar Monografias e TCC](#)
[Baixar livros Multidisciplinar](#)
[Baixar livros de Música](#)
[Baixar livros de Psicologia](#)
[Baixar livros de Química](#)
[Baixar livros de Saúde Coletiva](#)
[Baixar livros de Serviço Social](#)
[Baixar livros de Sociologia](#)
[Baixar livros de Teologia](#)
[Baixar livros de Trabalho](#)
[Baixar livros de Turismo](#)