UNIVERSIDADE DE SÃO PAULO ESCOLA POLITÉCNICA
DEPARTAMENTO DE ENGENHARIA MECÂNICA

Desenvolvimento de um Software de Otimização Topológica Aplicado a Peças e Estruturas num Domínio Bidimensional

Luís Augusto Motta Mello

Orientador: Emilio Carlos Nelli Silva

São Paulo
2002

UNIVERSIDADE DE SÃO PAULO ESCOLA POLITÉCNICA
DEPARTAMENTO DE ENGENHARIA MECÂNICA

Desenvolvimento de um Software de Otimização Topológica Aplicado a Peças e Estruturas num Domínio Bidimensional

Texto apresentado à Escola Politécnica da Universidade de São Paulo para obtenção do título de Engenheiro.

10,0
Emilio Carlos Nelli Silva

Luís Augusto Motta Mello

Orientador: Emilio Carlos Nelli Silva

Área de Concentração:
Engenharia Mecânica

São Paulo
2002

FICHA CATALOGRÁFICA

Motta Mello, Luís Augusto

Desenvolvimento de um Software de Otimização Topológica Aplicado a Peças e Estruturas num Domínio Bidimensional, por L.A.M. Mello. São Paulo: EPUSP, 2002. 54P. + anexos

Trabalho de formatura – Escola Politécnica da Universidade de São Paulo. Departamento de Engenharia Mecânica.

1. Otimização topológica 2. Estruturas 3. Rigidez 4. Múltiplos casos de carga 5. Bidimensional I. Universidade de São Paulo. Escola Politécnica. Departamento de Engenharia Mecânica. II. t.

Referencia dos Matemáticos

Otimização Matemática ; Topologia (Otimização)

SUMARIO

LISTA DE FIGURAS

LISTA DE ABREVIATURAS OU SIGLAS

OT Otimização Topológica

MEF Método dos Elementos Finitos

SIMP "Simple Isotropic Material with Penalization"

MGCSE Método dos Gradientes Conjugados para Sistemas Esparsos

MGBSE Método dos Gradientes Biconjugados para Sistemas Esparsos

PL Programação Linear

PLS Programação Linear Seqüencial

LISTA DE SÍMBOLOS

σ_{XX}	tensão normal na direção de x
σ_{YY}	tensão normal na direção de y
τ_{XY}	tensão de cisalhamento no plano xy
E	módulo de elasticidade
ν	coeficiente de Poisson
ε_{XX}	deformação normal na direção de x
ε_{YY}	deformação normal na direção de y
γ_{XY}	componente de cisalhamento da deformação no plano xy
u	deslocamento na direção de x
v	deslocamento na direçao de y
S	funções de forma
ξ	abscissa no sistema natural de coordenadas
η	ordenada no sistema natural de coordenadas
$[J]$	Jacobiano
t_e	espessura
$[K]^{(e)}$	matriz de rigidez do elemento
$\{U\}$	vetor de deslocamentos nodais
w_i	pesos para o Método de Gauss-Legendre
w_j	pesos para o Método de Gauss-Legendre
ξ_i	pontos de integração para o Método de Gauss-Legendre
η_i	pontos de integração para o Método de Gauss-Legendre
$\{F\}$	vetor de forças nodais
p	fator de penalidade
C_0	tensor isotrópico
ρ	"pseudo-densidade"
F_1	flexibilidade
F_2	flexibilidade linearizada

peso$_i$ peso para um determinado caso de carregamento

N número de nós da malha de elementos finitos

M número de elementos finitos

ID matriz de graus de liberdade não restritos e enumerados

edof matriz formada com valores representativos das conectividades dos elementos

fb vetor de carregamentos nodais

coord matriz de coordenadas nodais

conect matriz de conectividades nodais

ksize número de nós livres da estrutura

LM matriz de correspondência entre componentes da matriz de rigidez do elemento finito e da matriz de rigidez global

x0 "pseudo-densidades" iniciais

Vfrac volume máximo de material, dado como fração ou porcentagem do volume total

f vetor de carregamentos que engloba todos os casos de carga possíveis

num_load_cases número de casos de carga

dsplp nome da rotina de otimização usada

sign1 variável definida para cada elemento finito, auxiliar para cálculo dos limites móveis

sign2 variável definida para cada elemento finito, auxiliar para cálculo dos limites móveis

sign3 variável definida para cada elemento finito, auxiliar para cálculo dos limites móveis

NUMIT número máximo de iterações

ITERTOL limite para a variação da função objetivo

raio raio do filtro

cam número de camadas de vizinhos

RESUMO

Neste trabalho visa-se a implementação de um software (em Linguagem C) capaz de otimizar a topologia de uma peça, baseando-se no critério de máxima rigidez, restringido o peso total. As equações diferenciais que descrevem o equilíbrio do sistema são aproximadas por equações algébricas através do Método dos Elementos Finitos, obtendo-se então a chamada matriz de rigidez do sistema. As equações ou sistema de equações são resolvidas pelo Método dos Gradientes Conjugados para sistemas lineares esparsos (ou de matrizes esparsas) e a solução é usada no algoritmo de otimização propriamente dita, baseado na Programação Linear. Devido ao uso da Programação Linear como método numérico de otimização estrutural, são realizadas diversas iterações até que o resultado ótimo seja alcançado. Pretende-se por fim, ainda, o teste comparativo do programa e conseqüente validação, para uso posterior.

ABSTRACT

In this work the implementation of a software is sought (in C language) capable to optimize the topology of a piece, basing on the criterion of maxim stiffness, restricted the total weight. The differential equations that describe the equilibrium of the system are approximate for algebraic equations through the Finite Elements Method, and then being obtained the stiffness matrix of the system. The equations or system of equations are solved for the Conjugate Gradient Method for systems of linear equations and the solution is used in the optimization algorithm, based on the Linear Programming. Due to the use of the Linear Programming as numeric method of structural optimization, several iterations are accomplished until the optimum is reached. It is intended finally, the comparative test of the software and consequent validation, for subsequent use.

INTRODUÇÃO

O conceito de otimização sempre esteve presente na vida humana. Nas invenções mais antigas desenvolvidas pelo homem, como, por exemplo, a alavanca ou polias usadas para elevação de carga, nota-se a manifestação clara do desejo do homem de maximizar o rendimento mecânico. Mesmo nos dias de hoje, ao escolher um ou outro caminho que compreende um número finito de ruas a ligar dois lugares diferentes (o lugar de saída e o de chegada), uma pessoa está tentando minimizar o tempo gasto na realização de uma tarefa – o que, na maioria das vezes não acontece, uma vez que a função de excitação do sistema, dependente do tempo e da posição no domínio considerado (como o fenômeno climático conhecido como chuva), não é, normalmente, inicialmente conhecida, apesar do suposto desenvolvimento atual das ciências meteorológicas; sendo assim, a solução ótima varia com o tempo de maneira praticamente randômica, o que torna quase impossível a sua obtenção.

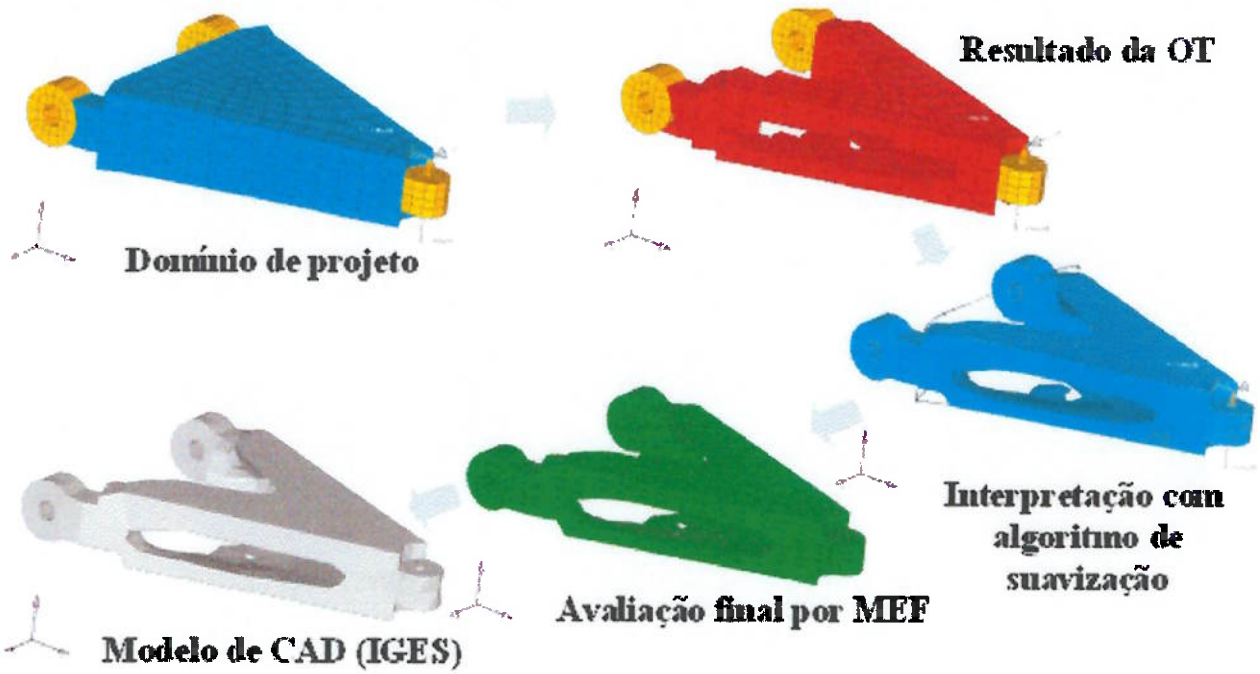
Existem basicamente dois métodos ou abordagens diferentes de obtenção de soluções ótimas. A primeira é conhecida como abordagem de análise e compreende a variação de parâmetros de projeto, de maneira aleatória, ou mesmo com algum direcionamento, em busca da otimização de um objetivo, ou função objetivo. Como um exemplo deste método, pode-se citar uma estrutura de treliça, sujeita a um dado carregamento e sustentada por alguns dos nós de suas extremidades. Suponha-se que a estrutura deva ser formada por dez barras de comprimentos iguais e com áreas de seção transversal podendo variar de forma descontínua, assumindo apenas dez valores diferentes (disponíveis), e que se quer maximizar a sua rigidez (da estrutura), restringindo-se o peso total. A abordagem apresentada consistiria em se construir um modelo do sistema e então calcular sua rigidez para as combinações possíveis das áreas. Os valores de rigidez seriam tabelados para melhor visualização e o maior valor, satisfazendo a restrição de peso imposta, seria escolhido como valor ótimo, juntamente com as áreas correspondentes.

Pode-se notar que este método é viável apenas para um número reduzido de parâmetros, ou variáveis de projeto.

A outra abordagem, utilizada inicialmente em áreas de conhecimento como na Economia, Logística ou Engenharia de Produção, é denominada síntese ou, como é mais conhecida, otimização. Pode ser definida de maneira genérica como *a busca do melhor resultado de uma dada operação, satisfeitas certas restrições,* ou *como a busca sistemática da solução ótima dentro de várias configurações possíveis.* São utilizados, para estas buscas, algoritmos matemáticos.

A otimização estrutural, área tratada neste trabalho, apresenta basicamente três abordagens diferentes: a paramétrica, a de forma e a topológica. A última provê os melhores resultados no que diz respeito à redução de peso da estrutura e otimização do objetivo. Portanto foi escolhida como base fundamental deste estudo.

Como ilustração, assim como para despertar o interesse do leitor, foi exposta a Figura 1, um exemplo de aplicação industrial. Nela, podem ser vistas as etapas complementares pelas quais passa a estrutura projetada - além da otimização propriamente dita, tema deste estudo.



Cortesia Altair Engineering, Inc., Michigan, EUA

Figura 1 Exemplo de aplicação industrial: braço de suspensão dianteira de um caminhão.

Este exemplo demonstra o procedimento adotado, mas levando-se em conta um domínio tridimensional. Como ilustração da otimização em domínios bidimensionais, mostra-se a Figura 2, na qual são vistos o carregamento considerado e os corpos original e otimizado de uma peça estrutural de sustentação do piso de um avião. A distribuição final tem a mesma rigidez apresentada pela estrutura original, mas com redução de, aproximadamente, 40% de material, o que representa um grande ganho - sobretudo para a indústria, para quem uma pequena economia num objeto pode significar poupar milhares de dólares.

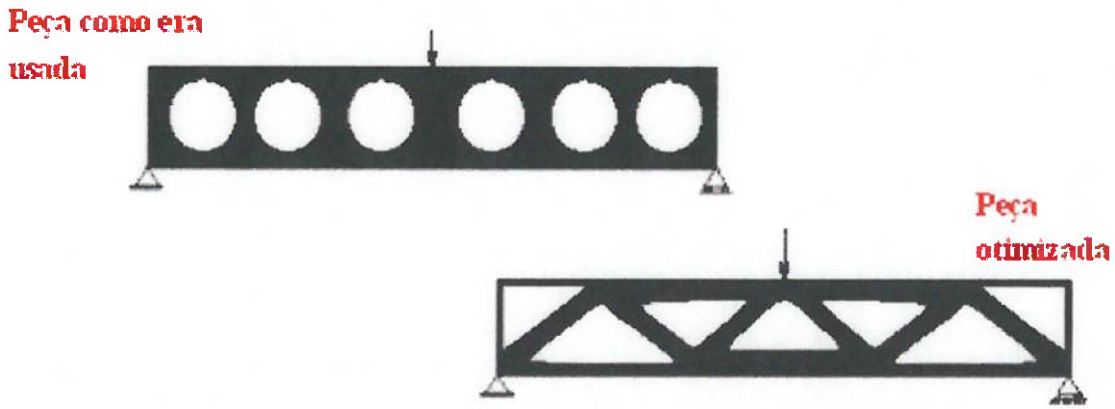


Figura 2 Exemplo de aplicação bidimensional: viga de sustentação do piso de um avião.

Neste trabalho visa-se a implementação de um software (em Linguagem C) capaz de otimizar a topologia de uma peça, baseando-se no critério de máxima rigidez, restringido o peso total. As equações diferenciais que descrevem o equilíbrio do sistema são aproximadas por equações algébricas através do Método dos Elementos Finitos, obtendo-se então a chamada matriz de rigidez do sistema. As equações ou sistema de equações são resolvidas pelo Método dos Gradientes Conjugados para sistemas lineares esparsos (ou de matrizes esparsas) e a solução é usada no algoritmo de otimização propriamente dita, baseado na Programação Linear. Devido ao uso da Programação Linear como método numérico de otimização estrutural, são realizadas diversas iterações até que o resultado ótimo seja alcançado. Pretende-se por fim, ainda, o teste comparativo do programa e conseqüente validação, para uso posterior.

Nos próximos capítulos, relacionar-se-á a fundamentação teórica necessária para o entendimento da implementação, serão introduzidos a forma de implementação do software e os resultados obtidos, e então divulgada a conclusão. A exposição dos fundamentos teóricos será feita em ordem, de acordo com o programa principal, para que se torne mais simples a sua compreensão (do programa).

FUNDAMENTOS TEÓRICOS

MÉTODO DOS ELEMENTOS FINITOS (MEF)

No presente trabalho, visa-se a implementação do modelo formulado à partir da simplificação para a qual se considera apenas o estado plano de tensões. O problema pode então ser tratado como possuidor de apenas duas dimensões. Exemplos deste tipo de simplificação podem ser vistos na Figura 3.

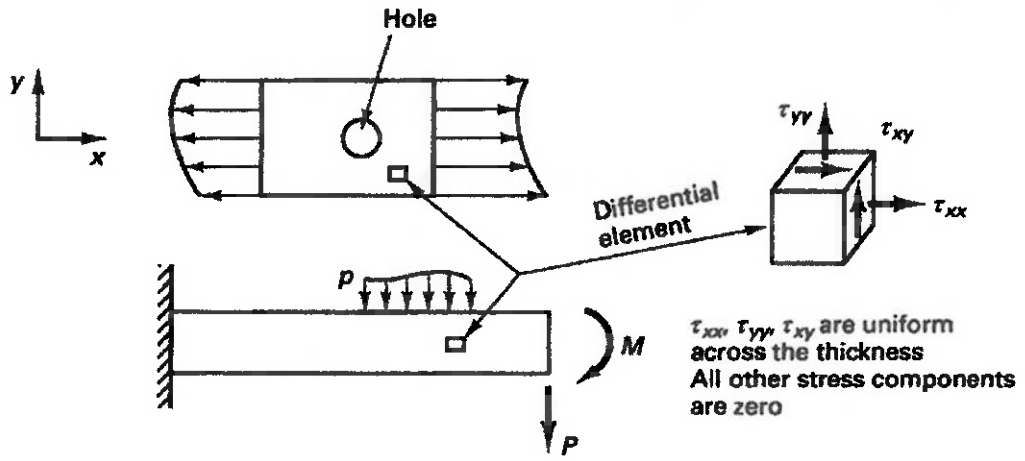


Figura 3 Condições de estado plano de tensões: uma chapa e uma viga engastada sob tensões no plano da folha.

Esta situação pode acontecer quando não existem forças agindo na direção Z, definida na Figura 4 juntamente a um cubo infinitesimalmente pequeno e que contém um ponto dentro de um material qualquer. As tensões existentes no cubo, provenientes da aplicação de forças externas, também são vistas na figura e representam o estado de tensões do ponto.

Para o caso da Figura 4, o estado de tensões se reduz a:

$$[\sigma]^T = [\sigma_{XX} \quad \sigma_{YY} \quad \tau_{XY}]$$

sendo σ_{XX} e σ_{YY} tensões normais e $\tau_{XY} = \tau_{YX}$ (a igualdade vem da condição de equilíbrio) as componentes de cisalhamento.

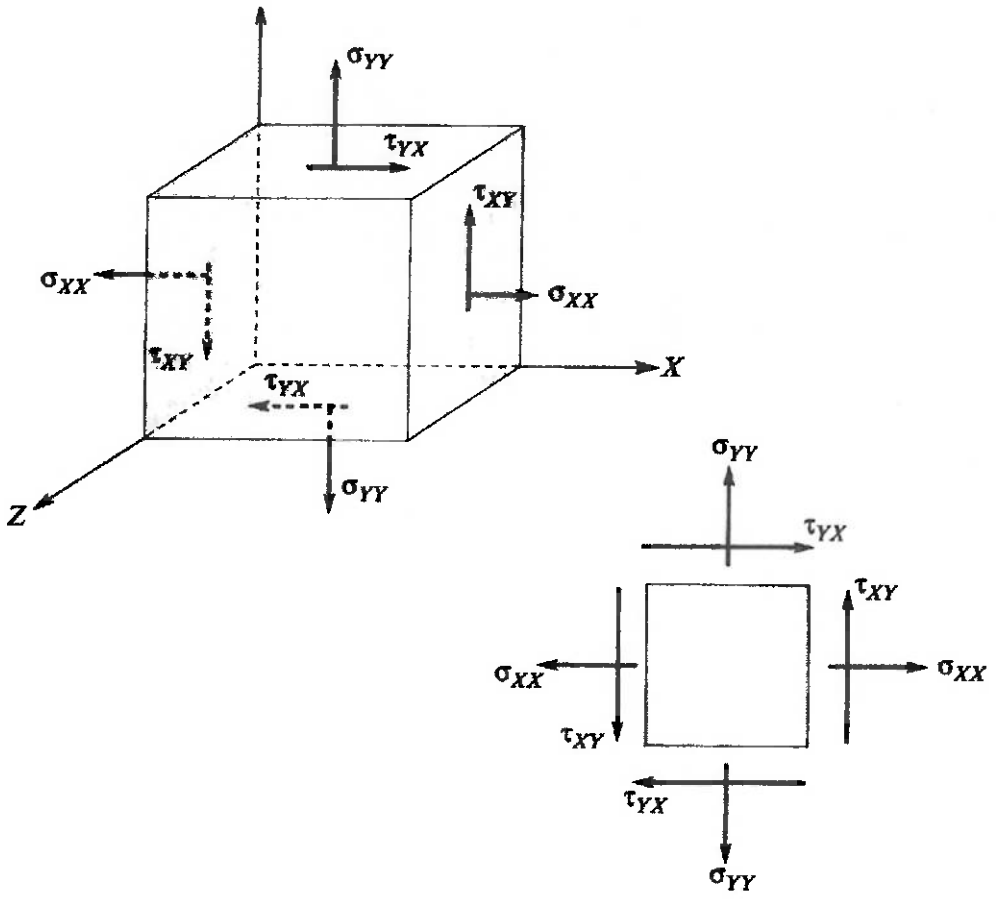


Figura 4 Simplificação obtida para estado plano de tensões.

Além de tensões, as forças aplicadas também provocam deslocamentos de pontos pertencentes a um corpo, que se torna deformado. Pode-se definir um vetor de deslocamentos que mede mudanças ocorridas na posição de tais pontos, o qual, descrito em termos de coordenadas cartesianas, rende:

$$\vec{\delta} = u(x, y, z)\vec{i} + v(x, y, z)\vec{j} + w(x, y, z)\vec{k}$$

onde:

$$u(x, y, z) = x' - x$$
$$v(x, y, z) = y' - y$$
$$w(x, y, z) = z' - z$$

As posições indicadas juntamente com o apóstrofo significam posições novas, obtidas após serem aplicados os carregamentos e as outras posições (x, y e z) referem-se à posições originais dos pontos.

Uma alternativa para se medir a mudança no formato da estrutura em estudo é introduduzida pelo conceito de deformações do corpo, que relacionam-se com as tensões através da Lei de Hooke, dada para o estado plano de tensões por:

$$\begin{Bmatrix} \sigma_{xx} \\ \sigma_{yy} \\ \tau_{xy} \end{Bmatrix} = \frac{E}{1 - \nu^2} \begin{bmatrix} 1 & \nu & 0 \\ \nu & 1 & 0 \\ 0 & 0 & \frac{1 - \nu}{2} \end{bmatrix} \begin{Bmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \gamma_{xy} \end{Bmatrix}$$

ou numa forma compacta:

$$\{\sigma\} = [\nu]\{\varepsilon\}$$

E é o módulo de elasticidade, ν é o coeficiente de Poisson, ε_{XX} e ε_{YY} são deformações normais e γ_{XY} é a componente de cisalhamento. Essas (deformações) também se relacionam aos deslocamentos estruturais à partir de:

$$\varepsilon_{xx} = \frac{\partial u}{\partial x} \quad \varepsilon_{yy} = \frac{\partial v}{\partial y} \quad \gamma_{xy} = \frac{\partial u}{\partial y} + \frac{\partial v}{\partial x}$$

Torna-se agora necessária a introdução do termo energia de deformação. Esta acumula-se no interior do material no qual são aplicados carregamentos e é resultado do trabalho realizados por estas forças externas ou carregamentos. Pode ser dada por:

$$\Lambda = \frac{1}{2}\int_V \left(\sigma_{xx}\varepsilon_{xx} + \sigma_{yy}\varepsilon_{yy} + \tau_{xy}\gamma_{xy}\right) dV$$

ou numa forma matricial:

$$\Lambda = \frac{1}{2}\int_V [\sigma]^T\{\varepsilon\}\, dV$$

que, considerando-se a Lei de Hooke simplificada discutida anteriormente, resulta em:

$$\Lambda = \frac{1}{2}\int_V \{\varepsilon\}^T[\nu]\{\varepsilon\}\, dV$$

Neste ponto, é introduzida o formulação do Método dos Elementos Finitos propriamente dita. Pode-se então representar os deslocamentos u e v dos pontos do material com auxílio de funções aproximadas definidas em cada elemento em que se subdivide a estrutura ou material em questão. No caso deste trabalho, optou-se por elementos quadrilaterais e, portanto, sendo também escolhidas equações polinomiais, foram obtidas funções ditas bilineares, que geram deslocamentos os quais variam linearmente nas bordas do elemento e apresentam variações não-lineares no interior. Os deslocamentos são ainda definidos num sistema alternativo de coordenadas adimensional denominado Sistema de Coordenadas Natural. O elemento fica desta forma definido como na Figura 5.

A transformação de um sistema de coordenadas qualquer para o Natural é extremamente benéfica, uma vez que força o elemento quadrilateral a ter uma geometria regular, sendo os limites desta geometria –1 e 1, tanto no eixo das ordenadas quanto no das abscissas. (os benefícios serão notados no momento do cálculo das matrizes de rigidez dos elementos, visto mais adiante)

No caso também deste trabalho, por comodidade de implementação, designou-se a chamada Formulação Isoparamétrica como componente essencial. Segundo esta, a mesma função interpoladora utilizada para aproximação dos deslocamentos será também utilizada para expressar a posição de qualquer ponto no interior do elemento.

Resolvendo o sistema de equações obtido da função interpoladora, pode-se então representar deslocamentos (ou posições) no interior do elemento através das funções de forma S e dos deslocamentos nodais U (ou posições nodais) do elemento, sendo que estes últimos (U's) serão definidos num sistema de coordenadas global (o sistema no qual se definem os deslocamentos e posições nodais é escolhido arbitrariamente em função da comodidade que representa para obtenção das matrizes de um elemento).

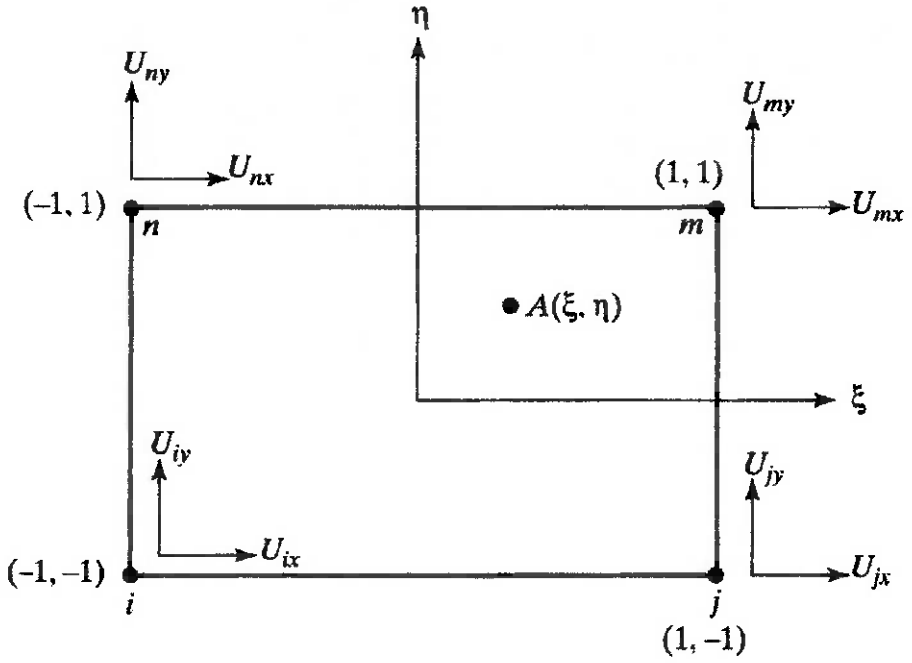


Figura 5 Elemento quadrilateral usado na formulação de problemas envolvendo o estado plano de tensões, definido no sistema natural de coordenadas e com deslocamentos definidos no sistema global de coordenadas.

Têm-se então:

$$u = S_i U_{ix} + S_j U_{jx} + S_m U_{mx} + S_n U_{nx}$$
$$v = S_i U_{iy} + S_j U_{jy} + S_m U_{my} + S_n U_{ny}$$

Ou numa notação matricial:

$$\begin{Bmatrix} u \\ v \end{Bmatrix} = \begin{bmatrix} S_i & 0 & S_j & 0 & S_m & 0 & S_n & 0 \\ 0 & S_i & 0 & S_j & 0 & S_m & 0 & S_n \end{bmatrix} \begin{Bmatrix} U_{ix} \\ U_{iy} \\ U_{jx} \\ U_{jy} \\ U_{mx} \\ U_{my} \\ U_{nx} \\ U_{ny} \end{Bmatrix}$$

E para o elemento isoparamétrico:

$$x = S_i x_i + S_j x_j + S_m x_m + S_n x_n$$
$$y = S_i y_i + S_j y_j + S_m y_m + S_n y_n$$

onde:

$$S_i = \frac{1}{4}(1 - \xi)(1 - \eta)$$

$$S_j = \frac{1}{4}(1 + \xi)(1 - \eta)$$

$$S_m = \frac{1}{4}(1 + \xi)(1 + \eta)$$

$$S_n = \frac{1}{4}(1 - \xi)(1 + \eta)$$

Como já foi dito, pode-se estabelecer que:

$$\{\varepsilon\} = \begin{Bmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \gamma_{xy} \end{Bmatrix} = \begin{Bmatrix} \dfrac{\partial u}{\partial x} \\ \dfrac{\partial v}{\partial y} \\ \dfrac{\partial u}{\partial y} + \dfrac{\partial v}{\partial x} \end{Bmatrix}$$

Nota-se, porém, que tanto *u* quanto *v* estão definidos em função das coordenadas naturais ξ e η, problema este contornado pela introdução da regra da cadeia, ou numa terminologia conhecida, da matriz denominada Jacobiano (*J*). Tem-se então:

$$\begin{Bmatrix} \dfrac{\partial f(x,y)}{\partial \xi} \\ \dfrac{\partial f(x,y)}{\partial \eta} \end{Bmatrix} = \overbrace{\begin{bmatrix} \dfrac{\partial x}{\partial \xi} & \dfrac{\partial y}{\partial \xi} \\ \dfrac{\partial x}{\partial \eta} & \dfrac{\partial y}{\partial \eta} \end{bmatrix}}^{[\mathbf{J}]} \begin{Bmatrix} \dfrac{\partial f(x,y)}{\partial x} \\ \dfrac{\partial f(x,y)}{\partial y} \end{Bmatrix}$$

e

$$\begin{Bmatrix} \dfrac{\partial f(x,y)}{\partial x} \\ \dfrac{\partial f(x,y)}{\partial y} \end{Bmatrix} = [\mathbf{J}]^{-1} \begin{Bmatrix} \dfrac{\partial f(x,y)}{\partial \xi} \\ \dfrac{\partial f(x,y)}{\partial \eta} \end{Bmatrix}$$

A inversa de J é:

$$[\mathbf{J}]^{-1} = \frac{1}{J_{11}J_{22} - J_{12}J_{21}} \begin{bmatrix} J_{22} & -J_{12} \\ -J_{21} & J_{11} \end{bmatrix} = \frac{1}{\det \mathbf{J}} \begin{bmatrix} J_{22} & -J_{12} \\ -J_{21} & J_{11} \end{bmatrix}$$

Para o tipo de elemento em questão, obtém-se:

$$[\mathbf{J}] = \frac{1}{4}\begin{bmatrix} [-(1-\eta)x_i + (1-\eta)x_j + (1+\eta)x_m - (1+\eta)x_n] \\ [-(1-\xi)x_i - (1+\xi)x_j + (1+\xi)x_m + (1-\xi)x_n] \end{bmatrix}$$
$$\begin{bmatrix} [-(1-\eta)y_i + (1-\eta)y_j + (1+\eta)y_m - (1+\eta)y_n] \\ [-(1-\xi)y_i - (1+\xi)y_j + (1+\xi)y_m + (1-\xi)y_n] \end{bmatrix} = \begin{bmatrix} J_{11} & J_{12} \\ J_{21} & J_{22} \end{bmatrix}$$

E retornando à energia de deformação, para o caso em estudo:

$$\Lambda^{(e)} = \frac{1}{2}\int_V \{\varepsilon\}^T[\nu]\{\varepsilon\}\,dV = \frac{1}{2}(t_e)\int_A \{\varepsilon\}^T[\nu]\{\varepsilon\}\,dA$$

onde t_e é a espessura do elemento e o índice (e) refere-se a elemento. À partir do exposto até agora, pode-se obter:

$$\{\varepsilon\} = \begin{Bmatrix} \dfrac{\partial u}{\partial x} \\ \dfrac{\partial v}{\partial y} \\ \dfrac{\partial u}{\partial y} + \dfrac{\partial v}{\partial x} \end{Bmatrix} = \overbrace{\frac{1}{\det \mathbf{J}}\begin{bmatrix} J_{22} & -J_{12} & 0 & 0 \\ 0 & 0 & -J_{21} & J_{11} \\ -J_{21} & J_{11} & J_{22} & -J_{12} \end{bmatrix}}^{[\mathbf{A}]} \begin{Bmatrix} \dfrac{\partial u}{\partial \xi} \\ \dfrac{\partial u}{\partial \eta} \\ \dfrac{\partial v}{\partial \xi} \\ \dfrac{\partial v}{\partial \eta} \end{Bmatrix}$$

e as derivadas parciais dos deslocamentos são:

$$\begin{Bmatrix} \frac{\partial u}{\partial \xi} \\ \frac{\partial u}{\partial \eta} \\ \frac{\partial v}{\partial \xi} \\ \frac{\partial v}{\partial \eta} \end{Bmatrix} = \overbrace{\frac{1}{4}\begin{bmatrix} -(1-\eta) & 0 & (1-\eta) & 0 & (1+\eta) & 0 & -(1+\eta) & 0 \\ -(1-\xi) & 0 & -(1+\xi) & 0 & (1+\xi) & 0 & (1-\xi) & 0 \\ 0 & -(1-\eta) & 0 & (1-\eta) & 0 & (1+\eta) & 0 & -(1+\eta) \\ 0 & -(1-\xi) & 0 & -(1+\xi) & 0 & (1+\xi) & 0 & (1-\xi) \end{bmatrix}}^{[\mathbf{D}]} \overbrace{\begin{Bmatrix} U_{ix} \\ U_{iy} \\ U_{jx} \\ U_{jy} \\ U_{mx} \\ U_{my} \\ U_{nx} \\ U_{ny} \end{Bmatrix}}^{\{\mathbf{U}\}}$$

Chega-se à expressão:

$$\{\boldsymbol{\varepsilon}\} = [\mathbf{A}][\mathbf{D}]\{\mathbf{U}\}$$

Nesta etapa mostra-se porque é benéfico definir os limites de um elemento como –1 e 1. Sendo transformado o termo diferencial dA = dxdy em dA = det Jdξdη, uma integral definida no domínio representado pela área de um elemento pode ser calculada com limites de integração descritos no sistema natural de coordenadas, de valores então iguais a –1 e 1. A equação da energia torna-se:

$$\Lambda^{(e)} = \frac{1}{2}(t_e)\int_A \{\boldsymbol{\varepsilon}\}^T[\boldsymbol{\nu}]\{\boldsymbol{\varepsilon}\}\,dA = \frac{1}{2}(t_e)\int_{-1}^{1}\int_{-1}^{1} \{\boldsymbol{\varepsilon}\}^T[\boldsymbol{\nu}]\{\boldsymbol{\varepsilon}\}\,\overbrace{\det \mathbf{J} d\xi d\eta}$$

Os deslocamentos nodais, por serem constantes no domínio, são "colocados para fora da integral" e a expressão resultante, derivada em relação a estes deslocamentos, resulta em $[K]^{(e)}\{U\}$. A matriz $[K]^{(e)}$ é denominada de matriz de rigidez do elemento e é dada por:

$$[\mathbf{K}]^{(e)} = t_e \int_{-1}^{1}\int_{-1}^{1} [[\mathbf{A}][\mathbf{D}]]^T [\boldsymbol{\nu}][\mathbf{A}][\mathbf{D}] \det \mathbf{J} d\xi d\eta$$

Nota-se que a obtenção requer a solução de uma integral. Esta é resolvida pelo Método de Gauss-Legendre, para o qual é conhecida a fórmula:

$$I = \int_{-1}^{1}\int_{-1}^{1} f(\xi, \eta) d\xi d\eta \cong \int_{-1}^{1} \left[\sum_{i=1}^{n} w_i f(\xi_i, \eta) \right] d\eta \cong \sum_{i=1}^{n}\sum_{j=1}^{n} w_i w_j f(\xi_i, \eta_j)$$

Os pesos w_i e w_j e os pontos ξ_i e η_i são dados em função dos limites de integração e do tipo de relação funcional a ser integrada, e padronizados (e tabelados) para limites iguais a -1 e 1 e para funções polinomiais. Daí a necessidade de uso dos valores -1 e 1, citada anteriormente.

No caso deste trabalho, foi utilizada a Quadratura Gaussiana (Método de Gauss-Legendre) de dois pontos, e prova-se que a somatória obtida representa a integral exata de um polinômio de grau igual a, no máximo, três, o que atende às necessidades vigentes.

Em posse da matriz do elemento, pode-se então obter a matriz de rigidez da estrutura, denominada global. Este procedimento é comum na implementação do Método dos Elementos Finitos e consiste numa soma de contribuições de cada elemento para a rigidez total da estrutura. Cada linha desta matriz pode ser interpretada como um conjunto de parcelas constituintes de uma das forças nodais - todas estas parcelas e a própria força nodal definidas no mesmo sistema de coordenadas - para deslocamentos dos nós da estrutura unitários (esta interpretação é válida para problemas estáticos).

Maiores detalhes sobre matrizes de rigidez podem ser obtidos nas referências [1], [2] e [3].

Para obtenção das já citadas forças ou carregamentos nodais, é necessário que, em primeiro lugar, calcule-se o trabalho realizado pelas forças externas, concentradas ou distribuídas. Este é então derivado em relação aos deslocamentos nodais e pode ser igualado, como matriz de carregamentos, à $[K]\{U\}$, onde K e U são matrizes globais. Neste procedimento nada mais se está fazendo a não ser igualar as derivadas do trabalho externo em relação a cada deslocamento, às da energia interna armazenada no material. Considerando-se apenas forças concentradas, caso tratado neste estudo, o trabalho é igual ao produto do carregamento pelo deslocamento correspondente. Derivado em relação ao deslocamento, logicamente fornece a carga nodal. É, portanto, obtido o sistema de equações linear:

$$[K]\{U\} = \{F\}$$

onde $\{F\}$ compreende forças externas (é o vetor de forças externas).

OTIMIZAÇÃO TOPOLÓGICA

A OT consiste num método computacional que gera a topologia ótima de estruturas. Basicamente, distribui o material no interior de um domínio fixo de forma a maximizar ou minimizar uma função custo especificada (por exemplo, máxima rigidez estrutural ou mínimo volume de material). O material em cada ponto do domínio pode, por exemplo, variar de um material do tipo A (ar, por exemplo) a um material do tipo B (por exemplo, um material sólido), assumindo materiais intermediários entre A e B, de acordo com uma lei de "mistura" definida, chamada modelo de material.

Um algoritmo de otimização é usado para se determinar, de forma iterativa, a distribuição ótima dos materiais, o que torna o processo rápido. Caso contrário, milhões de análises seriam necessárias para encontrá-la. A distribuição de um dado material é

representada, por exemplo, associando-se um valor de "densidade" (ou pseudo-densidade, conforme será discutido adiante no texto) a cada elemento (subdomínio), obtido da discretização do domínio inicial. Dessa forma, a OT combina, essencialmente, métodos de otimização com um método numérico de análise, no caso deste trabalho o MEF. Outros métodos numéricos de análise podem ser usados. No entanto, devem ser genéricos o suficiente, de forma a lidar com estruturas de formas complexas, resultantes da otimização. Os métodos de otimização aceleram o processo de busca da distribuição ótima de material, utilizando-se para isso da informação do gradiente (ou derivadas) da função custo em relação à quantidade de material em cada elemento. O cálculo deste gradiente é feito com base na análise estrutural (mais especificamente, à partir dos deslocamentos nodais) e será tratado, detalhadamente, adiante no texto.

Desde sua introdução, a OT vem ganhando destaque no meio acadêmico e na indústria. Torna o processo de projeto mais genérico, sistemático, otimizado, e independente da experiência específica de alguns engenheiros, fornecendo a topologia inicial, otimizada para uma certa aplicação do dispositivo a ser construído. A presença do engenheiro é necessária para a obtenção do projeto final e verificação do desempenho para o qual foi projetado (o que é feito com o auxílio de métodos numéricos e experimentais). Tem sido expandido, recentemente e com sucesso, para atuação em várias outras aplicações, como no projeto de atuadores piezoelétricos, antenas e motores eletromagnéticos, e mecanismos flexíveis [8].

No modelo de material adotado, a rigidez de cada elemento será tanto maior quanto maior for a sua "densidade", - ou "pseudo-densidade" (serão também tratadas simplesmente por densidades), já que se desconsidera o efeito da massa da estrutura – podendo variar de 0 (vazio, ou ar) a 1 (apesar de uma densidade igual a 0 ser perfeitamente possível para o problema de otimização, deve ser evitada no MEF devido à possibilidade de divisão por este mesmo valor). Uma pergunta que pode surgir neste momento é: por que não utilizar apenas 0 e 1 como valores a serem assumidos, uma vez que tal fato simplificaria em demasia a fabricação da estrutura obtida na OT? A resposta é a seguinte: não se pode garantir a obtenção da resposta do problema de OT, devido às

instabilidades numéricas geradas pela variação brusca causada pela parametrização discreta. Assim, admitindo a continuidade de valores, é garantido o alcance de solução.

O chamado método das densidades, ou, de outro modo, "Simple Isotropic Material with Penalization" (SIMP) engloba a continuidade das densidades e ainda introduz o conceito de fator de penalidade. É explicitado (o método) pela seguinte equação, modelo matemático do comportamento do material:

$$C(x) = \rho(x)^p C_0$$

O fator de penalidade p tem como função reduzir as densidades intermediárias no resultado final. O tensor C_0 é isotrópico e depende do módulo de elasticidade e do coeficiente de Poisson do material. Na implementação, simplesmente multiplica-se $\rho(x)^p$, onde ρ, "pseudo-densidade", é variável referente a um elemento finito (como já dito), por cada componente da matriz de rigidez do próprio elemento, o que dá origem à sua propriedade (ou rigidez) efetiva.

O ajuste do valor de p é discutido na literatura e pode-se mostrar que, para garantir a existência de uma estrutura formada quase que totalmente apenas por material com $\rho = 1$ e $\rho = 0$ (inexistência de material), para o estado plano de tensões (caso abordado), o fator deve ser maior ou igual a três. Porém é limitado superiormente, já que um valor muito alto faz o problema tornar-se aproximadamente discreto, e então experimentar a já citada não garantia de solução. Utilizou-se $p = 3$. Este valor de p, apesar de prevenir o aparecimento de densidades intermediárias, torna o problema não-convexo, ou seja, com vários mínimos locais. A ilustração de uma função convexa e uma não-convexa, para apenas uma dimensão, pode ser vista Figura 6.

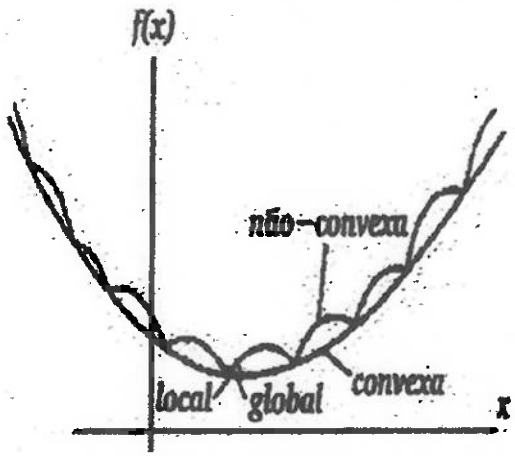


Figura 6 Funções convexa e não-convexa.

Uma explicação mais detalhada sobre os possíveis valores de p, bem como uma discussão sobre demais modelos de material encontrados na literatura são visto na referência [8]. É aqui apenas importante salientar que se pode descrever a prova da existência de soluções com pequenas quantidades de densidades intermediárias, fator extremamente importante para a fabricação, apesar de serem estas soluções ótimos locais, e não globais, o que seria o ideal. Porém, o chamado método da continuação pode dar origem a resultados melhores.

O método da continuação consiste em se obter a resposta do problema convexo (obtido com valor do fator de penalidade unitário), o qual conta com apenas um mínimo - alcançado independentemente do valor inicial dado às densidades dos elementos, valor este necessário para análise de elementos finitos, ou seja, obtenção de deslocamentos nodais - e partir desta para a resolução do problema não-convexo (obtido para valor de fator de penalidade maior do que a unidade). Partir da solução convexa significa utilizar como valores iniciais de densidades as respostas da otimização da função convexa. O resultado final estará sempre próximo do ótimo do problema convexo, e, provavelmente, será melhor do que uma solução encontrada para uma distribuição inicial de densidades qualquer e uso do fator de penalidade diferente de um. Computacionalmente, o método será discutido adiante.

OBTENÇÃO DOS DESLOCAMENTOS NODAIS

Sabe-se que nem todos os deslocamentos nodais definidos num malhamento de elementos finitos são responsáveis pelo equilíbrio de uma determinada força nodal. Assim, a matriz de rigidez global possuirá diversos valores iguais a zero. Isto remete ao uso de métodos de armazenamento de matrizes esparsas, que diminuem a quantidade de memória necessária para armazenamento de dados. Além disto, existem diversos procedimentos que se utilizam das matrizes definidas nestes métodos na solução de sistemas lineares, e reduzem o tempo de processamento computacional.

Um dos procedimentos de solução (neste caso, a obtenção dos deslocamentos) é o Método dos Gradientes Conjugados para Sistemas Esparsos (MGCSE) que, em um de seus algoritmos mais simples, resolve um sistema de equações lineares e apenas no caso em que a matriz A é simétrica e positiva definida, sendo o sistema igual a:

$$[A]\{x\} = \{b\}$$

O método é baseado na idéia de minimização de:

$$f(x) = \tfrac{1}{2}\{x\}[A]\{x\} - \{b\}\{x\}$$

Para este trabalho, será, porém, utilizado o Método dos Gradientes Biconjugados para Sistemas Esparsos (MGBSE), uma vez que se tem em mãos, da referência [4], o algoritmo correspondente implementado. Porém, este método engloba o MGCSE, sendo seu caso "um pouco mais genérico", para o qual permitem-se equações lineares não necessariamente positivas definidas ou simétricas. No entanto, não apresenta uma conexão simples e direta com a minimização de funções, como a apresenta o MGCSE, relação esta citada acima. Maiores detalhes sobre o MGCSE, o MGBSE ou sobre matrizes esparsas e métodos de armazenamento podem ser obtidos na referência [4].

PROGRAMAÇÃO LINEAR (PL) E PROGRAMAÇÃO LINEAR SEQUENCIAL (PLS)

Optou-se para a solução do problema de otimização pela PLS por ser esta abordagem bem conhecida (e, portanto, confiável) e pela comodidade apresentada pela existência de uma função praticamente pronta para uso (para Programação Linear), escrita em Linguagem Fortran de programação, que se encaixou perfeitamente nos requisitos deste trabalho.

A PLS é um caso especial da PL, para a qual o processo de otimização se dá iterativamente. É necessária quando se lida com funções não lineares (o caso em questão), linearizadas então em torno de um ponto (no caso unidimensional) para uso no algoritmo.

Se uma função é linearizada, representa o comportamento da não linear apenas num pequeno intervalo ao redor do ponto utilizado. Por outro lado, se são tomados vários pontos, pertencentes à função não linear, e esta última é aproximada por uma sucessão de linearizações em torno dos pontos, é representada de forma mais satisfatória.

A otimização por PL de uma relação não linear deve então ser levada a cabo iterativamente e, em cada iteração, ser restrita a um intervalo definido como aceitável (ou, em outras palavras, para o qual a divergência entre as duas funções – linearizada e não linear - é pequena), tomado em torno dos valores das variáveis obtidas na iteração anterior. O processo termina quando é verificada a condição (ou condições) verdadeira para o critério de parada.

Atendo-se agora à PL, pode-se dizer que compreende a identificação ou obtenção, de forma organizada, das N_i restrições dadas que devem ser satisfeitas pelas variáveis de projeto ótimas, ou seja, aquelas variáveis cujos valores, uma vez inseridos na função objetivo, ou a função a ser otimizada, tornam-na ótima. N_i é o número de variáveis. No caso deste trabalho, estas variáveis são as densidades dos elementos em que se divide a estrutura ou corpo estudado.

Este tipo de abordagem em forma de busca de restrições é particular de algoritmos baseados em PL. Isto porque se sabe de antemão que os valores máximos de uma função linear restringida (e de gradiente diferente de zero) ocorrem necessariamente nos limites destas restrições, o que não pode ser afirmado para funções não lineares. Um caso simples deste tipo de problema, para apenas duas variáveis independentes, pode ser visto na Figura 7.

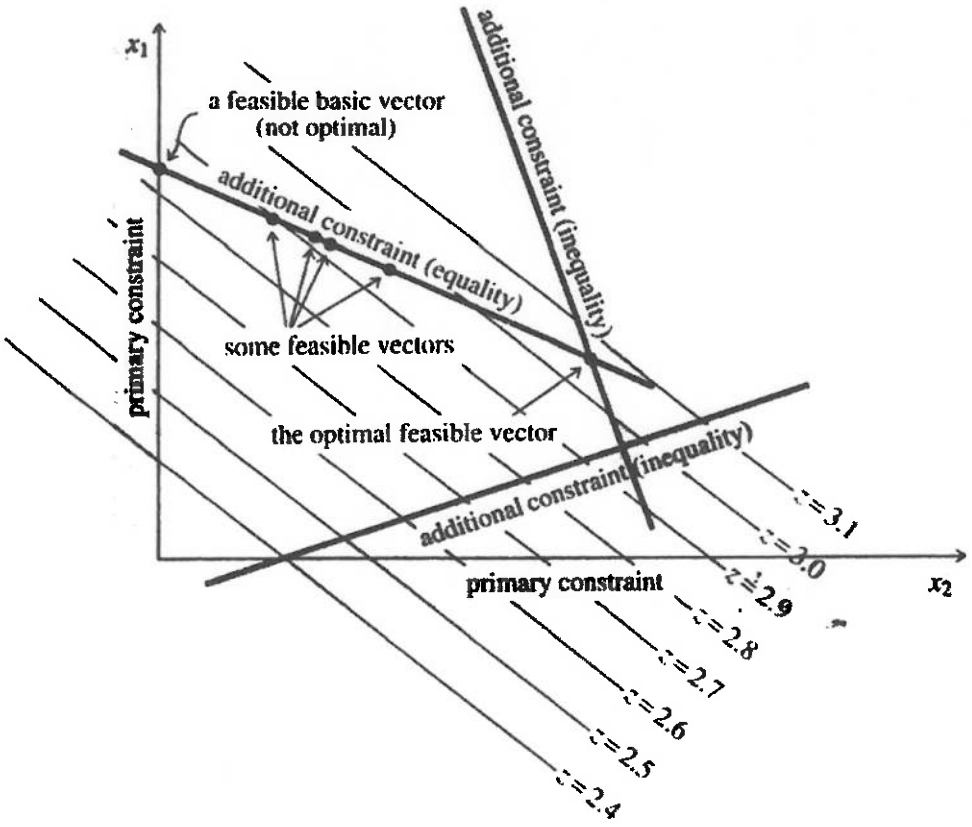


Figura 7 Visualização de um caso simples de problema de PL.

Esta representa um problema de maximização, sendo z a função objetivo, ou variável dependente, para a qual plotaram-se as curvas de nível. O vetor possível é aquele que satisfaz todas as restrições impostas e o vetor básico possível também se encontra nos limites da região permitida. Pode-se notar a existência tanto de restrições em forma de equações quanto na forma de inequações. O método de solução simplesmente procura entre vetores básicos possíveis até que o chamado vetor ótimo

possível seja encontrado. Não o faz aleatoriamente, mas com base nos componentes do gradiente da função objetivo (como já explicitado), relativos a cada variável, o que direciona a busca do ponto de ótimo, organizando-a.

Quer-se maximizar a rigidez. Então, indiretamente, minimiza-se a chamada flexibilidade F_1 da estrutura, tida como função objetivo e igual, por definição, ao dobro da energia elástica. É dada por:

$$F_1 = F^t U$$

e, pelo equilíbrio estático:

$$F_1 = U^t KU$$

Sendo as "pseudo-densidades" as variáveis de projeto, o gradiente, que contém as derivadas primeiras da função objetivo considerada, deve ser calculado com base nessas. Sendo ρ_i a "pseudo-densidade" do elemento finito i, tem-se que:

$$\frac{\partial F_1}{\partial \rho_i} = \frac{\partial \left(U^t KU\right)}{\partial \rho_i} = \frac{\partial \left(U^t\right)}{\partial \rho_i} KU + U^t \frac{\partial (K)}{\partial \rho_i} U + U^t K \frac{\partial (U)}{\partial \rho_i}$$

Após alguma manipulação matricial e depois da introdução da derivação da equação de equilíbrio ($[K]\{U\} = \{F\}$), chega-se, conforme é visto na referência [7] (nesta, as variáveis de projeto são as áreas das barras de treliça; mas o desenvolvimento, ou manipulação matricial, é o mesmo) na relação:

$$\frac{\partial F_1}{\partial \rho_i} = -U^t \frac{\partial K}{\partial \rho_i} U$$

Como já se disse, o problema não linear deve ser dividido em diversos lineares, abordagem esta adotada pelo PLS. Obtém-se então, à partir da expansão em série de

Taylor, função F_2, linearizada, conforme referência [7]. As constantes, obtidas quando da lenearização, não influenciam no processo de otimização; por isso são retiradas da equação. O problema de PL assume a forma:

$$F_2^{linear} = \frac{\partial F_1}{\partial \rho_1} \Big|_{\rho=\rho^0} \rho_1 + \frac{\partial F_1}{\partial \rho_2} \Big|_{\rho=\rho^0} \rho_2 + \ldots + \frac{\partial F_1}{\partial \rho_n} \Big|_{\rho=\rho^0} \rho_n$$

$$\sum_{i=1}^{M} \rho_i \leq \rho^*$$

$$\rho_i^{\min} \leq \rho_i \leq \rho_i^{\max}, \quad i = 1, \cdots M$$

Sendo o volume máximo restringido, o que é representado pela inequação que engloba a somatória das "pseudo-densidades", e sendo as restrições às "pseudo-densidades" (limites móveis) variadas a cada passo de linearização. Uma forma de cálculo de limites móveis é dada a seguir, sem que, no entanto, seja usada neste trabalho (a usada no trabalho é descrita no tópico referente à implementação numérica).

Um limite maior ou menor é imposto às variações de restrições, dependendo da taxa de crescimento da função objetivo com relação às variáveis de projeto, em cada iteração do processo, o que diminui consideravelmente o tempo computacional em relação à abordagem de limites fixos, se estes forem muito pequenos. Por outro lado, se apresentarem números exagerados, de modo a se economizar tempo, pode-se ignorar, numa iteração, o verdadeiro ponto de ótimo, o que dará origem à oscilação do valor da função objetivo. Assim, se, por exemplo, a função não linear apresenta valores baixos para as suas derivadas, pode-se admitir grandes valores para os limites móveis, já que, na região considerada, a relação não linear possui praticamente o mesmo comportamento de sua equivalente linear. Caso contrário, admitem-se pequenos limites móveis.

No caso de múltiplos casos de carregamento, o que muda é apenas o cálculo do gradiente, para o qual cada componente será:

$$\frac{\partial F}{\partial \rho_i} = -peso_1 U_1^t \frac{\partial K}{\partial \rho_i} U_1 - peso_2 U_2^t \frac{\partial K}{\partial \rho_i} U_2 + \cdots - peso_L U_L^t \frac{\partial K}{\partial \rho_i} U_L$$

Os índices 1, 2 e L referem-se aos casos de carregamento e cada *peso* deve ser ajustado para que o carregamento mais importante seja considerado como tal.

Fica assim definido o "subproblema" de otimização, explicitado pela minimização da função linear.

Maiores detalhes sobre a PLS podem ser obtidos na referência [5] e sobre PL, nas referências [4], [5] e [6]. Sobre limites móveis, na referência [10].

INSTABILIDADE DE TABULEIRO E COMPLEXIDADE ESTRUTURAL

Este tópico não é parte estrutural do programa, mas seu estudo teórico se faz muito importante, uma vez que introduz os principais problemas encontrados quando da utilização do método de OT.

Primeiramente, será citado o problema da instabilidade de tabuleiro, ou do tabuleiro de damas, ou do tabuleiro de xadrez, nomes estes encontrados na literatura. A literatura encontrada em [10] sugere que surge pela diferença nas ordens de interpolação dos campos de deslocamentos e de "pseudo-densidades". Manifesta-se na forma de descontinuidades da função $\rho(x,y)$ ("pseudo-densidade") - sendo x e y as coordenadas que definem o domínio da estrutura -, de tal forma que os vizinhos de vértice de um elemento apresentem valores de "pseudo-densidade" praticamente iguais e os vizinhos de aresta os apresentem também praticamente iguais, mas com valores extremamente maiores (ou menores), aproximadamente como num tabuleiro de damas. Dependente da discretização, a instabilidade de tabuleiro é indesejável, principalmente por não ser, de fato, uma solução ótima ao problema estrutural, mas uma solução ótima para a

representação matemática deste problema, e por representar dificuldades construtivas exacerbadas. Um exemplo é dado e corresponde Figura 8.

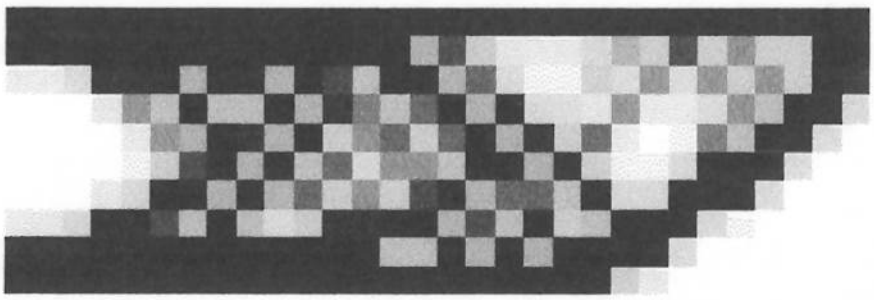


Figura 8 Estrutura obtida para discretização de 300 elementos, engastada do lado esquerdo e com carregamento aplicado no canto superior direito. Nota-se a presença de tabuleiro de damas.

A complexidade estrutural, por sua vez, ocorre devido ao aumento da discretização do domínio, o que, por outro lado, melhora a aproximação do campo de deslocamentos estruturais e, também, a interpretação dos contornos da estrutura. Um exemplo deste problema é visto na Figura 9.

Figura 9 Problema da complexidade estrutural.

Vários métodos foram propostos para o controle da instabilidade, como o uso de superelementos, ou seja, a união de dois ou mais elementos, o uso de elementos – de "pseudo-densidades" constantes - de oito ou nove nós e relações lineares entre as propriedades do material base e as propriedades efetivas do material estrutural, etc. Porém, o método mais difundido atualmente, sem que, no entanto, ataque as causas dos problemas, mas suas conseqüências, é o Método do Controle de Gradientes (MCG) [10].

O MCG pode ser aplicado na forma de filtros ou restrições na formulação de OT, e consiste em se restringir a variação espacial da variável de projeto, de tal forma que se evitem a complexidade estrutural e o tabuleiro de damas. Apesar de ser o controle de gradientes mais elegante matematicamente e efetivo, em sua forma de restrições, torna o programa lento. Neste trabalho, então, será usado o método de filtros, que apresenta resultados bastante interessantes do ponto de vista construtivo, principalmente quando aliado a um método de pós-processamento de eliminação de "pseudo-densidades" intermediárias do contorno estrutural, discutido adiante no texto. O filtro calcula, com base num certo número de elementos vizinhos considerados, o valor de uma quantidade (referente ao elemento possuidor dos vizinhos) importante para a otimização, como a "pseudo-densidade" ou o gradiente da função objetivo em relação à variável de projeto, ou outros, a cada iteração. Assim, o valor da nova quantidade, após a alteração realizada pela PL, será uma média, no caso deste trabalho ponderada pela distância entre os elementos e seus volumes.

Pretende-se aplicar o filtro aos limites móveis, o que gera alguns inconvenientes. São eles: o desempenho do filtro depende do método de determinação dos limites móveis e dos valores extremos definidos para estes limites [10]. Deve-se, para contornar estes problemas, impedir variações bruscas dos limites móveis e impor valores pequenos o suficiente para eles, de tal forma que o efeito do filtro não seja atenuado. Pode-se perceber, porém, que estas duas abordagens são, de fato, pré-requisitos para uso efetivo da PLS, o que elimina, pelo menos para o filtro, as limitações ou problemas na aplicação.

IMPLEMENTAÇÃO NUMÉRICA

Foram criadas geometrias arbitrárias no software ANSYS 5.4 e gerados arquivos de dados de entrada para o programa desenvolvido pelo aluno, do tipo ASCII. Um exemplo deste tipo de arquivo é visto no Anexo A. A construção de estruturas ou corpos quaisquer pode ser auxiliada por texto encontrado na referência [2] e no próprio manual do programa ANSYS, apesar de ser o citado na referência [9] pertencente a uma versão mais antiga do software usado.

Os dados de entrada são as conectividades de cada elemento, as coordenadas dos nós, locais e valores de aplicação de cargas concentradas, indicação de quais nós estão impedidos de se moverem (condições de contorno ou restrições), valores do coeficiente de Poisson e do módulo de elasticidade, o número de nós (*N*) e o número de elementos finitos (*M*). A conectividade de um elemento define os nós aos quais este elemento está conectado; assim, no caso de um elemento de quatro nós, quatro valores, que representam a numeração dos nós (previamente estabelecida pelo próprio ANSYS, sem a interferência, neste caso, do usuário), descrevem sua conectividade. Retornando ao tratamento de dados, estes estão salvos em arquivos com extensão ".txt", criados pelo software ANSYS, como já mencionado. À partir deles, foram criadas as matrizes fixas *ID*, ou matriz de graus de liberdade não restritos e enumerados, a matriz *edof*, formada com valores representativos das conectividades dos elementos (já orientada para criação da matriz *LM*), a *fb*, ou vetor de carregamentos nodais, a *coord*, ou matriz de coordenadas nodais e a *conect*, de conectividades nodais; tirou-se também o valor de *ksize*, ou número de nós livres da estrutura, para os quais se calculam deslocamentos. Com *ID* e *edof*, desenvolve-se então a matriz *LM*. Apenas *LM*, *fb*, *ksize*, *coord* e *conect* serão usados diretamente no MEF e na solução do sistema de equações resultante.

A criação das matrizes *LM*, *coord* e, finalmente, *conect* são importantes para a montagem da matriz de rigidez global de MEF, sendo que se calcula uma matriz de rigidez de um elemento sem ser considerada a existência dos outros nós e elementos da estrutura. Desta forma, necessita-se do conhecimento da correspondência entre as

posições das componentes de rigidez na matriz local a as posições das componentes de rigidez na matriz global, correspondência esta dada por *LM*. O leitor interessado em detalhes sobre a montagem da matriz de rigidez global e em mais explicações sobre a matriz *LM*, pode consultar as referências [1], [3] e [7].

Para o caso deste trabalho, a matriz do elemento terá oito linhas e colunas, uma vez que cada nó tem dois graus de liberdade, e a global (para a estrutura inteira) poderá apresentar um número muito maior (no caso, será definido, este número, pela razão de aspecto *ksize* x *ksize*).

Com a montagem da matriz de rigidez global finalizada, e em posse dos carregamentos aplicados, pode-se resolver o sistema de equações lineares, como já se disse, pelo MGBSE, e então se obtêm os deslocamentos nodais.

Algo a ser ressaltado é o seguinte fato: não foi aproveitado o benefício da utilização das matrizes esparsas no que diz respeito à economia de memória. A implementação direta da matriz global pelo método do armazenamento indexado, do modo como foi estudado em referência recomendada sobre o assunto ([4]), não era simples o suficiente para que se mostrasse viável neste trabalho. Em outras palavras, o desenvolvimento do programa ficaria bem mais complicado. Mas, a mesma referência bibliográfica ([4]) apresenta uma função em Linguagem C que armazena uma matriz esparsa pelo armazenamento indexado (da forma requerida pelo algoritmo de MGBSE), e assim pôde-se, ao menos, aproveitar a maior eficiência computacional no cálculo, introduzida pelo MGBSE, quando comparado, por exemplo, a algoritmos baseados no método de Gauss-Jordan.

As respostas do MEF podem ser consideradas bem próximas numa comparação entre o ANSYS e o software desenvolvido, e mostram precisão suficiente para a OT. Resultdos são mostrados no Anexo C.

No estudo do software de elementos finitos desenvolvido, como primeira etapa do projeto, notou-se a existência, em alguns casos de estrutura, de elementos triangulares na malha de MEF. Este problema (existente já que o software criado trabalha apenas com elementos de quatro lados) é contornado pelo próprio ANSYS, que gera quatro coordenadas, mesmo para o elemento de três lados, sendo que duas destas terão valores

iguais, ou seja, o mesmo nó é tomado duas vezes. O arquivo ASCII não é prejudicado, e, portanto, a rotina de leitura de dados não encontra qualquer tipo de problema. Assim, se uma linha deste arquivo contém os números "1 2 5 6", relativos à conectividade do elemento, a linha correspondente ao elemento triangular poderá, por exemplo, ser da forma: "1 2 6 6". A existência do "nó duplo" também não prejudica o funcionamento do software na etapa de análise por MEF (não há ocorrência de singularidade), na qual "se enxerga" o elemento como um "quadrilátero de três lados", sendo um deles de comprimento nulo.

Prosseguindo com a implementação, salienta-se a obtenção de alguns valores necessários, fornecidos por interface com o usuário. São estes as "pseudo-densidades" iniciais *x0*, o volume máximo de material *Vfrac*, dado como fração ou porcentagem do volume total (varia de zero a um), o raio do filtro e o número de iterações para eliminação de densidades intermediárias no contorno da estrutura final. O programa recebe o valor de *x0* e o repassa para todos os elementos, o que causará uma distribuição inicial de material uniforme. Outros modos de distribuir-se inicialmente material no domínio, como a distribuição randômica, podem ser usados, sobretudo em casos de teste de unicidade de solução [10].

Nesta etapa do programa, é criado o vetor de carregamentos *f*, que engloba todos os casos de carga possíveis, tendo o tamanho de *ksize*num_load_cases*, onde a última variável contém o número de casos. É usado na análise estrutural, para a qual *ksize* valores são tomados a cada cálculo de deslocamentos, ou seja, em cada caso de carregamento.

O loop de otimização é iniciado com o cálculo da matriz de rigidez global, para os valores de "pseudo-densidades" *x0* dados pelo usuário, e prossegue com a esparsificação desta matriz. Num loop menor, cada setor de *f*, de tamanho *ksize*, é então usado no cálculo dos deslocamentos nodais, e o componente do gradiente de cada elemento de discretização é, posteriormente, contemplado com o valor correspondente, advindo de fórmula de cálculo já definida anteriormente. Este loop recomeça para o caso de carga seguinte e os novos valores dos componentes do gradiente, referentes, é claro, a

um mesmo elemento, são somados aos anteriores. No fim do loop, é obtido o gradiente da função objetivo.

Calculam-se os limites móveis e a rotina de otimização é chamada, recebendo o gradiente e as restrições ao problema de minimização de trabalho, e devolvendo as novas densidades. Uma rotina, em linguagem computacional Fortran, foi usada, por representar economia de tempo. Esta tem o "nome" de *dsplp*.

Os limites móveis são calculados à partir de um valor inicial e limitados por valores superiores e inferiores. A sua variação será pequena, numa iteração, se *sign1*, ou seja, uma variável definida para cada elemento e representativa do sinal da densidade da iteração corrente menos a da iteração anterior, for positiva, *sign2*, a variável também definida para cada elemento e representativa do sinal da densidade da iteração anterior menos a da iteração anterior à anterior, for negativa e *sign3*, definida da mesma forma, mas para a iteração anterior à anterior e para a anterior a esta, for positiva, ou se *sign1* for negativa, *sign2* for positiva e *sign3* negativa. Em outras palavras, a variação dos limites móveis será pequena se a densidade estiver oscilando ao redor de um valor médio, o que pode ocorrer, no processo iterativo, quando se está próximo ao mínimo da função objetivo. Caso contrário, esta variação será maior.

A cada iteração, são testadas as condições de loop. Este pode terminar (e, então, serem gerados os arquivos de saída, tratados adiante no texto) caso o número máximo de iterações *NUMIT* seja excedido, ou caso a variação relativa da função objetivo seja menor que *ITERTOL*, tipicamente igual a 10^{-3}. Uma condição positiva para número máximo de iterações ultrapassado significa que se pode afirmar que a resposta não convergiu, enquanto que uma resposta afirmativa para a questão da variação pequena da função objetivo, rende a consideração da convergência. Verificada esta última, o usuário deve informar se quer continuar com as iterações. O caso afirmativo significa que o filtro poderá ser acionado, reativado ou desativado, e que o valor do fator de penalidade será modificado (do valor unitário para o valor de interesse três), para realização do método da continuação. Desativar o filtro e "rodar" o programa por mais três iterações, de maneira geral, elimina contornos estruturais não definidos (melhora a resolução

destes contornos), ou seja, de densidades intermediárias, o que facilita a interpretação dos resultados para posterior fabricação.

Finalmente, são gerados os arquivos de saída de dados para o software MATLAB e para o ANSYS que compreendem, no primeiro caso, os gráficos para volume e função objetivo por iteração e as densidades dos elementos finitos e, no segundo caso, dados já formatados para plotagem da estrutura final, com a correspondência entre cores e densidades definida previamente. No caso de malhas regulares, as densidades escritas no arquivo ".m" (de leitura do MATLAB) podem ser usadas para plotagem da estrutura, porém, com escalas de cores entre o preto e o branco, e com correspondência entre cores e densidades definida internamente pelo próprio software MATLAB.

Prosseguindo, o programa encerra com a liberação de memória alocada.

A listagem do programa desenvolvido é relatada no Anexo B, com partes julgadas menos importantes suprimidas.

RESULTADOS

A não ser em casos especiais, os quais serão previamente comunicados, a discretização será feita em elementos finitos regulares e isoparamétricos de quatro nós, numa razão de aspecto 3x1 (ou seja, comprimento divido por largura resulta em três). A visualização é feita no software MATLAB. Cada elemento tem uma unidade de lado, sendo a unidade de medida, uma qualquer. Os elementos têm sempre a unidade como comprimentos laterais para facilitar o uso do filtro, que pode ser pensado em termos de número de elementos. Assim, o raio do filtro será:

$$raio = cam \cdot \sqrt{2}$$

Sendo *cam* o número de camadas de vizinhos. Para que o leitor se situe melhor, a primeira camada possui oito vizinhos, o que pode ser visto na Figura 10. Obviamente, um número diferente para o raio pode ser adotado (por exemplo, o valor 1.2, para o qual seriam considerados apenas os vizinhos de aresta do elemento em questão, ou, em outras palavras, quatro vizinhos somente), ou seja, este não precisa ser discretizado da forma como foi aqui posto, através da equação vista acima. Mas, esta forma de utilização do filtro já é suficiente para a solução dos problemas existentes e facilita o trabalho do usuário.

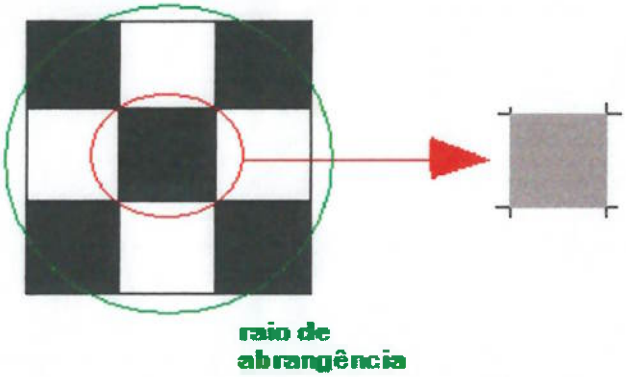


Figura 10 Efeito do filtro sobre densidade do elemento central e definição de raio (do filtro).

As estruturas a serem otimizadas, ou os domínios estruturais iniciais podem ser vistos nas figuras: Figura 11, Figura 12 e Figura 13. Cada figura irá corresponder a um chamado Caso. Assim, a estrutura engastada com carregamento inferior direito, compreende o Caso 1.

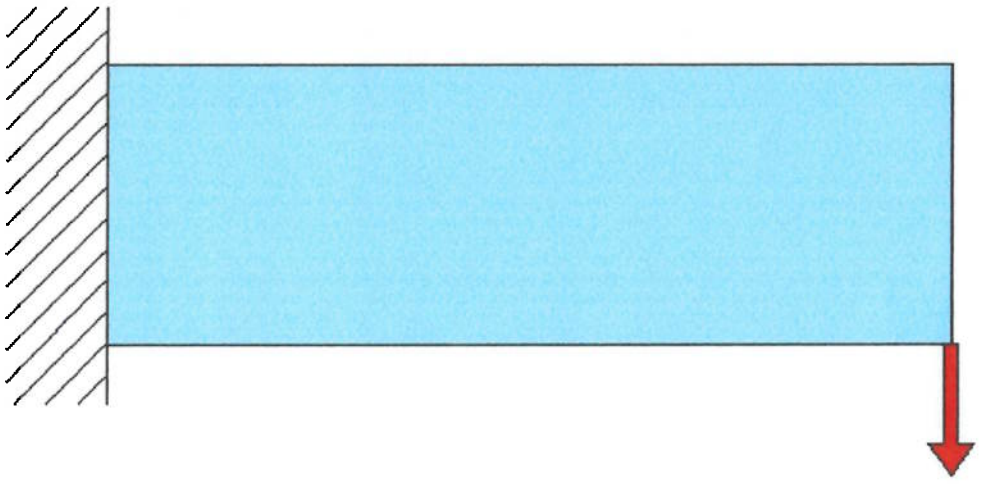


Figura 11 Situação referente ao Caso 1. A discretização foi feita com dois mil e setecentos elementos finitos.

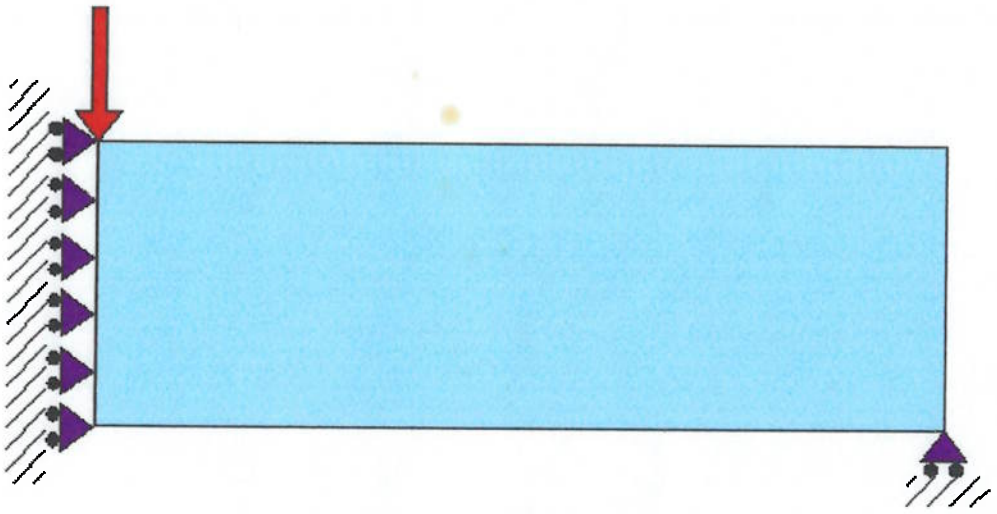


Figura 12 Situação referente ao Caso 2. A discretização foi feita com dois mil e quatrocentos elementos finitos.

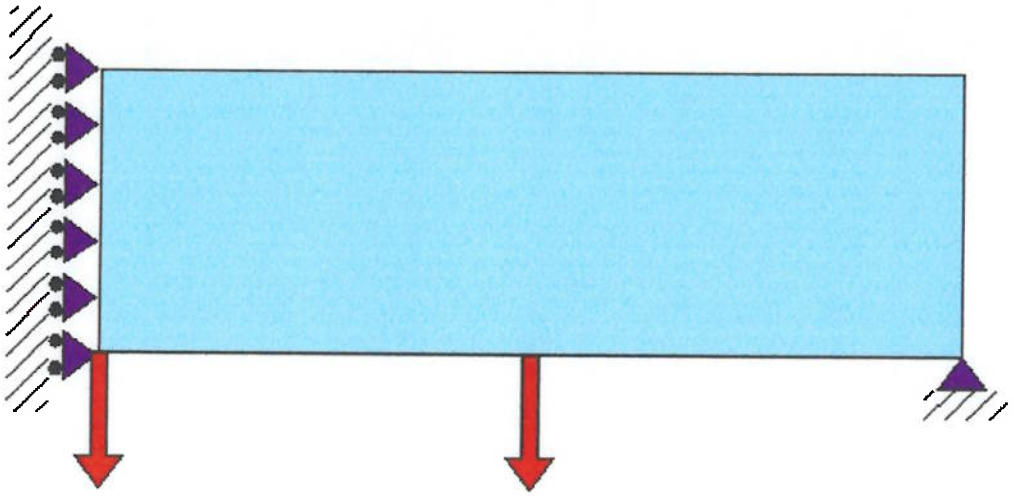


Figura 13 Situação referente ao Caso 3. A discretização foi feita com dois mil e quatrocentos elementos finitos.

Para os Casos 1, 2 e 3, com densidades iniciais uniformes no domínio e iguais a 0.5, volume restringido em sessenta por cento do domínio total considerado, fator de penalidade igual a três e com um raio abrangendo uma camada, obteve-se, respectivamente (em relação à numeração dos casos dada no início do parágrafo), as figuras: Figura 14, Figura 15 e Figura 16. Foram também postos aqui os gráficos do volume e da função objetivo (os gráficos para a função objetivo são também denominados de curvas de convergência), ambas as variáveis tomadas por iteração.

Demais curvas de convergência e de volume, para outros tipos de problema de otimização, serão omitidas.

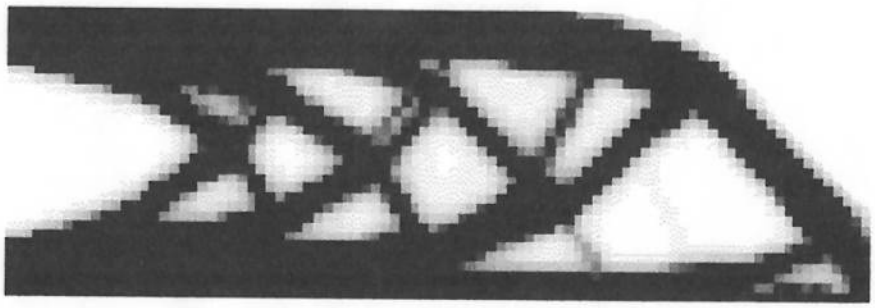


Figura 14 Estrutura obtida para o Caso 1. A função objetivo tem o valor final 11.334116.

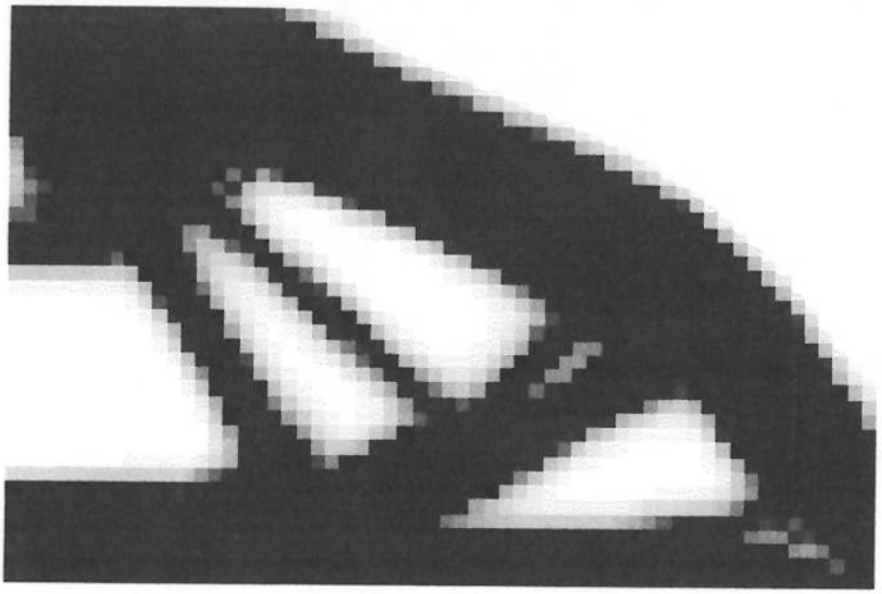


Figura 15 Estrutura obtida para o Caso 2. A função objetivo tem o valor final 2.799411.

Figura 16 Estrutura obtida para o Caso 3. A função objetivo tem o valor final 7.650370

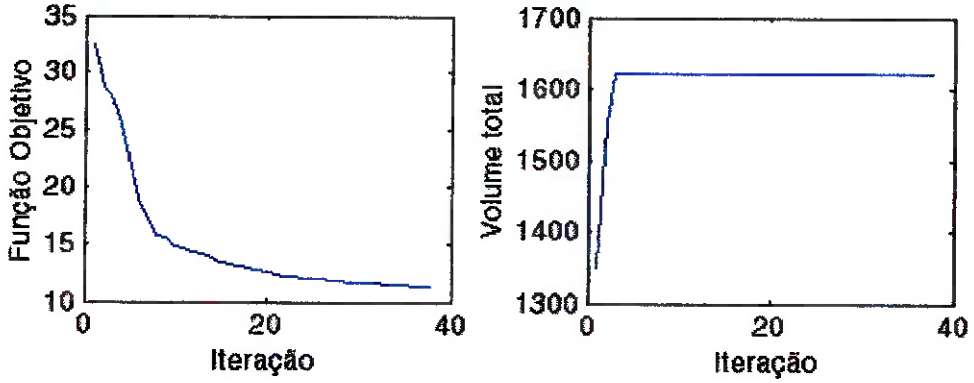


Figura 17 Curvas obtidas para o Caso 1.

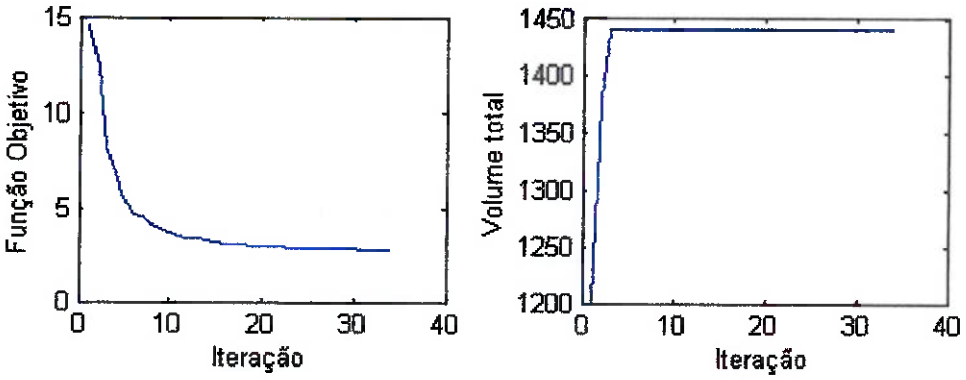


Figura 18 Curvas obtidas para o Caso 2.

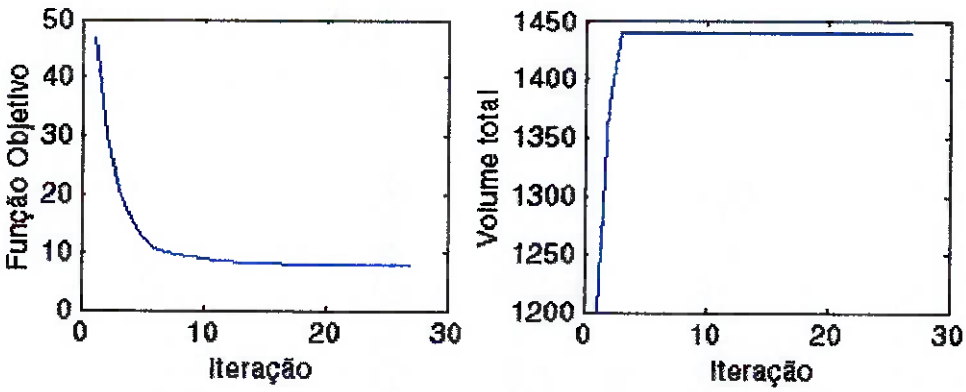


Figura 19 Curvas obtidas para o Caso 3.

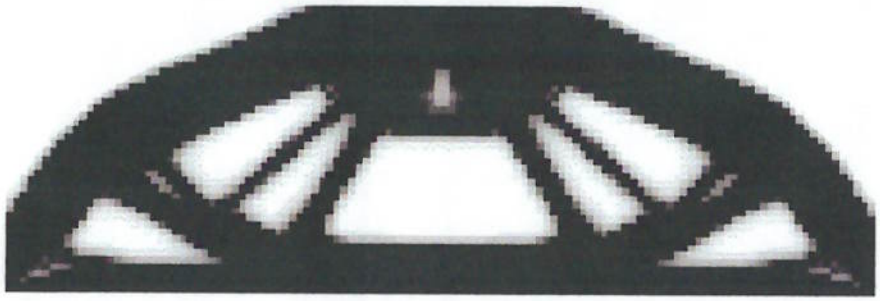


Figura 20 Caso 2 rebatido por simetria. Esta seria a estrutura real obtida.

Figura 21 Caso 3 rebatido por simetria. Esta seria a estrutura real obtida.

Pode-se observar, pelas figuras, a potencialidade do método. Nota-se, claramente, que, no local de aplicação das cargas (incluindo os apoios ou engastes ativos, ou seja, aqueles que "trabalham" para manter o equilíbrio, sendo então, as reações, diferentes de zero), como era de se supor, há a existência de material. Porém, nota-se também problemas de resolução de contornos e mesmo algumas estruturas

extremamente delgadas que, apesar de reforçarem o conjunto, imprimem grandes dificuldades de fabricação, devendo ser retiradas, com prejuízo da performance da peça.

Para solucionar os problemas, propõem-se o método da continuação aliado a um raio diferente para o filtro e o prolongamento, por mais algumas iterações, para eliminação das densidades intermediárias nos contornos. São obtidas as figuras: Figura 22, Figura 23 e Figura 24, para o caso do raio igual a 1.5 (uma camada), sessenta por cento de domínio ocupado, uso do método da continuação e três iterações para melhorar a resolução. Pelos valores finais obtidos para a função objetivo, pode-se notar que as estruturas geradas pelo método da continuação são mesmo mais rígidas, por estarem mais próximas do ótimo global. No Caso 1, porém, percebe-se que os mínimos obtidos foram praticamente iguais, o que pode mesmo ser identificado pela comparação entre figuras, ou pelos valores da função objetivo, a saber, 11.334116 para o Caso 1 sem a continuação e 11.248780 para o caso com continuação. Nesta situação, o que fez a diferença final foram as três iterações, que contribuem para deixar o resultado mais próximo de uma resposta com tabuleiro de damas, mais rígida.

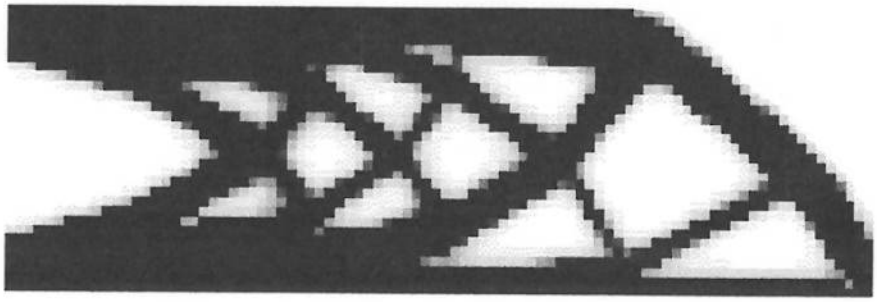


Figura 22 Estrutura obtida para o Caso 1 e o método da continuação ativo. A função objetivo tem o valor final 10.934071.

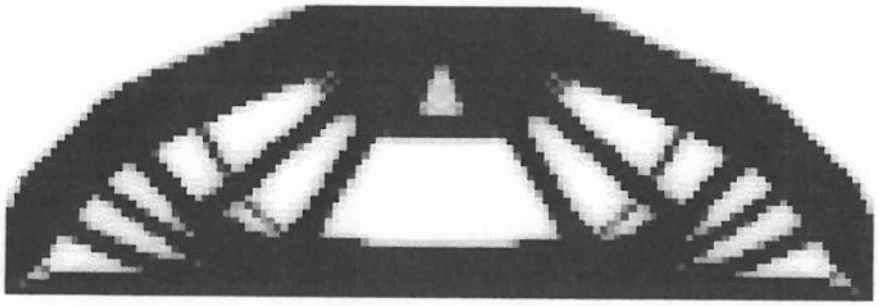


Figura 23 Estrutura obtida para o Caso 2 e o método da continuação ativo. A função objetivo tem o valor final 2.767086 (para apenas uma metade).

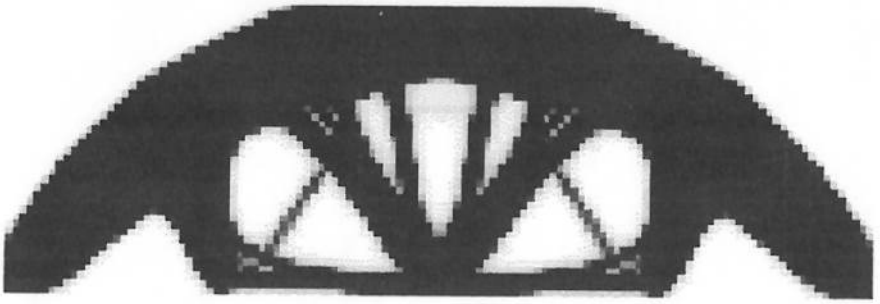


Figura 24 Resposta para o Caso 3 e o método da continuação ativo. A função objetivo tem o valor final 7.399160 (para apenas uma metade).

No Caso 1, mas para um volume de quarenta por cento do volume total considerado, é obtida a Figura 25. Esta mostra um resultado bem mais interessante do ponto de vista construtivo, o que sugere que se estude a aplicação deste valor de restrição (quarenta por cento do volume total). Com isto em mente, foram obtidas as figuras: Figura 26 e Figura 27, para os Casos 2 e 3, respectivamente. No caso da Figura 26, um maior valor de função objetivo foi obtido devido a um maior valor de carregamento aplicado. Optou-se por aumentar este carregamento pois o original gerava valores de flexibilidade muito próximos de zero, o que ocasionou problemas numéricos.

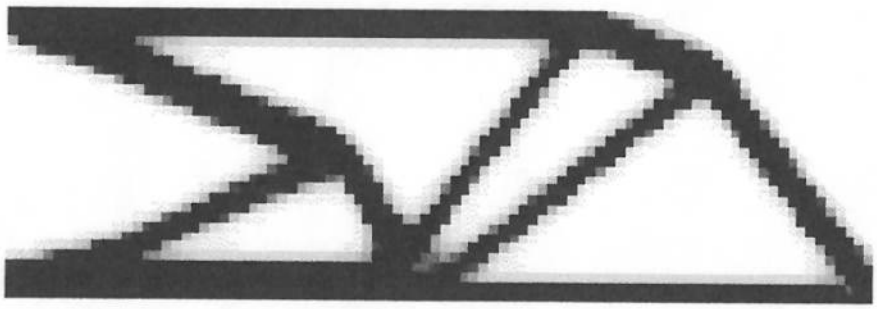


Figura 25 Estrutura obtida para o Caso 1, porém com restrição de quarenta por cento do volume, e o método da continuação ativo. A função objetivo tem o valor final 16.652338.

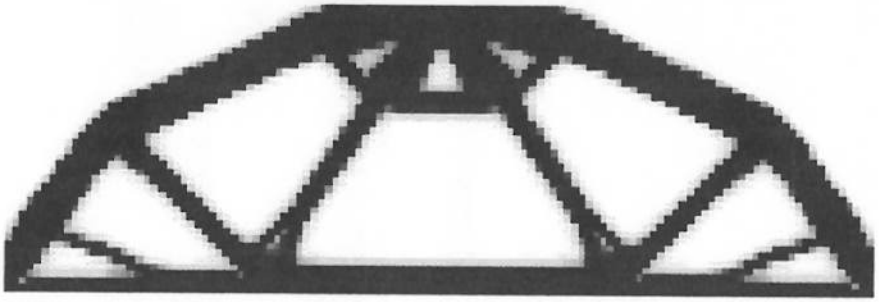


Figura 26 Estrutura obtida para o Caso 2, porém com restrição de quarenta por cento do volume, e o método da continuação ativo. A função objetivo tem o valor final 398.857108 (para apenas uma metade).

Figura 27 Estrutura obtida para o Caso 3, porém com restrição de quarenta por cento do volume, e o método da continuação ativo. A função objetivo tem o valor final 10.823697 (para apenas uma metade).

No caso de raios maiores, são plotadas as figuras: Figura 28, Figura 29, Figura 30 e Figura 31, as quais não mostraram bons resultados. O filtro força o método de otimização a estruturas com gradientes de material, no domínio, muito pouco acentuados, o que dá origem a estas "anomalias", as quais são, no que diz respeito à rigidez, estruturas de comportamento inferior.

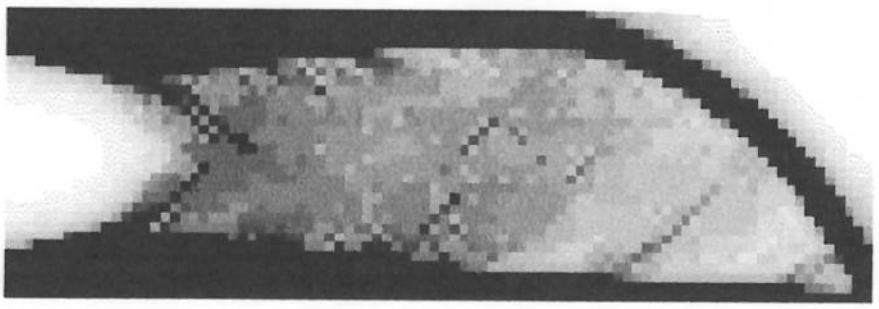


Figura 28 Estrutura obtida para o Caso 1, o raio abrangendo duas camadas de elementos e o método da continuação ativo. A função objetivo tem o valor final 13.486223.

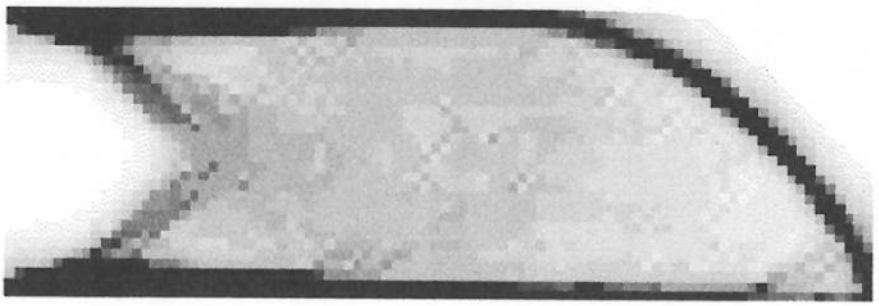


Figura 29 Estrutura obtida para o Caso 1, o raio abrangendo duas camadas de elementos, quarenta por cento do domínio permitido e o método da continuação ativo. A função objetivo tem o valor final 29.668639.

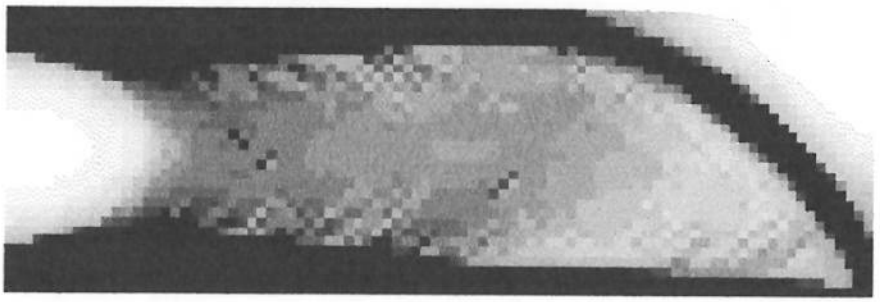


Figura 30 Estrutura obtida para o Caso 1, o raio abrangendo três camadas de elementos e o método da continuação ativo. A função objetivo tem o valor final 13.695132.

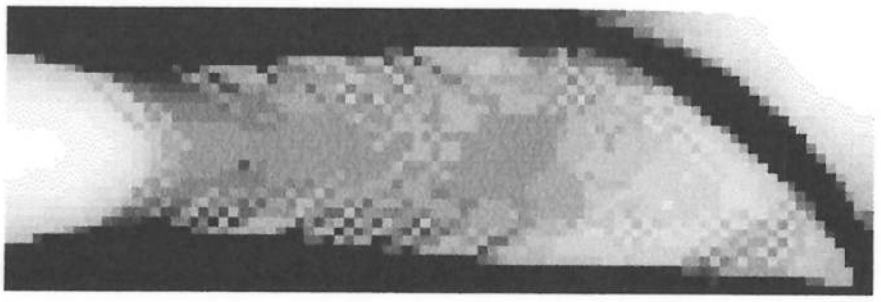


Figura 31 Estrutura obtida para o Caso 1, o raio abrangendo três camadas de elementos e o método da continuação ativo. A função objetivo tem o valor final 14.163698.

CONCLUSÃO

A próxima etapa do trabalho, no caso em que existisse uma maior preocupação com a fabricação da peça, seria a chamada interpretação do resultado, que nada mais é que a geração de um modelo, ou desenho, equivalente à estrutura obtida. Uma posterior análise por elementos finitos seria realizada, para verificação e validação, e então a etapa de CAM, ou fabricação assistida por computador, à partir do resultado validado, seria levada a cabo.

Neste trabalho, com maior valor acadêmico que industrial, no entanto, optou-se pela aplicação e estudo de métodos, como o da continuação, para melhor explicitar as nuanças da OT, preocupação esta inexistente fora do ambiente da graduação, onde se querem apenas resultados. Assim, mesmo podendo as soluções encontradas serem testadas pelo MEF, basta que sejam comparadas, qualitativamente, à resultados vistos na literatura.

Apesar da preocupação acadêmica, é inegável o quão interessante pode ser um método deste para a indústria. Para malhas relativamente "pesadas", ou seja, de aproximadamente três mil elementos, num computador GenuineIntel x86 Family 6 Model 8 Stepping 10, com 512MB RAM, atingiu-se, para a obtenção da resposta para um problema, dados o domínio, restrição e as demais entradas, o tempo de, aproximadamente, dez minutos.

Como continuação do trabalho, propõe-se, em primeiro lugar, o desenvolvimento de uma rotina de filtragem, a qual trabalhe com a noção de camadas, mesmo no caso de malhas irregulares, problema este de solução não trivial e que admite diversas variações de implementação (mas que não representará um grande desafio, no entanto). Além disto, pode-se propor o uso de uma interpolação linear, do mesmo modo que foi feito para a descrição do campo de deslocamentos no MEF, mas no caso das densidades estruturais, o que, como é sabido da literatura, pode eliminar o problema de tabuleiro de damas, sem que, no entanto, seja utilizado o filtro, o qual torna o programa lento.

Apesar de relativamente rápido, o programa ainda pode, em alguns pontos, ser melhorado e tornado ainda mais veloz. Isto é possível de ser feito em pontos como na montagem da matriz de rigidez global, à partir do conhecimento do fato de ser simétrica em relação à diagonal principal. Um outro ponto que aumenta o tempo é a alocação de memória, que deve ser reduzida a um mínimo, mesmo que, para tanto, o software se utilize de bastante memória, o que é feito colocando-se toda a alocação fora do loop de otimização.

Desde que tenha noções básicas de construção de modelos em CAD, sobre uso do software ANSYS e da OT, qualquer um pode utilizar esta nova ferramenta desenvolvida, tanto para projeto quanto para o aprofundamento nas técnicas de otimização topológica, campo da engenharia que se encontra em franca expansão, não só na área de estruturas mecânicas, mas também na área médica, como na obtenção de imagens do corpo humano através de um tomógrafo por impedância elétrica, e na área de micromecanismos, para obtenção de microdispositivos ótimos [8], como manipuladores de células, micropinças, microválvulas, sensores, etc.

BIBLIOGRAFIA

[1] Bathe, K. J., "Finite Elements Procedures", Englewood Cliffs, N.J.: Prentice-Hall, 1996.

[2] Moaveni, S., "Finite Element Analysis: Theory and Application with ANSYS", Prentice-Hall, Inc., New Jersey, 1999.

[3] Cook, R. D., Malkus, D. S., Plesha, M. E., "Concepts and Applications of Finite Element Analysis", third edition, John Wiley & Sons, Inc., 1989.

[4] Press, W. H., Teukolsky, S. A., Vetterling, W. T., Flannery, B. P., "Numerical Recipes in C – The Art of Scientific Computing", Cambrige University Press, 1999.

[5] Haftka, R. T., Gürdal, Z., "Elements of Structural Optimization", third edition, Kluwer Academic Publishers, Netherlands, 1999.

[6] Vanderplatz, G. N., "Numerical Optimization Techiniques for Engineering Design: with Applications", McGraw-Hill, 1984.

[7] Afonso, H. D., "Desenvolvimento de software para projeto de peças mecânicas otimizadas usando o método de Otimização Topológica", Relatório de Iniciação Científica, processo nº 00/01035-2, FAPESP, 2000.

[8] Lima, C. R., "Projeto de Mecanismos Flexíveis Usando o Método de Otimização Topológica", São Paulo, 2002. 146p. Dissertação (Mestrado) – Escola Politécnica, Universidade de São Paulo.

[9] Getting Started: for Revision 5.1. Houston, Swanson Analysis Systems, Inc., September 1994. 1st Revision.

[10] Cardoso, E. L., "Controle de Complexidade na Otimização Topológica de estruturas Contínuas", Porto Alegre, 2000. 120p. Dissertação (Mestrado) – Escola de Engenharia, Universidade Federal do Rio Grande do Sul.

ANEXO A

Exemplo de Arquivo de Entrada (gerado pelo ANSYSED 5.5; o programa listado no Anexo B deve ser ligeiramente modificado para lê-lo, uma vez que foi desenvolvido, o programa, para leitura de arquivos do ANSYS 5.4). Os dados não lidos foram suprimidos.

```
(...)
NUMOFF,NODE,     23
NUMOFF,ELEM,     14
(...)
(3i8,6e16.9)
       1       0       0 1.00000000     1.00000000
       2       0       0 2.00000000     7.00000000
       3       0       0 1.25000000     2.50000000
       4       0       0 1.50000000     4.00000000
       5       0       0 1.75000000     5.50000000
       6       0       0 8.00000000     3.00000000
       7       0       0 3.50000000     6.00000000
       8       0       0 5.00000000     5.00000000
       9       0       0 6.50000000     4.00000000
      10       0       0 10.0000000     1.00000000
      11       0       0 9.00000000     2.00000000
      12       0       0 4.00000000     2.00000000
      13       0       0 8.50000000     1.25000000
      14       0       0 7.00000000     1.50000000
      15       0       0 5.50000000     1.75000000
      16       0       0 2.50000000     1.50000000
      17       0       0 2.31866004     3.84876334
      18       0       0 4.07969155     3.48929813
      19       0       0 2.53708790     2.83451537
      20       0       0 2.93530951     4.71551128
      21       0       0 5.85529995     2.89618617
      22       0       0 7.24862177     2.32146292
      23       0       0 8.21077018     1.85574483
N,R5.3,LOC,     -1,
EBLOCK,19,SOLID
(19i7)
      1      1      1      1      0      0      0      0      4      0      1     13     23     22     14
      1      1      1      1      0      0      0      0      4      0      2     16     19      3      1
```

```
    1   1   1   1   0   0   0   0   4   0   3   19   17   4   3
    1   1   1   1   0   0   0   0   4   0   4   17   20   5   4
    1   1   1   1   0   0   0   0   4   0   5   20   7   2   5
    1   1   1   1   0   0   0   0   4   0   6   20   18   8   7
    1   1   1   1   0   0   0   0   4   0   7   18   21   9   8
    1   1   1   1   0   0   0   0   4   0   8   21   22   6   9
    1   1   1   1   0   0   0   0   4   0   9   22   23   11   6
    1   1   1   1   0   0   0   0   4   0   10   23   13   10   11
    1   1   1   1   0   0   0   0   4   0   11   18   19   16   12
    1   1   1   1   0   0   0   0   4   0   12   21   18   12   15
    1   1   1   1   0   0   0   0   4   0   13   14   22   21   15
    1   1   1   1   0   0   0   0   4   0   14   20   17   19   18
(...)
MPDATA,R5.0, 1,EX ,     1, 1, 38000.0000   ,
(...)
MPDATA,R5.0, 1,NUXY,     1, 1, 0.300000000   ,
(...)
D,    1,UX , 0.00000000   , 0.00000000
D,    1,UY , 0.00000000   , 0.00000000
D,    2,UX , 0.00000000   , 0.00000000
D,    2,UY , 0.00000000   , 0.00000000
D,    3,UX , 0.00000000   , 0.00000000
D,    3,UY , 0.00000000   , 0.00000000
D,    4,UX , 0.00000000   , 0.00000000
D,    4,UY , 0.00000000   , 0.00000000
D,    5,UX , 0.00000000   , 0.00000000
D,    5,UY , 0.00000000   , 0.00000000
D,    7,UX , 0.00000000   , 0.00000000
D,    7,UY , 0.00000000   , 0.00000000
F,    6,FX , 40.0000000   , 0.00000000
F,   10,FY , -20.0000000   , 0.00000000
(...)
```

ANEXO B

Listagem do Programa (as funções dsprsin.c, linbcg.c, nrutil.c, simplx.c e o "header" nrutil.h podem ser encontrados na referência [4]; as funções mais simples foram retiradas deste anexo)

```
#include <stdio.h>
#include <malloc.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>
#include "nrutil.h"
#include "funcs.h"

#define arquivo1  "2400_e_rx_id_rxy_f-50_di_f-50_mli.txt" /*input*/
#define arqANSYS  "2400_e_rx_id_rxy_f-50_di_f-50_mli_saida.txt"  /*output*/
#define arqMATLAB "2400_e_rx_id_rxy_f-50_di_f-50_mli.m"
#define TAM 150
#define NMAX (300000*TAM)

extern void dusrmt_();
extern void dsplp_();

unsigned long *ija;
double *sa;

void main () {
      //declaraçao de variaveis
      int *flag1, *pos_node, *pos_node_ksize, filter_en, opcao, contador, lprc, flag2;
      int i, j, k, p, M, N, count, *spc1, *LM, *ID, iter;
  int a1, a2, a3, d2, f2, compara, *conect, q[8], *izrov, *iposv, flag, it;
      int b1, b2, b3, b4, b5, b6, b7, b8, b9, b10, b11, dof1, dof2, dof3, dof4, *edof;
      int mrelas, nvars, *ind, *ibasis, *iwork;
  int lw, liw, info, lamat, lbm;
  double *fcases, *f, *filt_xupper, *filt_xlower, resultado_final, *resultado, penal;
      double *Kel, **K, ni, E, *fb, *ye, *xe, *sign1, *sign2, *sign3, *ml;
  double *ub, *bf, *ubif, err, passo, *loada3;
      double *xnew, *xold, *xmin, *xmax, *xupper, *xlower, *volume, *difer;
      double ube[8], tempmat[8], dcompl, objetivold;
      double *compliancia, *objetivo, *gradF, xmin0, xmax0;
      double valor1, valor2, *funobj, *volitera;
      double *bl, *bu, *prgopt, *dattrv, *primal, *duals, *work, *costs;
      unsigned long *count_forces, num_load_cases;
```

```
    unsigned long count_cases, count_cases_aux, fat1, fat2, fat3, *store;
    unsigned long ii, jj, kk, ksize;
 float *percent, *g, Vfrac, x0, radius, gg;
    float *a, **A;
    float a4, a5, f5, *coord;
 char *direction, *direction_ksize;
    char *temp, d1[5], d3[5], e1, e2, e3;
 char f1[5], f3[5], f4[5], f6, f7, f8, c1[14], c2[14], c3[15], c4[5], c5, c6[5], c7[5];
    char g1[15], g2[10], g3[5], g4[5];
    FILE *arquivo;
    FILE *arq1, *arq3;
    ////////////////////////////////////////////////////////////////////////////
    ////////////////////////////INICIO DA LEITURA DE DADOS/////////////////////////////
    ////////////////////////////////////////////////////////////////////////////
    /* Abertura do arquivo .txt */
    arq1=fopen(arquivo1,"r");
    temp = (char *)malloc(TAM*sizeof(char ));
    ////////////////////////////////////////////////////////////////////////////
    /* VALOR DE "N" (NUMERO DE NOS) */
    compara = -1;
    while ( compara > 0 || compara < 0) {
            fgets(temp, TAM, arq1);
            compara = strncmp( temp, "NUMOFF,NODE,", 12);
    }
    sscanf( temp, "%s %d", c1, &N ); /*le a string do temp e atribui a N como inteiro*/
    ////////////////////////////////////////////////////////////////////////////
    /* VALOR DE "M" (NUMERO DE ELEMENTOS) */
    fgets(temp, TAM, arq1);
    sscanf( temp, "%s %d", c2, &M ); /*le a string do temp e atribui a M como inteiro*/
    ////////////////////////////////////////////////////////////////////////////
    /* VETOR "COORD" (COORDENADAS DOS NOS) */
    compara = -1;
    while ( compara > 0 || compara < 0) {
            fgets(temp, TAM, arq1);
            compara = strncmp( temp, "(3i8,6e16.9)", 12);
    }
fgets(temp, TAM, arq1);
    coord = (float *)malloc(2*N*sizeof(float));
    count = 1;
    k=0;
    while (count<=N){
            sscanf( temp, "%d %d %d %f %f", &a1, &a2, &a3, &a4, &a5 );
            coord[k] = a4;
            coord[N+k] = a5;
            count = count+1;
```

```
            k++;
            fgets(temp, TAM, arq1);
        }
        ////////////////////////////////////////////////////////////////////////////
        /* VETOR "CONECT" (GRAUS DE LIBERDADE DOS NOS) */
        conect = (int *)malloc(4*M*sizeof(int ));
        fgets(temp, TAM, arq1);
        fgets(temp, TAM, arq1);
        fgets(temp, TAM, arq1);
        count=1;
        k=0;
        while (count<=M){
            fgets(temp, TAM, arq1);
            sscanf( temp, "%d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d",
                &b1, &b2, &b3, &b4, &b5, &b6, &b7, &b8, &b9, &b10, &b11, &dof1,
&dof2,
                &dof3, &dof4 );//b11 deve ser tirado ou adicionado, conforme a versão
do ANSYS
            conect[k*4] = dof1;
            conect[k*4+1] = dof2;
            conect[k*4+2] = dof3;
            conect[k*4+3] = dof4;
            count=count+1;
            k++;
        }
        ////////////////////////////////////////////////////////////////////////////
        /* MODULO DE ELASTICIDADE "E" */
        compara = -1;
        while ( compara > 0 || compara < 0) {
            fgets(temp, TAM, arq1);
            compara = strncmp( temp, "MPDATA,R5.0, 1,EX", 17);
        }
        sscanf( temp, "%s %s %c %s %s %lf", c3, c4, &c5, c6, c7, &E );
        ////////////////////////////////////////////////////////////////////////////
        /* COEFICIENTE DE POISSON "NI" */
        compara = -1;
        while ( compara > 0 || compara < 0) {
            fgets(temp, TAM, arq1);
            compara = strncmp( temp, "MPDATA,R5.0, 1,NUXY,", 20);
        }
        sscanf( temp, "%s %s %s %s %lf", g1, g2, g3, g4, &ni );
        printf("%f", ni);
        ////////////////////////////////////////////////////////////////////////////
        /* MATRIZ DE RESTRIÇOES "SPC1" (GRAUS DE LIBERDADE FIXOS) */
        spc1 = (int *)malloc(2*N*sizeof(int ));
```

```
for (k=0; k<2*N; k++) spc1[k]=1;
compara = -1;
while ( compara > 0 || compara < 0) {
        fgets(temp, TAM, arq1);
        compara = strncmp( temp, "D,", 2);
}
while ( compara == 0) {
        sscanf (temp, "%2s%d%3s", d1, &d2, d3);
        sscanf(d3, "%c %c %c", &e1, &e2, &e3);
        if (e3 == 'X'){
                spc1[2*d2-2]=0;
        }
        if (e3 == 'Y'){
                spc1[2*d2-1]=0;
        }
        fgets(temp, TAM, arq1);
        compara = strncmp( temp, "D,", 2);
}
////////////////////////////////////////////////////////////////////////////
/* MATRIZ DE FORÇAS "LOADA3" */
loada3 = (double *)malloc(2*N*sizeof(double));
pos_node = (int *)malloc(2*N*sizeof(int));
direction = (char *)malloc(2*N*sizeof(char));
for (k=0; k<2*N; k++) {
        loada3[k]=0;
        pos_node[k] = 0;
}
compara = strncmp( temp, "F,", 2);
while ( compara == 0 ) {
        sscanf( temp, "%2s%d%3s%s%f", f1, &f2, f3, f4, &f5 );
        sscanf(f3, "%c %c %c", &f6, &f7, &f8);
        if (f8 == 'X'){
                loada3[2*f2-2]=f5;
                pos_node[2*f2-2] = f2;
                direction[2*f2-2] = 'X';
        }
        if (f8 == 'Y'){
                loada3[2*f2-1]=f5;
                pos_node[2*f2-1] = f2;
                direction[2*f2-1] = 'Y';
        }
        fgets(temp, TAM, arq1);
        compara = strncmp( temp, "F,", 2);
}
fclose( arq1 );//final da leitura de dados
```

```
////////////////////////////////////////////////////////////////////////////
////////////////////////////MATRIZES FIXAS//////////////////////////////////
////////////////////////////////////////////////////////////////////////////
/* VETOR "ID" (MATRIZ DE DOF's NAO RESTRITOS ENUMERADOS) */
ID = (int *)malloc(2*N*sizeof(int));
count=1;
for (i=0; i<2*N; i++) {
        if (spc1[i]==0) {
                ID[i]=0;
        }
        else {
                ID[i]=count;
                count++;
        }
}
////////////////////////////////////////////////////////////////////////////
/* TAMANHO "SIZEM" DA MATRIZ REDUZIDA */
ksize = count-1;
////////////////////////////////////////////////////////////////////////////
/* VETOR DE CARGAS "FB" */
fb = (double *)malloc(ksize*sizeof(double));
pos_node_ksize = (int *)malloc(2*N*sizeof(int));
direction_ksize = (char *)malloc(2*N*sizeof(char));
for (ii=0; ii<ksize; ii++) {
        fb[ii]=0;
        pos_node_ksize[ii] = 0;
}
count=0;
count_cases = 0;
for (j=0; j<2*N; j++) {
        if (ID[j]!=0) {
                fb[count]=loada3[j];
                pos_node_ksize[count] = pos_node[j];
                direction_ksize[count] = direction[j];
                if (fb[count]!=0) {
                        count_cases++;
                }
                count++;
        }
}
////////////////////////////////////////////////////////////////////////////
/* MATRIZ "EDOF" (GRAUS DE LIBERDADE) */
edof = (int *)malloc(8*M*sizeof(int));
j=0;
for (i=0; i<4*M; i++) {
```

```
            edof[j] = 2*conect[i]-1;
            edof[j+1] = 2*conect[i];
            j=j+2;
    }
    ////////////////////////////////////////////////////////////////////////////
    /* VETOR "LM" */
    LM = (int *)malloc(8*M*sizeof(int));
    for (i=0; i<8*M; i++) LM[i]=0;/*para zerar o vetor inicial*/
    for (i=0; i<M; i++) {
            for (j=1; j<=4; j++) {
                    for (k=1; k<=2; k++) {
                            p=2*(j-1)+k;
                            LM[p-1+(8*i)]=ID[(k-1)+(edof[(j-1)*2+(8*i)]-1)];

                    }
            }
}
    free(temp);
    free(spc1);
    free(loada3);
    free(ID);
    free(edof);
    ////////////////////////////////////////////////////////////////////////////
    ////////////////////////ALOCAÇÃO DE MEMÓRIA////////////////////////////////
    ////////////////////////////////////////////////////////////////////////////

    // Valores necessários
    penal = 1;
    printf("Digite o valor das densidades iniciais xnew: ");
    scanf("%f",&x0);
    printf("\nDigite o valor da quantidade de material maxima permitida: ");
    scanf("%f",&Vfrac);
    printf("\nDeseja utilizar o filtro?(s = 1, n = 0): ");
    scanf("%d",&filter_en);
    if (filter_en){
            printf("\n  Digite o valor do raio para o filtro: ");
            scanf("%f",&radius);
    }
    printf("\n  Digite o numero de iteraçoes para melhorar a resolucao dos contornos: ");
    scanf("%d",&lprc);
    printf("\n\n");
    xmin0 = TOL;
    xmax0 = ITOL;
    // matriz das densidades
    for (i=0; i<M; i++) {
```

```
            xmin[i]=xmin0;
            xmax[i]=xmax0;
}
    //chute inicial para a rotina linbcg
    for (ii=0; ii<ksize; ii++){
            ub[ii]=0.0;
    }
    //vetor das densidades iniciais
    for (i=0; i<M; i++) {
            xnew[i]=x0;
    }
    //inicialização das variáveis auxiliares para cálculo dos limites
    //móveis
    for (i=0; i<M; i++) {
            sign1[i] = 1;
            sign2[i] = 1;
            sign3[i] = 1;
            ml[i] = 0.15;
    }
    ////////////////////////////////////////////////////////////////////////////
    //CRIAÇÃO DO VETOR f COM TODOS OS CASOS DE CARGA//////////////
    ////////////////////////////////////////////////////////////////////////////
    /*cálculo do número de casos de carga num_load_cases para alocação
    de memória e obtenção de condição de loop*/
    num_load_cases = 0;
    count_cases_aux = count_cases;
    while (count_cases_aux > 0) {
            fat1 = fatorial(count_cases); //fatorial: calcula o fatorial do argumento;
            fat2 = fatorial(count_cases_aux);
            fat3 = fatorial(count_cases - count_cases_aux);
            num_load_cases = num_load_cases + fat1/(fat2*fat3);
            count_cases_aux--;
    }
    //vetor de forças f (contém todos os "num_load_cases" casos)
    f = (double *)malloc(ksize*num_load_cases*sizeof(double ));
    percent = (float *)malloc(num_load_cases*sizeof(float ));
    num_load_cases = 0;
    count_cases_aux = count_cases - 1;
    for(kk = 0; kk < count_cases; kk++) {
            for(ii = 0; ii < count_cases; ii++) {
                    store[ii] = 0;
            }
            fat1 = fatorial(count_cases);
            fat2 = fatorial(count_cases_aux + 1);
            fat3 = fatorial(count_cases - count_cases_aux - 1);
```

```
            for (ii = 0; ii < fat1/(fat2*fat3); ii++) {
                    for(jj = 0; jj < ksize; jj++) {
                            fcases[jj] = 0;
                    }
                    count_forces[0] = 0;
                    flag1[0] = 0;
                    printf("Caso %d de cargas: ", num_load_cases);
                    vector_forces(direction_ksize, pos_node_ksize, flag1, store,
count_cases_aux + 1, count_forces, ksize, fcases, fb, count_cases_aux);
                    for (jj=0; jj < ksize; jj++){
                            f[jj + num_load_cases * ksize] = fcases[jj];
                    }
                    printf("\n\nDigite o peso deste caso para que a estrutura seja otimizada:
");
                    scanf("%f", g);
                    printf("\n");
                    percent[num_load_cases] = g[0];
                    num_load_cases++;
            }
            count_cases_aux--;
    }
    ////////////////////////INICIO DO LOOP DE OTIMIZAÇAO////////////////////////////
    for (i=0; i<(MAXITER+1); i++) {
            difer[i]=1;
    }
    for (i=0; i<MAXITER; i++) {
            volume[i]=0;
    }
    for (i=0; i<MAXITER; i++) {
            compliancia[i]=0;
    }
    // inicio do loop
    it=0;
    opcao = 1;
    flag2 = 0;
    contador = 0;
    while (opcao) {
            while (opcao) {
                    it = it+1;
                    //atribui a xold o valor de xnew
                    (...)
                    //Calculo do volume total
                    for (i=0; i<M; i++) {
                            volume[it-1] = volume[it-1]+xold[i];
                    }
```

```
//zera a matriz "K"
(...)
for(k=0; k<M; k++) {
        for(i = 0; i < 4; i++){
                xe[i] = coord[conect[i+k*4]-1];
                ye[i] = coord[conect[i+k*4]+N-1];

        }
        //Ke: modifica Kel, o qual recebe a matriz de rigidez do elemento;
        Ke(xe,ye,E,ni,Kel);
        ////////////////////////////////////////////////////////////////////////////
        ///////////////MATRIZ DE RIGIDEZ GLOBAL K///////////////
        ////////////////////////////////////////////////////////////////////////////
        for(j = 0; j < 8; j++){
                q[j] = LM[j+k*8];
        }
        for (i=1; i<=8; i++) {
                if (q[i-1]!=0) {
                        for (j=1; j<=8; j++) {
                                if (q[j-1]!=0) {
                                        K[q[i-1]][q[j-1]] = K[q[i-1]][q[j-1]]
+ pow(xold[k],penal)*Kel[(i-1)*8+(j-1)];
                                }
                        }
                }
        }
}
////////CALCULO DO DESLOCAMENTOS UB///////////////
////////////////////////////////////////////////////////////////////////////
//montagem da matriz esparsa
dsprsin(K, ksize, THRESH, NMAX, sa, ija);
//resoluçao do sistema K.UB = fb
for (i=0; i<M; i++) {
        gradF[i]=0;
}
for(kk = 0; kk < num_load_cases; kk++) {
        if(percent[kk] != 0) {
                for (ii=1; ii<=ksize; ii++){
                        bf[ii]=f[ii - 1 + kk * ksize];
                        ubif[ii]=ub[ii - 1];
                }
                linbcg(ksize, bf, ubif, ITOL, TOL, ITMAX, &iter, &err);

                for (jj=0; jj<ksize; jj++){
                        ub[jj] = ubif[jj+1];
```

```
                    }
                    for (ii=0; ii<ksize; ii++) {
                        compliancia[it-1] = compliancia[it-1] + ub[ii]*f[ii +
kk * ksize];
                    }

    ////////////////////////////////////////////////////////////////////////////
    ///////GRADIENTE DA FUNÇÃO OBJETIVO//////////////////
    ////////////////////////////////////////////////////////////////////////////

                    for (i=0; i<M; i++) {
                        for (k=0; k<8; k++) {
                            ube[k]=0;
                        }
                        //extrai de "ub" o vetor de deslocamento de cada
elemento
                        for (j=0; j<8; j++) {
                            if (LM[i*8+j]!=0) {
                                ube[j]=ub[LM[i*8+j]-1];
                            }
                        }
                        //calcula a derivada da compliancia de cada
elemento
                        for (k=0; k<8; k++) {
                            tempmat[k]=0;
                        }
                        for(k = 0; k < 4; k++){
                            xe[k] = coord[conect[k+i*4]-1];
                            ye[k] = coord[conect[k+i*4]+N-1];
                        }
                        Ke(xe,ye,E,ni,Kel);
                        for (j=0; j<8; j++) {
                            for (k=0; k<8; k++) {
                                tempmat[j]=tempmat[j]          +
ube[k]*Kel[(j*8)+k];

                            }
                        }
                        dcompl=0;
                        for (j=0; j<8; j++) {

                            dcompl      =      dcompl      +      -
penal*pow(xold[i],(penal-1))*ube[j]*tempmat[j];
                        }
                        gradF[i] = gradF[i] + percent[kk] * dcompl;
```

```
                }
        }
}
for (i=1; i<=M; i++) {
        costs[i] = gradF[i-1];
}
objetivo[it-1] = compliancia[it-1];
////////////////////////////////////////////////////////////////////////////
//////////////////////////////Limites Móveis//////////////////////////////
////////////////////////////////////////////////////////////////////////////

(...)
////////////////////////////////////////////////////////////////////////////
////////////////////Filtragem dos Limites Móveis/////////////////////
////////////////////////////////////////////////////////////////////////////

if (filter_en) {
        filter(M, N, coord, conect, xupper, filt_xupper, radius);
        filter(M, N, coord, conect, xlower, filt_xlower, radius);

}
////////////////////////////////////////////////////////////////////////////
////////////PROGRAMAÇÃO LINEAR (PL)/////////////////////
////////////////////////////////////////////////////////////////////////////

//Preparação dos dados para DSPLP (Programação Linear)
/* Rotina "DSPLP" que roda o LP em fortran
Recebe:
nvars   -> numero de variaveis de projeto
mrelas  -> numero de restricoes
flag    -> minimizacao (0) ou maximizacao (1)
bu e bl -> limites moveis superior e inferior (arrays com nvars posicoes)
costs   -> derivadas da funcao objetivo
Retorna:
info    -> flag de sucesso do LP
xnew    -> nova distribuicao das variaveis de projeto
*/
//números de váriaveis (nvars) e número de restrições (mrelas)
nvars = M;
mrelas = 1;
//Dimensiona arrays que serao utilizados na dsplp
bl=dvector(1,mrelas+nvars);
bu=dvector(1,mrelas+nvars);
prgopt=dvector(1,4);
dattrv=dvector(1,(2*mrelas*nvars)+1+nvars);
```

```
ind=ivector(1,mrelas+nvars);
primal=dvector(1,mrelas+nvars);
duals=dvector(1,mrelas+nvars);
ibasis=ivector(1,mrelas+nvars);
//Copia as informacoes de restricao lateral
if(filter_en) {
        for (i=1;i<=nvars;i++) {
                bu[i]=filt_xupper[i-1];
                bl[i]=filt_xlower[i-1];
                //Significa que a variavel e maior ou igual ao limite
                //inferior e menor ou igual ao limite superior
                ind[i]=3;
        }
}
else {
        for (i=1;i<=nvars;i++) {
                bu[i]=xupper[i-1];
                bl[i]=xlower[i-1];
                //Significa que a variavel e maior ou igual ao limite
                //inferior e menor ou igual ao limite superior
                ind[i]=3;
        }
}
//Copia o vetor com os valores das restricoes
bu[nvars+mrelas] = Vfrac*M;
ind[nvars+mrelas] = 2;
//Copia dos valores de A para o vetor Ax<=b
count=1;
for (i=1;i<=nvars;i++) {
        dattrv[count]=-i;
        count++;
        for  (k=1;k<=mrelas;k++) {
                dattrv[count]=k;
                count++;
                dattrv[count]=gradV[k][i];
                count++;
        }
}
//Fim de A
dattrv[count]=0;
//Minimização sem opções
prgopt[1]=1;
prgopt[2]=1;
//Dados auxiliares para dsplp
lamat=4*nvars+7;
```

```
lbm=8*mrelas;
lw=4*nvars+8*mrelas+lamat+lbm;
liw=nvars+11*mrelas+lamat+2*lbm;
work=dvector(1,lw);
iwork=ivector(1,liw);
//Chama o otimizador
dsplp_( dusrmt_,&mrelas, &nvars, &costs[1], &prgopt[1], &dattrv[1],
        &bl[1], &bu[1], &ind[1], &info, &primal[1], &duals[1],
        &ibasis[1], &work[1], &lw, &iwork[1], &liw);
//Resultado da otimização (info>0 -> OK)
printf("%d\n", info);
//Copia os valores de saida do otimizador
for (i=1;i<=nvars;i++) {
        xnew[i-1]=primal[i];
}
//Liberação dos arrays locais
free_dvector(bl,1,mrelas+nvars);
free_dvector(bu,1,mrelas+nvars);
free_dvector(prgopt,1,4);
free_dvector(dattrv,1,(2*mrelas*nvars)+1+nvars);
free_ivector(ind,1,mrelas+nvars);
free_dvector(primal,1,mrelas+nvars);
free_dvector(duals,1,mrelas+nvars);
free_ivector(ibasis,1,mrelas+nvars);
free_dvector(work,1,lw);
free_ivector(iwork,1,liw);
////////////////////////////////////////////////////////////////////////////
for (i=0; i<M; i++) {
        sign3[i] = sign2[i];
        sign2[i] = sign1[i];
        sign1[i] = xnew[i] - xold[i];
}
////////////////////////////////////////////////////////////////////////////
//Verificaçao da condiçoes de loop
if (it == 1) {
        objetivold = objetivo[it-1];
}
else {
        difer[it] = fabs((objetivo[it-1]-objetivold)/objetivold);
        objetivold = objetivo[it-1];
}
printf("iteracao %d\t obj: %f\t vol: %f\t difer: %f\n", it, objetivold,
volume[it-1], difer[it]);
if(flag2) {
        contador++;
```

```
                        printf("\n\n%d\n\n",contador);
                        if(contador==lprc) {
                                opcao = 0;
                                break;
                        }
                }
                else {
                        if (it==NUMIT) {
                                break;
                        }
                        else if (difer[it] < ITERTOL) {
                                printf("\nDeseja continuar a busca?(s = 1, n = 0): ");
                                scanf("%d", &opcao);
                                penal = 3;
                                if(opcao) {
                                        printf("\nDeseja ativar o filtro?(s = 1, n = 0): ");
                                        scanf("%d",&filter_en);

                                        if (filter_en){
                                                printf("\n    Digite o valor do raio para o
filtro: ");

                                                scanf("%f",&radius);
                                        }
                                }
                        }
                }
}
if (it==NUMIT) {
        break;
}

if(contador==lprc) {
        break;
}
printf("\nDeseja continuar para melhorar a resoluçao dos contornos?(s = 1, n = 0): ");
scanf("%d",&flag2);
if(flag2) {
        opcao = 1;
        filter_en = 0;
}
}//final do loop
//GERAÇAO DOS VETORES PARA PLOTAGEM DE GRAFICOS//////////////////
////////////////////////////////////////////////////////////////////////////
(...)
//////////////////////////CRIAÇAO DO ARQUIVO MATLAB////////////////////////////
```

```
////////////////////////////////////////////////////////////////////////////
(...)
////////////////////////////////////////////////////////////////////////////
(...)
//////////////////////LIBERAÇAO DE MEMÓRIA/////////////////////////////////
////////////////////////////////////////////////////////////////////////////
(...)
}

#include <stdio.h>

void vector_forces(char *direction_ksize, int *pos_node_ksize, int *flag, unsigned long
*store, unsigned long aux, unsigned long *count, unsigned long ksize, double *fcases, double
*fb, unsigned long count_cases_aux) {

	unsigned long i;
	for(i = store[count_cases_aux]; i < (ksize - count_cases_aux); i++) {
		if(fb[i] != 0) {
			fcases[i] = fb[i];
			count[0]++;
			if(store[count_cases_aux - 1] <= store[count_cases_aux] &&
count_cases_aux != 0) {
				store[count_cases_aux - 1] = i + 1;
			}
			store[count_cases_aux] = i;
			if (count_cases_aux != 0) {
				vector_forces(direction_ksize, pos_node_ksize, flag, store, aux,
count, ksize, fcases, fb, count_cases_aux - 1);
			}
			else {
				store[count_cases_aux]++;
			}
		}
		if(count[0] == aux) {
			printf("\n\nForca F%c de %f ", direction_ksize[i], fcases[i]);
			printf("aplicada ao no %d. ", pos_node_ksize[i]);
			break;
		}
		else if(flag[0] == 1) {
			flag[0] = 0;
			fcases[i] = 0;
			count[0]--;
			store[count_cases_aux - 1] = store[count_cases_aux];
		}
	}
```

```
        if(count[0] != aux) {
                flag[0] = 1;
        }
}

// header funcs.h

void nrerror(char error_text[]);
void dspr(double *a, unsigned long n, double thresh, unsigned long nmax, double sa[],
unsigned long ija[]);
void linbcg(unsigned long n, double b[], double x[], int itol, double tol, int itmax, int *iter,
double *err);
void simplx (float **a, int m, int n, int m1, int m2, int m3, int *icase, int izrov[], int iposv[]);
void Ke(double *xe, double *ye, double E, double v, double *Kel);
unsigned long fatorial(unsigned long num);
void vector_forces(char *direction_ksize, int *pos_node_ksize, int *flag, unsigned long
*store, unsigned long aux, unsigned long *count, unsigned long ksize, double *fcases, double
*fb, unsigned long count_cases_aux);
void filter(int M, int N, float *coord, int *conect, double *b, double *filt_b, float radius);
double dist(float *xei, float *yei, float *xej, float *yej);
double size_elem(float *xe, float *ye);


#define ITERTOL 1.0e-3
#define ITMAX 200
#define MAXITER 200
#define THRESH 1.0e-12
#define TOL 1.0e-3
#define ITOL 1
#define PI 3.1415192
#define NUMIT 100
#define minf 0.95
#define msup 1.05
#define mllower 0.04
#define mlupper 0.15
```

ANEXO C

Para a estrutura da Figura 32, foram obtidos os seguintes deslocamentos nodais, para cargas FY = 40, FY = -20 e FY = 40, aplicadas aos nós 6, 10 e 12, respectivamente:

	ANSYS 5.4		Software desenvolvido	
NODE	UX	UY	UX	UY
1	0.0000	0.0000	0.0000	0.0000
2	0.0000	0.0000	0.0000	0.0000
3	0.0000	0.0000	0.0000	0.0000
4	0.0000	0.0000	0.0000	0.0000
5	0.0000	0.0000	0.0000	0.0000
6	0.21173E-02	0.66067E-02	0.002117	0.006607
7	0.0000	0.0000	0.0000	0.0000
8	-0.47833E-03	0.26724E-02	-0.000478	0.002672
9	0.11642E-03	0.48237E-02	0.000116	0.004824
10	-0.71797E-02	-0.13228E-01	-0.007180	-0.013228
11	0.28627E-02	0.28139E-03	0.002863	0.00281
12	0.14355E-02	0.30810E-02	0.001436	0.003081
13	-0.28440E-02	0.45754E-02	-0.002844	0.004575
14	0.16573E-02	0.59531E-02	0.001657	0.005953
15	0.21195E-02	0.37504E-02	0.002120	0.003750
16	0.85436E-03	0.66905E-03	0.000854	0.000669
17	0.54250E-04	0.38021E-03	0.000054	0.000380
18	0.47532E-03	0.18909E-02	0.000475	0.001891
19	0.25872E-03	0.70756E-03	0.000259	0.000708
20	-0.28140E-03	0.52029E-03	-0.000281	0.000520
21	0.12685E-02	0.42731E-02	0.001269	0.004273
22	0.17282E-02	0.58939E-02	0.001728	0.005894
23	0.86899E-03	0.51530E-02	0.000869	0.005153

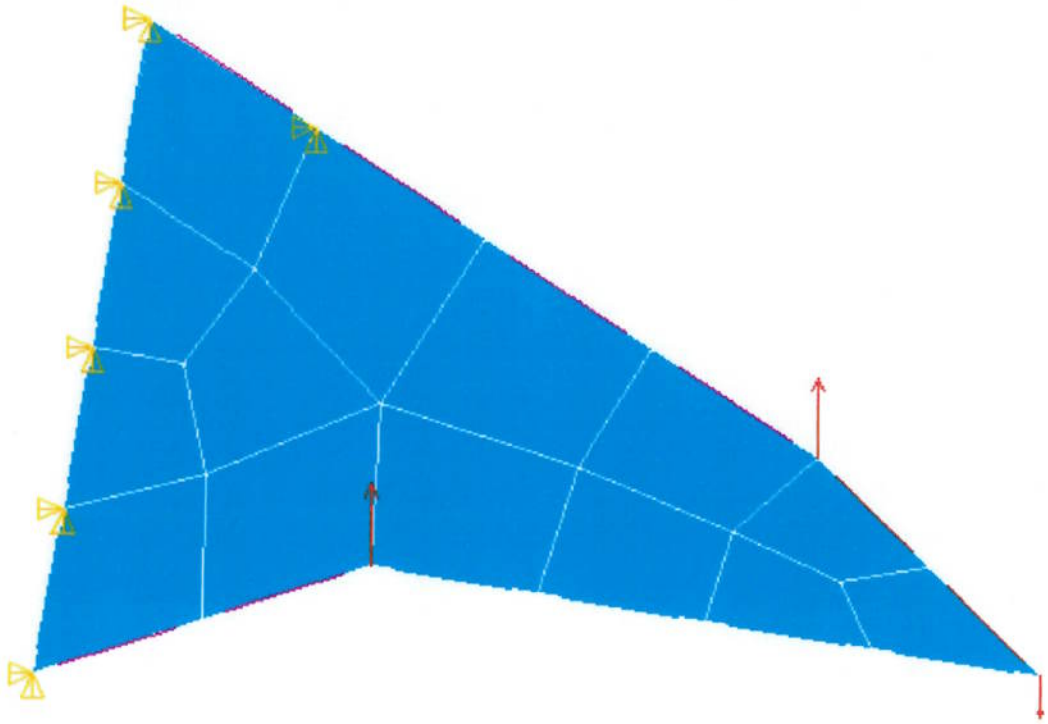


Figura 32 Teste da etapa de MEF do software de OT.