MÉTODOS NUMÉRICOS PARA PÓS-PROCESSAMENTO DE UMA OTIMIZAÇÃO TOPOLÓGICA (OT)

Evandro Pedro Alves de Mendonça[a], **Marcelino José de Lima Andrade**[a].

[a] *Núcleo de Tecnologia (NTI), Universidade Federal de Pernambuco (UFPE), Campus Acadêmico do Agreste (CAA), Rodovia BR-104, km 59, S/N, Nova Caruaru, CEP. 55.014-900, Caruaru-PE, Brasil, http://www.ufpe.br/caa*

Palavras Chave: suavização, contorno, estrutura, geometria, volume.

Resumo: Em diversas situações no cotidiano científico, sobretudo Engenharia e Design, utiliza-se o método de otimização topológica (TOM). Trata-se do procedimento numérico empregado para esculpir a distribuição mais adequada de material de uma estrutura dentro de um espaço determinado de design, submetido a um conjunto de cargas e condições de contorno. Um método muito importante e usado em diversas áreas, neste trabalho será explorada a etapa de pós-processamento de uma otimização topológica aplicada ao programa Matlab.

1 INTRODUÇÃO

A otimização topológica combina o Método de Elementos Finitos com fórmulas matemáticas de otimização, com intuito de proporcionar a melhor distribuição de material do espaço fixo de projeto. A abordagem material do método de otimização de layout foi proposto inicialmente por Bendsoe & Kikuchi (1988), considerando uma equação constitutiva homogeneizada que depende somente da densidade relativa do material.

A formulação para a determinação de topologias estruturais adequadas se utiliza de recursos tecnológicos capazes de proporcionar layouts eficientes. Essa inovação favorece a indústrias de diversos setores, tendo em vista que projetar peças e componentes mecânicos com alta rigidez e peso baixo se tornou uma necessidade comum.

Nesse trabalho é usado o método das splines cúbicas para interpolação dos pontos desejados. Além disso, para formar funções que deem contorno a uma figura desejada para que seja calculado posteriormente o seu volume. Mais detalhes sobre o método serão abordados na discussão dos resultados.

2 EXERCÍCIO PROPOSTO

Desenvolver os algoritmos necessários para a etapa de pós-processamento do seguinte problema, usando o MATLAB®.

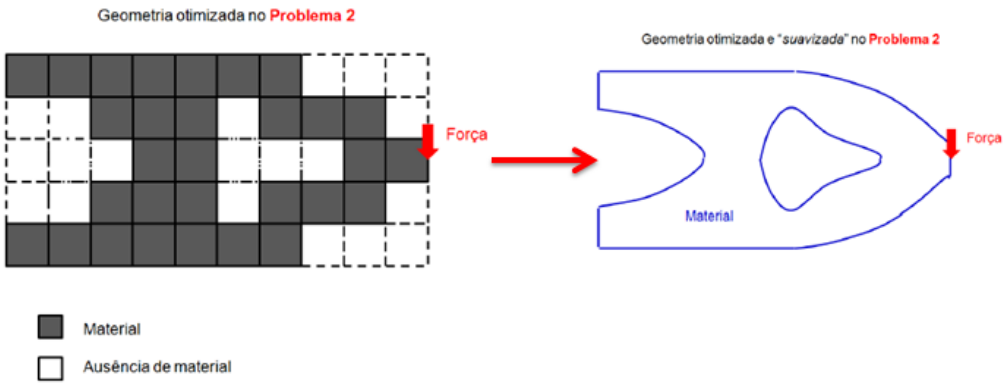

Figura 1: exercício proposto.

Primeiramente, plotamos a figura (peça) bruta no Matlab para comparar com a otimização, utilizado o código presente no anexo 1:

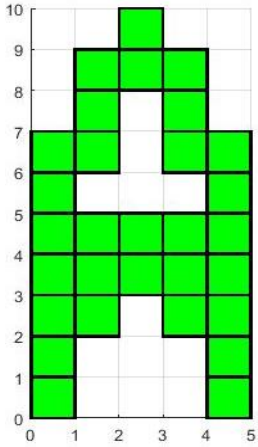

Figura 2: Imagem inicial plotada no MATLAB®.

Em seguida, foram definidos manualmente pontos que representassem a suavização da figura para que pudessem ser interpolados. A interpolação foi feita através do método das splines cúbicas. Uma vez interpolados os pontos, as curvas foram plotadas, formando a figura, e as integrais de cada curva foram calculadas e armazenadas. Após todo o processo de plotagem e cálculo de integrais, também foi chamada a função *PlotarGraficoImpar* para plotar a figura inicial no mesmo gráfico e, em seguida, foi feito o cálculo do volume.

Como a peça é de espessura unitária, para calcular o volume dela só é necessário calcular a área da figura. De posse das integrais calculadas ao longo do script, simplesmente calcula-se o volume pela seguinte expressão:

$$\boxed{Volume = Icima - Ibaixo - (Imeiosup - Imeioinf)}$$

Onde *Icima* é a variável que contém a integral da curva de cima, *Ibaixo*, da curva de baixo, *Imeiosup* e *Imeioinf*, das curvas do meio superior e inferior, respectivamente.

O resultado para o volume foi de aproximadamente 31,17 unidades de volume, que representa um erro de aproximadamente 3,9% do volume da figura inicial.

O script *posprocessamento*, que realiza todo esse processo, gera a seguinte resposta.

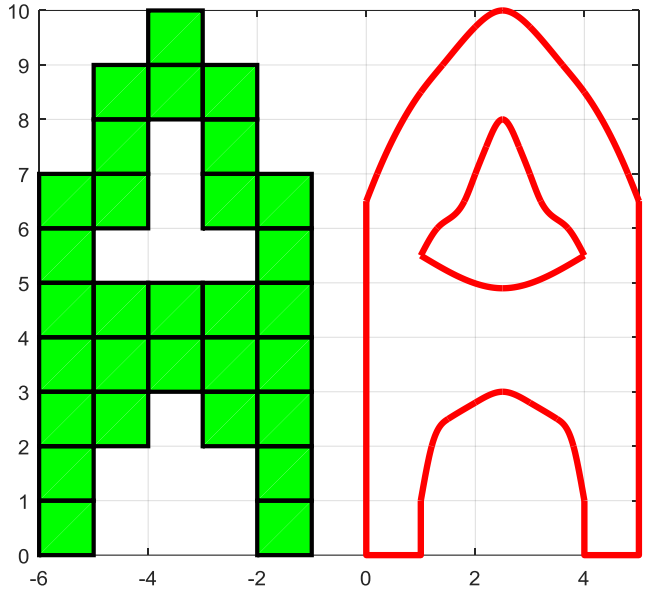

Figura 3: gráfico com figura inicial (à esquerda) e final (à direita).

```
>> tic; posprocessamento; toc;
O volume da figura suavizada é: 31.168059
Elapsed time is 7.062549 seconds.
```

Figura 4: resposta do script informando o volume total da peça final.

Todos os algoritmos usados nesse trabalho encontram-se nos Anexos (exceto a função *quad*, usada para calcular as integrais, pois é nativa do MATLAB®).

Devem-se levar em consideração alguns pontos sobre esse trabalho:

- Esse algoritmo só funciona para ESTA situação específica. A menor variação na peça inicial torna todo o algoritmo inválido. Esta é uma limitação desta solução, pois em uma situação real de uma indústria, por exemplo, não é viável, sendo preferível uma solução que seja automatizada e adaptável para qualquer peça.

- Não houve tempo suficiente para desenvolver algoritmos mais robustos e mais eficientes para o problema e, além de não ser o foco da disciplina, ainda falta bastante estudo teórico para desenvolver melhores soluções.
- Apesar o problema solicitar um volume da figura final igual ao da figura inicial, dadas as condições já faladas, um erro de 3,9% é aceitável.

3 CONCLUSÃO

Ao longo do trabalho, foram necessárias habilidades de raciocínio lógico para soluções de problemas, além do conhecimento prévio da disciplina de Cálculo Numérico, visto que era uma situação totalmente nova de acordo com a grade curricular da disciplina.

A Otimização Topológica pode auxiliar na redução de gastos corporativos em inovação tecnológica, tornando os processos mais dinâmicos e os produtos mais competitivos. Além disso, os ambientes de pré e pós-processamento oferecem análises que permitem adequar os produtos, proporcionando qualidade superior, podendo inclusive, ser aplicada nos produtos que a companhia já fábrica.

Há também outras formas, mais eficazes, de se resolver esse tipo de problema, mas que não pertencem ao escopo deste curso. São elas técnicas de Inteligência Artificial, que incluem reconhecimento de padrões, visão computacional, redes neurais artificiais, processamento digital de imagens e etc. Essas são técnicas computacionais usadas em diversas outras aplicações, mas que se encaixam perfeitamente à engenharia, como nessa situação.

Portanto, conclui-se que, apesar de não ter sido feita a melhor solução possível, dentro das limitações discutidas, pode-se afirmar que o objetivo foi alcançado com certa margem de tolerância.

REFERÊNCIAS

Chapra, S. C., e Canale, R. P. *Métodos Numéricos para Engenharia*. 5ª edição. Porto Alegre: AMGH, 2011.

Gilat, A., e Subramaniam, V. *Métodos Numéricos para Engenheiros e Cientistas: uma introdução com aplicações usando o MATLAB*. Porto Alegre: Bookman, 2008.

ANEXO 1

```
%declarando quadrados
[x1,y1] = formarquadrado(0,0);
[x2,y2] = formarquadrado(1,0);
[x3,y3] = formarquadrado(2,0);
[x4,y4] = formarquadrado(3,0);
[x5,y5] = formarquadrado(4,0);
[x6,y6] = formarquadrado(5,0);
[x7,y7] = formarquadrado(6,0);
[x8,y8] = formarquadrado(2,1);
[x9,y9] = formarquadrado(3,1);

[x10,y10] = formarquadrado(4,1);
[x11,y11] = formarquadrado(6,1);
[x12,y12] = formarquadrado(7,1);
[x13,y13] = formarquadrado(8,1);
[x14,y14] = formarquadrado(3,2);
[x15,y15] = formarquadrado(4,2);
[x16,y16] = formarquadrado(8,2);
[x17,y17] = formarquadrado(9,2);
[x18,y18] = formarquadrado(2,3);
[x19,y19] = formarquadrado(3,3);
[x20,y20] = formarquadrado(4,3);
[x21,y21] = formarquadrado(6,3);
[x22,y22] = formarquadrado(7,3);
[x23,y23] = formarquadrado(8,3);
[x24,y24] = formarquadrado(0,4);
[x25,y25] = formarquadrado(1,4);
[x26,y26] = formarquadrado(2,4);
[x27,y27] = formarquadrado(3,4);
[x28,y28] = formarquadrado(4,4);
[x29,y29] = formarquadrado(5,4);
[x30,y30] = formarquadrado(6,4);

%juntando tudo em duas matrizes
x = [x1 x2 x3 x4 x5 x6 x7 x8 x9 x10];
x = [x x11 x12 x13 x14 x15 x16 x17 x18 x19 x20];
x = [x x21 x22 x23 x24 x25 x26 x27 x28 x29 x30];
y = [y1 y2 y3 y4 y5 y6 y7 y8 y9 y10];
y = [y y11 y12 y13 y14 y15 y16 y17 y18 y19 y20];
y = [y y21 y22 y23 y24 y25 y26 y27 y28 y29 y30];

%apagando vetores
clear x1 x2 x3 x4 x5 x6 x7 x8 x9 x10
clear x11 x12 x13 x14 x15 x16 x17 x18 x19 x20
clear x21 x22 x23 x24 x25 x26 x27 x28 x29 x30
clear y1 y2 y3 y4 y5 y6 y7 y8 y9 y10
clear y11 y12 y13 y14 y15 y16 y17 y18 y19 y20
clear y21 y22 y23 y24 y25 y26 y27 y28 y29 y30

%plotando
figure(1);
patch(y,x,'g','LineWidth',2);
axis equal tight
grid on
```

Figura 5: Código da figura inicial.

ANEXO 2

```
function [ spline ] = montarspline3(x,y,i,a,h)
%Esta função cria uma spline cúbica.
%Parâmetros: [ spline ] = montarspline3(i,a,h)
%x: vetor de pontos x
%y: vetor de pontos y (imagens de x)
%i: ponto inicial do intervalo
%a: vetor de coeficientes
%h: vetor de tamanhos de intervalos

    syms X;
    spline = (a(i)/(6*h(i)))*((x(i+1)-X)^3);%1ª parte
    spline = spline + (a(i+1)/(6*h(i)))*((X-x(i))^3);%2ª parte
    spline = spline + ((y(i)/h(i))-((a(i)*h(i))/6))*(x(i+1)-X);%3ª parte
    spline = spline + ((y(i+1)/h(i))-((a(i+1)*h(i))/6))*(X-x(i));%4ª parte
end
```

Figura 6: Código para a montagem das Splines.

ANEXO 3

```
function [] = plotarspline3(spline,x)
%Plotar spline cúbica.
%Parâmetros: plotarspline3(spline,x)
%spline: spline cúbica
%x: vetor x

    for i = 1:length(spline)
        sp = inline(spline(i));%transformando em função inline
        x1 = x(i):0.005:x(i+1);%intervalo em x
        y1 = sp(x1);%cálculo das imagens no intervalo
        plot(x1,y1,'r','LineWidth',3);%plotando spline
        grid on; hold on;
    end
end
```

Figura 7: Código para a plotagem das Splines.

ANEXO 4

```
%%%PLOTANDO FIGURA%%%
a = [0 0 1 1]; b = [6.5 0 0 1]; %segmentos de reta da esquerda
plot(a,b,'r','LineWidth',3);
grid on; hold on;
a = [4 4 5 5]; b = [1 0 0 6.5]; %segmentos de reta da direita
plot(a,b,'r','LineWidth',3);

%Pontos da curva de baixo
x = [1,1.2,1.5,2,2.5,3,3.5,3.8,4];
y = [1,2,2.5,2.8,3,2.8,2.5,2,1];

CurvaDeBaixo = SplineCubica(x,y); %interponaldo com spline cúbica
plotarspline3(CurvaDeBaixo,x); %plotando

%calculando integral da curva de baixo
Ibaixo = 0;
for i = 1:length(CurvaDeBaixo)
    Ibaixo = Ibaixo + quad(inline(CurvaDeBaixo(i)),x(i),x(i+1));
end

 %Pontos da curva de cima
 x = [0 0.2 1 1.4 2 2.5 3 3.6 4 4.8 5];
 y = [6.5 7 8.5 9 9.7 10 9.7 9 8.5 7 6.5];

 CurvaDeCima = SplineCubica(x,y); %interponaldo com spline cúbica
 plotarspline3(CurvaDeCima,x); %plotando

 %calculando integral da curva de cima
 Icima = 0;
for i = 1:length(CurvaDeCima)
    Icima = Icima + quad(inline(CurvaDeCima(i)),x(i),x(i+1));
end

 %Pontos da curva do meio (parte superior)
 x = [1 1.3 1.8 2 2.2 2.5 2.8 3 3.2 3.7 4];
 y = [5.5 6 6.5 7 7.5 8 7.5 7 6.5 6 5.5];

 MeioSuperior = SplineCubica(x,y); %interponaldo com spline cúbica
 plotarspline3(MeioSuperior,x); %plotando

 %calculando integral da curva do meio superior
 Imeiosup = 0;
for i = 1:length(MeioSuperior)
    Imeiosup = Imeiosup + quad(inline(MeioSuperior(i)),x(i),x(i+1));
end
```

```
%Pontos da curva do meio (parte inferior)
x = [1 2.5 4];
y = [5.5 4.9 5.5];

MeioInferior = SplineCubica(x,y); %interponaldo com spline cúbica
plotarspline3(MeioInferior,x); %plotando

%calculando integral da curva do meio inferior
Imeioinf = 0;
for i = 1:length(MeioInferior)
    Imeioinf = Imeioinf + quad(inline(MeioInferior(i)),x(i),x(i+1));
end

axis equal tight %torna os eixos de igual tamanho e envolvendo somente a
                 %área da figura

PlotarGraficoImpar;

%%%CALCULANDO VOLUME%%%
Volume = Icima - Ibaixo - (Imeiosup - Imeioinf);
fprintf('O volume da figura suavizada é: %f\n',Volume);
clear;
```

Figura 8: Código do processamento para a resolução do problema.

ANEXO 5

```
function [spline] = SplineCubica(x,y)
%SplineCubica calcula a interpolação usando splines cúbicas.
%Parâmetros: [ Yint ] = SplineCubica(x,y,Xint)
%x: Vetor com as coordenadas x dos pontos dados.
%y: Vetor com as coordenadas y dos pontos dados.
%Xint: Coordenada x do ponto a ser interpolado.
%Yint: O valor interpolado de Xint.

    %%%%%VALIDAÇÃO%%%%%
    if isnumeric(x)==false||isnumeric(y)==false%|| ...
            %isnumeric(Xint)==false %um dos parâmetros não é numérico
        disp('Erro. Todos os parâmetros devem ser numéricos.');
        %Yint = 'erro';
        return;
    end
    if(((size(x,1)>1)&&(size(x,2)>1))||((size(y,1)>1)&&(size(y,2)>1)))
        disp('Erro! X ou Y não é um vetor.');
        %Yint = 'erro';
        return;
    elseif(length(x)~=length(y))%x e y devem ter mesmo número de elementos
        disp('Erro! Dimensões de X e Y não são equivalentes.');
        %Yint = 'erro';
        return;
    end

    %%%%%PROCESSAMENTO%%%%%
    n = length(x); %número de pontos
    ns = n-1; %número de splines
    h = zeros(1,ns); %vetor de tamanhos de intervalos
    for i = 1:ns
        h(i) = x(i+1)-x(i);%calculando tamanhos de intervalos
    end
    %MONTAGEM DO SISTEMA LINEAR PARA ENCONTRAR COEFICIENTES
    %Como a spline cúbica é natural, a(1)=a(n)=0
    b = zeros(n-2,1); %declarando vetor coluna inicialmente nulo
    for i = 1:n-2 %montando vetor resposta do sistema linear
        b(i) = 6*(((y(i+2)-y(i+1))/h(i+1))-((y(i+1)-y(i))/h(i)));
    end
    A = zeros(n-2); %declaração da matriz dos coeficientes (nula)
    j = 1; %variável auxiliar para colunas
    for i = 1:n-2 %calculando elementos da matriz A
        if i == 1 %primeira linha
            A(i,j) = 2*(h(i)+h(i+1)); %coeficiente de a(2)
            A(i,j+1) = h(i+1); %coeficiente de a(3)
        elseif i == n-2 %última linha
            A(i,j) = h(i); %coeficiente de a(n-3)
            A(i,j+1) = 2*(h(i)+h(i+1)); %coeficiente de a(n-2)
        else %linhas do meio
            A(i,j) = h(i); %coeficiente de a(i)
            A(i,j+1) = 2*(h(i)+h(i+1)); %coeficiente de a(i+1)
            A(i,j+2) = h(i+1); %coeficiente de a(i+2)
            j = j + 1; %incremento de j
        end
    end
    %SOLUÇÃO DO SISTEMA LINEAR
    format long;
    a = A\b;%vetor a contém os coeficientes de a(2) até a(n-1)
    a = [0;a;0];%adicionando os demais coeficientes

    %%%SAÍDA%%%
    %montando spline
    for i = 1:n-1 %i contém o ponto inicial do intervalo de interpolação
        spline(i) = montarspline3(x,y,i,a,h);%montando spline
    end
end
```

Figura 9: função que calcula as splines cúbicas.

ANEXO 6

```
function [ X,Y ] = formarquadrado(x,y)
%Esta função recebe um par ordenado (x,y) que representa o ponto inferior
%esquerdo de um quadrado unitário e retorna dois vetores contendo os
%vértices do quadrado.

    %%%VALIDAÇÃO%%%
    if isnumeric(x)==false||isnumeric(y)==false
        disp('Erro! Todos os parâmetros devem ser numéricos.');
        X = 'erro';
        Y = 'erro';
        return;
    end
    if size(x,1)~=1||size(x,2)~=1||size(y,1)~=1||size(y,2)~=1
        disp('Erro! Parâmetros não podem ser vetores ou matrizes.');
        X = 'erro';
        Y = 'erro';
        return;
    end

    %%%PROCESSAMENTO E SAÍDA%%%
    %declarando vetores de saída
    X = zeros(4,1);
    Y = zeros(4,1);
    %preenchendo vetores de saída
    X(1) = x; X(2) = x;
    X(3) = x+1; X(4) = x+1;
    Y(1) = y; Y(2) = y+1;
    Y(3) = y+1; Y(4) = y;
end
```

Figura 10: função que recebe as coordenadas do ponto inferior esquerdo de um quadrado unitário e retorna todos os vértices desse quadrado.