O método do adjunto médio em otimização de forma e aplicações

Vanessa Soares Borges da Silva

Dissertação apresentada
ao
Instituto de Matemática e Estatística
da
Universidade de São Paulo
para
obtenção do titulo
de
Mestre em Ciências

Programa: Matemática Aplicada
Orientador: Prof. Dr. Antoine Laurain

Durante o desenvolvimento deste trabalho a autora recebeu auxílio financeiro da CAPES

São Paulo, Maio de 2019

O método do adjunto médio em otimização de forma e aplicações

Esta é a versão original da dissertação elaborada pela candidata Vanessa Soares Borges da Silva, tal como submetida à Comissão Julgadora.

Resumo

SILVA, V. S. B. **O método do adjunto médio em otimização de forma e aplicações**. 2019. Dissertação (Mestrado) - Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo, 2019.

No presente trabalho abordamos conceitos de otimização de forma e os aplicamos utilizando um código educacional, baseado no método level set, escrito com o pacote FEniCS. Para tal, calculamos a derivada de forma distribuída, que se apresenta como uma integral no domínio. Essa derivada de forma distribuída é vantajosa para definir um algoritmo numérico de otimização de forma baseado no método level set, devido sua facilidade na implementação. Também apresentamos o método do adjunto médio como uma alternativa eficiente para o cálculo da derivada de forma distribuída.

Usando estes conceitos, estudamos um código educacional para minimização da compliance no contexto da elasticidade linear.

Através de vários exemplos clássicos da otimização de estruturas, demonstramos a eficiência da abordagem estudada.

Palavras-chave: Otimização de forma, método level set, elasticidade linear, derivada de forma.

Abstract

SILVA, V. S. B.. **The averaged adjoint method in shape optmization and aplications**. 2019. Dissertação (Mestrado) - Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo, 2019.

In the present work we present several concepts of shape optimization and apply them using an educational code based on the level set method,written using the FEniCS package. For this, we compute the so-called distributed shape derivative, which takes the form of a domain integral. This distributed expression of the shape derivative is advantageous for defining a numerical shape optimization algorithm based on the level set method, due to its ease of implementation. We also present the averaged adjoint method as an efficient alternative for the calculation of the distributed shape derivative.

Using these concepts, we study an educational code for compliance minimization in the context of linearized elasticity.

Through various classical examples of structural optimization, we demonstrate the efficiency of this approach.

Keywords: Shape optimization, level set method, linear elasticity, shape derivative.

Sumário

Capítulo 1

Introdução

Sigmund, com o código Matlab *"99 lines"* [19], deu início a cultura do compartilhamento e publicações de códigos educacionais para otimização estrutural. Uma atualização de tal código, ainda em Matlab, reduziu-o para 88 linhas e aumentou a velocidade da implementação.

Ambos os códigos citados acima se baseiam na abordagem de microestrutura isotrópica sólida com penalidade (SIMP). Nessa linha, é permitido que a densidade material assuma valores intermediários e as variáveis de otimização são densidades materiais da malha de elementos. Os valores intermediários são penalizados para assumir valores 0-1 e designs factíveis são obtidos usando técnicas de regularização.

Vários outros códigos tem sido publicados usando diferentes abordagens e plataformas além do Matlab. Uma outra categoria de abordagens para otimização topológica que surgiu depois do SIMP são os métodos level set. Eles consistem em representar a fronteira do domínio em movimento Ω como o conjunto de nível zero de uma função ϕ. Métodos level set são fundamentalmente diferentes dos métodos de densidade como o SIMP, já que a configuração da interface é mantida durante o processo de otimização ao invés de relaxar o problema usando densidade de material. Métodos level set foram introduzidos por Osher e Sethian no contexto do fluxo de curvatura média para facilitar a modelagem das mudanças topológicas durante a evolução da curva. Desde então, tem sido aplicado em muitos problemas de otimização de forma e perturbação de fronteira.

O código apresentado neste trabalho se encaixa na categoria dos métodos level set. Na abordagem usual, o conceito de derivada de forma é usado para calcular a sensibilidade da função objetivo. Usualmente os algoritmos são baseados no fato da derivada de forma ser uma distribuição na fronteira do domínio. Isso significa que a derivada é expressada como uma integral na fronteira e, então, estendida para o domínio inteiro ou por uma faixa estreita para o uso no método level set.

A derivada também pode ser escrita como uma integral no domínio, a qual é chamada de derivada volumétrica ou expressão de domínio da derivada de forma.

Do ponto de vista numérico, frequentemente a expressão de domínio é mais fácil de implementar que a expressão de fronteira por ser uma integral de volume. Outras vantagens da expressão de fronteira é mostrada por [3, 7]. Em [3], mostra-se que a discretização da expressão de fronteira geralmente não leva para a mesma expressão que a derivada de forma calculada após o problema ser discretizado. Em [7], os autores concluem que as expressões baseadas no volume para gradiente de forma frequentemente oferecem mais acurácia que o uso de fórmulas baseadas em traços nas fronteiras. O principal aspecto deste trabalho é de apresentar uma implementação compacta, ainda assim, eficiente do método level set para otimização estrutural de duas fases que se dá graças a derivada de forma distribuída.

A escolha do FEniCS para a implementação é motivada por sua notação próxima a matemática, a qual facilita a implementação de formulações variacionias complicadas. O projeto FEniCS (https://fenicsproject.org/) é um projeto colaborativo focado em soluções automatizadas de equações diferenciais por métodos de elementos finitos. FEniCS pode ser programado tanto em Python quanto em C++. Neste trabalho, focamos na linguagem Python por sua abordagem simples para iniciantes.

O presente trabalho se encontra estruturado da seguinte forma. No capítulo 2 introduzimos

definição da derivada para um funcional de forma e alguns resultados que facilita a compreenssão dos cálculos no decorrer no trabalho. No capítulo 3 apresentamos o método do ajunto médio e o teorema que nos permite calcular efetivamente a derivada de forma. Já no capítulo 4 exibimos a conexão entre a representação tensorial e a expressão usual de fronteira da derivada de forma. Por fim, no capítulo 5 apresentamos dois exemplos do cálculo da derivadade de forma por meio do método adjunto médio. No capítulo 6 apresentamos a otimização de forma para o caso da elasticidade linear usando material ersatz baseado em todos os conceitos apresentados nos capítulos anteriores. No capítulo 7 introduzimos o método level set com a discretização da equação de Hamilton-Jacobi usada na implementação numérica descrita no capítulo 8 justamente com algumas explicações dos códigos usados que são encontrados no apêndice. Finalmente no último capítulo aprensentamos alguns resultados que mostram a eficiência da abordagem estudada no presente trabalho.

Capítulo 2

Derivada de forma

Em se tratando de problemas de otimização, após a garantia da existência de solução começamos a busca por candidatos a solução ótima. Em dimensão finita, por exemplo, para $J : \mathbb{R}^n \to \mathbb{R}$ uma função de custo suficientemente suave, as condições necessárias de otimalidade são dadas pela equação $\nabla J(x) = 0$.

A mesma abordagem se dá na otimização de forma, mas agora estamos lidando com funcionais cujo domínio é um conjunto de formas, ou seja, subconjuntos de $\mathbb{R}^n$. Nesse contexto uma certa dificuldade é encontrada ao introduzir o conceito de derivada já que os espaços de formas não possuem de imediato uma estrutura vetorial ou de variedade suave.

Na matemática, o conceito de diferenciabilidade se apresenta naturalmente para funções definidas em espaços vetoriais topológicos. A noção de diferenciabilidade se adapta bem a estrutura de variedades Riemanianas, mesmo se a definição de diferenciabilidade for mais complicada. Os conjuntos de forma não tem naturalmente uma estrutura vetorial nem uma estrutura de variedade Riemaniana. Assim, para se obter noções de diferenciabilidade precisamos trabalhar com subconjuntos ou caminhos particulares desses conjuntos de forma.

No presente trabalho, para contornar tal situação, foi obtido uma estrutura vetorial atrelada a esses conjuntos de formas considerando perturbações de um domínio inicial, onde cada perturbação está associada a um parâmetro $t \in \mathbb{R}$ e, introduzido uma família de domínios parametrizados, pode-se então definir uma derivada de forma.

Assim, neste capítulo veremos o método da velocidade que nos fornecerá uma perturbação de domínio que possibilitará definir uma derivada para um funcional de forma. Também apresentaremos alguns resultados que facilitarão os cálculos no decorrer deste trabalho.

2.1 Preliminares para algumas definições

Definição 2.1. *Seja $p \in \mathbb{R}$ com $1 < p < \infty$ e Ω um subconjunto de $\mathbb{R}^n$. Denotamos por $L^p(\Omega)$ o espaço das funções $f : \Omega \to \mathbb{R}$ mensuráveis tais que $\int_\Omega |f(x)|^p \, dx < \infty$.*

Definição 2.2. *Seja Ω um subconjunto aberto de $\mathbb{R}^n$. O conjunto $\mathcal{C}_0^\infty(\Omega)$ é o conjunto de todas as funções $\Phi : \Omega \to \mathbb{R}$ infinitamente diferenciável cujo suporte é um subconjunto compacto de Ω.*

Definição 2.3. *Seja $p \in \mathbb{R}$ com $1 < p < \infty$ e Ω um subconjunto de $\mathbb{R}^n$. Dizemos que uma função $f : \Omega \to \mathbb{R}$ pertence ao espaço denotado por $L^p_{loc}(\Omega)$, quando esta for mensurável e dado qualquer subconjunto compacto K de Ω temos que $\int_K |f(x)|^p \, dx < \infty$. Ou seja,*

$$f|_K \in L^p(\Omega), \ \ \forall K \subset\subset \Omega$$

Definição 2.4. *Seja Ω um conjunto aberto e $\gamma \in (0, 1]$ um número real. Uma função $f : \Omega \to \mathbb{R}$ é dita Hölder contínua com expoente γ se existir uma constante real C tal que:*

$$|f(x) - f(y)| \leq C \, |x - y|^\gamma, \quad \forall x, y \in \Omega.$$

Definição 2.5. *Sejam $\Omega \in \mathbb{R}^n$ um aberto, $1 < p < \infty$ e k um inteiro não negativo. Definimos o espaço de Sobolev $W^{k,p}(\Omega)$ como sendo*

$$W^{k,p}(\Omega) := \{f \in L^p(\Omega) \mid D^\alpha f \in L^p(\Omega) \text{ para todo multi-índice } \alpha \text{ tal que } |\alpha| \leq k\}.$$

Onde $D^\alpha f$ denota a derivada no sentido fraco.

Definição 2.6. *i) Se $f : \Omega \to \mathbb{R}$ é limitada e contínua, temos*

$$\|f\|_{\mathcal{C}(\Omega)} := \sup_{x \in \Omega} |f(x)|.$$

ii) A γ-ésima seminorma de Hölder de $f : \Omega \to \mathbb{R}$ é definida como

$$[f]_{\mathcal{C}^{0,\gamma}(\Omega)} := \sup_{\substack{x,y \in \Omega \\ x \neq y}} \left\{ \frac{|f(x) - f(y)|}{|x - y|^\gamma} \right\}.$$

iii) A γ-ésima norma de Hölder de $f : \Omega \to \mathbb{R}$ é definida como

$$\|f\|_{\mathcal{C}^{0,\gamma}(\Omega)} := \|f\|_{\mathcal{C}(\Omega)} + [f]_{\mathcal{C}^{0,\gamma}(\Omega)}.$$

Definição 2.7. *O espaço de Hölder $\mathcal{C}^{k,\alpha}(\Omega)$ consiste de todas as funções $f : \Omega \to \mathbb{R}^n$ que pertencem ao espaço $\mathcal{C}^k(\Omega)$ das funções k vezes diferenciáveis cuja norma*

$$\|f\|_{\mathcal{C}^{k,\gamma}(\Omega)} := \sum_{|\alpha| \leq k} \|D^\alpha f\|_{\mathcal{C}^0(\Omega)} + \sum_{|\alpha| = k} [D^\alpha f]_{\mathcal{C}^{0,\gamma}(\Omega)}$$

é finito.

Aqui $\alpha = (\alpha_1, \ldots, \alpha_n)$ é um multi índice e sua norma é dada por $|\alpha| = \alpha_1 + \ldots + \alpha_n$. A derivada de ordem α é dada por

$$D^\alpha f(x) := \frac{\partial^{|\alpha|} f(x)}{\partial_{x_1}^{\alpha_1} \ldots \partial_{x_n}^{\alpha_n}}.$$

Assim, o espaço $\mathcal{C}^{k,\gamma}(\Omega)$ consiste nas funções f k vezes diferenciáveis e cujas k-ésimsa derivadas parciais são Hölder contínua com expoente γ.

2.2 Estrutura das derivadas de forma

Seja $\mathcal{D} \subset \mathbb{R}^n$ aberto e limitado, de classe $\mathcal{C}^1$ e $\mathcal{P}(\mathcal{D})$ o conjunto das partes de $\mathcal{D}$. Defina para $k \geq 0$ e $0 \leq \alpha \leq 1$

$$\mathcal{C}_c^{k,\alpha}(\mathcal{D}, \mathbb{R}^n) := \{\theta \in \mathcal{C}^{k,\alpha}(\mathcal{D}, \mathbb{R}^n) \mid \theta \text{ possui suporte compacto em } \mathcal{D}\}. \tag{2.1}$$

Dado também um domínio $\Omega \subset \mathcal{D}$ com fronteira pelo menos $\mathcal{C}^1$, introduzimos o espaço dos campos vetoriais

$$\mathcal{C}_{\partial\Omega}^{k,\alpha}(\mathcal{D}, \mathbb{R}^n) := \{\theta \in \mathcal{C}_c^{k,\alpha} \mid \theta \cdot n = 0 \text{ em } \partial\Omega\}, \tag{2.2}$$

onde n é o vetor normal unitário apontado para fora de Ω. Vamos considerar perturbações de um domínio Ω usando campos θ não autônomos. Na maioria das vezes, apenas precisamos de campos autônomos para definir derivadas de forma, mas, em alguns casos particulares, campos não autônomos podem ser úteis. Vamos exigir a seguinte hipótese para o campo θ.

Hipótese 1. *Seja $\tau > 0$ e $\theta : [0,\tau] \times \overline{\mathcal{D}} \to \mathbb{R}^n$ tal que*

$$\theta(\cdot, x) \in \mathcal{C}([0,\tau];\mathbb{R}^n), \quad \forall\, x \in \overline{\mathcal{D}}, \tag{2.3}$$

$$\|\theta(\cdot, y) - \theta(\cdot, x)\|_{\mathcal{C}([0,\tau];\mathbb{R}^n)} \leq c_\theta |y - x|, \quad \forall\, x, y \in \overline{\mathcal{D}}, \tag{2.4}$$

$$\theta(t, x) \cdot n(x) = 0, \quad \forall\, x \in \partial\mathcal{D},\ \forall\, t \in [0,\tau]. \tag{2.5}$$

Para $\tau > 0$, introduzimos uma família de transformações $T_t^\theta(x) := T(t,x)$ como a solução da equação diferencial ordinária

$$\begin{cases} \frac{d}{dt}x(t,X) &= \theta(t, x(t,X)), \qquad \text{para } t \in [0,\tau], \\ x(0,X) &= X \in \mathbb{R}^n. \end{cases} \tag{2.6}$$

Observação 2.1. *Note que, por construção, para $t = 0$ temos $T_0^\theta = I$, onde I denota o operador identidade.*

Veremos no Teorema 2.1 que para τ suficientemente pequeno, o sistema (2.6) possui uma única solução. A transformação T_t^θ nos permite definir uma família de domínios $\Omega_t = T_t^\theta(\Omega)$ que nos será útil para definir a derivada de um funcional de forma. Usaremos a notação $T_t(\Omega)$ em vez de $T_t^\theta(\Omega)$ para simplificar.

O seguinte teorema fornece informações sobre a regularidade de T_t.

Teorema 2.1. *Seja $\tau > 0$ e θ satisfazendo a Hipótese 1, então a transformação T_t definida por (2.6) e sua inversa T_t^{-1} satisfaz*

1) T é $\mathcal{C}^1$ com respeito a t:

$$T(\cdot, X) \in \mathcal{C}([0,\tau];\mathbb{R}^n), \quad \forall X \in \overline{\mathcal{D}}. \tag{2.7}$$

2) T é Lipschitz com respeito a x:

$$\exists L > 0, \quad \|T(\cdot, Y) - T(\cdot, X)\|_{\mathcal{C}([0,\tau];\mathbb{R}^n)} \leq L|Y - X|, \quad \forall X, Y \in \overline{\mathcal{D}}. \tag{2.8}$$

3) T é bijetora com respeito a x:

$$\forall t \in [0,\tau], \quad X \mapsto T_t(X) : \overline{\mathcal{D}} \to \overline{\mathcal{D}} \text{ é bijetora.} \tag{2.9}$$

4) T^{-1} é contínua com respeito a t:

$$T^{-1}(\cdot, x) \in \mathcal{C}([0,\tau];\mathbb{R}^n), \quad \forall x \in \overline{\mathcal{D}}. \tag{2.10}$$

5) T^{-1} é Lipschitz com respeito a x:

$$\exists L > 0, \quad \|T^{-1}(\cdot, y) - T^{-1}(\cdot, x)\|_{\mathcal{C}([0,\tau];\mathbb{R}^n)} \leq L|y - x|, \quad \forall x, y \in \overline{\mathcal{D}}. \tag{2.11}$$

Demonstração. Defini-se o seguinte sistema aumentado em $[0,\tau]$

$$\begin{cases} \dfrac{dx(t)}{dt} = \theta(t, x(t)), \quad x(0) = X \in \overline{\mathcal{D}}, \\ \dfrac{dx_0(t)}{dt} = 1, \quad x_0(0) = 0. \end{cases} \tag{2.12}$$

Introduzindo $\hat{x}(t) := (x_0(t), x(t)) \in \mathbb{R}^{n+1}$, podemos escrever (2.12) como

$$\begin{cases} \dfrac{d\hat{x}(t)}{dt} = \hat{\theta}(\hat{x}(t)), \\ \hat{x}(0) = (0, X) \in \hat{\mathcal{D}} := \mathbb{R}^+ \times \overline{\mathcal{D}}, \end{cases} \tag{2.13}$$

com $\hat{\theta}(\hat{x}) := (1, \tilde{\theta}(\hat{x}))$, e para $\hat{x} = (x_0, x)$:

$$\begin{aligned} \tilde{\theta}(\hat{x}) &:= \theta(x_0, x) \quad \text{para } 0 \le x_0 \le \tau \text{ e } x \in \overline{\mathcal{D}}, \\ \tilde{\theta}(\hat{x}) &:= \theta(\tau, x) \quad \text{para } \tau < x_0 \text{ e } x \in \overline{\mathcal{D}}. \end{aligned} \tag{2.14}$$

É fácil conferir que os sistemas (2.12) e (2.13) são equivalentes em $[0, \tau]$ e que $\hat{x}(t) = (t, x(t))$. O campo de velocidade $\hat{\theta}$ é contínuo em cada $\hat{x} \in \hat{\mathcal{D}}$ já que

$$\hat{\theta}(\hat{y}) - \hat{\theta}(\hat{x}) = (0, \tilde{\theta}(y_0, y) - \tilde{\theta}(x_0, x)),$$

e para $0 \le x_0, y_0 \le \tau$ temos, usando Hipótese (1),

$$\begin{aligned} |\tilde{\theta}(y_0, y) - \tilde{\theta}(x_0, x)| &= |\theta(y_0, y) - \theta(x_0, x)| \\ &\le |\theta(y_0, y) - \theta(y_0, x)| + |\theta(y_0, x) - \theta(x_0, x)| \\ &\le c_\theta |y - x| + |\theta(y_0, x) - \theta(x_0, x)|. \end{aligned}$$

Agora introduzimos o cone contingente de Boulingand $\mathcal{T}_{\mathfrak{D}}(x)$ para $\mathfrak{D} \subset \mathbb{R}^{n+1}$ em $x \in \mathfrak{D}$:

$$\mathcal{T}_{\mathfrak{D}}(x) := \bigcap_{\epsilon > 0} \bigcap_{\alpha > 0} \bigcup_{0 < h < \alpha} \left[\frac{1}{h}(\mathfrak{D} - x) + \epsilon B \right],$$

onde B é o disco unitário em $\mathbb{R}^{n+1}$. Usando (2.5), pode-se mostrar que para $\mathfrak{D} = \hat{\mathcal{D}}$ temos, para $\hat{x} = (x_0, x) \in \hat{\mathcal{D}}$ e $0 \le x_0 \le \tau$,

$$\hat{\theta}(\hat{x}) = (1, \theta(\hat{x})) \in \mathcal{T}_{\hat{\mathcal{D}}}(\hat{x}) = \mathcal{T}_{\mathbb{R}^+}(x_0) \times \mathcal{T}_{\overline{\mathcal{D}}}(x).$$

Além disso, $\hat{\theta}(\hat{\mathcal{D}})$ é limitado e $\mathbb{R}^{n+1}$ possui dimensão finita. Usando a versão do teorema de Nagumo [12], existe uma solução viável $\hat{x}$ para (2.13), que significa, em particular, que

$$\hat{x}(t) = (t, x(t)) \in \hat{\mathcal{D}} = \mathbb{R}^+ \times \overline{\mathcal{D}} \qquad \text{para qualquer} \quad t \in [0, \tau].$$

Consequentemente, existe uma solução T de (2.6) tal que $T(t, X) = x(t) \in \overline{\mathcal{D}}$ em $[0, \tau]$. A unicidade segue da condição de Lipschitz (2.4).

Agora provaremos (2.8). Usando (2.6) e (2.4) temos

$$\begin{aligned} \left| \frac{d}{dt}(T_t(Y) - T_t(X)) \right| &= |\theta(t, T_t(Y)) - \theta(t, T_t(X))| \\ &\le c|T_t(Y) - T_t(X)| \end{aligned} \tag{2.15}$$

e

$$\int_0^t \frac{d}{ds}(T_s(Y) - T_s(X))\, ds = T_t(Y) - T_t(X) - Y + X,$$

que nos leva, usando (2.15), a

$$\begin{aligned} |T_t(Y) - T_t(X)| &\le |Y - X| + \left| \int_0^t \frac{d}{ds}(T_s(Y) - T_s(X)) ds \right| \\ &\le |Y - X| + \int_0^t c|T_s(Y) - T_s(X)| ds. \end{aligned}$$

Aplicando o lema de Gronwall,

$$|T_t(Y) - T_t(X)| \le |Y - X| \exp\left(\int_0^t c ds \right).$$

Para τ suficientemente pequeno, obtemos

$$|T_t(Y) - T_t(X)| \leq |LY - X|,$$

para todo $t \in [0, \tau]$e para alguma constante $L > 0$, o que prova (2.8).

Agora provaremos (2.9). Seja $X \in \overline{\mathcal{D}}$ e defina

$$y(s) := T_{t-s}(X), \quad 0 \leq s \leq t.$$

Então,

$$\frac{dy}{ds}(s) = -\theta(t-s, y(s)), \quad 0 \leq s \leq t, \quad y(0) = T_t(X). \tag{2.16}$$

Para cada $x \in \overline{\mathcal{D}}$, podemos provar da mesma maneira que acima que a equação diferencial

$$\frac{dy}{ds}(s) = -\theta(t-s, y(s)), \quad 0 \leq s \leq t, \quad y(0) = x \in \overline{\mathcal{D}} \tag{2.17}$$

possui uma solução única em $C^1([0,t], \mathbb{R}^n)$, i.e $y(s) \in \overline{\mathcal{D}}$ para todo $s \in [0,t]$, já que pela hipótese (2.5) temos $-\theta(t,x) \in T_{\overline{\mathcal{D}}}$, $\forall t \in [0,\tau]$ e $\forall x \in \overline{\mathcal{D}}$.

A solução de (2.17) define um mapeamento Lipschitziano

$$x \mapsto S_t(x) = y(s) : \overline{\mathcal{D}} \to \overline{\mathcal{D}},$$

tal que

$$\exists c > 0, \quad |S_t(y) - S_t(x)| \leq c|y - x|, \quad \forall t \in [0,\tau], \forall x, y \in \overline{\mathcal{D}}. \tag{2.18}$$

Agora, tendo em vista (2.16) e (2.17) temos

$$S_t(T_t(X)) = y(t) = T_{t-t}(X) = X \Rightarrow S_t \circ T_t = I \quad \text{em } \overline{\mathcal{D}}.$$

Para obter a outra identidade, considere a seguinte função

$$z(r) = y(t-r; x),$$

onde $y(\cdot, x)$ é a solução de (2.17). Por definição, temos

$$\frac{dz}{dr}(r) = \theta(r, z(r)), \quad z(0) = y(t,x)$$

e então

$$\begin{aligned} x = y(0,x) = z(t) = T_t(y(t,x)) = T_t(S_t(x)) \\ \Rightarrow T_t \circ S_t = I \text{ em } \overline{\mathcal{D}} \Rightarrow S_t = T_t^{-1} : \overline{\mathcal{D}} \to \overline{\mathcal{D}}, \end{aligned}$$

o que prova (2.9). A continuidade Lipschitz uniforme (2.11) segue de (2.18) e (2.9).

Finalmente provaremos (2.10). Dado $t \in [0,\tau]$, tome uma sequência arbitrária $\{t_n\}$, $t_n \to t$. Usando (2.7) e (2.9), para cada $x \in \overline{\mathcal{D}}$, existe $X \in \overline{\mathcal{D}}$ tal que

$$T_t(X) = x \quad \text{e } T_{t_n} \to T_t(X) = x.$$

Também,

$$T_{t_n}^{-1}(x) - T_t^{-1}(x) = T_{t_n}^{-1}(T_t(X)) - T_t^{-1}(T_t(X)) = T_{t_n}^{-1}(T_t(X)) - T_{t_n}^{-1}(T_{t_n}(X)).$$

Usando (2.11) obtemos

$$|T_{t_n}^{-1}(x) - T_t^{-1}(x)| = |T_{t_n}^{-1}(T_t(X)) - T_{t_n}^{-1}(T_{t_n}(X))| \leq c|T_t(X) - T_{t_n}(X)|.$$

Finalmente usando (2.7) obtemos (2.10). □

Observação 2.2. *Teorema 2.1 mostra, em particular, que para $t \in [0,\tau]$, o fluxo $T_t : \overline{\mathcal{D}} \to \overline{\mathcal{D}}$ é um homeomorfismo e, consequentemente, mapeia fronteira em fronteira e interior em interior.*

Definido a transformação T_t^θ nos permite definir uma família de domínios

$$\Omega_t := T_t^\theta(\Omega), \tag{2.19}$$

que permite introduzir o conceito de derivada de forma.

Seja $J : \mathcal{B} \to \mathbb{R}$ um funcional de forma definido para algum conjunto admissível $\mathcal{B} \subset \mathcal{P}(\mathcal{D})$. Introduzimos, então, a seguinte noção de diferenciabilidade com respeito a forma.

Definição 2.8. *A semiderivada Euleriana de J em Ω na direção $\theta \in \mathcal{C}_c^{0,1}(\mathcal{D},\mathbb{R}^n)$ é definida pelo seguinte limite, quando este existir,*

$$dJ(\Omega)(\theta) := \lim_{t \searrow 0} \frac{J(\Omega_t) - J(\Omega)}{t}. \tag{2.20}$$

i) Dizemos que J é diferenciável com respeito a forma Ω se possuir a semiderivada Euleriana em Ω para todo $\theta \in \mathcal{C}_c^\infty(\mathcal{D},\mathbb{R}^n)$, e a função

$$\begin{aligned} dJ(\Omega) : \mathcal{C}_c^\infty(\mathcal{D},\mathbb{R}^n) &\to \mathbb{R}^n, \\ \theta &\mapsto dJ(\Omega)(\theta), \end{aligned}$$

*é linear e contínua. Nesse caso, $dJ(\Omega)$ é chamado de **gradiente de forma** e $dJ(\Omega)(\theta)$ é chamada de **derivada de forma** em Ω.*

ii) A derivada de forma $dJ(\Omega)(\theta)$ é de ordem finita se existir um inteiro $l \geq 0$ e uma constante $c > 0$ tal que para cada compacto $K \subset \mathcal{D}$,

$$|dJ(\Omega)(\theta)| \leq c\|\theta\|_l, \quad \forall \theta \in \mathcal{C}_c^\infty(K,\mathbb{R}^n), \tag{2.21}$$

onde $\|\theta\|_l := \sum_{|\alpha|\leq l} |D^\alpha \theta|_\infty$. Definimos a ordem de $dJ(\Omega)$ como sendo o menor inteiro $l \geq 0$ tal que (2.21) seja verificada.

2.3 O teorema de estrutura

Nosso objetivo nessa seção é descrever as propriedades da derivada de forma em um nível abstrato e enfatizar que todas as representações da derivada de forma satisfazem o mesmo teorema de estrutura. A derivada de forma da Definição 2.8 possui uma estrutura particular. Intuitivamente, o funcional de forma permanece constante pelas transformações T_t que não alteram Ω, isto é, $T_t(\Omega) = \Omega$ mesmo se alguns pontos no interior de Ω mudem. Consequentemente, a derivada de forma é zero. Essa propriedade é válida para Ω aberto ou fechado. Matematicamente, essa ideia é expressa pelo seguinte teorema.

Teorema 2.2. *Seja $\Omega \subset \mathcal{B}$ aberto ou fechado. Seja $\theta \in \mathcal{C}_c^{0,1}(\mathcal{D},\mathbb{R}^n)$ um campo vetorial com suporte compacto em Ω satisfazendo a Hipótese 1. Denote por T_t seu fluxo definido em (2.6). Então temos*

$$dJ(\Omega)(\theta) = 0.$$

Demonstração. Por hipótese, θ possui suporte compacto em Ω e podemos aplicar o Teorema 2.1 com Ω no lugar de $\mathcal{D}$. Assim $T_t : \overline{\Omega} \to \overline{\Omega}$ é um homeomorfismo e temos que $T_t(\overline{\Omega}) = \overline{\Omega}$ para todo $t \in [0,\tau]$. Se Ω for fechado, então $\Omega = \overline{\Omega}$ e portanto $T_t(\Omega) = \Omega$. Agora, se Ω for aberto, temos que $\overline{\Omega} = \Omega \cup \partial\Omega$ e $T_t(\partial\Omega) = \partial\Omega$. Assim, usando o fato de $T_t : \overline{\Omega} \to \overline{\Omega}$ ser um homeomorfismo, temos

$$T_t(\Omega) \cup T_t(\partial\Omega) = T_t(\Omega \cup \partial\Omega) = T_t(\overline{\Omega}) = \overline{\Omega} = \Omega \cup \partial\Omega = \Omega \cup T_t(\partial\Omega),$$

o que implica que $T_t(\Omega) = \Omega$. Portanto, para Ω fechado ou aberto temos

$$\lim_{t \searrow 0} \frac{J(\Omega_t) - J(\Omega)}{t} = \lim_{t \searrow 0} \frac{J(\Omega) - J(\Omega)}{t} = 0. \tag{2.22}$$

Assim, a derivada de forma se anula para tal campo de vetores θ. □

Note que a derivada de forma de $J(\Omega)$ sempre existe para campos de vetores com suporte compacto em Ω, mesmo se não existir para outros campos. Uma importante consequência desse teorema, também para métodos numéricos, é que independentemente da representação da derivada de forma e da regularidade de Ω, os valores de θ fora da fronteira de Ω não têm influência na derivada de forma. Esse fato está expresso no seguinte corolário.

Corolário 2.1. *Seja $\Omega \in \mathcal{B}$ um conjunto com fronteira $\mathcal{C}^1$. Suponha que J seja diferenciável com respeito a Ω. Seja $\theta \in \mathcal{C}^{0,1}_{\partial\Omega}(\mathcal{D}, \mathbb{R}^n)$. Então,*

$$dJ(\Omega)(\theta) = 0.$$

Isso nos leva a um resultado fundamental de otimização de forma. Quando a fronteira do domínio Ω e o campo velocidade θ são suficientemente suaves, a derivada de forma possui uma estrutura particular: ela se restringe a fronteira $\partial\Omega$ e depende apenas do componente normal do campo velocidade θ na fronteira.

Teorema 2.3 (Teorema de estrutura)**.** *Suponha $\Gamma := \partial\Omega$ compacto e J um funcional de forma diferenciável. Denote a derivada de forma por*

$$dJ(\Omega) : \mathcal{C}^\infty_c(\mathcal{D}, \mathbb{R}^n) \to \mathbb{R}, \qquad \theta \mapsto dJ(\Omega)(\theta). \tag{2.23}$$

Se $dJ(\Omega)$ for de ordem $k \geq 0$ e Γ de classe $\mathcal{C}^{k+1}$, então existe uma distribuição $g : \mathcal{C}^k(\Gamma) \to \mathbb{R}$ tal que

$$dJ(\Omega)(\theta) = g(\theta_{|\Gamma} \cdot n), \tag{2.24}$$

Demonstração. Ver [6, Corolário 1, p. 480-481]. □

Antes de começar a calcular uma derivada de forma em um problema de otimização propriamente dito, iremos apresentar alguns resultados que usaremos com frequência nos cálculos desenvolvidos nos próximos capítulos.

Lema 2.4. *Seja T_t solução de (2.6) com $\theta \in \mathcal{C}^{0,1}_c(\mathcal{D}, \mathbb{R}^n)$. Temos então que:*

1) $\frac{d}{dt} DT_{t|_{t=0}} = D\theta$,

2) $\frac{d}{dt} (DT_t)^{-1}_{|t=0} = -D\theta$,

3) $\frac{d}{dt} (DT_t)^{\mathsf{T}}_{|t=0} = (D\theta)^{\mathsf{T}}$,

4) $\frac{d}{dt} (DT_t)^{-\mathsf{T}}_{|t=0} = -(D\theta)^{\mathsf{T}}$.

onde $(D\theta)^{\mathsf{T}}$ denota a transposta de $(D\theta)$.

Demonstração. 1) Pela definição de T_t e pela regra da cadeia temos

$$\frac{d}{dt} DT_t(x) = D\frac{d}{dt} T_t(x) = D(\theta \circ T_t) = D\theta(T_t(x)) DT_t(x). \tag{2.25}$$

Assim,

$$\frac{d}{dt}DT(x)_{|t=0} = D\theta(T_t(x))DT_t(x)|_{t=0} = D\theta(x).$$

2) Como $I = (DT_t)^{-1}(DT_t)$, temos que

$$\begin{aligned} 0 &= \frac{d}{dt}[(DT_t)^{-1}DT_t)] \\ \Rightarrow \quad & \frac{d}{dt}(DT_t)^{-1}DT_t = -(DT_t)^{-1}\frac{d}{dt}DT_t \\ \Rightarrow \quad & \frac{d}{dt}(DT_t)^{-1}|_{t=0} = -(DT_t)^{-1}\left(\frac{d}{dt}DT_t\right)(DT_t)^{-1}|_{t=0} \\ \Rightarrow \quad & \frac{d}{dt}(DT_t)^{-1}|_{t=0} = -D\theta. \end{aligned}$$

3)

$$\begin{aligned} & \frac{d}{dt}(DT_t)^{\mathsf{T}} = \left(\frac{d}{dt}(DT_t)\right)^{\mathsf{T}} = DT_t^{\mathsf{T}}D\theta^{\mathsf{T}} \\ \Rightarrow & \frac{d}{dt}(DT_t)^{\mathsf{T}}_{|t=0} = (DT_t^{\mathsf{T}}D\theta^{\mathsf{T}})_{t=0} = D\theta^{\mathsf{T}}. \end{aligned}$$

4) Basta repetir o raciocínio utilizado para calcular $(DT_t)^{-1}$ usando a igualdade $I = (DT_t)^{-\mathsf{T}}(DT_t)^{\mathsf{T}}$.

□

Em muitos problemas de otimização de forma os funcionais de forma são expressos como integrais definidas em um domínio Ω ou na fronteira desse domínio $\Gamma := \partial\Omega$. Suponha $f \in W^{1,1}_{loc}(\mathbb{R}^n)$ que independe de $t \in [0,\tau]$, $g \in H^2(\mathbb{R}^n)$, Γ de classe C^2 tal que o vetor normal unitário n é classe C^1. O domínio perturbado $\Omega_t = T_t(\Omega)$ com fronteira $\Gamma_t = \partial\Omega_t$ é definido por (2.19).

Consideramos os seguintes funcionais de forma:

$$J_1(t) = \int_{\Omega_t} f(t,y)\,dy, \tag{2.26}$$

$$J_2(t) = \int_{\Gamma_t} g(y)d\Gamma_t(y). \tag{2.27}$$

Usando a mudança de variável $y = T_t(x)$, temos

$$J_1(t) = \int_{\Omega} f(t,T_t(x))h(t,x)\,dx, \tag{2.28}$$

$$J_2(t) = \int_{\Gamma} g((T_t(x)) \circ T_t h_\Gamma(t)d\Gamma(x). \tag{2.29}$$

onde $h(t,x) = |\det(DT_t(x))|$ é o Jacobiano da transformação $y = T_t(x)$ e $h_\Gamma(t) = |M(x,t)\,n|$, com $M(x,t) = (\det DT_t)(DT_t)^{-\mathsf{T}}$ é a matriz dos cofatores de DT_t.

Note que, ao realizar a mudança de variável apenas reescrevemos as funções (2.26) de tal forma que elas passam a ser representadas por uma integral definida no domínio fixo Ω e não mais em um domínio variável Ω_t. Assim, com as funções (2.26) escritas da forma (2.28), aparece o parâmetro t no integrando facilitando o cálculo da derivada de forma.

Podemos ver isso no seguinte exemplo.

Exemplo 1. *Para qualquer subconjunto mensurável $\Omega \subset \mathbb{R}^n$, o funcional de volume está bem definido por*

$$J(\Omega) = \int_{\Omega} dx.$$

Para Ω com volume finito e $\theta \in \mathcal{D}^1(\mathbb{R}^n,\mathbb{R}^n)$, seja $\Omega_t = T_t(\Omega)$ o domínio perturbado. Usando a

mudança de variável $y = T_t(x)$, podemos escrever o funcional de forma que depende do domínio perturbado da seguinte maneira

$$J(\Omega_t) = \int_{\Omega_t} dy = \int_{\Omega} |\det(DT_t)|\, dx = \int_{\Omega} \det(DT_t)\, dx,$$

para t pequeno. A expansão de $f(t) := \det(DT_t)$ para t pequeno é

$$f(t) = f(0) + tf'(0) + o(t^2) \;\Rightarrow \det(DT_t) = 1 + t \operatorname{div}\theta + o(t^2).$$

Assim a derivada é dada por

$$\begin{aligned} dJ(\Omega)(\theta) &= \lim_{t\searrow 0} \frac{J(\Omega_t) - J(\Omega)}{t} \\ &= \lim_{t\searrow 0} \frac{\int_\Omega 1 + t\operatorname{div}\theta + o(t^2)\, dx - \int_\Omega dx}{t} \\ &= \lim_{t\searrow 0} \int_\Omega \operatorname{div}\theta + o(t)\, dx = \int_\Omega \operatorname{div}\theta. \end{aligned}$$

Uma vez que os funcionais de forma são escritos da forma (2.28), isto é, com as integrais em domínios independentes de t, precisamos calcular as derivadas dos integrandos. O seguinte teorema fornece resultados que permitem calcular tais derivadas.

Lema 2.5. *Seja U um subconjunto de $\mathbb{R}$ aberto e $A(t) = a_{ij}(t)_{1\le i,j\le N}$ uma função com valores matriciais diferenciável em U. Denote $d(t) := \det A(t)$, então $d(t)$ é diferenciável e satisfaz*

$$d'(t) = \operatorname{tr}(\operatorname{Adj}(A(t))A'(t)) \text{ para } t \in U, \tag{2.30}$$

onde $\operatorname{Adj}(A(t))$ é a adjunta de $A(t)$, i.e. a transposta da matriz de cofatores:

$$\operatorname{Adj}(A(t)) := \{\operatorname{cof} a_{ji}\}_{1\le i,j\le N}.$$

Além disso, se para cada $t \in U$, $A(t)$ for inversível, então

$$d'(t) = [\operatorname{tr}(A^{-1}(t)A'(t))]d(t). \tag{2.31}$$

Demonstração. Seja $a(t) = \{\alpha_1(t), \alpha_2(t), \ldots, \alpha_n(t)\}$ onde $\alpha_i(t)$ é a i-ésima linha de $A(t)$. Então,

$$d'(t) = \sum_{i=1}^{n} \det(\alpha_1(t), \alpha_2(t), \ldots, \alpha_i'(t), \ldots, \alpha_n(t)).$$

Como $\operatorname{Adj}(A(t)) = \operatorname{cof} a_{ji}$ temos que

$$\operatorname{Adj}(A(t))A'(t) = \left\{ \sum_{k=1}^{n} \operatorname{cof} a_{ki}(t) a_{kj}'(t) \right\}_{1\le i,j\le N}.$$

Portanto,

$$\operatorname{tr}(\operatorname{Adj}(A(t))A'(t)) = \sum_{i=1}^{n} \left\{ \sum_{k=1} n \operatorname{cof} a_{ki}(t) a_{ki}'(t) \right\} = \sum_{k=1}^{n} \left\{ \sum_{i=1}^{n} \operatorname{cof} a_{ki}(t) a_{ki}'(t) \right\}.$$

Usando a expansão de Laplace para determinantes, temos

$$\sum_{i=1}^{n} \operatorname{cof} a_{ki}(t) a_{ki}'(t) = \det(\alpha_1(t), \alpha_2(t), \ldots, \alpha_i'(t), \ldots, \alpha_n(t)).$$

Portanto,

$$\operatorname{tr}(\operatorname{Adj}(A(t))(A'(t))) = \sum_{k=1}^{n} \det(\alpha_1(t), \alpha_2(t), \ldots, \alpha_i'(t), \ldots, \alpha_n(t)) = d'(t).$$

Se para todo $t \in U$, $A(t)$ for inversível, então pela regra de Cramer, $\operatorname{Adj}(A(t)) = d(t)A^{-1}(t)$ para $t \in U$. Assim, obtemos

$$d'(t) = \operatorname{tr}(d(t)A^{-1}(t)A'(t)) = [\operatorname{tr}(A^{-1}(t)A'(t))]d(t).$$

□

Teorema 2.6. *Seja $\theta \in \mathcal{C}_c^{0,1}(\mathbb{R}^n, \mathbb{R}^n)$ que satisfaz a Hipótese 1.*

1) Seja $f \in L^1(\mathbb{R}^n)$, então

$$\lim_{t \searrow 0} \|f \circ T_t - f\|_{L^1(\mathbb{R}^n)} = 0. \tag{2.32}$$

2) Se $\theta \in \mathcal{C}_{loc}^1(\mathbb{R}^n, \mathbb{R}^n)$ e $f \in W^{1,1}(\mathbb{R}^n)$, a função

$$t \mapsto f \circ T_t \in W_{loc}^{1,1}(\mathbb{R}^n)$$

está bem definida e para cada t temos

$$\frac{d}{dt} f \circ T_t = (\nabla f \cdot \theta) \circ T_t \in L_{loc}^1(\mathbb{R}^n). \tag{2.33}$$

Consequentemente a função $t \mapsto f \circ T_t$ pertence a $\mathcal{C}^1([0,\tau]; L_{loc}^1(\mathbb{R}^n)) \cap \mathcal{C}^0([0,\tau]; W_{loc}^{1,1}(\mathbb{R}^n))$.

3) Seja $f \in W_{loc(\mathbb{R}^n)}^{1,1}$. Para cada $t \in [0,\tau]$, a função

$$f \mapsto f \circ T_t : W_{loc}^{1,1}(\mathbb{R}^n) \to W_{loc}^{1,1}(\mathbb{R}^n) \tag{2.34}$$

e sua inversa são localmente Lipschitz e

$$\nabla(f \circ T_t) = DT_t^{\mathsf{T}} \nabla f \circ T_t. \tag{2.35}$$

4) Se $\theta \in \mathcal{C}_{loc}^1(\mathbb{R}^n, \mathbb{R}^n)$, então a função

$$t \mapsto h(t, \cdot) : [0,\tau] \to \mathcal{C}_{loc}^0(\mathbb{R}^n)$$

é diferenciável e

$$\frac{dh(t,\cdot)}{dt} = [\operatorname{div} \theta] \circ T_t h(t,\cdot) \in \mathcal{C}_{loc}^0(\mathbb{R}^n). \tag{2.36}$$

Consequentemente a função $t \mapsto h(t,\cdot)$ pertence a $\mathcal{C}^1([0,\tau]; \mathcal{C}_{loc}^0(\mathbb{R}^n))$.

Demonstração. 1) Seja $f_p \in \mathcal{C}_0^\infty(\mathbb{R}^n)$. Assim,

$$\begin{aligned} \|f \circ T_t - f\|_{L^1(\mathbb{R}^n)} &= \|f \circ T_t - f + f_p - f_p + f_p \circ T_t - f_p \circ T_t\|_{L^1(\mathbb{R}^n)} \\ &\leq \|f - f_p\|_{L^1(\mathbb{R}^n)} + \|f_p - f_p \circ T_t\|_{L^1(\mathbb{R}^n)} + \|f_p \circ T_t - f \circ T_t\|_{L^1(\mathbb{R}^n)} \end{aligned}$$

Usando a mudança de variável $y = T_t(x)$ no último termo e o fato do Jacobiano da transformação usada ser uniformemente limitada, temos que

$$\|f \circ T_t - f\|_{L^1(\mathbb{R}^n)} \leq C\|f - f_p\|_{L^1(\mathbb{R}^n)} + \|f_p - f_p \circ T_t\|_{L^1(\mathbb{R}^n)}.$$

Como $f_p \in \mathcal{C}_0^\infty(\mathbb{R}^n)$, temos $\|f_p - f_p \circ T_t\|_{L^1(\mathbb{R}^n)} \to 0$ quando $t \searrow 0$ para p fixo, e $\|f - f_p\|_{L^1(\mathbb{R}^n)}$ pode ser tornar arbitrariamente pequeno pela escolha de f_p, pois $\mathcal{C}_0^\infty(\mathbb{R}^n)$ é denso em $L^1(\mathbb{R}^n)$. Isso prova 2.32.

2) Provaremos (2.33) em $t = 0$. Seja $f_p \in \mathcal{C}_0^\infty(\mathbb{R}^n)$. Temos a expansão

$$\begin{aligned} & f_p(T_t(x)) - f_p(x) - t\nabla f_p \cdot \theta(T_t(x)) - t\nabla f_p(x) \cdot \epsilon(t,x) \\ = & \int_0^1 [\nabla f_p(x + s(T_t(x) - x)) - \nabla f_p(x)] \cdot (T_t(x) - x)\, ds. \end{aligned} \tag{2.37}$$

onde $\epsilon(t,\cdot) \to 0$ em $L^\infty(\mathbb{R}^n)$. Integrando com respeito a x e definindo

$$\eta_t(f_p) = t^{-1}\|f_p(T_t(x)) - f_p(x) - t\nabla f_p \cdot \theta(T_t(x))\|_{L^1(\mathbb{R}^n)},$$

$$e(t, f_p)(x) := \int_0^1 |\nabla f_p(x - s(T_t(x) - x)) - \nabla f_p(x)|\, ds.$$

Usando (2.37), obtemos

$$\begin{aligned} \eta_t(f_p) &= \left\| \frac{f_p(T_t(x)) - f_p(x) - t\nabla f_p \cdot \theta(T_t(x))}{t} \right\|_{L^1(\mathbb{R}^n)} \\ &= \|\nabla f_p(x) \cdot \epsilon(t,x) + \int_0^1 [\nabla f_p(x + s(T_t(x) - x)) - \nabla f_p(x)] \cdot (T_t(x) - x)\|_{L^1(\mathbb{R}^n)}, \end{aligned}$$

o que fornece

$$\eta_t(f_p) \le \|\nabla f_p\|_{L^1(\mathbb{R}^n)} \|\epsilon(t,\cdot)\|_{L^\infty(\mathbb{R}^n)} + c_1 \|e(t, f_p)\|_{L^1(\mathbb{R}^n)}, \tag{2.38}$$

onde c_1 é um limitante uniforme para $\|T_t - I\|_{L^\infty}$. Denotando $C(f_p)$ a medida do suporte de f_p, temos as estimativas

$$\|e(t, f_p)\|_{L^1(\mathbb{R}^n)} \le C(f_p)\|\nabla f_p\|_{L^\infty(\mathbb{R}^n)} \|T_t - I\|_{L^\infty(\mathbb{R}^n)}, \tag{2.39}$$

$$\|e(t, f_p)\|_{L^1(\mathbb{R}^n)} \le 2\|\nabla f_p\|_{L^1(\mathbb{R}^n)} \|T_t\|_{W^{1,\infty}(\mathbb{R}^n)}, \tag{2.40}$$

onde usamos a mudança de variável $z = x + s(T_t(x) - x)$ para obter (2.40).

Agora, seja $f \in W^{1,1}(\mathbb{R}^n)$ e $f_p \in \mathcal{C}_0^\infty(\mathbb{R}^n)$ que converge para f em $W^{1,1}(\mathbb{R}^n)$. Podemos passar o limite $f_p \to f$ em (2.38) e obtemos a mesma desigualdade, mas com f ao invés de f_p, i.e,

$$\eta_t(f) \le \|\nabla f\|_{L^1(\mathbb{R}^n)} \|\epsilon(t,\cdot)\|_{L^\infty(\mathbb{R}^n)} + c_1 \|e(t, f)\|_{L^1(\mathbb{R}^n)}. \tag{2.41}$$

Também temos

$$e(t, f) \le e(t, f - f_p) + e(t, f_p) \tag{2.42}$$

assim, usando (2.39),(2.40) e (2.42), obtemos

$$\|e(t, f)\|_{L^1(\mathbb{R}^n)} \le 2\|\nabla(f - f_p)\|_{L^1(\mathbb{R}^n)} \|T_t\|_{W^{1,\infty}(\mathbb{R}^n)} + C(f_p)\|\nabla f_p\|_{L^\infty(\mathbb{R}^n)} \|T_t - I\|_{L^\infty(\mathbb{R}^n)}.$$

Usando o fato de $\|T_t\|_{W^{1,\infty}(\mathbb{R}^n)}$ ser uniformemente limitada obtemos $\|e(t, f)\|_{L^1(\mathbb{R}^n)} \to 0$ quando $t \to 0$ e finalmente $\eta_t \to 0$ usando (2.38).

3) A função dada em (2.34) e sua inversa são localmente Lipschitz, pois T e T^{-1} o são com respeito a x, veja itens 2) e 3) do Teorema 2.1. A expressão (2.35) é obtida pela regra da cadeia.

4) Aqui usaremos o Lema 2.5 onde $A(t) = DT_t$ e $d(t) = \det A(t) = \det DT_t = h(t,x)$. Assim,

temos que

$$\begin{aligned}
\frac{d}{dt}h(t,x) = \frac{d}{dt}\det DT_t(x) &= [\operatorname{tr}(DT_t^{-1}(x)D\theta(T_t)DT_t(x))]h(t,x) \\
&= [\operatorname{tr}(D\theta(T_t(x)DT_tDT_t^{-1})]h(t,x) \\
&= [\operatorname{tr} D\theta(T_t(x))]h(t,x) \\
&= [\operatorname{div}\theta]\circ T_t(x)h(t,x).
\end{aligned}$$

□

No caso em que o funcional de forma está definido na fronteira, vimos que com a mudança de variável no integrando acaba aparecendo a expressão do Jacobiano tangencial

$$h_\Gamma(t) = h(t)|(DT_t)^{-\mathsf{T}}n|.$$

Com o item 4 do Teorema 2.6 temos que $h_\Gamma(t)$ é diferenciável em $t = 0$ em $\mathcal{C}^0(\Gamma)$. Considerando a expressão

$$\begin{aligned}
h_\Gamma(t) &= h(t)|DT_t^{-\mathsf{T}}n| \\
&= h(t)\langle DT_t^{-\mathsf{T}}n, DT_t^{-\mathsf{T}}n\rangle^{\frac{1}{2}}
\end{aligned}$$

Podemos calcular sua derivada como

$$\begin{aligned}
h'_\Gamma(0) &= h'(0) + \frac{h(0)}{2}\frac{2\langle \frac{d}{dt}DT_t^{-\mathsf{T}}n|_{t=0}, DT_0^{-\mathsf{T}}n\rangle}{\langle DT_0^{-\mathsf{T}}n, DT_0^{-\mathsf{T}}n\rangle} \\
&= \operatorname{div}\theta - \langle D\theta n, n\rangle.
\end{aligned}$$

Capítulo 3

Adjunto Médio

Neste capítulo introduzimos o método do adjunto médio [20, 10], que é um método para calcular derivadas de forma de funcionais que dependem de soluções de uma EDP. Para ilustrar as ideias principais dessa abordagem, começaremos com a descrição de um problema similar em dimensão finita, que facilita a compreensão do método quando estudado no contexto de otimização de forma.

3.1 O Lagrangiano em dimensão finita

Seja $f : \mathbb{R}^{m+n} \to \mathbb{R}$ de classe $\mathcal{C}^k$, $h : \mathbb{R}^{m+n} \to \mathbb{R}^m$ de classe $\mathcal{C}^k$ e considere o problema

$$\text{minimizar} \quad f(x) \tag{3.1}$$

$$\text{sujeito a} \quad x \in \mathbb{R}^{m+n} \quad \text{e} \quad h(x) = 0. \tag{3.2}$$

É conhecido que se $u \in \mathbb{R}^{m+n}$ é um ponto crítico para (3.1)-(3.2), então existem multiplicadores de Lagrange $\{p_i\}_{i=1}^m$ com $p_i \in \mathbb{R}$ tal que

$$\begin{aligned} h(u) &= 0, \\ \nabla f(u) &= \sum_{i=1}^m p_i \nabla h_i(u). \end{aligned}$$

Em vista desse resultado, definimos o *Lagrangiano* como

$$\mathcal{L}(x, \lambda) := f(x) - \sum_{i=1}^m \lambda_i h_i(x) = f(x) - \lambda \cdot h(x),$$

onde $\lambda = (\lambda_1, \lambda_2, \ldots, \lambda_m) \in \mathbb{R}^m$ e $(h_1(x), h_2(x), \cdots, h_m(x)) \in \mathbb{R}^m$. O Lagrangiano possui as seguintes propriedades:

$$\begin{aligned} \mathcal{L}(u, \lambda) &= f(u), \quad \forall \lambda \in \mathbb{R}^m, \\ \nabla_x \mathcal{L}(u, p) &= \nabla f(u) - \sum_{i=1}^m p_i \nabla h_i(u) = 0, \\ \nabla_\lambda \mathcal{L}(x, \lambda) &= -h(x). \end{aligned}$$

Neste trabalho, estamos interessados no caso onde as funções f e h são parametrizadas pelo escalar $t \in \mathbb{R}$, i.e. consideramos funções $f : \mathbb{R} \times \mathbb{R}^{m+n} \to \mathbb{R}$ e $h : \mathbb{R} \times \mathbb{R}^{m+n} \to \mathbb{R}^m$, ambas de classe C^k. Nesse contexto podemos considerar o problema parametrizado por t:

$$\text{minimizar} \quad f(t, x) \tag{3.3}$$

$$\text{sujeito a} \quad x \in \mathbb{R}^{m+n} \quad \text{e} \quad h(t, x) = 0. \tag{3.4}$$

Seja $u(t)$ um ponto crítico parametrizado de (3.3)-(3.4), defini-se então o Lagrangiano parametrizado como

$$\mathcal{L}(t,x,\lambda) := f(t,x) - \sum_{i=1}^{m} \lambda_i h_i(t,x) = f(t,x) - \lambda \cdot h(t,x). \tag{3.5}$$

Observe que temos

$$\begin{aligned} \mathcal{L}(t,u(t),\lambda) &= f(t,u(t)), \ \ \forall \lambda \in \mathbb{R}^m. && (3.6)\\ \nabla_x \mathcal{L}(t,u(t),p(t)) &= \nabla_x f(t,u(t)) - \sum_{i=1}^{m} p_i(t) \nabla_x h_i(t,u(t)) = 0, && (3.7) \end{aligned}$$

onde $p_i(t)$ são multiplicadores de Lagrange parametrizados por t. Nosso interesse é o cálculo da derivada em $t = 0$ da função custo $J(t) := f(t,u(t))$. Assim, supondo que $u(t)$ e f são deriváveis em $t = 0$, para todo $\lambda \in \mathbb{R}^m$, temos por causa de (3.6) que

$$J'(0) = \frac{d}{dt}[f(t,u(t))]|_{t=0} = \frac{d}{dt}[\mathcal{L}(t,u(t),\lambda)]|_{t=0}, \ \ \forall \lambda \in \mathbb{R}^m. \tag{3.8}$$

Obtemos

$$J'(0) = \partial_t \mathcal{L}(0,u(0),\lambda) + \nabla_x \mathcal{L}(0,u(0),\lambda) \cdot u'(0).$$

Como essa igualdade é válida para qualquer λ podemos escolher $\lambda = p(0) \in \mathbb{R}^m$ com $p(t) = (p_1(t), p_2(t), \ldots, p_m(t))$ que nos fornece, tendo em vista (3.7)

$$J'(0) = \partial_t \mathcal{L}(0,u(0),p(0)) + \nabla_x \mathcal{L}(0,u(0),p(0)) \cdot u'(0) = \partial_t \mathcal{L}(0,u(0),p(0)). \tag{3.9}$$

Isso mostra que para calcular $J'(0)$, precisamos conhecer $u(0)$ e $p(0)$, mas não precisamos calcular $u'(0)$. Essa é a propriedade chave usada para abordagens Lagrangianas em otimização de forma, como veremos na seção seguinte.

No entanto, ainda existe um obstáculo para se usar tal propriedade, que é o fato de termos assumido a existência de $u'(0)$. A prova desse fato pode ser similar ao cálculo de $u'(0)$, contudo queremos evitar tudo isso. A principal ideia do método adjunto médio é introduzir um *adjunto médio* $\tilde{p}(t)$ que satisfaz

$$\begin{aligned} \mathcal{L}(t,u(t),\tilde{p}(t)) - \mathcal{L}(t,u(0),\tilde{p}(t)) &= 0, && (3.10)\\ \tilde{p}(0) &= p(0). && (3.11) \end{aligned}$$

Assim, usando (3.10) temos

$$\begin{aligned} &\mathcal{L}(t,u(t),\tilde{p}(t)) - \mathcal{L}(0,u(0),\tilde{p}(t))\\ &= \mathcal{L}(t,u(t),\tilde{p}(t)) - \mathcal{L}(t,u(0),\tilde{p}(t)) + \mathcal{L}(t,u(0),\tilde{p}(t)) - \mathcal{L}(0,u(0),\tilde{p}(t)) && (3.12)\\ &= \mathcal{L}(t,u(0),\tilde{p}(t)) - \mathcal{L}(0,u(0),\tilde{p}(t)). \end{aligned}$$

Então, para o funcional de custo, usando (3.6)

$$\begin{aligned} \frac{f(t,u(t)) - f(0,u(0))}{t} &= \frac{\mathcal{L}(t,u(t),\lambda) - \mathcal{L}(0,u(0),\lambda)}{t}\\ &= \frac{\mathcal{L}(t,u(t),\tilde{p}(t)) - \mathcal{L}(0,u(0),\tilde{p}(t))}{t}, && (3.13) \end{aligned}$$

pois vale para todo $\lambda \in \mathbb{R}^m$. Assim, temos, usando (3.13) e (3.12)

$$\begin{aligned} J'(0) &= \lim_{t\to 0} \frac{\mathcal{L}(t,u(t),\tilde{p}(t)) - \mathcal{L}(0,u(0),\tilde{p}(t))}{t}\\ &= \lim_{t\to 0} \frac{\mathcal{L}(t,u(0),\tilde{p}(t)) - \mathcal{L}(0,u(0),\tilde{p}(t))}{t}. \end{aligned}$$

Finalmente, se conseguirmos provar que

$$\lim_{t\to 0}\frac{\mathcal{L}(t,u(0),\tilde{p}(t))-\mathcal{L}(0,u(0),\tilde{p}(t))}{t}=\partial_t\mathcal{L}(0,u(0),p(0)), \tag{3.14}$$

então teremos

$$J'(0)=\partial_t\mathcal{L}(0,u(0),p(0)).$$

Dessa forma, obtemos (3.9) sem precisar provar a existência de $u'(0)$. Em contrapartida, temos que provar a hipótese (3.14), que corresponderá a Hipótese (3) da seção 3.2.

A propriedade (3.10) pode ser obtida simplesmente tomando $\tilde{p}(t)$ que satisfaz

$$\int_0^1 \nabla_x\mathcal{L}(t,su(t)+(1-s)u(0),\tilde{p}(t))\,ds=0. \tag{3.15}$$

De fato, multiplicando (3.15) por $(u(t)-u(0))$, obtemos

$$\begin{aligned}&\int_0^1 (u(t)-u(0))\cdot\nabla_x\mathcal{L}(t,su(t)+(1-s)u(0),\tilde{p}(t))ds\\&=\int_0^1\frac{d}{ds}\mathcal{L}(t,su(t)+(1-s)u(0),\tilde{p}(t))ds\\&=\mathcal{L}(t,u(t),\tilde{p}(t))-\mathcal{L}(t,u(0),\tilde{p}(t))=0.\end{aligned}$$

Considerando (3.5) com $\lambda=\tilde{p}_i(t)$, a equação (3.15) também pode ser escrita da forma

$$\int_0^1 \nabla_x f(t,su(t)+(1-s)u(0))\,ds=\sum_{i=1}^m\tilde{p}_i(t)\int_0^1\nabla_x h_i(t,su(t)+(1-s)u(0))\,ds. \tag{3.16}$$

Fazendo $t=0$ em (3.16) obtemos

$$\nabla_x f(0,u(0))-\sum_{i=1}^m\tilde{p}_i(0)\nabla_x h_i(0,u(0))=0, \tag{3.17}$$

Fazendo a diferença entre (3.17) e (3.7) em $t=0$, obtemos

$$\sum_{i=1}^m(p_i(0)-\tilde{p}_i(0))\nabla_x h_i(0,u(0))=0.$$

Em forma matricial escreve-se

$$D_x h(0,u(0))^{\mathsf{T}}(p(0)-\tilde{p}(0))=0,$$

onde $D_x h(0,u(0))$ é a matriz Jacobiana de h em $(0,u(0))$. Se $D_x h(0,u(0))$ tem posto máximo, então obtemos $\tilde{p}(0)=p(0)$ que corresponde a (3.11). Na próxima seção, seguiremos a mesma linha de raciocínio para problemas de otimização de forma, mas com a dificuldade adicional de que a variável $u(t)$ representará a solução de uma EDP que pertence a um espaço de funções de dimensão infinita.

3.2 Derivada de forma via método do adjunto médio

Métodos Lagrangianos em otimização de forma nos permitem calcular a derivada de forma de funções que dependem das soluções de equações diferenciais parciais eficientemente, pois eles fornecem imediatamente a equação do adjunto. Veja [6] para a descrição de tal método no caso linear. Aqui o método do adjunto médio é um método tipo Lagrangiano introduzido em [20]. Com essa abordagem a computação da derivada de forma é rápida, a recuperação da expressão da fronteira é conveniente e, não é necessário a suposição de pontos de sela, diferentemente em [6].

Sejam dois espaços vetoriais $E = E(\Omega)$, $F = F(\Omega)$ e $\tau > 0$ dados e considere a parametrização $\Omega_t = T_t(\Omega)$ para $t \in [0,\tau]$. Nosso objetivo é diferenciar funções de forma do tipo $J(\Omega_t)$ que podem ser escritas, usando o Lagrangiano, como $J(\Omega_t) = \mathcal{L}(\Omega_t, u^t, \hat{\psi})$, onde $u^t \in E(\Omega_t)$ e $\hat{\psi} \in F(\Omega_t)$. O principal recurso do Lagrangiano é que para calcular a derivada de $J(\Omega_t)$ precisaremos apenas calcular a derivada de $\mathcal{L}(\Omega_t, \hat{\varphi}, \hat{\psi})$ com respeito a t, resultado que será dado pelo Teorema 3.1.

Como $\mathcal{L}(\Omega_t, \hat{\varphi}, \hat{\psi})$ frequentemente é constituído de integrais em $T_t(\Omega)$, usando uma mudança de variável podemos reescrever essas integrais em integrais no domínio fixo Ω e, consequentemente, transferir a dependência de t para o integrando. Contudo, no processo aparece as funções compostas $\hat{\varphi} \circ T_t \in E(\Omega)$ e $\hat{\psi} \circ T_t \in F(\Omega)$, cujas derivadas não são calculadas diretamente já que $\hat{\varphi}$ e $\hat{\psi}$ são definidas nos espaços $E(\Omega_t)$ e $F(\Omega_t)$.

Felizmente, para calcular a derivada de forma podemos reparametrizar o problema considerando $\mathcal{L}(\Omega_t, \varphi \circ T_t^{-1}, \psi \circ T_t^{-1})$ em vez de $\mathcal{L}(\Omega_t, \hat{\varphi}, \hat{\psi})$, onde $\varphi \in E(\Omega)$, $\psi \in F(\Omega)$. Assim, a mudança de variável nas integrais leva as funções φ e ψ no integrando, qual está definido em espaços fixos.

Assim levamos em consideração funções gerais do tipo $G : [0,\tau] \times E \times F \to \mathbb{R}$,

$$G(t, \varphi, \psi) := \mathcal{L}(T_t(\Omega), \varphi \circ T_t^{-1}, \psi \circ T_t^{-1}).$$

O resultado principal dessa seção mostra que para obter a derivada de forma de $\mathcal{L}$, é suficiente calcular a derivada de G com respeito a t enquanto atribuímos valores apropriados de φ e ψ.

Além disso, consideramos a seguinte forma específica

$$G(t, \varphi, \psi) := a(t, \varphi, \psi) + b(t, \varphi), \tag{3.18}$$

onde $a : [0,\tau] \times E \times F \to \mathbb{R}$ e $b : [0,\tau] \times E \to \mathbb{R}$ são funções tais que $\psi \mapsto a(t, \varphi, \psi)$ é linear para todo $t \in [0,\tau]$ e $\varphi \in E$.

Vamos assumir que para cada $t \in [0,\tau]$ a equação

$$d_\psi G(t, u^t, 0; \hat{\psi}) = a(t, u^t, \hat{\psi}) = 0, \qquad \forall \hat{\psi} \in F, \tag{3.19}$$

admite uma solução única $u^t \in E$. Admitiremos também as seguintes suposições para G.

Hipótese 2. *Para todo $(t, \psi) \in [0,\tau] \times F$*

(i) $[0,1] \ni s \mapsto G(t, su^t + (1-s)u^0, \psi)$ *é absolutamente contínua.*

(ii) $[0,1] \ni s \mapsto d_\varphi G(t, su^t + (1-s)u^0, \psi; \hat{\varphi})$ *pertence a* $L^1(0,1)$ *para todo* $\hat{\varphi} \in E$.

Para $t \in [0,\tau]$ introduzimos a equação do adjunto médio associado a u^t e u^0 : encontrar $p^t \in F$ tal que

$$\int_0^1 d_\varphi G(t, su^t + (1-s)u^0, p^t; \hat{\varphi})\, ds = 0, \quad \forall \hat{\varphi} \in E. \tag{3.20}$$

Note que, para todo $t \in [0,\tau]$,

$$G(t, u^t, p^t) - G(t, u^0, p^t) = \int_0^1 d_\varphi G(t, su^t + (1-s)u^0, p^t; u^t - u^0)\, ds = 0. \tag{3.21}$$

Podemos agora enunciar o principal resultado dessa seção

Hipótese 3. *Assumimos que*

$$\lim_{t \searrow 0} \frac{G(t, u^0, p^t) - G(0, u^0, p^t)}{t} = \partial_t G(0, u^0, p^0).$$

Teorema 3.1. *Suponha que as Hipóteses 2 e 3 são satisfeitas e que exista uma única solução p^t da equação do adjunto médio 3.20. Então, para qualquer $\psi \in F$ obtemos*

$$\frac{d}{dt} b(t, u^t)|_{t=0} = \frac{d}{dt} G(t, u^t, \psi)|_{t=0} = \partial_t G(0, u^0, p^0). \tag{3.22}$$

Demonstração. Denote $g(t) := G(t, u^t, p^t) - G(0, u^0, p^t)$ e considerando (3.19) temos

$$g(t) = b(t, u^t) - b(0, u^0) = G(t, u^t, \psi) - G(0, u^0, \psi), \ \forall \psi \in F.$$

Usando a definição do adjunto médio p^t, obtemos

$$g(t) = \underbrace{G(t, u^t, p^t) - G(t, u^0, p^t)}_{=0 \text{ devido a } (3.21)} + G(t, u^0, p^t) - G(0, u^0, p^t).$$

Observando que $g(0) = 0$ e usando Hipótese 3 obtemos

$$\begin{aligned}
\frac{d}{dt} b(t, u^t)|_{t=0} &= g'(0) \\
&= \lim_{t \searrow 0} \frac{g(t)}{t} \\
&= \lim_{t \searrow 0} \frac{G(t, u^0, p^t) - G(0, u^0, p^t)}{t} \\
&= \partial_t G(0, u^0, p^0).
\end{aligned}$$

□

Capítulo 4

Representação tensorial das derivadas de forma

Nessa seção vamos identificar a representação tensorial da derivada de forma que corresponde a uma vasta classe de problemas estudados na literatura para otimização de forma com restrição de EDP. Essa representação tensorial possui várias propriedades interessantes. Em particular, exibimos a conexão entre essa representação tensorial e a expressão usual de fronteira da derivada de forma.

4.1 Definições e propriedades

Definição 4.1 (Gradiente tangencial). *Seja $\Omega \subset \mathbb{R}^d$, $\Gamma = \partial\Omega$ de classe C^2 e $g \in C^1(\Gamma)$. O gradiente tangencial ∇_Γ é definido como*

$$\nabla_\Gamma g = \nabla \tilde{g} - (\nabla \tilde{g} \cdot n)n \text{ em } \Gamma, \tag{4.1}$$

onde $\tilde{g} \in \mathcal{C}^1(\mathbb{R}^d)$ é uma extensão de g e n o vetor normal unitário apontando para fora de Ω.

Definição 4.2 (Divergência tangencial). *Seja θ uma função vetorial em $\mathcal{C}^1(\Gamma, \mathbb{R}^n)$. A divergência tangencial $\operatorname{div}_\Gamma$ é definida como*

$$\operatorname{div}_\Gamma \theta = \operatorname{div} \tilde{\theta} - D\tilde{\theta} n \cdot n \text{ em } \Gamma, \tag{4.2}$$

onde $\tilde{\theta} \in \mathcal{C}^1(\mathbb{R}^n, \mathbb{R}^n)$ é uma extensão de θ. Também temos a fórmula

$$\operatorname{div}_\Gamma \theta = \operatorname{tr}(D_\Gamma \theta),$$

onde

$$D_\Gamma \theta = D\tilde{\theta} - (D\tilde{\theta} n) \otimes n.$$

Definição 4.3 (Curvatura aditiva). *Suponha $\Omega \subset \mathbb{R}^2$ de classe C^2. A curvatura aditiva de Γ é definida por $\mathcal{H} := \operatorname{div}_\Gamma n$.*

Definição 4.4. *Seja $\Omega \in \mathcal{B}$ um conjunto com fronteira $\mathcal{C}^k$, $k \geq 1$, onde $\mathcal{B} \subset \mathcal{P}(\mathcal{D})$ é um conjunto admissível e $\mathcal{D} \subset \mathbb{R}^n$. Dizemos que uma função de forma diferenciável J de ordem k admite uma representação tensorial se existirem tensores $\boldsymbol{S}_l \in L^1(\mathcal{D}, \mathcal{L}^l(\mathbb{R}^n, \mathbb{R}^n))$ e $\mathfrak{S}_l \in L^1(\partial\Omega; \mathcal{L}^l(\mathbb{R}^n, \mathbb{R}^n))$, $l = 0, \ldots, k$, tais que*

$$dJ(\Omega)(\theta) = \sum_{l=0}^{k} \int_{\mathcal{D}} \boldsymbol{S}_l : D^l \theta dx + \int_{\partial\Omega} \mathfrak{S}_l : D^l_\Gamma \theta ds, \qquad \forall \theta \in \mathcal{C}^k_c(\mathcal{D}, \mathbb{R}^n). \tag{4.3}$$

Aqui, $\mathcal{L}^l(\mathbb{R}^n, \mathbb{R}^n)$ denota o espaço das funções multilineares de $\underbrace{\mathbb{R}^n \times \ldots \times \mathbb{R}^n}_{l \text{ vezes}}$ em $\mathbb{R}^n$.

Observação 4.1. *(a) Um caso particular da representação tensorial (4.3) é o tensor energia-momento de Eshelby, veja [13].*

(b) Quando J é diferenciável com respeito à forma Ω, então por definição $\theta \mapsto dJ(\Omega)(\theta)$ é uma distribuição, e se $\partial\Omega$ é compacto, a distribuição $\theta \mapsto dJ(\Omega)(\theta)$ é de ordem finita.

(c) Se $dJ(\Omega)$ for de ordem $k = 1$ e $|dJ(\Omega)(\theta)| \leq C\|\theta\|_{H^1(\mathcal{D},\mathbb{R}^n)}$ para todo $\theta \in \mathcal{C}_c^\infty(\mathcal{D},\mathbb{R}^n)$, então pela densidade de $\mathcal{C}_c^\infty(\mathcal{D},\mathbb{R}^n)$ em $H_0^1(\mathcal{D},\mathbb{R}^n)$ a derivada $dJ(\Omega)$ se estende a um funcional contínuo em $H_0^1(\mathcal{D},\mathbb{R}^n)$, isto é,

$$|\widehat{dJ(\Omega)(\theta)}| \leq c\|\theta\|_{H^1(\mathcal{D},\mathbb{R}^n)}, \qquad \theta \in H_0^1(\mathcal{D},\mathbb{R}^n).$$

Pelo teorema de Riesz, obtemos o campo de vetor W em $H^1(\mathcal{D},\mathbb{R}^n)$ tal que

$$\forall \theta \in H_0^1(\mathcal{D},\mathbb{R}^n), \qquad \widehat{dJ(\Omega)(\theta)} = \int_{\mathcal{D}} DW : D\theta + W \cdot \theta dx,$$

e isso define a representação tensorial (4.3) com $\boldsymbol{S}_1 = DW$, $\boldsymbol{S}_0 = W$, $\mathfrak{S}_1 = 0$ e $\mathfrak{S}_0 = 0$.

(d) A hipótese de que Ω seja um conjunto de classe $\mathcal{C}^k$ pode ser enfraquecida se $\mathfrak{S}_l \equiv 0$ para todo $0 \leq l \leq k$.

Lema 4.1. *Seja $a, b, c \in \mathbb{R}^n$ vetores colunas e $M \in \mathbb{R}^{n,n}$ uma matriz quadrada, $f \in \mathcal{C}^1(\mathbb{R}^n,\mathbb{R}^n)$, $g \in \mathcal{C}^1(\mathbb{R}^n)$ então temos*

$$M : (b \otimes c) = (Mc) \cdot b, \tag{4.4}$$
$$(a \otimes b)c = (b \cdot c)a, \tag{4.5}$$
$$D(gf) = gDf + f \otimes \nabla g, \tag{4.6}$$
$$\operatorname{div} \theta = Id : D\theta, \tag{4.7}$$

onde Id é a matriz identidade.

A representação tensorial (4.3) não é única no sentido que existem várias formas de escolher os tensores $\mathbf{S}_l$ e $\mathfrak{S}_l$. Isso é expresso pelo fato desses tensores estarem correlacionados. Descrevemos essas relações para o caso $k = 1$ na Proposição 4.1, que descreve a ligação entre a representação tensorial e a representação de fronteira usual (2.24) da derivada de forma.

Proposição 4.1. *Seja Ω um subconjunto de $\mathcal{D}$ com fronteira $\mathcal{C}^1$. Suponha que a derivada $dJ(\Omega)$ possui a representação*

$$dJ(\Omega)(\theta) = \int_D \boldsymbol{S}_1 : D\theta + \boldsymbol{S}_0 \cdot \theta dx + \int_{\partial\Omega} \mathfrak{S}_1 : D_\Gamma \theta + \mathfrak{S}_0 \cdot \theta ds. \tag{4.8}$$

Se $\boldsymbol{S}_l$ for de classe $W^{1,1}$ em Ω e $\mathcal{D}\backslash\overline{\Omega}$ então, indicando por $+$ e $-$ as restrições dos tensores em Ω e $D\backslash\overline{\Omega}$, respectivamente, temos

$$-\operatorname{div}(\boldsymbol{S}_1^+) + \boldsymbol{S}_0^+ = 0 \qquad \text{em } \Omega, \tag{4.9}$$
$$-\operatorname{div}(\boldsymbol{S}_1^-) + \boldsymbol{S}_0^- = 0 \qquad \text{em } D\backslash\overline{\Omega}. \tag{4.10}$$

Além disso, podemos reescrever a representação tensorial como uma distribuição na fronteira:

$$dJ(\Omega)(\theta) = \int_{\partial\Omega} [(\boldsymbol{S}_1^+ - \boldsymbol{S}_1^-)n] \cdot \theta + \mathfrak{S}_1 \cdot D_\Gamma \theta + \mathfrak{S}_0 \cdot \theta \, ds,$$

onde n denota o vetor normal orientado para fora de Ω.

Se a fronteira $\partial\Omega$ for C^2 e $g_1 \in W^{1,1}(\partial\Omega; \mathcal{L}^1(\mathbb{R}^n,\mathbb{R}^n))$, então obtemos uma distribuição mais regular, a expressão de fronteira da derivada de forma:

$$dJ(\Omega)(\theta) = \int_{\partial\Omega} g_1 \theta \cdot n \, ds, \tag{4.11}$$

onde

$$g_1 := [(S_1^+ - S_1^-)n] \cdot n + \mathfrak{S}_0 \cdot n + \mathfrak{S}_1 \cdot D_\Gamma n - \operatorname{div}_\Gamma(\mathfrak{S}_1^T n) + \mathcal{H}(\mathfrak{S}_1^T n \cdot n). \tag{4.12}$$

Demonstração. Aplicando Teorema 2.2 temos

$$dJ(\Omega)(\theta) = \int_{\mathcal{D}} \mathbf{S}_1 : D\theta + \mathbf{S}_0 \cdot \theta dx + \int_{\partial\Omega} \mathfrak{S}_1 \cdot D_\Gamma \theta + \mathfrak{S}_0 \cdot \theta ds = 0, \qquad \forall \theta \in \mathcal{C}_c^1(\Omega \cup (\mathcal{D} \backslash \bar{\Omega}), \mathbb{R}^n).$$

Integrando por partes mostramos (4.9).

Então, quando $\partial\Omega$ é $\mathcal{C}^1$, substituindo (4.9) na expressão da derivada de forma e usando a fórmula de Green obtemos

$$\begin{aligned}
\int_{\mathcal{D}} \mathbf{S}_1 : D\theta &= \int_{\Omega} \mathbf{S}_1 : D\theta + \int_{\mathcal{D}\backslash\overline{\Omega}} \mathbf{S}_1 : D\theta \\
dJ(\Omega)(\theta) &= \int_{\partial\Omega} \mathfrak{S}_1 : D_\Gamma \theta + \mathfrak{S}_0 \cdot \theta ds + \int_{\partial\Omega} [(\mathbf{S}_1^+ - \mathbf{S}_1^-)n] \cdot \theta \, ds \\
&+ \int_{\Omega} (-\operatorname{div}(\mathbf{S}_1^+) + \mathbf{S}_0^+) \cdot \theta dx + \int_{\mathcal{D}\backslash\overline{\Omega}} (-\operatorname{div}(\mathbf{S}_1^-) + \mathbf{S}_0^-) \cdot \theta dx \\
&= \int_{\partial\Omega} [(\mathbf{S}_1^+ - \mathbf{S}_1^-)n] \cdot \theta + \mathfrak{S}_1 : D_\Gamma \theta + \mathfrak{S}_0 \cdot \theta ds, \qquad \forall \theta \in \mathcal{C}_c^1(\mathcal{D}, \mathbb{R}^n).
\end{aligned}$$

Manteremos a mesma notação n para a extensão da normal de uma vizinhança de $\partial\Omega$. Seja $\theta \in \mathcal{C}^1(\overline{\mathcal{D}}, \mathbb{R}^n)$ e defina $\theta_\tau := \theta - (\theta \cdot n)n$ a parte tangencial de θ. Então, $\theta_\tau \cdot n = 0$ em $\partial\Omega$ e consequentemente do Teorema 2.3 temos $dJ(\Omega)(\theta_\tau) = 0$ que nos leva, em vista de (4.13):

$$\begin{aligned}
dJ(\Omega)(\theta) &= dJ(\Omega)((\theta \cdot n)n) \\
&= \int_{\partial\Omega} ((\mathbf{S}_1^+ - \mathbf{S}_1^-)n \cdot n)(\theta \cdot n) + (\mathfrak{S}_0 \cdot n)(\theta \cdot n) ds \\
&+ \int_{\partial\Omega} \mathfrak{S}_1 \cdot D_\Gamma n(\theta \cdot n) + n \cdot \mathfrak{S}_1 \nabla_\Gamma (\theta \cdot n) ds,
\end{aligned}$$

onde usamos (4.6). Finalmente, usando $\mathfrak{S}_1 \in W^{1,1}(\partial\Omega; \mathcal{L}^1(\mathbb{R}^n, \mathbb{R}^n))$ integramos por partes na fronteira $\partial\Omega$ para transformar o último termo em (4.6)

$$\int_{\partial\Omega} n \cdot \mathfrak{S}_1 \nabla_\Gamma (\theta \cdot n) ds = \int_{\partial\Omega} (-\operatorname{div}_\Gamma(\mathfrak{S}_1^T n) + \mathcal{H}(\mathfrak{S}_1^T n \cdot n))(\theta \cdot n) ds.$$

Portanto

$$dJ(\Omega)(\theta) = \int_{\partial\Omega} ((\mathbf{S}_1^+ - \mathbf{S}_1^-)n \cdot n)(\theta \cdot n)) + (\mathfrak{S}_0 \cdot n)(\theta \cdot n) ds \tag{4.13}$$

$$+ \int_{\partial\Omega} \mathfrak{S}_1 \cdot D_\Gamma n(\theta \cdot n) + (-\operatorname{div}_\Gamma(\mathfrak{S}_1^T n) + \mathcal{H}(\mathfrak{S}_1^T n \cdot n))(\theta \cdot n) ds, \tag{4.14}$$

que podemos reescrever como (4.11). □

Observação 4.2. *Na Proposição 4.1, se $\mathfrak{S}_1 \equiv 0$, podemos ainda obter (4.9) quando Ω é apenas Lipschitz.*

Corolário 4.1. *Suponha que as hipóteses da Proposição 4.1 são satisfeitas. Suponha que o tensor $\mathfrak{S}_1 : \partial\Omega \to \mathcal{L}(\mathbb{R}^n, \mathbb{R}^n)$ tem a forma $\mathfrak{S}_1 = \alpha(I - n \otimes n)$, onde $\alpha \in C^0(\partial\Omega)$. Então (4.11) simplifica para*

$$dJ(\Omega)(\theta) = \int_{\partial\Omega} g_1 \theta \cdot n ds, \tag{4.15}$$

onde g_1 é dado por

$$g_1 := [(S_1^+ - S_1^-)n] \cdot n + \mathfrak{S}_0 \cdot n + \alpha\mathcal{H}.$$

Demonstração. Primeiramente $\mathfrak{S}_1^T = \alpha(I - n \otimes n)^T = \mathfrak{S}_1$ e $\mathfrak{S}_1^T n = \alpha(I - n \otimes n)n = \alpha(n - (n \cdot n)n) = 0$, assim os dois últimos termos em (4.12) desaparecem. Em relação ao terceiro termo de (4.12)

escrevemos:

$$\mathfrak{S}_1 \cdot D_\Gamma n = \alpha(I - n \otimes n) \cdot D_\Gamma n = \alpha(\mathrm{tr}(D_{\Gamma n}) - (n \otimes n) \cdot D_\Gamma n) = \alpha(\mathrm{div}_\Gamma\, n - (D_\Gamma nn) \cdot n) = \alpha\mathcal{H}.$$

onde usamos $(D_\Gamma nn) \cdot n = 0$ e $\mathrm{div}_\Gamma\, n = \mathcal{H}$. □

Usando a abordagem do método do adjunto médio, podemos chegar na representação (4.3) da derivada de forma. Então a Proposição 4.1 pode ser usada para obter imediatamente a expressão de fronteira do gradiente de forma correspondente a essa representação tensorial.

Capítulo 5

Derivada de forma para o problema de Robin.

Nessa seção vamos calcular a derivada de forma para um funcional do tipo mínimo quadrado usando o método do adjunto médio. Esse tipo de funcional é bastante usado nas aplicações, particularmente em problemas inversos e de controle ótimo. A ideia é de minimizar a distância na norma L^2 entre um estado desejado e o estado que queremos controlar. Assim, seja $\mathcal{D} \subset \mathbb{R}^n$ um conjunto compacto, $\mathcal{O} = \{\Omega \subset \mathcal{D}, \Omega \text{ aberto e mensurável}\}$, $f \in L^2(\mathcal{D})$ e $z \in H^1(\mathcal{D})$. Considere o seguinte problema de otimização de forma

$$\text{minimizar } J(\Omega) = \frac{1}{2}\int_\Omega (u-z)^2, \tag{5.1}$$

$$\text{sujeito a } \Omega \in \mathcal{O} \text{ e } u \text{ solução de (5.3).} \tag{5.2}$$

Aqui, $u \in H^1(\Omega)$ é a solução do problema de Robin

$$\int_\Omega \nabla u \cdot \nabla v + \alpha \int_{\partial\Omega} vu = \int_\Omega fv, \quad \forall v \in H^1(\Omega). \tag{5.3}$$

Se u for suficientemente regular, ele é solução do problema cuja formulação forte é

$$-\Delta u = f \text{ em } \Omega, \tag{5.4}$$

$$\frac{\partial u}{\partial n} + \alpha u = 0 \text{ em } \partial\Omega. \tag{5.5}$$

Assim, seguindo a linha de raciocínio do Capítulo 2, definimos o Lagrangiano como

$$\mathcal{L}(\Omega, \varphi, \psi) = \int_\Omega \nabla\varphi \cdot \nabla\psi + \alpha \int_{\partial\Omega} \psi\varphi - \int_\Omega f\psi + \frac{1}{2}\int_\Omega (\varphi - z)^2.$$

Com isso temos que

$$\mathcal{L}(\Omega, u, \psi) = J(\Omega), \quad \forall \psi \in H^1(\Omega).$$

Note que $\mathcal{L}_\psi(\Omega, \varphi, \psi)(\hat{\psi}) = \int_\Omega \nabla\varphi \cdot \nabla\hat{\psi} + \alpha \int_{\partial\Omega} \hat{\psi}\varphi - \int_\Omega f\hat{\psi}$, portanto

$$\mathcal{L}_\psi(\Omega, u, \psi)(\hat{\psi}) = 0, \ \forall \hat{\psi} \in H^1(\Omega), \tag{5.6}$$

já que supomos u solução de (5.3). Assim, temos que a condição de otimalidade (5.6) nos fornece a restrição exigida para nosso problema de otimização.

Para obter a equação do adjunto calculamos a derivada parcial com respeito a φ na direção $\hat{\varphi}$ em $(\varphi, \psi) = (u, p)$.

$$\mathcal{L}_\varphi(\Omega, u, p)(\hat{\varphi}) = \int_\Omega \nabla\hat{\varphi} \cdot \nabla p + \alpha \int_{\partial\Omega} \hat{\varphi} p + \int_\Omega (u - z)\hat{\varphi}.$$

Impondo a condição de otimalidade obtemos a equação do adjunto

$$\text{Encontrar } p \in H^1(\Omega): \quad \int_\Omega \nabla\hat{\varphi}\cdot\nabla p + \alpha\int_{\partial\Omega}\hat{\varphi}p = -\int_\Omega (u-z)\hat{\varphi}, \quad \forall \varphi \in H^1$$

Agora para calcular a derivada de forma com respeito a Ω pelo método do adjunto médio, vamos mostrar que as hipóteses do Teorema 3.1 são satisfeitas e assim aplicá-lo. Seguindo as notações já usadas no Capítulo 2, introduzimos $G:[0,\tau]\times E\times F\to\mathbb{R}$ com $E=F=H^1(\Omega)$ e

$$\begin{aligned}
G(t,\varphi,\psi) &:= \mathcal{L}(T_t(\Omega),\varphi\circ T_t^{-1},\psi\circ T_t^{-1})\\
&= \int_{\Omega_t}\nabla(\varphi\circ T_t^{-1})\cdot\nabla(\psi\circ T_t^{-1}) + \alpha\int_{\partial\Omega_t}(\psi\circ T_t^{-1})(\varphi\circ T_t^{-1}) - \int_{\Omega_t} f(\psi\circ T_t^{-1})\\
&\quad + \frac{1}{2}\int_{\Omega_t}(\varphi\circ T_t^{-1}-z)^2.
\end{aligned}$$

Usamos então a seguinte notação mais específica

$$G(t,\varphi,\psi) := a(t,\varphi,\psi) + b(t,\varphi),$$

onde $a:[0,\tau]\times E\times F\to\mathbb{R}$, $b:[0,\tau]\times E\to\mathbb{R}$.

$$\begin{aligned}
a(t,\varphi,\psi) &:= \int_{\Omega_t}\nabla(\varphi\circ T_t^{-1})\cdot\nabla(\psi\circ T_t^{-1}) + \alpha\int_{\partial\Omega_t}(\psi\circ T_t^{-1})(\varphi\circ T_t^{-1}) - \int_{\Omega_t} f(\psi\circ T_t^{-1}),\\
b(t,\varphi) &:= \frac{1}{2}\int_{\Omega_t}(\varphi\circ T_t^{-1}-z)^2.
\end{aligned}$$

Aplicando o Teorema 3.1, temos $dJ(\Omega)(\theta) = \partial_t G(0,u,p)$ onde p é o estado adjunto solução da equação

$$\partial_\varphi G(0,u,p) = 0.$$

Lema 5.1. *Para todo $\phi\in H^1(\Omega)$ temos que*

$$\nabla(\phi\circ T_t^{-1}) = [(DT_t)^{-1}\circ T_t^{-1}]^{\mathsf{T}}\nabla\phi\circ T_t^{-1}.$$

Demonstração. Note que, para um $z\in\mathbb{R}^n$ arbitrário, temos

$$\begin{aligned}
T_t\circ T_t^{-1} = I \quad &\Rightarrow \quad DI(y)z = D(T_t\circ T_t^{-1}(y))z\\
&\Rightarrow \quad z = D(T_t\circ T_t^{-1}(y))z\\
&\Rightarrow \quad z = DT_t(T_t^{-1}(y))(DT_t^{-1}(y)z)\\
&\Rightarrow \quad [DT_t(T_t^{-1}(y))]^{-1}z = DT_t^{-1}(y)z\\
&\Rightarrow \quad [DT_t]^{-1}\circ T_t^{-1} = DT_t^{-1}.
\end{aligned} \tag{5.7}$$

Assim, usando (5.7) obtemos

$$\begin{aligned}
\nabla(\phi\circ T_t^{-1})(y)\cdot h &= \nabla\phi(T_t^{-1}(y))\cdot DT_t^{-1}(y)h\\
&= \nabla\phi(T_t^{-1}(y))\cdot[(DT_t)^{-1}\circ T_t^{-1}(y)]h\\
&= [(DT_t)^{-1}\circ T_t^{-1}(y)]^{\mathsf{T}}\nabla\phi\circ T_t^{-1}(y)\cdot h.
\end{aligned}$$

Portanto,

$$\nabla(\phi\circ T_t^{-1}) = [(DT_t)^{-1}\circ T_t^{-1}]^{\mathsf{T}}\nabla\phi\circ T_t^{-1}.$$

□

Pelo Lema 5.1 temos que

$$\begin{aligned} a(t,\varphi,\psi) &= \int_{\Omega_t} \nabla(\varphi\circ T_t^{-1})\cdot\nabla(\psi\circ T_t^{-1}) + \alpha\int_{\partial\Omega_t}(\psi\circ T_t^{-1})(\varphi\circ T_t^{-1}) - \int_{\Omega_t} f(\psi\circ T_t^{-1}) \\ &= \int_{\Omega_t}[(DT_t)^{-1}\circ T_t^{-1}(y)][(DT_t)^{-1}\circ T_t^{-1}(y)]^{\mathsf{T}}\nabla\varphi\circ T_t^{-1}(y)\cdot\nabla\psi\circ T_t^{-1}(y) \\ &\quad +\alpha\int_{\partial\Omega_t}(\psi\circ T_t^{-1}(y))(\varphi\circ T_t^{-1}(y)) - \int_{\Omega_t} f(\psi\circ T_t^{-1}(y)). \end{aligned}$$

Agora, com a mudança de variável $y = T_t(x)$, temos

$$a(t,\varphi,\psi) = \int_\Omega A(t)\nabla\varphi\cdot\nabla\psi + \alpha\int_{\partial\Omega}(\psi\varphi)h_\Gamma(t) - \int_\Omega (f\circ T_t)\psi h(t), \tag{5.8}$$

$$b(t,\varphi) = \frac{1}{2}\int_\Omega(\varphi - z\circ T_t)^2 h(t), \tag{5.9}$$

sendo $h(t)$ e $h_\Gamma(t)$ os Jacobianos já definidos na Seção 2.3.

Para verificar a Hipótese (3), temos que

$$\begin{aligned} \lim_{t\to 0}\frac{G(t,u^0,p^t)-G(0,u^0,p^t)}{t} &= \lim_{t\to 0}\int_\Omega \nabla u^0\cdot\nabla p^t\frac{A(t)-A(0)}{t} + \frac{1}{2}(u^0 - z\circ T_t)^2\frac{h(t)-h(0)}{t} \\ &\quad - p^t(f\circ T_t)\frac{h(t)-h(0)}{t} + \alpha\int_{\partial\Omega}(u^0p^t)\frac{h_\Gamma(t)-h_\Gamma(0)}{t}. \end{aligned}$$

Pelo Teorema 2.6 sabemos que $h(t)$ é diferenciável e garantimos a existência do limite $\lim_{t\to 0}\frac{h(t)-h(0)}{t}$. Assim, pela diferenciabilidade de $h(t)$, de $(DT_t)^{-1}$ e de seu transposto, os limites

$$\lim_{t\to 0}\frac{A(t)-A(0)}{t} \quad\text{e}\quad \lim_{t\to 0}\frac{h_\Gamma(t)-h_{\Gamma(0)}}{t},$$

também existem. Por último, o seguinte lema garante que $p^t \to p^0$ quando t tende a zero e a Hipótese 3 (ver Seção 3.2) é satisfeita.

Lema 5.2. *Existe uma constante $c > 0$ tal que*

$$\|p^t - p^0\|_{H^1(\Omega)} \le ct, \qquad \forall t\in[0,\tau].$$

Lema 5.3. *Temos $A'(0) = (\operatorname{div}\theta)I - D\theta - D\theta^{\mathsf{T}}$.*

Demonstração. Pela regra da cadeia temos que

$$\begin{aligned} A'(t) &= h'(t)[DT_t^{-1}DT_t^{-\mathsf{T}}] + h(t)\frac{d}{dt}[DT_t^{-1}DT_t^{-\mathsf{T}}] \\ &= h'(t)[DT_t^{-1}DT_t^{-\mathsf{T}}] + h(t)[\frac{d}{dt}DT_t^{-1}DT_t^{-\mathsf{T}} + DT_t^{-1}\frac{d}{dt}DT_t^{-\mathsf{T}}]. \end{aligned}$$

Pelo Lema 2.4 e pela Observação 2.1, temos, para $t = 0$,

$$\begin{aligned} A'(0) &= h'(0)I + h(0)[-D\theta - D\theta^{\mathsf{T}}] \\ &= (\operatorname{div}\theta)I - D\theta - D\theta^{\mathsf{T}}. \end{aligned}$$

□

Aplicando o Teorema 3.1, temos que, usando as notações $u = u^0$ e $p = p^0$

$$\begin{aligned}
dJ(\Omega)(\theta) &= \partial_t G(0,u,p) \\
&= \partial_t \Big(\int_\Omega A(t)\nabla u \cdot \nabla p - h(t)p(f \circ T_t) + \frac{1}{2}(u - z \circ T_t)^2 h(t) + \alpha \int_{\partial\Omega} (up) h_\Gamma(t) \Big)|_{t=0} \\
&= \int_\Omega A'(0)\nabla u \cdot \nabla p - h'(0)p(f \circ T_0) - h(0)p(\nabla f \cdot \theta) \circ T_0 + \frac{1}{2}(u - z \circ T_t)^2 h'(0) \\
&\quad -h(0)(u - z \circ T_0)(\nabla z \cdot \theta) \circ T_0 + \alpha \int_{\partial\Omega} (up) h'_\Gamma(0) \\
&= \int_\Omega ((\operatorname{div}\theta) I - D\theta - D\theta^\mathsf{T})\nabla u \cdot \nabla p + \frac{1}{2}\operatorname{div}\theta (u - z)^2 - \operatorname{div}\theta p f - p\nabla f \cdot \theta - (u - z)(\nabla z \cdot \theta) \\
&\quad +\alpha \int_{\partial\Omega} up(\operatorname{div}\theta - D\theta n \cdot n).
\end{aligned}$$

Organizando $\mathrm{d}J(\Omega)(\theta)$ de forma que satisfaça a Proposição 4.1, temos

$$\begin{aligned}
dJ(\Omega)(\theta) = \int_\Omega & -D\theta\nabla u \nabla p - D\theta^\mathsf{T}\nabla u \nabla p + \operatorname{div}\theta(\nabla u \cdot \nabla p + \frac{1}{2}(u - z)^2 - pf) \\
& -(p\nabla f + (u - z)\nabla z) \cdot \theta + \alpha \int_{\partial\Omega} up \operatorname{div}\theta - upD\theta n \cdot n.
\end{aligned}$$

Utilizando as propriedades do cálculo tensorial, vistas no capítulo anterior, temos seguinte resultado.

Lema 5.4. *Seja $\theta \in C^1(\mathcal{D}, \mathbb{R}^n)$, então*

1) $D\theta : Id = \operatorname{div}\theta$.

2) $D\theta^\mathsf{T}\nabla u \cdot \nabla p = D\theta : (\nabla u \otimes \nabla p)$.

3) $\operatorname{div}\theta - D\theta n \cdot n = (Id - n \otimes n) D_\Gamma \theta$.

4) $D\theta \nabla u \cdot \nabla p = D\theta : (\nabla p \otimes \nabla u)$.

Demonstração. 2) Observe que

$$(D\theta)^\mathsf{T}\nabla u \cdot \nabla p = \nabla u \cdot D\theta \nabla p = (D\theta \nabla p) \cdot \nabla u.$$

Usando (4.4) do Lema 4.1 temos $(D\theta\nabla p) \cdot \nabla u = D\theta : (\nabla u \otimes \nabla p)$.

3) Definimos a derivada tangencial de θ ao longo de Γ como

$$D_\Gamma \theta := D\theta - (D\theta\, n) \otimes n \Rightarrow D\theta = D_\Gamma\theta + (D\theta\, n) \otimes n.$$

Substituindo em (4.7) temos

$$(Id - n \otimes n) \cdot (D_\Gamma\theta + (D\theta\, n) \otimes n) = (Id - n \otimes n) \cdot (D_\Gamma\theta) + (Id - n \otimes n) \cdot ((D\theta\, n) \otimes n).$$

De fato, usando Lema 4.1 e $n \cdot n = 1$ temos

$$\begin{aligned}
(I - n \otimes n) : [(D\theta\, n) \otimes n] &= I : [(D\theta\, n) \otimes n] - (n \otimes n) : [(D\theta\, n) \otimes n] \\
&= n \cdot D\theta\, n - [(n \otimes n) \cdot n] \cdot D\theta\, n \\
&= n \cdot D\theta\, n - n \cdot D\theta\, n \\
&= 0.
\end{aligned}$$

Portanto,

$$\operatorname{div}\theta - D\theta\, n \cdot n = (I - n \otimes n) \cdot (D_\Gamma\theta).$$

4) Basta usar (4.4) do Lema 4.1.

□

Utilizando os resultados acima temos que,

$$\begin{aligned}
dJ(\Omega)(\theta) &= \int_\Omega -D\theta\nabla u\cdot\nabla p-(\nabla u\cdot\nabla p)D\theta(\nabla u\otimes\nabla p)+[\nabla u\cdot\nabla pId+\frac{1}{2}(u-z)^2Id-pfId]:D\theta \\
&\quad -[p\nabla f+(u-z)\nabla z]\cdot\theta+\int_{\partial\Omega}\alpha up(Id-n\otimes n)D_\Gamma\theta \\
&= \int_\Omega D\theta:[-\nabla p\otimes\nabla u-\nabla u\otimes\nabla p+(\nabla u\cdot\nabla p)Id+\frac{1}{2}(u-z)^2Id-pfId] \\
&\quad +[-p\nabla f-(u-z)\cdot\nabla z]\cdot\theta+\int_{\partial\Omega}\alpha up(Id-n\otimes n)D_\Gamma\theta.
\end{aligned}$$

Então, temos que

$$dJ(\Omega)(\theta)=\int_\Omega \mathbf{S}_1:D\theta+\mathbf{S}_0\cdot D\theta+\int_{\partial\Omega}\mathfrak{S}_1:D_\Gamma\theta+\mathfrak{S}_0\cdot\theta,$$

com

$$\begin{aligned}
\mathbf{S}_1 &= -\nabla p\otimes\nabla u-\nabla u\otimes\nabla p+(\nabla u\cdot\nabla p+\frac{1}{2}(u-z)^2-pf)Id, \\
\mathbf{S}_0 &= -p\nabla f-(u-z)\cdot\nabla z, \\
\mathfrak{S}_1 &= \alpha up(Id-n\otimes n), \\
\mathfrak{S}_0 &= 0.
\end{aligned}$$

Usando o Corolário 4.1 temos

$$\begin{aligned}
g_1 &= [-\nabla p\otimes\nabla u-\nabla u\otimes\nabla p+\nabla u\cdot\nabla pId+\frac{1}{2}(u-z)^2Id-pfId]\cdot n+\alpha\mathcal{H} \\
&= [(-\nabla p\otimes\nabla u)n]\cdot n-[(-\nabla u\otimes\nabla p)n]\cdot n+[(-\nabla u\cdot\nabla p)n]\cdot n+\left[\frac{1}{2}(u-z)^2n\right]\cdot n-[(pfId)n]\cdot n+\alpha\mathcal{H} \\
&= -2\partial_n u\partial_n p+\nabla u\cdot\nabla p+\frac{1}{2}(u-z)^2-pf+\alpha\mathcal{H}.
\end{aligned}$$

Utilizando a expressão do gradiente tangencial $\nabla_\Gamma u=\nabla u-(\nabla u\cdot n)n$ temos

$$\nabla u=\nabla_\Gamma u+(\nabla u\cdot n)n,$$

$$\nabla p=\nabla_\Gamma p+(\nabla p\cdot n)n.$$

Assim,

$$\begin{aligned}
g_1 &= -2\partial_n u\partial_n p+(\nabla_\Gamma u+\partial_n un)\cdot(\nabla_\Gamma p+\partial_n pn)+\frac{1}{2}(u-z)^2-pf+\alpha\mathcal{H} \\
&= -\partial_n u\partial_n p+\nabla_\Gamma u\cdot\nabla_\Gamma p+\partial_n p(\nabla_\Gamma u\cdot n)+\frac{1}{2}(u-z)^2-pf+\alpha\mathcal{H}.
\end{aligned}$$

Capítulo 6

Otimização de forma para elasticidade linear com dois materias

6.1 Abordagem usando o material ersatz

Usa-se a abordagem do material ersatz, que é comumente usada em otimização de estruturas, ver [1, 21]. É conveniente usar nessas circunstâncias, pois permite trabalhar em um domínio fixo $\mathcal{D}$ invés de um domínio variável Ω, mas também pode criar problemas de instabilidade como apontado em [4]. A ideia do método do material ersatz é que um domínio fixo Ω é preenchido com dois materiais homogêneos com diferentes tensores de elasticidade Hooke A_0 e A_1 definidos por

$$A_i\xi = 2\mu_i\xi + \lambda_i(\mathrm{Tr}\,\xi)Id, \;\; i = 0, 1,$$

com módulos de Lamé λ_i e μ_i para $i = 0, 1$, Id é a matriz identidade e ξ é uma matriz. O primeiro material se encontra no subconjunto aberto Ω de $\mathcal{D}$ e o outro material preenche o complemento. Assim a lei de Hooke é escrita em $\mathcal{D}$ como

$$A_\Omega = A_0\chi_\Omega + \epsilon A_0\chi_{\mathcal{D}\backslash\Omega}, \tag{6.1}$$

onde χ_Ω denota a função característica de Ω, e ϵ é um parâmetro pequeno dado. Assim, a região $\mathcal{D}\backslash\Omega$ representa uma "fase fraca"enquanto que Ω é a "fase forte". O otimização ainda é em relação ao conjunto variável Ω, mas aqui Ω está incorporado no conjunto fixo maior $\mathcal{D}$. Note que calculamos a derivada de forma para a equação diferencial parcial (EDP) incluindo o material ersatz, diferente do que usualmente é feito na literatura.

Seja $\Omega \subset \mathcal{D} \subset \mathbb{R}^m$, $m = 2, 3$, onde $\mathcal{D}$ é um domínio fixo cuja fronteira $\partial\mathcal{D}$ está particionada em quatro subconjuntos Γ_d, Γ_n, Γ_s e Γ. Uma condição de fronteira de Dirichlet (respectivamente Neumann) é imposta em Γ_d (respectivamente Γ). Em Γ_n, uma condição de Neumann não homogênea é imposta, que representa uma carga superfície dada $g \in H^{-\frac{1}{2}}(\Gamma_n)^m$. A interface livre entre as fases fraca e forte é $\partial\Omega$. Uma mola com rigidez k_s é acoplada na fronteira Γ_s, e é modelada com uma condição de fronteira de Robin; essa condição é usada para mecanismos. Seja $H^1_d(\mathcal{D})^m$ o espaço dos campos de vetores em $H^1(\mathcal{D})^m$ que satisfazem as condições homogêneas de Dirichlet em Γ_d. Define-se um domínio parametrizado $\Omega_t = T^\theta_t(\Omega)$ como em (2.19), e assumimos que $T^\theta_t = I$ em $\Gamma_d \bigcup \Gamma_n \bigcup \Gamma_s$ onde I é a identidade.

Na abordagem usando o material ersatz, o campo de deslocamento $u \in H^1_d(\mathcal{D})^m$ é a solução do sistema elástico linearizado

$$
\begin{aligned}
-\operatorname{div} A_\Omega e(u) &= 0 \text{ em } \mathcal{D}, && (6.2)\\
u &= 0 \text{ em } \Gamma_d, && (6.3)\\
A_\Omega e(u) &= g \text{ em } \Gamma_n, && (6.4)\\
A_\Omega e(u) &= 0 \text{ em } \Gamma, && (6.5)\\
A_\Omega e(u) &= -k_s u \text{ em } \Gamma_s, && (6.6)
\end{aligned}
$$

onde o gradiente simetrizado é $e(u) := \frac{Du+Du^\mathsf{T}}{2}$, e Du^T denota o transposto de Du. Considere o seguinte funcional

$$
J(\Omega) := c_1 \int_{\mathcal{D}} A_\chi e(u(x)) : e(u(x)) + c_2 \int_{\mathcal{D}} F_{\mathcal{D}}(x, u(x)) + c_3 \int_{\Gamma_m} F_\Gamma(x, u(x)). \tag{6.7}
$$

Assumimos que $F_{\mathcal{D}}$ e F_Γ são funções suaves de seus argumentos. O conjunto Γ_m é um subconjunto da fronteira $\partial\Omega$. Essa função genérica abrange vários casos importantes como compliance e alguns funcionais usados para mecanismo flexível. O caso $(c_1, c_2, c_3) = (1, 0, 0)$ corresponde à compliance e o caso de mecanismo flexível pode ser feito por uma escolha apropriada de $F_{\mathcal{D}}$ e F_Γ, o que será discutido mais adiante.

Denotamos por u_t a solução de (6.2)-(6.6) com Ω_t no lugar de Ω. Definido $u^t := u_t \circ T_t$ e usando a regra da cadeia temos que

$$
\begin{aligned}
Du^t = D(u_t \circ T_t^{-1} \circ T_t) = [(Du_t) \circ T_t] DT_t \quad &\Rightarrow \quad [(Du_t) \circ T_t] = Du^t (DT_t)^{-1} && (6.8)\\
&\Rightarrow \quad [(Du_t)^\mathsf{T} \circ T_t] = (DT_t)^{-\mathsf{T}} (Du^t)^\mathsf{T}. && (6.9)
\end{aligned}
$$

De (6.8) e (6.9) obtemos

$$
\begin{aligned}
e(u^t \circ T_t^{-1}) \circ T_t &= e(u_t) \circ T_t\\
&= \frac{Du_t \circ T_t + (Du_t)^\mathsf{T} \circ T_t}{2}\\
&= \frac{Du^t (DT_t)^{-1} + (DT_t)^{-\mathsf{T}} (Du^t)^\mathsf{T}}{2}.
\end{aligned}
$$

Assim, podemos introduzir a seguinte notação

$$
E(t, u^t) := e(u^t \circ T_t^{-1}) \circ T_t = \frac{Du^t (DT_t)^{-1} + (DT_t)^{-\mathsf{T}} (Du^t)^\mathsf{T}}{2}. \tag{6.10}
$$

A formulação variacional da EDP é: encontrar $u_t \in H_d^1(\mathcal{D})^m$ tal que

$$
\int_{\mathcal{D}} A_{\Omega_t} e(u_t) : e(v_t) - \int_{\Gamma_s} k_s u_t \cdot v_t = \int_{\Gamma_n} g \cdot v_t, \tag{6.11}
$$

para todo $v_t \in H_d^1(\mathcal{D})^m$. Continuamos com a mudança de variável $x \mapsto T_t(x)$ em (6.11), que nos leva a

$$
\int_{\mathcal{D}} [A_{\Omega_t} e(u_t)] \circ T_t : e(v_t) \circ T_t \xi(t) - \int_{\Gamma_s} k_s u_t \cdot v_t = \int_{\Gamma_n} g \cdot v_t, \quad \forall v_t \in H_d^1(\mathcal{D})^m, \tag{6.12}
$$

onde $h(t) = |\det DT_t|$ é o Jacobiano da transformação $x \mapsto T_t(x)$. Note que o Jacobiano em T_t aparece nas integrais em Γ_n e Γ_s, já que assumimos que $T_t = I$ em $\partial\mathcal{D}$. Em vista de (**??**), podemos escrever (6.12) como

$$
\int_{\mathcal{D}} A_\Omega E(t, u^t) : E(t, v) \xi(t) - \int_{\Gamma_s} k_s u^t \cdot v = \int_{\Gamma_n} g \cdot v, \tag{6.13}
$$

para todo $v \in H_d^1(\mathcal{D})^m$. De modo similar, temos

$$J(\Omega_t) = c_1 \int_{\mathcal{D}} A_{\Omega_t} e(u_t) : e(u_t) + c_2 \int_{\mathcal{D}} F_{\mathcal{D}}(x, u_t(x)) + c_3 \int_{\Gamma_s} F_\Gamma(x, u_t(x)),$$

e usando a mudança de variável $x \mapsto T_t(x)$ obtemos

$$\begin{aligned} J(\Omega_t) =& c_1 \int_{\mathcal{D}} A_\Omega E(t, u^t) : E(t, u^t) h(t) + c_2 \int_{\mathcal{D}} F_{\mathcal{D}}(T_t(x), u^t(x)) h(t)(x) \\ &+ c_3 \int_{\Gamma_s} F_\Gamma(x, u^t(x))\, ds_x \end{aligned} \tag{6.14}$$

6.2 Derivada de forma usando o Lagrangiano

Para calcular a derivada de forma de $J(\Omega)$ usamos o método do adjunto médio. Escrevendo A no lugar de A_Ω, por simplicidade, temos

$$\begin{aligned} G(t, \varphi, \psi) :=& c_1 \int_{\mathcal{D}} AE(t, \varphi) : E(t, \varphi) h(t) + c_2 \int_{\mathcal{D}} F_{\mathcal{D}}(T_t(x), \varphi(x)) h(t)(x) \\ &+ c_3 \int_{\Gamma_s} F_\Gamma(x, \varphi(x)) + \int_\Omega AE(t, \varphi) : E(t, \psi) h(t) - \int_{\Gamma_s} k_s \varphi \cdot \psi - \int_{\Gamma_n} g \cdot \psi. \end{aligned}$$

onde $\varphi, \psi \in H^1_d(\mathcal{D})^m$. Tendo em vista (6.13) e (6.14), temos $J(\Omega_t) = G(t, u^t, \psi)$ para todo $\psi \in H^1_d(\mathcal{D})^m$. Assim, a derivada de forma pode ser calculada como

$$dJ(\Omega)(\theta) = \frac{d}{dt}(G(t, u^t, \psi))|_{t=0}.$$

O adjunto é dado como solução da seguinte primeira condição de otimalidade

$$\partial_\varphi G(0, u, p)(\hat{\varphi}) = 0, \quad \forall \hat{\varphi} \in H^1_d(\mathcal{D})^m,$$

que nos leva, usando $A = A^\mathsf{T}$, a

$$\begin{aligned} &2c_1 \int_\Omega AE(0, u) : E(0, \hat{\varphi}) + c_2 \int_{\mathcal{D}} \partial_u F_{\mathcal{D}}(T_t(x), u(x)) \cdot \hat{\varphi} + c_3 \int_{\Gamma_s} \partial_u F_\Gamma(x, u(x)) \cdot \hat{\varphi} \\ &- \int_{\Gamma_s} k_s \hat{\varphi} \cdot p + \int_\Omega AE(0, \hat{\varphi}) : E(0, p) = 0, \quad \forall \hat{\varphi} \in H^1_d(\mathcal{D})^m. \end{aligned}$$

Como $E(0, v) = e(v)$ para todo $v \in H^1_d(\mathcal{D})^m$, e $A = A^\mathsf{T}$, obtemos a equação do adjunto

$$\begin{aligned} \int_\Omega Ae(p) : e(\hat{\varphi}) - \int_{\Gamma_s} k_s p \cdot \hat{\varphi} =& -2c_1 \int_\Omega Ae(u) : e(\hat{\varphi}) - c_2 \int_{\mathcal{D}} \partial_u F_{\mathcal{D}}(T_t(x), u(x)) \hat{\varphi} \\ &- c_3 \int_{\Gamma_s} \partial_u F_\Gamma(x, u(x)) \cdot \hat{\varphi} \end{aligned}$$

para todo $\hat{\varphi} \in H^1_d(\mathcal{D})^m$. Dando seguimento obtemos

$$\begin{aligned} dJ(\Omega)(\theta) =& \partial_t G(0, u, p) = c_1 \int_\Omega A \partial_t E(0, u) : E(0, u) \\ &+ c_1 \int_\Omega AE(0, u) : \partial_t E(0, u) + AE(0, u) : E(0, u) \operatorname{div} \theta \\ &+ c_2 \int_{\mathcal{D}} \partial_x F_{\mathcal{D}}(x, u(x)) \cdot \theta(x) + F_{\mathcal{D}}(x, u(x)) \operatorname{div} \theta + \int_\Omega A \partial_t E(0, u) : E(0, p) \\ &+ \int_\Omega AE(0, u) : \partial_t E(0, p) + AE(0, u) : E(0, p) \operatorname{div} \theta \end{aligned}$$

Note que $E(0,v) = e(v)$ para todo $\hat{\varphi} \in H^1_d(\mathcal{D})^m$, então

$$\partial_t E(0,v) = \frac{-DvD\theta - D\theta^\mathsf{T} Dv^\mathsf{T}}{2}.$$

Usando a propriedade $A = A^\mathsf{T}$, obtemos

$$\begin{aligned} dJ(\Omega)(\theta) = &- \int_\Omega \frac{1}{2} Du^\mathsf{T}(Ae(p) + (Ae(p))^\mathsf{T}) : D\theta - \int_\Omega \frac{1}{2} Dp^\mathsf{T}(Ae(u) + (Ae(u))^\mathsf{T}) : D\theta \\ &+ \int_\Omega (Ae(u) : e(p) + c_1 Ae(u) : e(u)) \operatorname{div}\theta - c_1 \int_\Omega (Du^\mathsf{T} Ae(u) + Du^\mathsf{T}(Ae(u))^\mathsf{T}) : D\theta \\ &+ c_2 \int_\mathcal{D} \partial_x F_\mathcal{D}(x, u(x)) \cdot \theta(x) + F_\mathcal{D}(x, u(x)) \operatorname{div}\theta. \end{aligned}$$

Usando que $(Ae(v))^\mathsf{T} = Ae(v)$, temos

$$dJ^{\mathrm{vol}}(\Omega)(\theta) = \int_\mathcal{D} \mathbf{S}_1 : D\theta + \mathbf{S}_0 \cdot \theta, \tag{6.15}$$

com

$$\mathbf{S}_1 = -Du^\mathsf{T} A_\Omega e(p) - Dp^\mathsf{T} A_\Omega e(u) + (A_\Omega e(u) : e(p) + c_1 A_\Omega e(u) : e(u) + c_2 F_\mathcal{D}(\cdot, u)) Id, \tag{6.16}$$

$$\mathbf{S}_0 = \partial_x F_\mathcal{D}(\cdot, u). \tag{6.17}$$

Fórmula (6.15) é conveniente para implementação numérica usando FEniCS.

6.3 Compliance

O caso da compliance é obtido com $(c_1, c_2, c_3) = (1, 0, 0)$. Que nos leva a $\mathbf{S}_0 \equiv 0$, $p = -2u$ e

$$\mathbf{S}_1 = 2Du^\mathsf{T} A_\Omega e(u) - A_\Omega e(u) : e(u) Id. \tag{6.18}$$

6.4 Múltiplas cargas

No caso da compliance para múltiplas cargas e compliance, temos um conjunto de forças $\{g_i\}_{i \in I}$ e a compliance é a soma das compliances associadas a cada força g_i:

$$J(\Omega) := \sum_{i \in I} \int_{\Gamma_n} g_i \cdot u_i,$$

onde u_i é a solução da elasticidade linearizada correspondente a g_i. Assim, a derivada de forma nesse caso é

$$dJ^{\mathrm{vol}}(\Omega)(\theta) = \int_\Omega \mathbf{S}_1 : D\theta, \tag{6.19}$$

com $\mathbf{S}_1 := \sum_{i \in I} 2Du_i^\mathsf{T} A_\Omega e(u_i) - A_\Omega e(u_i) : e(u_i) Id$.

6.5 Comparação: derivadas de forma com e sem material ersatz

Usualmente, a expressão de fronteira da derivada de forma é usada nos métodos level set e calculada para o problema sem o material ersatz, no entanto, o sistema elástico é resolvido usando o material ersatz na parte numérica. Essa pequena incompatibilidade é justificada pelo fato do tensor do material ersatz ter pequena amplitude. A razão de se tolerar essa incompatibilidade na parte numérica, provavelmente se deve ao fato de a expressão de fronteira da derivada de forma

naquele caso ser impraticável lidar numericamente, já que requer o cálculo do pulo do gradiente ao longo da interface em movimento entre a fase forte e fraca; ver (6.29).

De qualquer maneira, é mais preciso usar uma apropriada derivada de forma que corresponde ao material ersatz na implementação numérica, em ordem de se evitar incompatibilidade. Outra vantagem em usar a fórmula exata para a abordagem ersatz é que tal fórmula é válida para qualquer valor de ϵ e não somente para um ϵ suficientemente pequeno. Mostramos nessa seção que calcular e implementar a fórmula da derivada de forma de distribuição não é mais difícil para a abordagem ersatz.

Primeiro, comparamos a derivada de forma distribuída com ou sem material ersatz. Considere o sistema elástico seguinte para o caso sem material ersatz:

$$\begin{aligned} -\operatorname{div} Ae(u) &= 0 \text{ em } \Omega, && (6.20)\\ u &= 0 \text{ em } \Gamma_d, && (6.21)\\ Ae(u)n &= g \text{ em } \Gamma_n, && (6.22)\\ Ae(u)n &= 0 \text{ em } \Gamma, && (6.23)\\ Ae(u)n &= -k_s u \text{ em } \Gamma_s. && (6.24) \end{aligned}$$

Também assumimos aqui que Γ_d, Γ_n e Γ_s são fixos, mas nesse caso $\Gamma = \partial\Omega\backslash(\Gamma_d \cup \Gamma_n \cup \Gamma_s)$ é a fronteira livre. O tensor de elasticidade de Hooke A satisfaz $A\xi = 2\mu\xi + \lambda \operatorname{tr}(\xi) Id$, onde $\xi \in \mathbb{R}^{m\times m}$, μ, λ são os parâmetros de Lamé. A formulação variacional de (6.24) consiste em encontrar $u \in H^1_d(\Omega)^m$ tal que

$$\int_\Omega Ae(u) : e(v) - \int_{\Gamma_s} k_s u \cdot v = \int_{\Gamma_n} g \cdot v, \quad \forall v \in H^1_d(\Omega)^m.$$

O funcional de custo nesse caso é

$$J_0(\Omega_t) := c_1 \int_\Omega Ae(u) : e(u) + c_2 \int_\Omega F_{\mathcal{D}}(x, u(x))\, dx + c_3 \int_{\Gamma_s} F_\Gamma(x, u(x))\, ds_x.$$

Um cálculo similar ao encontrado na seção 6.2 nos leva a

$$dJ_0^{\mathrm{vol}}(\Omega)(\theta) = \int_\Omega \mathbf{S}_1 : D\theta + \mathbf{S}_0 \cdot \theta$$

com

$$\begin{aligned} \mathbf{S}_1 &= -Du^\mathsf{T} Ae(p) - Dp^\mathsf{T} Ae(u) + (Ae(u) : e(p) + c_1 Ae(u) : e(u) + c_2 F_{\mathcal{D}}(x,u)) Id, && (6.25)\\ \mathbf{S}_0 &= \partial F_{\mathcal{D}}(x,u). && (6.26) \end{aligned}$$

No caso $(c_1, c_2, c_3) = (1, 0, 0)$, que corresponde ao caso da compliance, temos $\mathbf{S}_0 \equiv 0$ e $p = -2u$ o que nos leva a

$$\mathbf{S}_1 = 2Du^\mathsf{T} Ae(u) - Ae(u) : e(u) Id, \text{ em } \Omega. \tag{6.27}$$

Note que (6.27) é semelhante a (6.18), a principal diferença é que (6.18) é definida em $\mathcal{D}$ e (6.27) é definida em Ω. Assim, do ponto de vista numérico, (6.18) não é mais difícil de calcular que (6.27).

Agora comparamos a expressão de fronteira da derivada de forma com ou sem material ersatz no caso da compliance. Suponha que Ω seja $\mathcal{C}^2$ e usando (6.27) e (4.11) da Proposição 4.1, obtemos a expressão de fronteira da derivada de forma

$$dJ_0^{\mathrm{surf}}(\Omega)(\theta) = \int_{\partial\Omega} (\mathbf{S}_1 n \cdot n)\theta \cdot n = \int_\Gamma (\mathbf{S}_1 n \cdot n)\theta \cdot n,$$

desde que $\partial\Omega\backslash\Gamma$ seja fixo. Então calculamos

$$\begin{aligned} \mathbf{S}_1 n \cdot n &= 2Du^\mathsf{T} Ae(u) n \cdot n - Ae(u) : e(u)\\ &= 2Ae(u) n \cdot n \cdot Dun - Ae(u) : e(u). \end{aligned}$$

Em Γ, temos que $Ae(u)n = 0$ o que nos leva a

$$dJ_0^{\text{surf}}(\Omega)(\theta) = \int_\Gamma -Ae(u) : e(u)\theta \cdot n, \tag{6.28}$$

que é um caso particular da fórmula em [1, Teorema 7].

No caso do material ersatz, aplicando a Proposição 4.1 e supondo $\theta = 0$ em $\partial\mathcal{D}$ e $\partial\Omega$ de classe C^2, obtemos a expressão de fronteira

$$dJ^{\text{surf}}(\Omega)(\theta) = \int_{\partial\Omega} [(\mathbf{S}_1^+ - \mathbf{S}_1^-)n] \cdot n\theta \cdot n,$$

com

$$[(\mathbf{S}_1^+ - \mathbf{S}_1^-)n] \cdot n = -[\![Ae(u) : e(u)]\!] + 2A_0 e(u)^+ n \cdot Du^+ n - 2\epsilon A_0 e(u)^- n \cdot Du^- n,$$

onde expoentes $(\cdot)^+$ e $(\cdot)^-$ denotam restrições de Ω e $\mathcal{D}\backslash\Omega$, respectivamente. Também

$$[\![v]\!] := \gamma_\Omega(v) - \gamma_{\mathcal{D}\backslash\Omega}(v)$$

denota o salto de uma função v sobre a interface $\partial\Omega$; aqui $\gamma_\Omega(v)$ é o traço de $v_{|\Omega}$ em $\partial\Omega$. Usando a condição de transmissão $A_0 e(u)^+ n = \epsilon A_0 e(u)^- n$ em $\partial\Omega$, obtemos

$$dJ^{\text{surf}}(\Omega)(\theta) = \int_{\partial\Omega} (2\epsilon A_0 e(u^-)n \cdot [\![Du]\!]n)\theta \cdot n - \int_{\partial\Omega} [\![Ae(u) : e(u)]\!]\theta \cdot n. \tag{6.29}$$

As duas principais diferenças entre (6.29) e (6.28) são o termo de pequena perturbação

$$2\epsilon A_0 e(u^-)n \cdot [\![Du]\!]n$$

e o fato que $[\![Ae(u) : e(u)]\!]$ ser um salto sobre a interface $\partial\Omega$.

6.6 Direção de descida máxima

Nesse trabalho estamos interessados em problemas de otimização de forma do tipo

$$\min_{\Omega\epsilon\mathcal{B}} J(\Omega),$$

onde $\mathcal{B} \subset \mathcal{P}$ é algum conjunto admissível de formas. Resolvido o problema do cálculo da derivada, utilizamos um método de busca linear adaptado ao contexto de funcionais de forma. Assim, assuma que $J : \mathcal{B} \to \mathbb{R}$ seja diferenciável com respeito a forma $\Omega \subset \mathcal{D} \subset \mathbb{R}^n$.

Definição 6.1. *O campo de vetor $\theta \in C_c^{0,1}(\mathcal{D}, \mathbb{R}^n)$ é chamado de direção de descida para J em Ω se existir um ϵ tal que*

$$J(\Phi_t^\theta(\Omega)) < J(\Omega) \quad \forall t \in (0, \epsilon).$$

Se a derivada de forma de J em Ω na direção θ existir e for uma direção de descida, então por definição

$$dJ(\Omega)(\theta) < 0.$$

Para nosso método de gradiente, estamos procurando uma direção de descida θ. Quando $dJ(\Omega;\theta)$ é escrito usando a expressão da fronteira

$$dJ^{\text{surf}}(\Omega)(\theta) = \int_{\partial\Omega} G(\Omega)\theta \cdot n,$$

então uma escolha pode ser tomar $\theta = -G(\Omega) \cdot n$. Contudo, essa escolha assume que $G(\Omega)$ e $\partial\Omega$ são um tanto regulares e, na prática, isso pode nos levar a θ com uma regularidade baixa e um comportamento instável do algoritmo. Uma escolha melhor é encontrar uma direção de descida mais

suave encontrando $\theta \in \mathbb{H}(\partial\Omega)$ tal que

$$\mathcal{B}(\theta, \xi) = -dJ^{\text{surf}}(\Omega)(\xi) \quad \forall \xi \in \mathbb{H}(\partial\Omega), \tag{6.30}$$

onde $\mathbb{H}(\partial\Omega)$ é um espaço de Sobolev apropriado de campos de vetor em $\partial\Omega$ e $\mathcal{B} : \mathbb{H}(\partial\Omega) \times \mathbb{H}(\partial\Omega) \to \mathbb{R}$, $k \geq 1$, é uma forma bilinear definida positiva em $\partial\Omega$.

No nosso caso usamos a expressão distribuída (6.15) da derivada de forma, desta forma usamos a forma bilinear definida positiva $\mathcal{B} : \mathbb{H}(\mathcal{D}) \times \mathbb{H}(\mathcal{D}) \to \mathbb{R}$, onde $\mathbb{H}(\mathcal{D})$ é um espaço de Sobolev apropriado de campos de vetor em $\mathcal{D}$). Assim, nosso problema é encontrar $\theta \in \mathbb{H}(\mathcal{D})$ tal que

$$\mathcal{B}(\theta, \xi) = -dJ^{vol}(\Omega)(\xi) \quad \forall \xi \in \mathbb{H}(\mathcal{D}). \tag{6.31}$$

Com essa escolha, a solução θ de (6.31) é definida em todo $\mathcal{D}$ e é uma direção de descida já que $dJ(\Omega)(\theta) = -\mathcal{B}(\theta, \theta) < 0$ se $\theta \neq 0$.

Também é possível combinar as duas abordagens substituindo $dJ^{vol}(\Omega)(\theta)$ por $dJ^{surf}(\Omega)(\theta)$ em (6.31). Isso foi feito em [5] onde um melhoramento da convergência do método level set foi observado. Em nosso algoritmo, escolhemos $\mathbb{H}(\mathcal{D}) = H^1(\mathcal{D})^m$ e

$$\mathcal{B}(\theta, \xi) = \int_{\mathcal{D}} \alpha_1 D\theta : D\xi + \alpha_2 \theta \cdot \xi, \tag{6.32}$$

com $\alpha_1 = 1$ e $\alpha_2 = 0.1$, com condição de fronteira $\theta \cdot n = 0$ em $\partial\mathcal{D}$.

Capítulo 7

Método level set

Para representar a evolução geométrica numericamente, escolhemos o método level set devido sua facilidade em lidar com mudanças topológicas. Métodos usando splines, por exemplo, não conseguem lidar tão bem apesar de ser mais preciso na representação da geometria.

Métodos level set foram introduzidos por Osher e Sethian [15] para facilitar a modelagem da propagação de interfaces quando se deparava com mudanças topológicas durante a evolução de uma curva, como surgimento de quinas acentuadas, fusão e divisão de formas. A principal ideia de um método level set é representar implicitamente a fronteira de um domínio em movimento $\Omega_t \subset \mathcal{D} \in \mathbb{R}^n$ como um conjunto de nível zero de uma função contínua $\phi(\cdot, t) : \mathcal{D} \to \mathbb{R}$. Por exemplo, seja $\Omega = \{|(x, y)| \leq 1\}$ o disco unitário em $\mathbb{R}^2$, sabemos que sua fronteira é o círculo unitário. Mas pode ser dada de forma implícita como o conjunto de nível zero da função $\phi(x, y) = x^2 + y^2 - 1 = 0$, ou seja, $\partial\Omega = \{(x, y) \in \mathbb{R}^2 \mid f(x, y) = 0\}$.

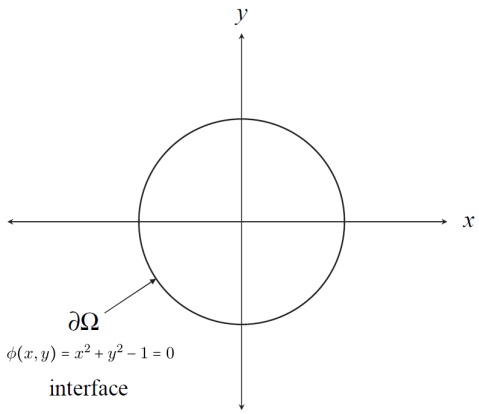


Figura 7.1: *Representação implícita da curva* $x^2 + y^2 = 1$

7.1 Introdução

Considere a família de domínios $\Omega_t \subset \mathcal{D}$ definidos como

$$\Omega_t := \{x \in \mathcal{D} \mid \phi(x, t) < 0\}, \tag{7.1}$$

onde $\phi : \mathcal{D} \times \mathbb{R}^+ \to \mathbb{R}$ é alguma função contínua. Chamamos tal função de *função level set*, pois se assumirmos $|\nabla\phi(\cdot, t)| \neq 0$ para qualquer $t \in [0, T]$ no conjunto $\{x \in \mathcal{D} \mid \phi(x, t) = 0\}$ então temos

$$\partial\Omega_t = \{x \in \mathcal{D} \mid \phi(x, t) = 0\}, \tag{7.2}$$

ou seja, a fronteira $\partial\Omega_t$ é o conjunto de nível zero da função $\phi(\cdot,t)$. Assim, se explicitarmos uma função level set podemos reconstruir o domínio correspondente para cada Ω_t.

Agora vamos supor que Ω_t pode ser representado como $\Omega_t = T_t^{\Omega}$, como em (2.19). Para tanto, seja $x(t)$ um ponto em movimento da fronteira $\partial\Omega_t$, com velocidade $\dot{x}(t) = \theta(x(t))$ de acordo com (2.6). Diferenciando a relação $\phi(x(t),t) = 0$ com respeito a t, obtemos a equação de Hamilton-Jacobi:

$$\partial_t\phi(x(t),t) + \theta(x(t)) \cdot \nabla\phi(x(t),t) = 0, \text{ em } \partial\Omega_t \times \mathbb{R}^+, \tag{7.3}$$

que então é estendida para todo $\mathcal{D}$ via seguinte equação

$$\partial_t\phi(x,t) + \theta(x) \cdot \nabla\phi(x,t) = 0, \text{ em } \mathcal{D} \times \mathbb{R}^+, \tag{7.4}$$

ou alternativamente para $U \times \mathbb{R}^+$, onde U é uma vizinhança de $\partial\Omega_t$.

Originalmente, o método level set foi projetado para rastrear interfaces suaves em movimento ao longo da direção normal a fronteira. Teoricamente, isso é sustentado pelo Teorema 2.3: se o domínio Ω_t e o gradiente de forma forem suficientemente suaves então a derivada de forma somente depende de $\theta \cdot n$ em $\partial\Omega_t$. Nesse caso, podemos escolher um campo vetorial $\theta = \vartheta_n n$ para a otimização e notando que uma extensão de $\mathcal{D}$ do vetor normal unitário n apontando para fora de Ω_t é dado por $n = \frac{\nabla\phi}{|\nabla\phi|}$, obtemos de (7.4) a *equação level set*:

$$\partial_t\phi + \vartheta_n|\nabla\phi| = 0, \text{ em } \mathcal{D} \times \mathbb{R}^+. \tag{7.5}$$

Assim, para encontrar a função level set, temos que ter de antemão uma condição inicial $\phi(x,0) = \phi_0(x)$ e um campo vetorial θ para então obtê-la por meio da solução da equação diferencial

$$\partial_t\phi + \vartheta_n|\nabla\phi| = 0, \text{ em } \mathcal{D} \times \mathbb{R}^+, \tag{7.6}$$

$$\phi(x,0) = \phi_0(x). \tag{7.7}$$

A condição inicial acompanhando a equação Hamilton-Jacobi acima pode ser escolhida como sendo a função distância sinalizada para a fronteira inicial $\partial\Omega_0$ a fim de satisfazer a condição $|\nabla u| \neq 0$ em $\partial\Omega$, i.e.

$$\phi_0(x) = \begin{cases} d(x,\partial\Omega_0), & \text{se} \quad x \in \Omega_0^c, \\ -d(x,\partial\Omega_0), & \text{se} \quad x \in \Omega_0. \end{cases}$$

Com a escolha de tal condição inicial tem-se $|\nabla\phi| = 1$, o que garante um bom comportamento no processo de representação implícita de interfaces, já que seu gradiente não assume valores extremos.

7.2 Método level set e derivada de forma distribuída

No caso da derivada de forma distribuída, nós não estendemos ϑ_n para $\mathcal{D}$, ao invés obtemos diretamente a direção de descida θ definida em $\mathcal{D}$ resolvendo (6.31), onde $dJ^{\text{vol}}(\Omega)(\theta)$ é dado por (6.15). Assim, diferente do método level set usual, θ não é necessariamente normal a $\partial\Omega_t$ e ϕ não é regido por (7.5), mas pela equação (7.4).

Em otimização de forma, ϑ_n geralmente depende da solução de uma ou várias EDPs e de suas derivadas. Como a fronteira $\partial\Omega_t$, de modo geral, não coincide com os nós da malha onde ϕ e as soluções das equações diferenciais parciais estão definidas na aplicação numérica, o calculo e a extensão de ϑ_n pode exigir interpolação das funções em $\partial\Omega_t$ definidas apenas nos pontos da malha, complicando a implementação numérica e introduzindo erros de interpolação. Essa é uma questão particular de problemas de interfaces, tais como os problemas de elasticidade com material ersatz, onde ϑ_n é o salto de uma função através da interface, como em (6.29), a qual requer várias interpolações e propenso a erros. Já na derivada de forma distribuída, θ só precisa estar bem definida nos pontos da malha.

7.3 Discretização da equação de Hamilton-Jacobi

Seja $\mathcal{D}$ o quadrado unitário $\mathcal{D} = (0,1)\times(0,1)$. Para a discretização da equação Hamilton-Jacobi (7.4), primeiro definimos a malha correspondente a $\mathcal{D}$. Introduzimos os nós P_{ij} cujas coordenadas são dadas por $(i\Delta x, j\Delta y)$, $1 \le i,j \le N$ onde Δx e Δy são os passos da discretização nas direções x e y respectivamente. Denote também $t^k = k\Delta t$ o tempo discreto para $k \in \mathbb{N}$, onde Δt é o passo do tempo. Estamos procurando por uma aproximação $\phi_{ij}^k \approx \phi(P_{ij}, t^k)$.

No método level set usual, a equação (7.5) é discretizada usando o esquema upwind explícito proposto por Osher e Sethian [14, 15, 18]. Esse esquema é apropriado para resolver (7.5), mas não é adequado para discretizar (7.4) que precisamos para nossa aplicação. A equação (7.4) é da forma

$$\partial_t \phi + H(\nabla\phi) = 0, \text{ em } \mathcal{D}\times\mathbb{R}^+, \tag{7.8}$$

onde $H(\nabla\phi) := \theta\cdot\nabla\phi$ é o chamado Hamiltoniano. Foi usado o fluxo de Lax-Friedrichs, ver [16], que se escreve para nosso caso:

$$\hat{H}^{LF}(p^-,p^+,q^-,q^+) = H\left(\frac{p^-+p^+}{2}, \frac{q^-+q^+}{2}\right) - \frac{1}{2}(p^+-p^-)\alpha^x - \frac{1}{2}(p^+-p^-)\alpha^y. \tag{7.9}$$

onde $\alpha^x = |\theta_x|$, $\alpha^y = |\theta_y|$, $\theta = (\theta_x, \theta_y)$ e

$$p^- = D_x^- \phi_{ij} = \frac{\phi_{ij} - \phi_{i-1,j}}{\nabla x}, \tag{7.10}$$

$$p^+ = D_x^+ \phi_{ij} = \frac{\phi_{i+1,j} - \phi_{ij}}{\nabla x}, \tag{7.11}$$

$$q^- = D_y^- \phi_{ij} = \frac{\phi_{ij} - \phi_{i,j-1}}{\nabla y}, \tag{7.12}$$

$$q^+ = D_y^+ \phi_{ij} = \frac{\phi_{i,j+1} - \phi_{ij}}{\nabla y}, \tag{7.13}$$

são as aproximações backward e forward da derivada de ϕ em x, y no ponto P_{ij}, respectivamente. Usando um passo de Euler explícito para a discretização do tempo, o esquema numérico correspondente a (7.4) é

$$\phi_{ij}^{k+1} = \phi_{ij}^k - \Delta t \hat{H}^{LF}(p^-,p^+,q^-,q^+), \tag{7.14}$$

onde p^-, p^+, q^-, q^+ são calculados para ϕ_{ij}^k.

7.4 Reinicialização

Para estabilidade numérica, a solução da equação level set (7.4) não pode ser muito achatada ou íngreme. Isso é satisfeito a princípio se ϕ é uma função distância, i.e. $|\nabla\phi| = 1$. Mesmo que inicializemos ϕ usando uma função distância sinalizada, a solução ϕ da equação (7.4) geralmente não permanece próximo a uma função distância, assim regularmente reinicializamos ϕ.

Apresentamos aqui de forma breve o procedimento para a reinicialização introduzida em [17]. A reinicialização é feita resolvendo a seguinte equação do tipo Hamilton-Jacobi:

$$\partial_\tau \varphi + S(\phi)(|\nabla\varphi| - 1) = 0 \text{ em } \mathcal{D}\times\mathbb{R}^+, \tag{7.15}$$

$$\varphi(x,0) = \phi(x,t),\ x \in \mathcal{D}, \tag{7.16}$$

onde $S(\phi)$ é a aproximação da função distância sinalizada

$$S(\phi) = \frac{\phi}{\sqrt{\phi^2 + |\nabla\phi|^2\epsilon^2}}, \tag{7.17}$$

com $\epsilon = \min(\Delta x, \Delta y)$.

Para a discretização foi usado o esquema upwind explícito padrão,

$$\varphi_{ij}^{k+1} = \varphi_{ij}^{k} - \Delta t K(p^-, p^+, q^-, q^+), \tag{7.18}$$

onde

$$K(p^-, p^+, q^-, q^+) = \max(S(\phi_{ij}), 0)K^+ + \min(S(\phi_{ij}), 0)K^-, \tag{7.19}$$

e

$$K^+ = [\max(p^-, 0)^2 + \min(p^+, 0)^2 + \max(q^-, 0)^2 + \min(q^+, 0)^2]^{\frac{1}{2}}, \tag{7.20}$$

$$K^- = [\min(p^-, 0)^2 + \max(p^+, 0)^2 + \min(q^-, 0)^2 + \max(q^+, 0)^2]^{\frac{1}{2}}, \tag{7.21}$$

e onde p^-, p^+, q^-, q^+ são calculados para ϕ_{ij}^k usando (7.10).

Capítulo 8

Implementação Numérica

Para a implementação numérica foi utilizado FEniCS 2017.1 [2, 8, 11], que é uma plataforma de computação de código aberto usada para resolver equações diferenciais parciais. Tal possui interface de alto nível para Python e C++.

8.1 Introdução

O fluxograma 8.1 nos fornece uma visão geral do algoritmo ultilizado na implementação numérica. Com isso, tomaremos ele como base para explicar as linhas de códigos, que se encontra no apêndice, para minimização da compliance de seis casos: cantilever, cantilever assimétrico, half-wheel, bridge, MBB beam e cantilever duas forças.

O código está dividido em dois arquivos chamados `init.py` e `compliance.py`, sendo o segundo o arquivo principal que contém as linhas de código que executa algoritmo descrito no fluxograma.

Já o arquivo `init.py`, serve para identificar quais dos seis casos se deseja implementar e retornar os dados exclusivos do caso escolhido como, por exemplo, o chute inicial da função level set que no código denotamos por `phi_mat`, e as condições de fronteira. Assim, os dados de saída de `init.py` compõem uma parte dos dados iniciais necessários para a execução do arquivo principal `compliance.py`. O que veremos com mais detalhes na seção que se segue.

8.2 Parâmetros dependentes dos casos e `init.py`

Como estamos lidando com a minimização da compliance com restrição de volume, o funcional que minimizaremos será

$$\mathcal{J}(\Omega) := J(\Omega) + \Lambda V(\Omega), \tag{8.1}$$

onde Λ é uma constante, $V(\Omega)$ é o volume de Ω e $J(\Omega)$ a função compliance denotada por

$$J(\Omega) = \int_{\mathcal{D}} A_\Omega e(u(x)) : e(u(x))dx.$$

A tabela 8.1 fornece todos os parâmetros que variam de acordo com o caso selecionado como declarado no código e o que representam.

O domínio $\mathcal{D}$ em todos os casos é um retângulo $\mathcal{D} = [0, lx] \times [0, ly]$, onde cada caso possui tamanhos dos lados `lx` e `ly` específicos. A variável `Load` é a posição pontual onde uma força será aplicada, por exemplo `Load = [Point(lx, 0.5)]` para o caso cantilever, o que significa que um peso é aplicado no ponto $(l_x, 0.5)$.

Uma vez especificado os números de divisões `Nx, Ny` nas direções x e y , dois tipos de discretização de $\mathcal{D}$ são realizadas. Uma usando a malha `mesh` construída usando a linha

```
mesh = RectangleMesh(Point(0.0,0.0),Point(lx,ly),Nx,Ny,'crossed'
```

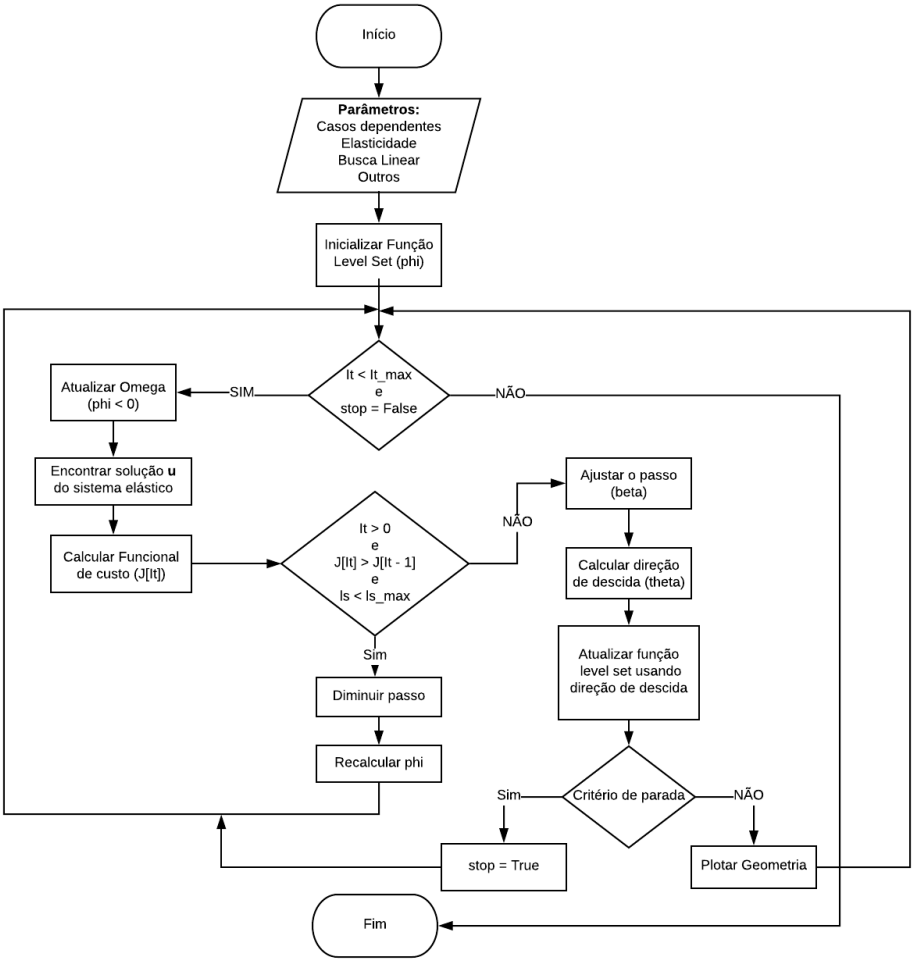


Figura 8.1: *Fluxograma do algoritmo para minimização da compliance no contexto de linearização da elasticidade*

outra referente a malha cartesiana cujos pontos são definidos por

```
XX,YY = np.meshgrid(np.linspace(0.0,lx,Nx+1),np.linspace(0.0,ly,Ny+1)).
```

A malha `mesh` é usada no algoritmo para calcular numericamente a solução u do sistema elástico (6.2) - (6.6) e a direção de descida θ a partir da equação (6.30) pelo método de elementos finitos.

A classe `RectangleMesh` cria uma malha em um retângulo em 2D gerado pelos pontos do retângulo (0, 0) e (lx, ly). O argumento opcional `crossed` significa que cada quadrado da malha é dividido em quatro triângulos definidos pelas diagonais transversais do quadrado. Escolhemos `lx, ly, Nx, Ny` com lx Nx^{-1} = ly Ny^{-1}. A escolha do argumento `crossed` é necessária para se ter um deslocamento u simétrico e para manter a simetria do design ao longo das iterações caso o problema for simétrico, por exemplo o caso cantilever. Note que para preservar a simetria das soluções precisamos escolher um número ímpar de divisões `Nx` ou `Ny`, dependendo da orientação da simetria. Por exemplo, no caso do cantilever simétrico, podemos escolher `Ny`=75 já que o eixo de simetria é a linha $y = 1/2$, e `Nx`=150.

Já a malha cartesiana é utilizada para a inicialização da função level set, na implementação do esquema numérico (7.14) que resolve a equação de Hamilton-Jacobi e também para a reinicialização (7.18). Em `compliance.py` tudo que for definido na malha cartesiana é diferenciada pelo sufixo `mat`. Por exemplo, `phi` é uma função definida em `mesh` enquanto `phi_mat` é função correspondente definida na malha cartesiana.

A primeira inicialização da função level set se dá por meio de `phi_mat` em `init.py`, com cada

Tabela 8.1: *Parâmetros dependentes dos casos*

Notação	O que representa
`Lag`	Multiplicador de Lagrange Λ para restrição de volume
`lx, ly`	Tamanho dos lados do domínio $\mathcal{D}$ retangular
`Nx, Ny`	Número de subintervalos dos lados `lx, ly`, respectivamente
`Load`	Posição do peso ou carga aplicada em um ponto do domínio $\mathcal{D}$
`bcd`	Lista que contém as condições de fronteira
`mesh`	Malha usada para resolver equações via elementos finito
`phi_mat`	Função level set inicial ϕ cujo domínio compõe a malha cartesiana
`Vvec`	Espaço de funções vetoriais

caso com sua inicialização particular.

No caso do cantilever simétrico escolhemos,

$$\begin{aligned}\phi(x,y) = &-\cos(8\pi x/l_x)\cos(4\pi y) - 0.4 + \max(200(0.01 - x^2 - (y - l_y/2)^2), 0) \\ &+ \max(100(x + y - l_x - l_y + 0.1), 0) + \max(100(x - y - l_x + 0.1), 0),\end{aligned} \tag{8.2}$$

Os coeficientes dentro do cosseno determinam o número inicial de componentes conexas dentro do domínio. Aqui (8.2) corresponde a dez "buracos"iniciais dentro do domínio (mais alguns semi-buracos na fronteira de $\mathcal{D}$).

A razão para se adicionar três termos max em (8.2) é específica na nossa abordagem. Como $\theta \in \mathcal{C}_c^{0,1}(\mathcal{D}, \mathbb{R}^n)$, temos $\theta = 0$ nos cantos do retângulo $\mathcal{D}$. Assim sendo, a forma em uma pequena vizinhança das quinas não irá mudar, e se iniciarmos com uma inicialização inapropriada acabaríamos com um pequeno corte com o material correto em determinados cantos. Dependendo do problema, é fácil de ver qual deve ser a distribuição correta do material para o design final.

Outro problema pode aparecer em pontos da fronteira que estão no eixo de simetria para os problemas simétricos. De fato, devido a suavidade de θ e a simetria do problema obteremos $\theta_x = 0$ ou $\theta_y = 0$ nesses pontos e a forma não mudará. Por exemplo no caso do cantilever simétrico, esse problema surge no ponto $[0, l_y/2]$. Não há nenhum problema em $[l_x, l_y/2]$, já que esse é o ponto em que o peso é aplicado, então dever ser fixo de qualquer jeito. Isso explica o termo $\max(200(0.01 - x^2 - (y - l_y/2)^2), 0)$ em (8.2).

Por último, temos as definições das condições de fronteiras. A fronteira Γ_d, que depende do caso, é definida usando a classe `DirBd`, e instanciada por `dirBd = DirBd()` em `init.py`. Colocamos a marca 1 para `dirBd` e marcamos as outras fronteiras com 0, e introduzimos a medida de fronteira `ds`. A condição de fronteira de Dirichlet em Γ_d é definida usando

```
DiriichletBC(Vvec, (0.0, 0.0), boundaries, 1)
```

Quando a fronteira Γ_d é desconectada, como no caso do half-wheel, a variável `bcd` é definida como uma lista de condições de fronteira. Para o cantilever, `bcd` possui apenas um elemento. Para o caso halfwheel definimos uma classe adicional `DirBd2` para definir condições de fronteira `bcd`, pois há dois tipos de condições de Dirichlet.

Na subseções seguintes, fornecemos as informações dos parâmetros que dependem dos casos encontrados em `init.py`, como condições de fronteiras, posição da carga/peso e inicialização da função level set.

8.2.1 Cantilever assimétrico

Aqui tomamos `Load = [Point(lx, 0.0)]` para o caso do cantilever assimétrico, para se ter uma carga no canto inferior direito. A inicialização é feita de modo que começa com a fase material

onde o peso é aplicado:

$$\phi(x,y) = -\cos(6\pi x/l_x)\cos(4\pi y) - 0.4 + \max(100(x+y-l_x-l_y+0.1),0).$$

8.2.2 Half-wheel

Para o half-wheel temos `lx, ly = [2.0, 1.0]`. A posição da carga é dada por `Load = [Point(lx/2, 0.0)]`. No canto $(0.0, 0.0)$ precisa-se condições de Dirichlet pontuais e condições de rolamento em $(l_x, 0.0)$. Para isso usamos as seguintes fronteiras em `init.py`:

```
class DirBd(SubDomain):
  def inside(self, x, on_boundary):
    return abs(x[0])< tol and abs(x[1])< tol
class DirBd2(SubDomain):
  def inside(self, x, on_boundary):
    return abs(x[0]-lx)< tol and abs(x[1])< tol
dirBd,dirBd2 = [DirBd(),DirBd2()]
```

onde `tol = 1E-14`. Onde as suas partes da fronteira são marcadas com diferentes números

```
dirBd.mark(boundaries, 1)
dirBd2.mark(boundaries, 2)
```

e definimos o vetor com as condições de fronteira como

```
bcd = [DirichletBC(Vvec, (0.0,0.0), dirBd, method='pointwise'),\
DirichletBC(Vvec.sub(1), 0.0, dirBd2,method='pointwise')]
```

O método `pointwise` é usado já que `dirBd2` é um único ponto. Note aqui que a condição de rolamento é alcançada colocando a componente `Vvec.sub(1)` sendo 0, de fato `Vvec.sub(1)` representa a componente y de uma função vetorial do espaço `Vvec`. Nas linhas 50-51 de `compliance.py`, condições aproximadas de Dirichlet para θ são aplicadas em `dirBd1` e `dirBd2` como cantos que devem ser fixos.

A inicialização precisa de um ajustes para encaixar no caso half-wheel. Escolhemos

$$\begin{aligned}\phi(x,y) = &-\cos(3\pi(x-1))\cos(7\pi y) - 0.3 + \min(5/l_y(y-1)+4,0)\\ &+\max(100(x+y-l_x-l_y+0.1),0) + \max(100(-x+y-l_y+0.1),0).\end{aligned} \tag{8.3}$$

8.2.3 Bridge

Esse caso é similar ao half-wheel. A principal diferença é a condição de Dirichlet pontual no canto inferior direito, que corresponde a

```
DirichletBC(Vvec, (0.0,0.0), dirBd2,method='pointwise')
```

Tomamos também a seguinte inicialização

$$\phi(x,y) = -\cos(4\pi(x-1))\cos(4\pi y) - 0.2 + \max(100(y-l_y+0.05),0).$$

8.2.4 MBB-beam

Definimos o MBB-beam como no paper original [19]. Usamos a simetria do problema para calcular a solução somente da metade direita do domínio. Assim, impomos a condição de rolamento do lado esquerdo do domínio computacional $\mathcal{D}$, que corresponde a $u \cdot n = 0$. Tomamos `lx = 3.0` e `ly = 1.0`, e `Nx, Ny` devem ser escolhidos de acordo para manter a malha regular. Por exemplo, podemos escolher `Nx = 150, Ny = 50`. Tomamos `Load = [Point(0.0, 1.0)]`.

Também temos condições de rolamento pontuais no canto inferior direito de $\mathcal{D}$. Em `init.py` isso corresponde as seguintes definições de fronteira:

```
class DirBd(SubDomain):
  def inside(self, x, on_boundary):
  return near(x[0],.0)
class DirBd2(SubDomain):
  def inside(self, x, on_boundary):
  return abs(x[0]-lx) < tol and abs(x[1]) < tol
dirBd,dirBd2 = [DirBd(),DirBd2()]}
```

Então as fronteiras são marcadas com diferentes números:

```
dirBd.mark(boundaries, 1)
dirBd2.mark(boundaries, 2)
```

Definimos as condições de fronteira em duas fronteiras `dirBd` e `dirBd2`:

```
bcd = [DirichletBC(Vvec.sub(0), 0.0, boundaries, 1),\
  DirichletBC(Vvec.sub(1), 0.0, dirBd2,method='pointwise')]}
```

O termo `+ds(2)` nas linhas 50-51 é ativo já que `dirBd2` não é vazio, como para o caso half-wheel. Também escolhemos a inicialização apropriada:

$$\phi(x,y) = -\cos(4/l_x\pi x)\cos(4\pi y) - 0.4 + \max(100(x+y-l_x-l_y+0.1),0) + \max(5/l_y(y-1)+4,0).$$

8.2.5 Casos com cargas múltiplas

Para esse caso temos que `Load = [Point(lx, 0.0), Point(lx, 1.0)]` é uma lista. Na linha 30 de `compliance.py`, `U` é uma lista com dois elementos correspondentes as duas cargas. Isso explica o comando de repetição `for` em `_shape_der`(ver linha 132). Ilustramos o caso de cargas múltiplas com um problema cantilever com duas cargas aplicadas no canto inferior e superior direito, os dois com mesma intensidade para obter um a simetria no design. Usamos a inicialização

$$\phi(x,y) = -\cos(4\pi(x-0.5))\cos(4\pi(y-0.5)) - 0.6 + \max(50(y-l_y+0.1),0) - \max(50(-y+0.1),0).$$

8.3 Outros parâmetros e `compliance.py`

Além dos parâmetros que dependem dos casos, temos os parâmetros do sistema elástico ou utilizados na busca linear e outros que são inicializados nas primeiras linhas de código de `compliance.py`. Os principais parâmetros se encontram na tabela 8.2 e daremos mais detalhes de alguns deles nas subseções seguintes.

Tabela 8.2: *Parâmetros dependentes dos casos*

Notação	O que representa
`eps_er`	Coeficiente do material ersatz ϵ
`E, nu, mu, lmbda`	Parâmetros de elasticidade
`ls`	Contador para a busca linear
`ls_max`	Número máximo de buscas
`beta`	Tamanho do passo utilizado no método gradiente
`It`	Contador que acompanha as iterações no loop principal
`It_max`	Número máximo de iterações

8.3.1 Inicialização da função level set

A função level set usada em `compliance.py` para definir e atualizar o domínio Ω, está definida na malha `mesh` e não na malha cartesiana para melhorar a precisão dos cálculos. Assim, a função level set `phi` é uma função no espaço `V` (ver seção 8.3.2) obtida a partir da matriz `phi_mat`, inicializada em `init.py`, por meio da função `_comp_lsf`. Esse processo é detalhado em 8.3.3.

8.3.2 Elementos finito

Na linha 28 e no arquivo `init.py` definimos os espaços de elementos finitos associado a `mesh`:

```
V = FunctionSpace(mesh, 'CG', 1)
Vvec = VectorFunctionSpace(mesh, 'CG', 1)
```

Aqui `CG` é uma abreviação para continuous Galerkin é o último argumento é o grau do elemento, que significa que foi escolhido elementos de lagrange linear por partes. Como escolhemos a malha com diagonais cruzadas, cada quadrado possui um vértice adicional em seu centro, onde as diagonais se cruzam. Portanto o número total de vértices é

```
dofsV_max = (Nx+1)*(Ny+1) + Nx*Ny
```

Definimos também `dofsVvec_max = 2*dofsV_max` na linha 37, que representa os graus de liberdade do espaço de função vetorial `Vvec`.

8.3.3 Função `_comp_lsf`

Aqui explicamos o mecanismo para se obter `phi` de `phi_mat`. A função `phi_mat` é atualizada toda iteração pela função `_hj` na linha 100, e precisamos de `phi` para definir o novo conjunto `Omega` na linhas 42-44. Observe que o conjunto de vértices de `mesh` são os vértices da malha cartesiana mais os vértices dos centros dos quadrados onde as diagonais se encontram. Assim, calculamos os valores de `phi` no centro dos quadrados usando interpolação.

Isso é feito pela função `_comp_lsf` (linhas 173-182) da seguinte forma. Primeiro, nas linhas 33-34, `dofsV` e `dofsVvec` são as coordenadas dos vértices associados com os graus de liberdade. Eles são usados nas linhas 35-36 para definir `px, py` e `pxvec, pyvec`, que assumem valores inteiros e são usados por `_comp_lsf` para encontrar a correspondência entre as entradas da matriz `phi_mat` e as entradas de `phi`. Em `_comp_lsf`, precisamente na linha 175, conferimos se o vértice associado a `px, py` correspondem ao vértice da malha cartesiana. Se este for o caso, o valor de `phi` será igual ao valor de `phi_mat`, caso contrário o vértice será o centro do quadrado e o valor de `phi` nesse ponto será o valor médio dos quatro vértices do quadrado; veja linhas 179-181. Assim a sáida de `_comp_lsf` será a função `phi` definida em `mesh`.

8.3.4 Atualização do domínio

O loop principal começa na linha 55. Na linha 57, Ω é instanciado por `omega = Omega()`. Isso inicializa `omega` ou o atualiza se `phi` tiver sido atualizado dentro do loop. Na linha 59, `omega` é marcado com o número 1 e o seu complementar com o número 0 na linha 58. Definimos a medida de integração para os subdomínios Ω e $\mathcal{D}\backslash\Omega$ usando

```
1    dx = Measure('dx') (subdomain\_data=domains)
```

Usamos `dx(1)` para integrar em Ω e `dx(0)` para integrar em $\mathcal{D}\backslash\Omega$; ver linha 68 como exemplo.

8.3.5 Resolvendo o sistema elástico

Agora podemos calcular `U`, a solução do sistema elástico nas linhas 61-62, usando `_solve_PDE` nas linhas 116-127. Usamos um solver LU para resolver o sistema; ver linhas 125-126. Note que

U é uma lista já que consideramos o caso geral de várias cargas. Assim o tamanho da lista U é o tamanho de Load; ver linha 30.

8.3.6 Atualização do funcional de custo

Nas linhas 64-70 calculamos a compliance, o volume de omega e o funcional de custo J correspondente a (8.1). Observe que o comando par o cálculo da compliance é parecido com a notação matemática, i.e. se assemelha a seguinte fórmula matemática:

$$J(\Omega) = \epsilon \int_{\mathcal{D}\setminus\Omega} \mathbf{S}_1 : D\theta + \int_{\Omega} \mathbf{S}_1 : D\theta, \quad \mathbf{S}_1 = 2\mu e(u) : e(u) + \lambda \mathrm{tr}(e(u))^2.$$

Lembre que a compliance é a soma no caso de várias cargas, isso explica o loop for na linha 65.

8.3.7 Busca linear e critério de parada

A busca linear começa na linha 72. Se o critério

```
J[It] > J[It - 1]
```

for satisfeito, então rejeitamos o passo atual. Nesse caso reduzimos o tamanho do passo beta multiplicando-o por gamma na lina 74. Também, retomamos os valores anteriores de phi_mat e phi, os quais armazenamos em phi_mat_old e phi_old, ver linha 75. Então precisamos recalcular phi_mat e phi nas linhas 76-77 usando o novo passo beta.

Se o passo não for rejeitado, então vamos para a próxima iteração começando na linha 85. Se o passo foi aceito na primeira iteração na busca linear, para acelerar a velocidade do algoritmo, aumentamos o passo de referência beta0 colocando

```
beta0 = min(beta0 / gamma2, 1)
```

para tomar passos maiores, já que gamma2 é menor que 1. Aqui beta0 é mantido menor que 1 para estabilizar o esquema numérico para a equação Hamilton-Jacobi, veja o passo do tempo dt na linha 150. Escolhemos gamma2 = 0.8 em nossos exemplos; ver linha 21.

Se o número máximo de buscas linear ls_max for atingido, diminuímos na linha 85 o passo de referência beta0 colocando

```
beta0 = max(beta0 * gamma2, 0.1*beta0\_init)
```

Impomos um limite inferior 0.1*beta0_init em beta0 para que o tamanho do passo não fique tão pequeno. Note que beta é resetado para beta0 na linha 88.

8.3.8 Direção de descida

FEniCS usa a Unified Form Language (UFL) para a representação das formulações fracas de equações diferenciais parciais, que resulta em uma notação intuitiva próxima a da usada na matemática. Isso pode ser visto nas linhas 49-50, onde definimos a matriz av que corresponde a forma bilinear (6.32). Aqui theta e xi são funções em Vvec, xi é a função teste em (6.31), enquanto theta corresponde a θ em (6.31). Em nosso código usamos a notação th para θ quando θ é a direção de descida. O coeficiente 1.0e4 nas condições de fronteira para av:

```
1.0e4*(inner(dot(theta,n),dot(xi,n)) * (ds(0) + ds(1) +ds(2)))
```

força $\theta \cdot n$ próximo de zero em $\partial\mathcal{D}$, que corresponde a restrição $\theta \in \mathcal{C}_c^{0,1}(\mathcal{D}, \mathbb{R}^n)$. Para casos onde dirBD2 não está definido o termo ds(2) não afeta o cálculo.

Montamos a matriz para a EDP de th e define-se o solver LU nas linhas 50-53, antes de começar o loop principal. De fato, a forma bilinear $\mathcal{B}$ em (6.32) é independente de Ω. Assim, reutilizamos a fatoração do solver LU para resolver a EDP para th usando o parâmetro reuse_factorization

na linha 53. Isso permite uma economia de cálculos, mas para as malhas maiores seria apropriado usar abordagens mais eficientes como métodos de Krylov.

Na linha 90, calculamos a direção de descida `th`. A função `_shape_der` nas linhas 129-139 resolve a EDO para θ, i.e. implementa (6.31)-(6.32) usando a expressão de volume da derivada (6.15) e (6.18). A formulação variacional usada no código para o caso de uma carga é: encontrar $\theta \in H_d^k(\mathcal{D})^m$ tal que

$$\int_{\mathcal{D}} \alpha_1 D\theta : D\xi + \alpha_2 \theta \cdot \xi = -d\mathcal{J}^{\mathrm{vol}}(\Omega, \xi), \forall \xi \in H^1(\mathcal{D})^m \tag{8.4}$$

$$\text{onde} \qquad d\mathcal{J}^{\mathrm{vol}}(\Omega)(\xi) = \int_{\mathcal{D}} (2Du^{\mathsf{T}} A_\Omega e(u) - A_\Omega e(u) : e(u) I_d) : D\xi + \Lambda \int_\Omega \operatorname{div} \xi,$$

com $\alpha_1 = 1$ e $\alpha_2 = 0.1$. Caso várias cargas são aplicadas o lado direito de (8.4) deve ser substituído pela soma das cargas `u` em `u_vec`.

O lado direto de (8.4) é feito na linha 136, mas precisamos integrar separadamente em Ω e $\mathcal{D}\backslash\Omega$ usando `dx(1)` e `dx(0)`, respectivamente. O sistema é resolvido na linha 138 usando o solver definido na linha 52.

Na linha 90 obtemos a direção de descida `th` no espaço `Vvec`. Para atualizar `phi` precisamos de `th` na malha cartesiana. Como já explicado, apenas precisamos extrair os valores apropriados de `th` já que a malha cartesiana está incluído em `mesh`. Isso é feito nas linhas 91-97, e a função correspondente à malha cartesiana é `th_mat`.

8.3.9 Atualização da função level set

Então, para a atualização da função level set `phi_mat` usamos a subfunção `_hj`. A subfunção `_hj` nas linhas 141-152 segue exatamente o processo de discretização descrito na seção 7.3. Nas linhas 143-146, as quantias `Dxm, Dxp, Dyp, Dym` correspondem a p^-, p^+, q^+, q^-, respectivamente. Na linha 142, tomamos 10 passos da atualização do Hamilton-Jacobi, que é um padrão, no entanto eurístico, para acelerar a convergência. Em ordem a estabilizar o esquema numérico, escolhemos o passo do tempo como

```
dt = beta*lx / (Nx*maxv)
```

onde `maxv` é igual a $\max(|\theta_1| + |\theta_2|)$, `lx/Nx` e o tamanho da célula, e o tamanho de passo `beta` é menor que 1 todo momento em vista da linha 86. Na linha 99, salvamos as versões atuais de `phi` e `phi_mat` nas variáveis `phi_old, phi_mat_old`, para usar no caso em que o passo é rejeitado durante a busca linear. na linha 102, a função `phi` é extrapolada para `phi_mat` usando `_comp_lsf.`

8.3.10 Reinicialização da função level set

A cada 5 iterações, reinicializamos a função level set na linha 101. Isso é feito pela subfunção `_reinit` nas linhas 154-157. A reinicialização segue o procedimento descrito na seção 7.4. Nas linhas 161-164, `Dxm, Dxp, Dyp, Dym` correspondem a p^-, p^+, q^+, q^-, respectivamente. Nas linhas 165 até 168, `Kp` e `Km` correspondem a K^+ e K^- de (7.20)-(7.21), respectivamente. Linha 169 corresponde a (7.19) e linha 70 a atualização (7.18).

A função `signum` é a aproximação da função sinal de ϕ correspondendo a $S(\phi)$, definida em (7.17). Para calcular `signum`, usamos `lx/Nx` para ϵ_s, e $|\nabla\phi|$ é calculado usando diferenças finitas simétrica, que á dada por `Dxs` e `Dys` nas linhas 155-158.

8.3.11 Critério de parada

Finalmente nas linhas 104-105 conferimos se o critério de parada

```
if It>20 and max(abs(J[It-5:It] - J[It -1])) < 2.0*J[It-1]/Nx**2:
```

é satisfeito. Aqui, `It-1` é a iteração corrente. Isso significa que o algoritmo para se o máximo da diferença entre o valor do funcional de custo na iteração vigente e o valor das quatro iterações anteriores for menor que um certo limite. Para se tomar passos menores quando a malha fica mais fino, foi determinado euristicamente o limite `2.0*J[It-1]/Nx**2`, que depende do tamanho da malha `Nx`.

Capítulo 9

Resultados

Neste capítulo exploramos diferentes inicializações, discretizações e testamos alguns multiplicadores de Lagrange com o intuito de analisar o comportamento do algoritmo e qual a influência dessas alterações no design ótimo.

9.1 Influência da inicialização

Fixado a discretização com $(N_x, N_y) = (302, 151)$ e o multiplicador de Lagrange $\Lambda = 40$ para o caso do cantilever e $\Lambda = 70$ para o caso do cantilever assimétrico, geramos algumas inicializações a partir das seguintes expressões

$$\begin{aligned}\phi_1(x,y) = &-\cos(\mathrm{n}\pi x/l_x)\cos(4\pi y) - 0.4 + \max(200(0.01 - x^2 - (y - l_y/2)^2), 0) \\ &+ \max(100(x + y - l_x - l_y + 0.1), 0) + \max(100(x - y - l_x + 0.1), 0),\end{aligned} \tag{9.1}$$

$$\phi_2(x,y) = -\cos(\mathrm{n}\pi x/l_x)\cos(4\pi y) - 0.4 + \max(100(x + y - l_x - l_y + 0.1), 0), \tag{9.2}$$

onde ϕ_1 corresponde ao caso do cantilever e ϕ_2 ao caso assimétrico.

Para cada valor de n nas expressões acima obtemos uma malha inicial com uma determinada quantidade de componentes conexas, quanto maior o valor de n mais componentes conexas a inicialização terá. Nos dois casos começamos com $n = 6$ e fomos aumentando em duas unidades e obtivemos os resultados das figuras 9.1 e 9.2.

Concluímos que aumentando o número de componentes conexas na inicialização gera um design também com mais componentes conexas. Assim, pode-se dizer que a topologia do design é influenciada pela inicialização.

9.2 Influência do multiplicador de Lagrange em diferentes malhas

Aqui exploramos como a variação do multiplicador de Lagrange em diferentes malhas influencia o design otimizado. Para tal análise fixamos uma inicialização para os dois casos observados e observamos a topologia final dos designs gerados pelo algoritmo.

Nessa abordagem usamos o caso do cantilever e do half-wheel com as seguintes inicializações, respectivamente,

$$\begin{aligned}\phi(x,y) = &-\cos(6\pi x/l_x)\cos(4\pi y) - 0.4 + \max(200(0.01 - x^2 - (y - l_y/2)^2), 0) \\ &+ \max(100(x + y - l_x - l_y + 0.1), 0) + \max(100(x - y - l_x + 0.1), 0),\end{aligned}$$

$$\begin{aligned}\phi(x,y) = &-\cos(3\pi(x-1))\cos(7\pi y) - 0.3 + \min(5/l_y(y-1) + 4, 0) \\ &+ \max(100(x + y - l_x - l_y + 0.1), 0) + \max(100(-x + y - l_y + 0.1), 0).\end{aligned}$$

Os resultados são mostrados nas figuras 9.3, 9.4, 9.5, 9.6 e 9.7. É interessante observar nesses

resultados que a topologia do design otimizado depende de Λ e também do tamanho da malha. Uma malha mais fina gera uma topologia mais complexa para um Λ fixo.

Observamos que há uma tendência do número de componentes conexas diminuir com o aumento de Λ.

Por fim pode-se concluir que o algoritmo é robusto em relação a mudança dos parâmetros. Em alguns casos, o algoritmo pode falhar, mas isso é devido a parâmetros extremos, como no caso half-wheel com $\Lambda = 80$ e malha $(Nx, Ny) = (150, 75)$ onde o multiplicador de Lagrange é grande com uma malha não fina suficiente.

Para finalizar a tabela 9.8 apresenta o design otimizado para os casos MBB_beam, cantilever com duas cargas e bridge.

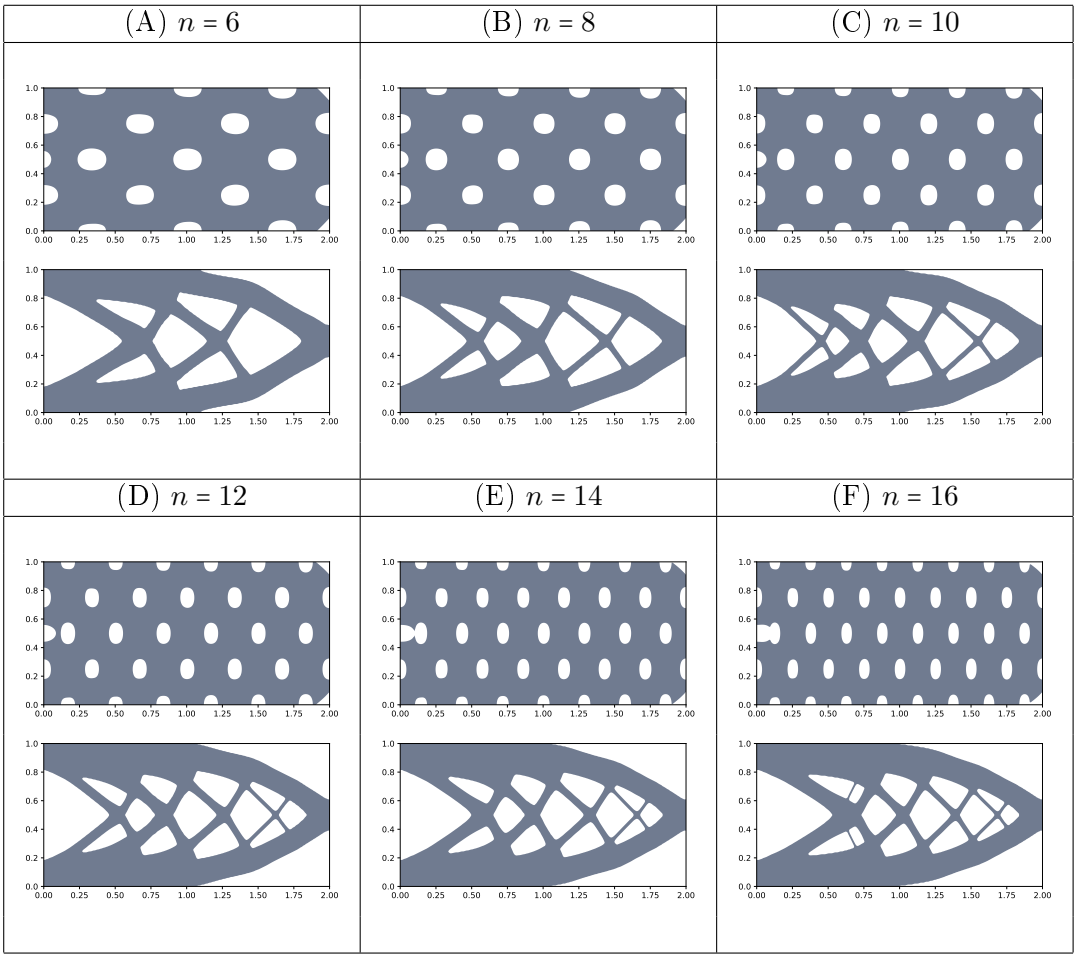


Figura 9.1: *Design otimizado para o cantilever simétrico,* $\Lambda = 40$, $(N_x, N_y) = (302, 151)$

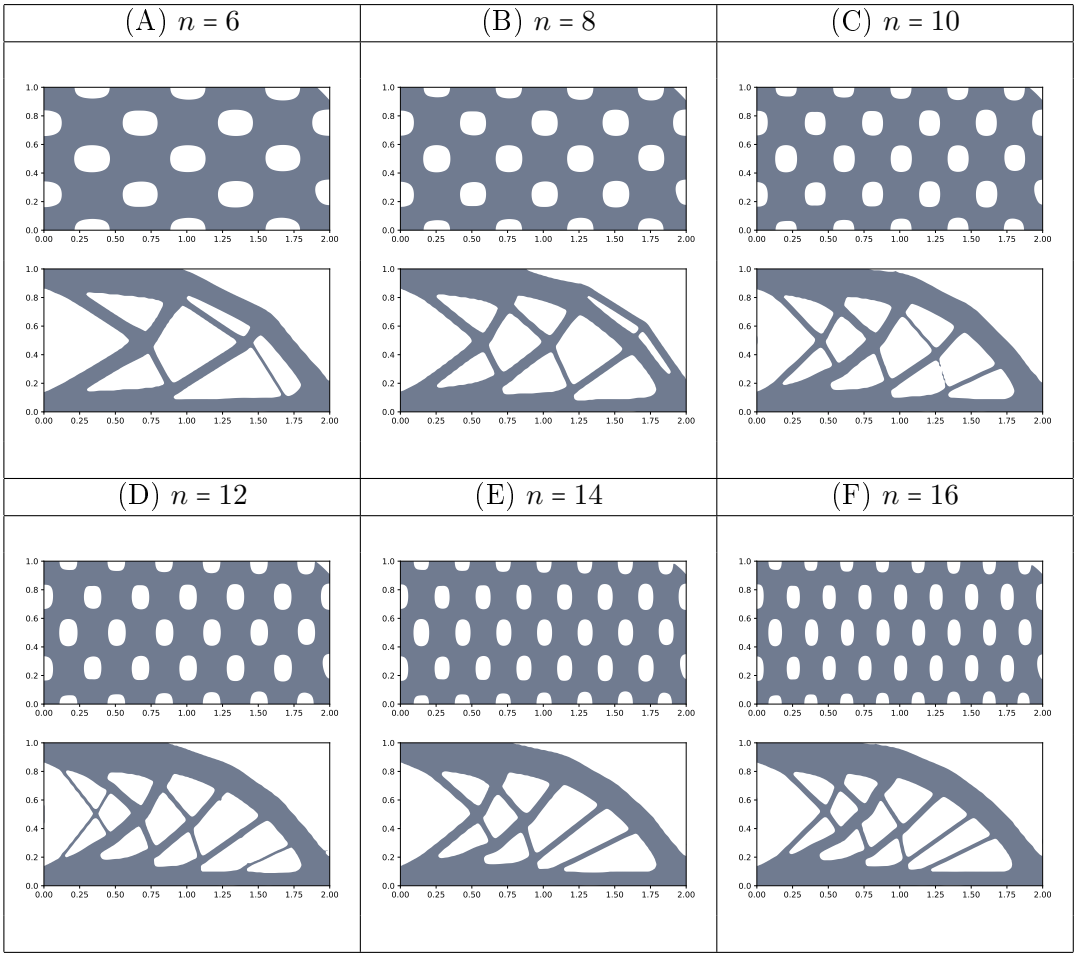
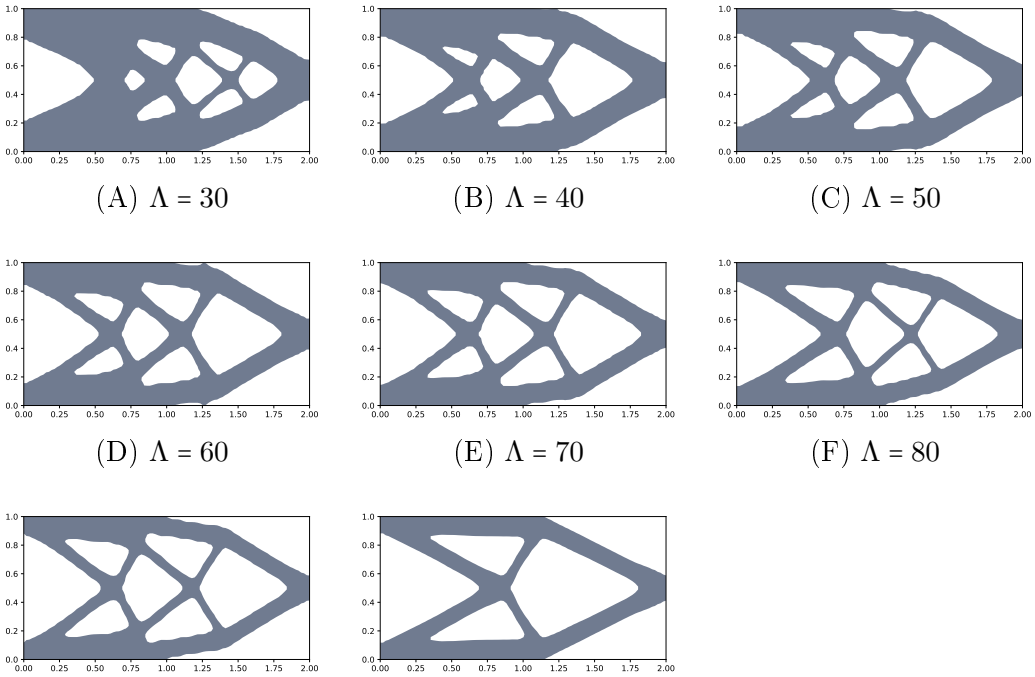


Figura 9.2: *Design otimizado para o cantilever assimétrico,* $\Lambda = 70$, $(N_x, N_y) = (302, 151)$

(Nx, Ny) = (102, 51)



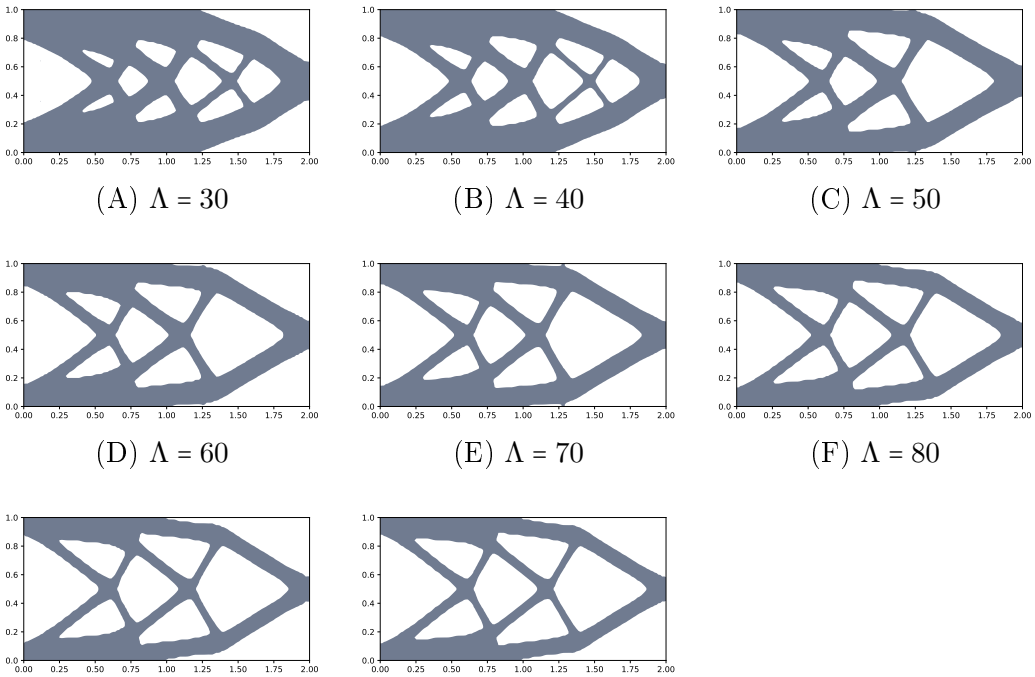
(A) Λ = 30 (B) Λ = 40 (C) Λ = 50

(D) Λ = 60 (E) Λ = 70 (F) Λ = 80

(G) Λ = 90 (H) Λ = 100

Figura 9.3: *Design otimizado do cantiveler simétrico com diferentes Λ e (Nx, Ny) = (102, 51).*

(Nx, Ny) = (150, 75)



(A) Λ = 30 (B) Λ = 40 (C) Λ = 50

(D) Λ = 60 (E) Λ = 70 (F) Λ = 80

(G) Λ = 90 (H) Λ = 100

Figura 9.4: *Design otimizado do cantiveler simétrico com diferentes Λ e (Nx, Ny) = (150, 75).*

(Nx, Ny) = (202, 101)

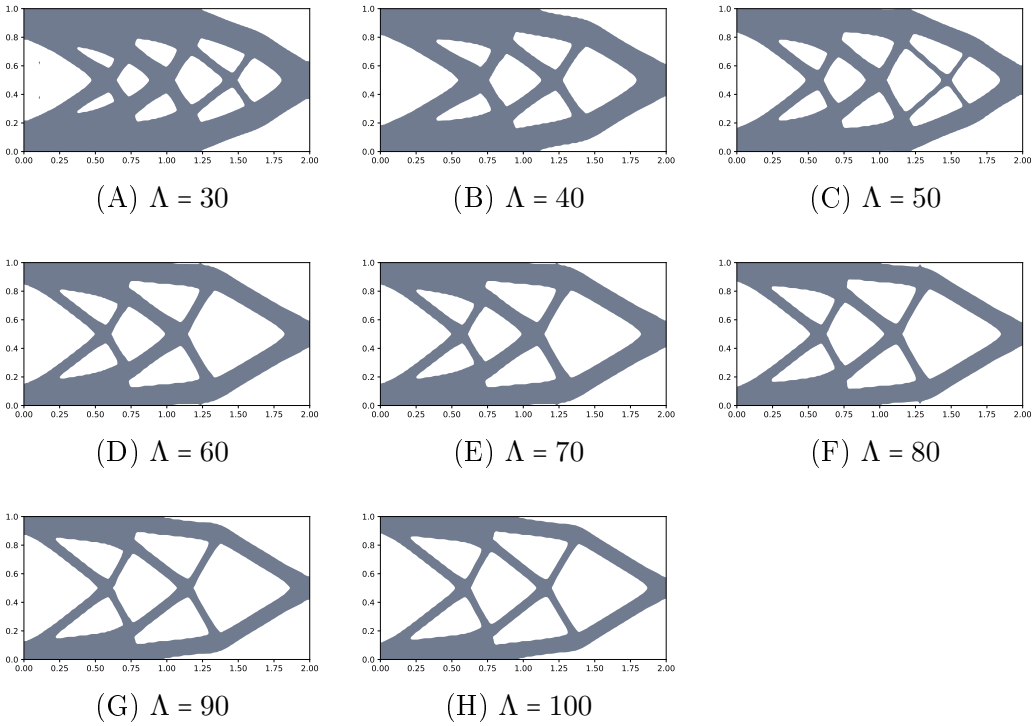


Figura 9.5: *Design otimizado do cantiveler simétrico com diferentes Λ e (Nx, Ny) = (202, 101).*

(Nx, Ny) = (302, 151)

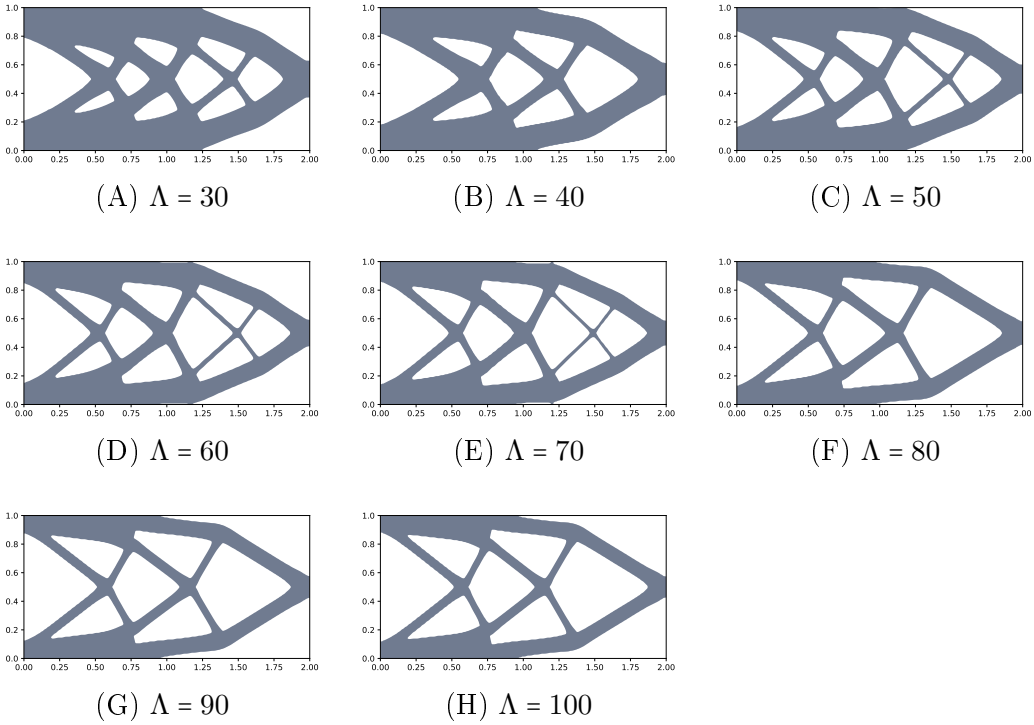


Figura 9.6: *Design otimizado do cantiveler simétrico com diferentes Λ e (Nx, Ny) = (302, 151)*

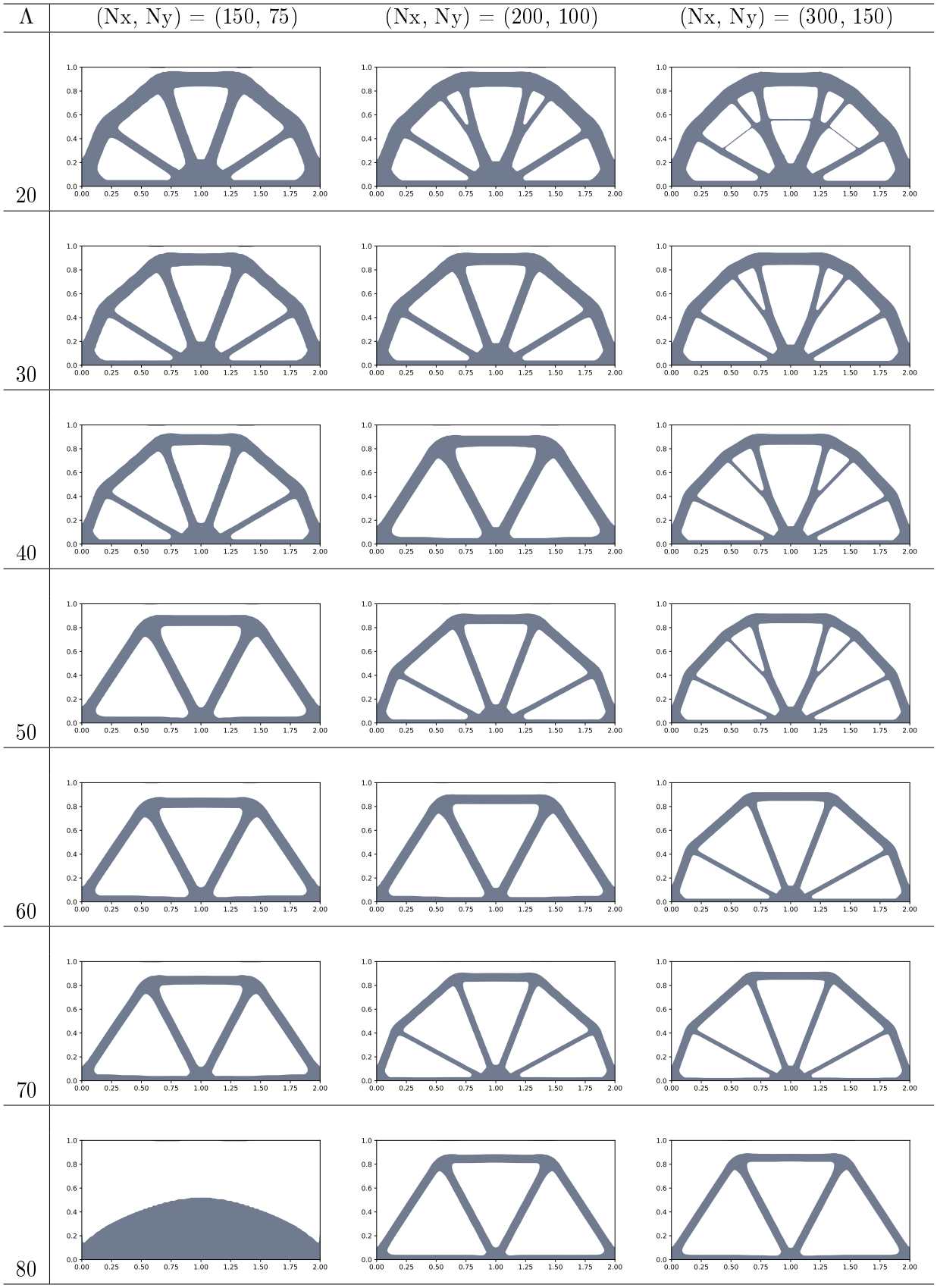
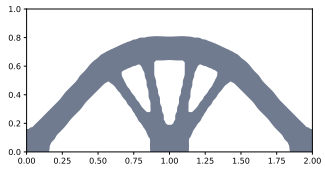
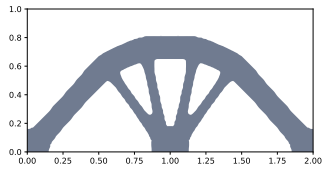


Figura 9.7: *Design otimizado do caso bridge com vários valores de* Λ *e de* (N_x, N_y).

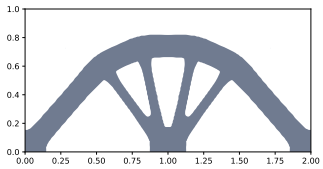
Brigde - Λ = 20



(A) (Nx, Ny) = (150, 75)

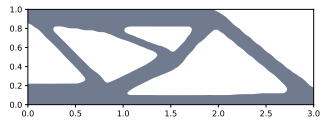


(B) (Nx, Ny) = (200, 100)

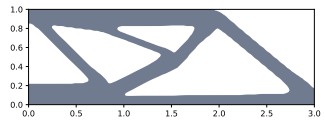


(C) (Nx, Ny) = (300, 150)

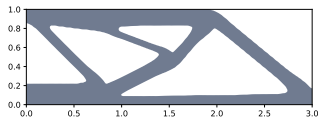
MBB beam - Λ = 130



(D) (Nx, Ny) = (150, 50)

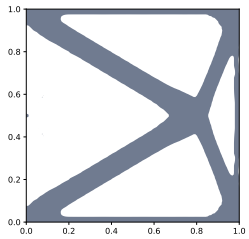


(E) (Nx, Ny) = (240, 80)

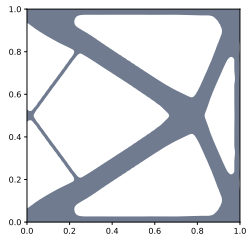


(F) (Nx, Ny) = (300, 100)

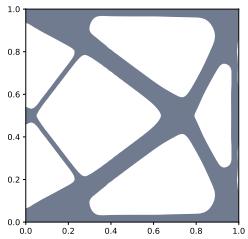
Cantilever com duas cargas - Λ = 100



(G) (Nx, Ny) = (121, 121)



(H) (Nx, Ny) = (151, 151)



(Nx, Ny) = (175, 175)

Figura 9.8: *Design otimizado dos casos bridge, MBB beam e cantilever com duas cargas.*

Apêndice

9.3 Init.py

```
from dolfin import *
import numpy as np

def init(Name):
  tol = 1E-14 # tolerance for coordinate comparisons

  if Name == 'cantilever':
    Lag,Nx,Ny = [40.0,302,151]
  if Name == 'cantilever_asymmetric':
    Lag,Nx,Ny = [70.0,202,101]
  if Name == 'half_wheel':
    Lag,Nx,Ny = [50.0,200,100]
  if Name == 'bridge':
    Lag,Nx,Ny = [30.0,200,100]
  if Name == 'MBB_beam':
    Lag,Nx,Ny = [130.0,150,50]
  if Name == 'cantilever_twoforces':
    Lag,Nx,Ny = [60.0,121,121]

  if Name == 'cantilever' or Name == 'cantilever_asymmetric':
    lx,ly = [2.0,1.0]
    XX,YY = np.meshgrid(np.linspace(0.0,lx,Nx+1),np.linspace(0.0,ly,Ny+1))
    #define boundary conditions
    mesh = RectangleMesh(Point(0.0,0.0),Point(lx,ly),Nx,Ny,'crossed')
    Vvec = VectorFunctionSpace(mesh, 'CG', 1)
    class DirBd(SubDomain):
      def inside(self, x, on_boundary):
        return near(x[0],.0)
    dirBd = DirBd()
    boundaries = FacetFunction("size_t", mesh)
    boundaries.set_all(0)
    dirBd.mark(boundaries, 1)
    ds = Measure("ds")(subdomain_data=boundaries)
    bcd = [DirichletBC(Vvec, (0.0,0.0), boundaries, 1)]

  if Name == 'cantilever':
    ####################
    # Cantilever case
    ####################
    Load = [Point(lx, 0.5)]
    # Initialize level set function
    phi_mat = -np.cos(8.0/lx*pi*XX) * np.cos(4.0*pi*YY) - 0.4\
     + np.maximum(200.0*(0.01-XX**2-(YY-ly/2)**2),.0)\
     + np.maximum(100.0*(XX+YY-lx-ly+0.1),.0) +
        np.maximum(100.0*(XX-YY-lx+0.1),.0)
```

```
45
46  if Name == 'cantilever_asymmetric':
47    ##########################
48    # Asymmetric cantilever
49    ##########################
50    Load = [Point(lx, 0.0)]
51    # Initialize level set function
52    phi_mat = -np.cos(6.0/lx*pi*XX) * np.cos(4.0*pi*YY) - 0.4\
53    + np.maximum(100.0*(XX+YY-lx-ly+0.1),.0)
54
55  if Name == 'half_wheel' or Name == 'bridge':
56    lx,ly = [2.0,1.0]
57    XX,YY = np.meshgrid(np.linspace(0.0,lx,Nx+1),np.linspace(0.0,ly,Ny+1))
58    #define boundary conditions
59    mesh = RectangleMesh(Point(0.0,0.0),Point(lx,ly),Nx,Ny,'crossed')
60    Vvec = VectorFunctionSpace(mesh, 'CG', 1)
61    class DirBd(SubDomain):
62      def inside(self, x, on_boundary):
63        return abs(x[0])< tol and abs(x[1])< tol
64    class DirBd2(SubDomain):
65      def inside(self, x, on_boundary):
66        return abs(x[0]-lx)< tol and abs(x[1])< tol
67    dirBd,dirBd2 = [DirBd(),DirBd2()]
68    boundaries = FacetFunction("size_t", mesh)
69    boundaries.set_all(0)
70  dirBd.mark(boundaries, 1)
71  dirBd2.mark(boundaries, 2)
72  ds = Measure("ds")(subdomain_data=boundaries)
73  Load = [Point(lx/2, 0.0)]
74
75  if Name == 'half_wheel':
76    ####################
77    # Half-wheel case
78    ####################
79    bcd = [DirichletBC(Vvec, (0.0,0.0), dirBd, method='pointwise'),\
80           DirichletBC(Vvec.sub(1), 0.0, dirBd2,method='pointwise')]
81    # Initialize level set function
82    phi_mat = -np.cos((3.0*pi*(XX-1.0))) * np.cos(7*pi*YY) - 0.3\
83    + np.minimum(5.0/ly *(YY-1.0) + 4.0,0) \
84    + np.maximum(100.0*(XX+YY-lx-ly+0.1),.0) +
        np.maximum(100.0*(-XX+YY-ly+0.1),.0)
85
86  if Name == 'bridge':
87    ####################
88    # Bridge case
89    ####################
90    bcd = [DirichletBC(Vvec, (0.0,0.0), dirBd, method='pointwise'),\
91    DirichletBC(Vvec, (0.0,0.0), dirBd2,method='pointwise')]
92    # Initialize level set function
93    phi_mat = -np.cos((4.0*pi*(XX-1.0))) * np.cos(4*pi*YY) - 0.2 \
94    + np.maximum(100.0*(YY-ly+0.05),.0)
95
96  if Name == 'MBB_beam':
97    ####################
98    # MBB beam case
99    ####################
100   lx,ly = [3.0,1.0]
101   XX,YY = np.meshgrid(np.linspace(0.0,lx,Nx+1),np.linspace(0.0,ly,Ny+1))
102   #define boundary conditions
```

```
    mesh = RectangleMesh(Point(0.0,0.0),Point(lx,ly),Nx,Ny,'crossed')
    Vvec = VectorFunctionSpace(mesh, 'CG', 1)
    class DirBd(SubDomain):
      def inside(self, x, on_boundary):
        return near(x[0],.0)
    class DirBd2(SubDomain):
      def inside(self, x, on_boundary):
        return abs(x[0]-lx) < tol and abs(x[1]) < tol
    dirBd,dirBd2 = [DirBd(),DirBd2()]
    boundaries = FacetFunction("size_t", mesh)
    boundaries.set_all(0)
    dirBd.mark(boundaries, 1)
    dirBd2.mark(boundaries, 2)
    ds = Measure("ds")(subdomain_data=boundaries)
    bcd = [DirichletBC(Vvec.sub(0), 0.0, boundaries, 1),\
           DirichletBC(Vvec.sub(1), 0.0, dirBd2,method='pointwise')]
    Load = [Point(0.0, 1.0)]
    # Initialize level set function
    phi_mat = -np.cos(4.0/lx*pi*XX) * np.cos(4.0*pi*YY) - 0.4\
    + np.maximum(100.0*(XX+YY-lx-ly+0.1),.0)+ np.minimum(5.0/ly *(YY-1.0)
       + 4.0,0)

  if Name == 'cantilever_twoforces':
    ########################
    # Cantilever two forces
    ########################
    lx,ly = [1.0,1.0]
    XX,YY = np.meshgrid(np.linspace(0.0,lx,Nx+1),np.linspace(0.0,ly,Ny+1))
    #define boundary conditions
    mesh = RectangleMesh(Point(0.0,0.0),Point(lx,ly),Nx,Ny,'crossed')
    Vvec = VectorFunctionSpace(mesh, 'CG', 1)
    class DirBd(SubDomain):
      def inside(self, x, on_boundary):
        return near(x[0],.0)
    dirBd = DirBd()
    boundaries = FacetFunction("size_t", mesh)
    boundaries.set_all(0)
    dirBd.mark(boundaries, 1)
    ds = Measure("ds")(subdomain_data=boundaries)
    bcd = [DirichletBC(Vvec, (0.0,0.0), boundaries, 1)]
    Load = [Point(lx, 0.0),Point(lx, 1.0)]
    # Initialize level set function
    phi_mat = -np.cos(4.0*pi*(XX-0.5)) * np.cos(4.0*pi*(YY-0.5)) - 0.6 \
      - np.maximum(50.0*(YY-ly+0.1),.0)- np.maximum(50.0*(-YY+0.1),.0)

  return Lag,Nx,Ny,lx,ly,Load,Name,ds,bcd,mesh,phi_mat,Vvec
```

9.4 compliance.py

```
# ----------------------------------------------------------------------
# FEniCS 2017.1 code for level set-based structural optimization.
#
# ----------------------------------------------------------------------
from dolfin import *
from init import *
from matplotlib import cm, pyplot as pp
import numpy as np, sys, os
pp.switch_backend('Agg')
```

```
set_log_level(ERROR)
# ----------------------------------------------------------------------
def _main():
  Lag, Nx, Ny, lx, ly, Load, Name, ds, bcd, mesh, phi_mat, Vvec = init(sys.argv[1])
  eps_er, E, nu = [0.001, 1.0, 0.3] # Elasticity parameters
  mu, lmbda = Constant(E / (2 * (1 + nu))), Constant(E * nu / ((1 + nu) * (1 - 2 * nu)))
  # Create folder for saving files
  rd = os.path.join(os.path.dirname(__file__), \
  Name + '/LagVol=' + str(np.int_(Lag)) + '_Nx=' + str(Nx))
  if not os.path.isdir(rd): os.makedirs(rd)
  # Line search parameters
  beta0_init, ls, ls_max, gamma, gamma2 = [0.5, 0, 3, 0.8, 0.8]
  beta0 = beta0_init
  beta = beta0
  # Stopping criterion parameters
  ItMax, It, stop = [int(1.5 * Nx), 0, False]
  # Cost functional and function space
  J = np.zeros(ItMax)
  V = FunctionSpace(mesh, 'CG', 1)
  VolUnit = project(Expression('1.0', degree=2), V) # to compute volume
  U = [0] * len(Load) # initialize U
  # Get vertices coordinates
  gdim = mesh.geometry().dim()
  dofsV = V.tabulate_dof_coordinates().reshape((-1, gdim))
  dofsVvec = Vvec.tabulate_dof_coordinates().reshape((-1, gdim))
  px, py = [(dofsV[:, 0] / lx) * 2 * Nx, (dofsV[:, 1] / ly) * 2 * Ny]
  pxvec, pyvec = [(dofsVvec[:, 0] / lx) * 2 * Nx, (dofsVvec[:, 1] / ly) * 2 * Ny]
  dofsV_max, dofsVvec_max = ((Nx + 1) * (Ny + 1) + Nx * Ny) * np.array([1, 2])
  # Initialize phi
  phi = Function(V)
  phi = _comp_lsf(px, py, phi, phi_mat, dofsV_max)
  # Define Omega = {phi<0}
  class Omega(SubDomain):
    def inside(self, x, on_boundary):
      return .0 <= x[0] <= lx and .0 <= x[1] <= ly and phi(x) < 0
  domains = CellFunction("size_t", mesh)
  dX = Measure('dx')
  n = FacetNormal(mesh)
  # Define solver to compute descent direction th
  theta, xi = [TrialFunction(Vvec), TestFunction(Vvec)]
  av = assemble((inner(grad(theta), grad(xi)) + 0.1 * inner(theta, xi)) * dX \
    + 1.0e4 * (inner(dot(theta, n), dot(xi, n)) * (ds(0) + ds(1) + ds(2))))
  solverav = LUSolver(av)
  solverav.parameters['reuse_factorization'] = True
  # ---------- MAIN LOOP ----------------------------------------------
  while It < ItMax and stop == False:
    # Update and tag Omega = {phi<0}, then solve elasticity system.
    omega = Omega()
    domains.set_all(0)
    omega.mark(domains, 1)
    dx = Measure('dx')(subdomain_data=domains)
    for k in range(0, len(Load)):
      U[k] = _solve_pde(Vvec, dx, ds, eps_er, bcd, mu, lmbda, Load[k])
```

```
  # Update cost functional
  compliance = 0
  for u in U:
    eU = sym(grad(u))
    S1 = 2.0 * mu * inner(eU, eU) + lmbda * tr(eU) ** 2
    compliance += assemble(eps_er * S1 * dx(0) + S1 * dx(1))
  vol = assemble(VolUnit * dx(1))
  J[It] = compliance + Lag * vol
  # ------- LINE SEARCH ----------------------------------------
  if It > 0 and J[It] > J[It - 1] and ls < ls_max:
    ls += 1
    beta *= gamma
    phi_mat, phi = [phi_mat_old, phi_old]
    phi_mat = _hj(th_mat, phi_mat, lx, ly, Nx, Ny, beta)
    phi = _comp_lsf(px, py, phi, phi_mat, dofsV_max)
    print('Line search iteration : %s' % ls)
  else:
    print('************ ITERATION NUMBER %s' % It)
    print('Function value : %.2f' % J[It])
    print('Compliance     : %.2f' % compliance)
    print('Volume fraction : %.2f' % (vol / (lx * ly)))
    # Decrease or increase line search step
    if ls == ls_max: beta0 = max(beta0 * gamma2, 0.1 * beta0_init)
    if ls == 0:  beta0 = min(beta0 / gamma2, 1)
    # Reset beta and line search index
    ls, beta, It = [0, beta0, It + 1]
    # Compute the descent direction th
    th = _shape_der(Vvec, U, eps_er, mu, lmbda, dx, solverav, Lag)
    th_array = th.vector().array()
    th_mat = [np.zeros((Ny + 1, Nx + 1)), np.zeros((Ny + 1, Nx + 1))]
    for dof in xrange(0, dofsVvec_max, 2):
      if np.rint(pxvec[dof]) % 2 == .0:
        cx, cy = np.int_(np.rint([pxvec[dof] / 2, pyvec[dof] / 2]))
        th_mat[0][cy, cx] = th_array[dof]
        th_mat[1][cy, cx] = th_array[dof + 1]
    # Update level set function phi using descent direction th
    phi_old, phi_mat_old = [phi, phi_mat]
    phi_mat = _hj(th_mat, phi_mat, lx, ly, Nx, Ny, beta)
    if np.mod(It, 5) == 0: phi_mat = _reinit(lx, ly, Nx, Ny, phi_mat)
    phi = _comp_lsf(px, py, phi, phi_mat, dofsV_max)
    # ------------ STOPPING CRITERION --------------------------
    if It > 20 and max(abs(J[It - 5:It] - J[It - 1])) < 2.0 * J[It - 1] /
        Nx ** 2:
      stop = True
    # ------------ Plot Geometry -------------------------------
    if np.mod(It, 10) == 0 or It == 1 or It == ItMax or stop == True:
      pp.close()
      pp.contourf(phi_mat, [-10.0, .0], extent=[.0, lx, .0, ly], \
        cmap=cm.get_cmap('bone'))
      pp.axes().set_aspect('equal', 'box')
      pp.show()
      pp.savefig(rd + '/it_' + str(It) + '.pdf', bbox_inches='tight')
  return
# ----------------------------------------------------------------------
def _solve_pde(V, dx, ds, eps_er, bcd, mu, lmbda, Load):
  u, v = [TrialFunction(V), TestFunction(V)]
  S1 = 2.0 * mu * inner(sym(grad(u)), sym(grad(v))) + lmbda * div(u) *
      div(v)
  A = assemble(S1 * eps_er * dx(0) + S1 * dx(1))
```

```
  b = assemble(inner(Expression(('0.0', '0.0'), degree=2), v) * ds(2))
  U = Function(V)
  delta = PointSource(V.sub(1), Load, -1.0)
  delta.apply(b)
  for bc in bcd: bc.apply(A, b)
  solver = LUSolver(A)
  solver.solve(U.vector(), b)
  return U
# ----------------------------------------------------------------------
def _shape_der(Vvec, u_vec, eps_er, mu, lmbda, dx, solver, Lag):
  xi = TestFunction(Vvec)
  rv = 0.0
  for u in u_vec:
    eu, Du, Dxi = [sym(grad(u)), grad(u), grad(xi)]
    S1 = 2 * mu * (2 * inner((Du.T) * eu, Dxi) - inner(eu, eu) * div(xi)) \
      + lmbda * (2 * inner(Du.T, Dxi) * div(u) - div(u) * div(u) * div(xi))
      rv += -assemble(eps_er * S1 * dx(0) + S1 * dx(1) + Lag * div(xi) * dx(1))
  th = Function(Vvec)
  solver.solve(th.vector(), rv)
  return th
# ----------------------------------------------------------------------
def _hj(v, psi, lx, ly, Nx, Ny, beta):
  for k in range(10):
    Dym = Ny * np.repeat(np.diff(psi, axis=0), [2] + [1] * (Ny - 1), axis=0) / ly
    Dyp = Ny * np.repeat(np.diff(psi, axis=0), [1] * (Ny - 1) + [2], axis=0) / ly
    Dxm = Nx * np.repeat(np.diff(psi), [2] + [1] * (Nx - 1), axis=1) / lx
    Dxp = Nx * np.repeat(np.diff(psi), [1] * (Nx - 1) + [2], axis=1) / lx
    g = 0.5 * (v[0] * (Dxp + Dxm) + v[1] * (Dyp + Dym)) \
      - 0.5 * (np.abs(v[0]) * (Dxp - Dxm) + np.abs(v[1]) * (Dyp - Dym))
    maxv = np.max(abs(v[0]) + abs(v[1]))
    dt = beta * lx / (Nx * maxv)
    psi = psi - dt * g
  return psi
# ----------------------------------------------------------------------
def _reinit(lx, ly, Nx, Ny, psi):
  Dxs = Nx * (np.repeat(np.diff(psi), [2] + [1] * (Nx - 1), axis=1) \
        + np.repeat(np.diff(psi), [1] * (Nx - 1) + [2], axis=1)) / (2 * lx)
  Dys = Ny * (np.repeat(np.diff(psi, axis=0), [2] + [1] * (Ny - 1), axis=0) \
        + np.repeat(np.diff(psi, axis=0), [1] * (Ny - 1) + [2], axis=0)) / (2 * ly)
  signum = psi / np.power(psi ** 2 + ((lx / Nx) ** 2) * (Dxs ** 2 + Dys ** 2), 0.5)
  for k in range(0, 2):
    Dym = Ny * np.repeat(np.diff(psi, axis=0), [2] + [1] * (Ny - 1), axis=0) / ly
    Dyp = Ny * np.repeat(np.diff(psi, axis=0), [1] * (Ny - 1) + [2], axis=0) / ly
    Dxm = Nx * np.repeat(np.diff(psi), [2] + [1] * (Nx - 1), axis=1) / lx
    Dxp = Nx * np.repeat(np.diff(psi), [1] * (Nx - 1) + [2], axis=1) / lx
    Kp = np.sqrt((np.maximum(Dxm, 0)) ** 2 + (np.minimum(Dxp, 0)) ** 2 \
          + (np.maximum(Dym, 0)) ** 2 + (np.minimum(Dyp, 0)) ** 2)
    Km = np.sqrt((np.minimum(Dxm, 0)) ** 2 + (np.maximum(Dxp, 0)) ** 2 \
```

```
            + (np.minimum(Dym, 0)) ** 2 + (np.maximum(Dyp, 0)) ** 2)
    g = np.maximum(signum, 0) * Kp + np.minimum(signum, 0) * Km
    psi = psi - (0.5 * lx / Nx) * (g - signum)
  return psi
# ----------------------------------------------------------------------
def _comp_lsf(px, py, phi, phi_mat, dofsV_max):
  for dof in range(0, dofsV_max):
    if np.rint(px[dof]) % 2 == .0:
      cx, cy = np.int_(np.rint([px[dof] / 2, py[dof] / 2]))
      phi.vector()[dof] = phi_mat[cy, cx]
    else:
      cx, cy = np.int_(np.floor([px[dof] / 2, py[dof] / 2]))
      phi.vector()[dof] = 0.25 * (phi_mat[cy, cx] + phi_mat[cy + 1, cx] \
      + phi_mat[cy, cx + 1] + phi_mat[cy + 1, cx + 1])
  return phi
```

Referências Bibliográficas

[1] Grégoire Allaire, François Jouve, and Anca-Maria Toader. Structural optimization using sensitivity analysis and a level-set method. *J. Comput. Phys.*, 194(1):363–393, 2004. 31, 36

[2] Martin Alnæs, Jan Blechta, Johan Hake, August Johansson, Benjamin Kehlet, Anders Logg, Chris Richardson, Johannes Ring, Marie Rognes, and Garth Wells. The fenics project version 1.5. *Archive of Numerical Software*, 3(100), 2015. 43

[3] Martin Berggren. A unified discrete-continuous sensitivity analysis method for shape optimization. In *Applied and numerical partial differential equations*, volume 15 of *Comput. Methods Appl. Sci.*, pages 25–39. Springer, New York, 2010. 1

[4] Marc Dambrine and Djalil Kateb. On the ersatz material approximation in level-set methods. *ESAIM Control Optim. Calc. Var.*, 16(3):618–634, 2010. 31

[5] Frédéric de Gournay. Velocity extension for the level-set method and multiple eigenvalues in shape optimization. *SIAM J. Control Optim.*, 45(1):343–367, 2006. 37

[6] M. C. Delfour and J.-P. Zolésio. *Shapes and geometries*, volume 22 of *Advances in Design and Control*. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, second edition, 2011. Metrics, analysis, differential calculus, and optimization. 9, 17

[7] R. Hiptmair, A. Paganini, and S. Sargheini. Comparison of approximate shape gradients. *BIT*, 55(2):459–485, 2015. 1

[8] H.P. Langtangen and A. Logg. *Solving PDEs in Python: The FEniCS Tutorial I*. Simula SpringerBriefs on Computing. Springer International Publishing, 2017. 43

[9] Antoine Laurain. A level set-based structural optimization code using FEniCS. *Struct. Multidiscip. Optim.*, 58(3):1311–1334, 2018.

[10] Antoine Laurain and Kevin Sturm. Distributed shape derivative *via* averaged adjoint method and applications. *ESAIM Math. Model. Numer. Anal.*, 50(4):1241–1267, 2016. 15

[11] A. Logg, K.-A. Mardal, and G. N. Wells, editors. *Automated Solution of Differential Equations by the Finite Element Method*, volume 84 of *Lecture Notes in Computational Science and Engineering*. Springer, 2012. 43

[12] Mitio Nagumo. Über die Lage der Integralkurven gewöhnlicher Differentialgleichungen. *Proc. Phys.-Math. Soc. Japan (3)*, 24:551–559, 1942. 6

[13] Antonio André Novotny and Jan Sokoł owski. *Topological derivatives in shape optimization*. Interaction of Mechanics and Mathematics. Springer, Heidelberg, 2013. 21

[14] Stanley Osher and Ronald Fedkiw. *Level set methods and dynamic implicit surfaces*, volume 153 of *Applied Mathematical Sciences*. Springer-Verlag, New York, 2003. 41

[15] Stanley Osher and James A. Sethian. Fronts propagating with curvature-dependent speed: algorithms based on Hamilton-Jacobi formulations. *J. Comput. Phys.*, 79(1):12–49, 1988. 39, 41

[16] Stanley Osher and Chi-Wang Shu. High-order essentially nonoscillatory schemes for Hamilton-Jacobi equations. *SIAM J. Numer. Anal.*, 28(4):907–922, 1991. 41

[17] Danping Peng, Barry Merriman, Stanley Osher, Hongkai Zhao, and Myungjoo Kang. A PDE-based fast local level set method. *J. Comput. Phys.*, 155(2):410–438, 1999. 41

[18] J. A. Sethian. *Level set methods and fast marching methods*, volume 3 of *Cambridge Monographs on Applied and Computational Mathematics*. Cambridge University Press, Cambridge, second edition, 1999. Evolving interfaces in computational geometry, fluid mechanics, computer vision, and materials science. 41

[19] O. Sigmund. A 99 line topology optimization code written in matlab. *Structural and Multidisciplinary Optimization*, 21(2):120–127, 2014. 1, 46

[20] Kevin Sturm. Minimax Lagrangian approach to the differentiability of nonlinear PDE constrained shape functions without saddle point assumption. *SIAM J. Control Optim.*, 53(4):2017–2039, 2015. 15, 17

[21] Michael Yu Wang, Xiaoming Wang, and Dongming Guo. A level set method for structural topology optimization. *Comput. Methods Appl. Mech. Engrg.*, 192(1-2):227–246, 2003. 31