BUSCA DIRETA EM MINIMIZAÇÃO IRRESTRITA

Lucas Garcia Pedroso * Maria Aparecida Diniz-Ehrhardt

DMA – IMECC – UNICAMP

Resumo

Neste trabalho, voltamos nossa atenção para estratégias de busca direta, que são métodos de minimização que não fazem uso de derivadas ou de suas aproximações. Abordamos um algoritmo proposto por Lucidi e Sciandrone para problemas irrestritos, que usa um critério de decréscimo suficiente para garantir convergência global, no sentido que todo ponto de acumulação da seqüência de aproximações para o minimizador é um ponto estacionário do problema. Tal algoritmo mescla dois diferentes tipos de métodos de busca direta, a saber, busca linear e busca padrão, com o propósito de aproveitar as vantagens de cada estratégia. Motivados pelos interessantes resultados teóricos deste trabalho, realizamos alguns testes computacionais, especialmente em problemas clássicos de minimização irrestrita.

Abstract

In this work we are interested in direct search procedures: methods for unconstrained minimization that do not require derivatives of the objective function, neither their approximations. We consider an algorithm due to Lucidi and Sciandrone, in which global convergence is guaranteed by using a sufficient decrease of f. The algorithm combines two different subclasses of direct search methods, that is, line search and pattern search methods, taking the advantages of each different strategy. Its interesting theoretical results motivated us to report numerical experiments with the objective of analysing its computational performance.

*Bolsista de mestrado da Capes entre março e agosto de 2003 e da Fapesp entre setembro de 2003 e fevereiro de 2005.

1 Introdução

Neste trabalho, consideraremos o problema irrestrito

$$Minimizar f(x), \tag{1}$$

onde $f : \mathbb{R}^n \to \mathbb{R}$ é continuamente diferenciável. Dentre os diversos métodos para tratar o problema (1), os mais usuais são baseados no cálculo de derivadas de $f(x)$. Citemos, por exemplo, o método de Newton e os métodos Quase-Newton [1, 9]. Apesar da indiscutível eficiência de tais algoritmos, parte da comunidade científica trabalha em métodos que não usam gradientes ou mesmo aproximações locais para a função f, conhecidos como *métodos de busca direta*.

O termo busca direta surgiu em 1961 em um artigo de Robert Hooke e T. A. Jeeves [5]. Refere-se a um método iterativo em que um conjunto de pontos é testado a cada iteração, associado a uma estratégia que usa somente avaliações da função f para definir a aproximação seguinte para o minimizador.

Desde então, a pesquisa em torno da busca direta aponta na direção da análise de sua confiabilidade. Longe de querer competir com os métodos clássicos de minimização, os algoritmos de busca direta são de crescente interesse por apresentarem, em geral, bons resultados teóricos de convergência, por serem fáceis de implementar e por representarem uma alternativa em ocasiões onde métodos mais elaborados falham.

Dentre todos os métodos que não usam derivadas, o de Nelder-Mead [12] está entre os mais utilizados. Essa popularidade reside nos bons resultados práticos, especialmente se levada em conta a simplicidade do algoritmo. A partir de um simplex no $\mathbb{R}^n$, a idéia do método é, a cada iteração, substituir o vértice com o valor menos desejado da função objetivo ou, quando isso não for possível, reduzir as dimensões do simplex. Apesar de não possuir resultados expressivos de convergência e de apresentar comportamento insatisfatório em alguns problemas [17], a relevância do método de Nelder-Mead ainda é grande, pois é consagrada por anos de uso na computação científica.

Mesmo tendo a difícil tarefa de competir com o algoritmo de Nelder-Mead, a pesquisa de novos métodos de busca direta que sejam mais confiáveis é crescente. Exemplo disso é o algoritmo proposto por Lucidi e Sciandrone [8]. Os autores se baseiam em diferentes tipos de método de busca direta com o objetivo de elaborar uma estratégia que apresente bons resultados teóricos. Um aspecto interessante do

algoritmo é a possibilidade de se escolher as direções de busca, desde que estas satisfaçam alguns critérios. O conveniente critério de decréscimo suficiente adotado dá ao algoritmo resultados de convergência global.

Na Seção 2 iremos discutir sobre a definição de busca direta e relacionar algumas de suas subclasses. Na Seção 3 trataremos do algoritmo proposto por Lucidi e Sciandrone [8], que possui promissores resultados de convergência. Na Seção 4, destacaremos os aspectos mais importantes de dois métodos que não usam derivadas: o método de Nelder-Mead e o Algoritmo das Direções Aleatórias [2]. Por fim, seguirão alguns experimentos numéricos e análise dos resultados (Seção 5).

2 Métodos de Busca Direta

É importante observar que o simples fato de um método não fazer uso de derivadas não é suficiente para caracterizá-lo como de busca direta. Métodos que não calculam derivadas são conhecidos como *derivative-free*. Em tais procedimentos, há liberdade para, por exemplo, aproximar o gradiente da função objetivo por diferenças finitas ou interpolar a função por um polinômio, com o intuito de atualizar a aproximação do minimizador para a iteração seguinte.

Dentro do amplo conjunto dos métodos *derivative-free*, estão os de busca direta. Um método de busca direta, além de não calcular ou aproximar as derivadas, não faz uso do valor explícito da função. Apenas a ordenação entre os pontos testados é necessária, por exemplo, dados os pontos $\{x_1, \ldots, x_m\}$ basta conhecer os índices $i_1, \ldots, i_m$ tais que

$$f(x_{i_1}) \leq f(x_{i_2}) \leq \cdots \leq f(x_{i_m}).$$

Dentre os diversos tipos de métodos, destacaremos três: métodos simplex, métodos de busca linear e métodos de busca padrão [7]. A maioria dos algoritmos encontrados na literatura se enquadra (ou ao menos se aproxima) de alguma dessas categorias.

2.1 Métodos Simplex

O primeiro método simplex é devido a Spendley, Hext e Himsworth (1962) [15]. Um simplex é um conjunto de $n+1$ pontos em $\mathbb{R}^n$. Trabalhamos sempre com simplex não degenerados, ou seja, se $\{x_1, \ldots, x_{n+1}\}$ são os pontos que definem o simplex (também conhecidos como vértices), pediremos que o conjunto $\{x_2 - x_1, \ldots, x_{n+1} - x_1\}$ seja linearmente independente. O valor da função nos

vértices é conhecido.

A idéia de utilizar apenas $n+1$ pontos por iteração para definir um algoritmo de busca direta é razoável, visto que $n+1$ pontos seriam suficientes, por exemplo, para aproximar o gradiente da função objetivo por diferenças finitas. Muitos métodos de busca direta calculam a função entre $2n$ e 2^n vezes por iteração, de onde já podemos deduzir uma vantagem dos métodos simplex: são econômicos, no que diz respeito a avaliações de função por iteração, quando comparados a outros métodos de busca direta.

A essência desses métodos está em substituir os vértices do simplex, em geral o pior deles (ou seja, aquele que tem o valor menos desejado da função objetivo). A maneira mais usual de fazê-lo é refletindo o ponto através do centróide da face oposta, como mostra o exemplo em $\mathbb{R}^2$.

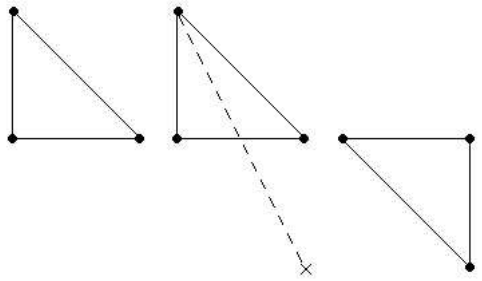

Figura 1: Reflexão de um vértice através do centróide da face oposta.

O mais famoso método simplex é sem dúvida o método de Nelder e Mead [12]. Abordaremos esse algoritmo na Seção 4.1.

2.2 Métodos de Busca Linear

Os métodos de busca linear têm espírito semelhante aos métodos baseados em derivadas. Para uma aproximação x_k, escolhemos uma direção d_k e associamos à iteração corrente a função de uma variável $\phi(\alpha) = f(x_k + \alpha d_k)$, $\alpha \in \mathbb{R}$. A intenção é encontrar α_k que defina o novo vetor $x_{k+1} = x_k + \alpha_k d_k$. Para tanto, é desejável que sejam feitas poucas avaliações de f, e é usual pedir que

$$f(x_{k+1}) \leq f(x_k).$$

Diferentes escolhas das direções e estratégias de encontrar α_k definem os diferentes métodos. A liberdade que esse tipo de algoritmo permite na escolha das direções

e passos exige, normalmente, o uso de alguns artifícios (decréscimo suficiente de f, por exemplo) para assegurar convergência global [14].

2.3 Métodos de Busca Padrão

Essa classe de algoritmos analisa a função objetivo em um padrão de pontos, para então decidir qual será a próxima aproximação para o minimizador. O padrão mais intuitivo é definido pelas coordenadas cartesianas.

O método das variações locais [13], também conhecido como método das direções alternadas ou busca coordenada, consiste em, a partir de um ponto x_k, buscar um ponto x_{k+1} onde haja decréscimo da função, utilizando para isso as direções coordenadas, como mostra a Figura (2).

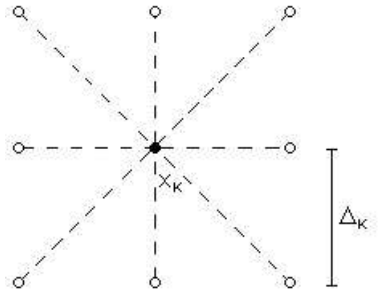

Figura 2: Padrão em $\mathbb{R}^2$ definido pelas direções coordenadas e pelo passo Δ_k.

Se nenhum candidato a x_{k+1} fornecer valor da função menor que $f(x_k)$ (fracasso), o tamanho do passo Δ_k é reduzido e um novo padrão de pontos é analisado. Caso contrário (sucesso), aceitamos x_{k+1} e, a partir dele, buscamos x_{k+2} com $f(x_{k+2}) < f(x_{k+1})$ e assim por diante. Já o algoritmo de Hooke e Jeeves [5], que introduziu na comunidade científica o conceito de busca direta, é semelhante ao de variações locais, exceto pelo fato de que, quando há sucesso em uma iteração k, não consideramos, para a próxima iteração, o padrão de pontos ao redor de x_{k+1}, e sim o padrão de pontos ao redor de $x_{k+1} + (x_{k+1} - x_k)$. A idéia é aproveitar a direção $x_{k+1} - x_k$, que é uma direção promissora. Ver figura (3)

Há vários trabalhos que abordam a questão da convergência para cada algoritmo de busca padrão. Mas um artigo de V. Torczon [16] sintetiza a essência de todos os métodos de busca padrão e demonstra a sua convergência. Neste trabalho, a autora mostra que, para uma iteração k e uma aproximação x_k para a solução, são

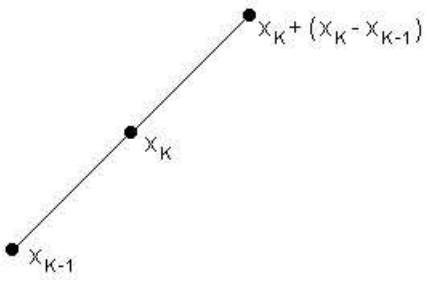

Figura 3: Ponto que definirá o padrão de pontos em $\mathbb{R}^2$ para o método de Hooke e Jeeves.

necessárias duas componentes para se definir um padrão: uma base $B \in \mathbb{R}^{n\times n}$ e uma matriz geradora $C_k \in Z^{n\times p}, p > 2n$. C_k deve ter a forma

$$C_k = [\ M_k \quad -M_k \quad L_k\ ] = [c_k^1 \cdots c_k^p],$$

com $M_k \in M \subset Z^{n\times n}$, onde M é um conjunto finito de matrizes não singulares e $L_k \in Z^{n\times(p-2n)}$ deve conter pelo menos uma coluna de zeros. O padrão de pontos é definido pelas colunas da matriz $P_k = BC_k$, de onde, dado o passo Δ_k, os candidatos a x_{k+1} são $\bar{x}_{k+1}^i = x_k + \Delta_k B c_k^i$. O passo Δ_k é reduzido por um fator pré-definido quando há fracasso ou pode até aumentar (dependendo do método) em caso de sucesso, mas sempre multiplicado por um conjunto finito e pré-definido de fatores.

Se tivermos um algoritmo que possa ser decomposto nas componentes descritas acima, e se, além disso, ele satisfizer algumas condições sobre os passos exploratórios e sobre o processo de atualização de Δ_k, esse algoritmo pode ser considerado um algoritmo de busca padrão. A autora demonstra que, sob certas condições, a seqüência $\{x_k\}$ satisfaz

$$\liminf_{k\to\infty} ||\nabla f(x_k)|| = 0.$$

Se além disso, forem impostas hipóteses extras sobre o algoritmo (que são naturalmente satisfeitas para diversos métodos de busca padrão), fica garantida a convergência forte, no sentido que

$$\lim_{k\to\infty} ||\nabla f(x_k)|| = 0.$$

Uma particularidade da busca padrão é que esta não necessita de critérios de decréscimo suficiente para garantir a convergência.

Há interessantes trabalhos que mesclam idéias de diferentes tipos de métodos de busca direta, almejando aproveitar as vantagens de cada um em algoritmos mais

competitivos em relação aos métodos baseados em derivadas. Um exemplo desses algoritmos híbridos é o proposto por Lucidi e Sciandrone, abordado na próxima seção.

3 Algoritmo de Lucidi e Sciandrone

Os métodos de busca padrão costumam avaliar a função objetivo em várias direções diferentes a cada iteração. Isso pode dar uma boa idéia do comportamento local de f em torno de um ponto x_k, tornando possível identificar boas direções (direções onde há decréscimo da função). Já os métodos de busca linear, podem dar passos grandes quando boas direções são encontradas. A idéia do trabalho de Lucidi e Sciandrone [8] foi juntar elementos desses dois tipos de métodos para criar um esquema de algoritmo globalmente convergente. Assumiremos sempre que a função objetivo satisfaz as hipóteses abaixo:

1. $f(x)$ é continuamente diferenciável,
2. $L(x_0)$ é compacto, onde $L(x_0) = \{x \in I\!R^n \mid f(x) \leq f(x_0)\}$, e x_0 é o chute inicial.

Os autores afirmam que um bom conjunto de direções para um método de busca direta deve, de alguma maneira, suprir a falta do gradiente, que é o melhor termômetro do comportamento local da função. Tais direções devem ser suficientes para caracterizar se um ponto é estacionário, caso contrário devem indicar uma direção de descida a partir deste ponto. Se o conjunto de direções p_k^i, $i = 1, \ldots, r$ satisfizer a condição C1 abaixo, saberemos que se trata de direções que preencherão a lacuna deixada pelo não uso das derivadas:

Condição C1: Dada uma seqüência de pontos $\{x_k\}$, as seqüências de direções $\{p_k^i\}$, $i = 1, \ldots, r$, são limitadas e tais que

$$\lim_{k\to\infty} ||\nabla f(x_k)|| = 0 \Leftrightarrow \lim_{k\to\infty} \sum_{i=1}^{r} \min\{0, \nabla f(x_k)^T p_k^i\} = 0.$$

Dois exemplos de direções de busca que satisfazem C1 são:

1. p_k^i tais que as seqüências $\{p_k^i\}$, com $i = 1, \ldots, r$ são limitadas, e todo ponto limite $(\bar{p}^1, \ldots, \bar{p}^r)$ da seqüência $\{(p_k^1, \ldots, p_k^r)\}$ é tal que os vetores $\bar{p}^i$ geram

positivamente o $\mathbb{R}^n$. Direções que satisfazem essas condições são fáceis de construir. Basta tomar, por exemplo

$$p_k^i = e^i, \qquad i = 1, \ldots, n$$

e fazer

$$p_k^{n+i} = -e^i, \qquad i = 1, \ldots, n$$

ou

$$p_k^{n+1} = -\sum_{i=1}^{n} e^i.$$

2. p_k^i, $i = 1, \ldots, n$ uniformemente linearmente independente e p_k^{n+1} da forma

$$p_k^{n+1} = \frac{x_k - x_k^{max}}{\xi_k},$$

onde $x_k^{max} = \text{argmax}_{i=1,\ldots,n}\{f(x_k + \xi_k p_k^i)\}$ e $\xi_k \to 0$ quando $k \to \infty$.

A proposição abaixo descreve um conjunto de condições para convergência global:

Proposição 1 *Sejam $\{x_k\}$ uma seqüência de pontos, $\{p_k^i\}$, $i = 1, \ldots, r$ seqüências de direções e suponha que as seguintes condições são satisfeitas:*

1. $f(x_{k+1}) \leq f(x_k)$,
2. $\{p_k^i\}$, $i = 1, \ldots, r$ *satisfaz a condição C1,*
3. *Existem seqüências de pontos $\{y_k^i\}$ e seqüências de escalares positivos ξ_k^i, $i = 1, \ldots, r$ tais que $f(y_k^i + \xi_k^i p_k^i) \geq f(y_k^i) - o(\xi_k^i)$, $\lim_{k\to\infty} \xi_k^i = 0$ e $\lim_{k\to\infty} ||x_k - y_k^i|| = 0$.*

Então $\lim_{k\to\infty} ||\nabla f(x_k)|| = 0$.

Demonstração: Ver [8].

Esta proposição, juntamente com a condição C1, cria um molde de algoritmos de busca direta globalmente convergentes: se as direções de busca do algoritmo satisfazem C1, basta demonstrar que a seqüência $\{x_k\}$ gerada por este satisfaz a Proposição 1. E como esta resume algumas hipóteses comuns à busca padrão e busca linear, esta proposição é adequada para guiar a definição de algoritmos mistos. Dois algoritmos deste tipo são propostos pelos autores. A diferença entre eles está na escolha das direções. No algoritmo abaixo, pedimos que as seqüências $\{p_k^i\}$, $i = 1, \ldots, r$, satisfaçam a condição C1.

Algoritmo 1: Sejam $k = 0$, $x_0 \in \mathbb{R}^n$, $\tilde{\alpha}_0^i > 0$, $i = 1, \ldots, r$, $\gamma > 0$, δ, $\theta \in (0, 1)$.

1. Faça $i = 1$ e $y_k^1 = x_k$.

2. Se $f(y_k^i + \tilde{\alpha}_k^i p_k^i) \leq f(y_k^i) - \gamma(\tilde{\alpha}_k^i)^2$, então
   calcule α_k^i por **Procedimento LS**$(\tilde{\alpha}_k^i, y_k^i, p_k^i, \gamma, \delta)$
   e faça $\tilde{\alpha}_{k+1}^i = \alpha_k^i$;
   caso contrário $\alpha_k^i = 0$ e $\tilde{\alpha}_{k+1}^i = \theta\tilde{\alpha}_k^i$.
   Faça $y_k^{i+1} = y_k^i + \alpha_k^i p_k^i$.

3. Se $i < r$, faça $i = i + 1$ e volte para o passo 2.

4. Encontre x_{k+1} tal que
   $f(x_{k+1}) \leq f(y_k^{r+1})$,
   faça $k = k + 1$, e volte para o passo 1.

Procedimento LS$(\tilde{\alpha}_k^i, y_k^i, p_k^i, \gamma, \delta)$: Encontre $\alpha_k^i = \min\{\delta^{-j}\tilde{\alpha}_k^i : j = 0, 1, \ldots\}$ tal que

$$f(y_k^i + \alpha_k^i p_k^i) \leq f(y_k^i) - \gamma(\alpha_k^i)^2,$$

$$f(y_k^i + \tfrac{\alpha_k^i}{\delta} p_k^i) \geq \max\{f(y_k^i + \alpha_k^i p_k^i), f(y_k^i) - \gamma(\tfrac{\alpha_k^i}{\delta})^2\}.$$

A cada iteração k, o comportamento da função é analisado em todas as direções de busca p_k^i. Quando algum p_k^i cumpre o critério de decréscimo suficiente, o procedimento de busca linear tenta dar um passo significativo nesta direção. A imposição de decréscimo suficiente permite uma liberdade maior na escolha das direções.

É interessante observar que, para cada $i \in [1, 2, \ldots, r]$, adotamos uma seqüência de passos $\{\alpha_k^i\}_{k=1}^{\infty}$ diferente. Isso permite que o comportamento da função em cada um dos r conjuntos de direções $\{p_k^i\}_{k=1}^{\infty}$ seja analisado independentemente, fato útil principalmente se as direções de busca são as mesmas a cada iteração $(p_k^i = \bar{p}^i, i = 1, \ldots, r)$.

Por fim, observamos que, no Passo 4, há a liberdade para o uso de qualquer esquema de extrapolação, que pode ser uma boa idéia para se obter melhores resultados práticos.

O Algoritmo 1 goza do seguinte teorema de convergência :

Teorema 1 *Seja $\{x_k\}$ uma seqüência gerada pelo Algoritmo 1. Suponha que as seqüências de direções $\{p_k^i\}_{i=1}^r$ satisfazem a condição C1. Então o Algoritmo 1 está bem definido e temos que*

$$\lim_{k\to\infty} ||\nabla f(x_k)|| = 0.$$

Uma demonstração do teorema acima está intimamente relacionada com a Proposição 1 [8]. Com a expressão "bem definido", queremos dizer que o procedimento de busca linear sempre termina em uma quantidade finita de passos.

O algoritmo seguinte tem a mesma essência de exploração das direções do Algoritmo 1, diferindo apenas no conjunto de direções. As direções $\{p_k^i\}$, $i = 1, \ldots, n$ devem ser uniformemente linearmente independentes.

Algoritmo 2: Sejam $k = 0$, $x_0 \in I\!R^n$, $c > 0$, $\tilde{\alpha}_0^i > 0$, $i = 1, \ldots, n+1$, $\gamma > 0$, δ, $\theta \in (0, 1)$.

1. Faça $i = 1$ e $y_k^1 = x_k$, $V_k = \{y_k^1\}$, $S_k = \{\emptyset\}$.

2. Se $f(y_k^i + \tilde{\alpha}_k^i p_k^i) \leq f(y_k^i) - \gamma(\tilde{\alpha}_k^i)^2$, então
   calcule α_k^i por **Procedimento LS**$(\tilde{\alpha}_k^i, y_k^i, p_k^i, \gamma, \delta)$
   e faça $\tilde{\alpha}_{k+1}^i = \alpha_k^i$, $V_k = V_k \cup \{y_k^i + \alpha_k^i p_k^i\}$, $S_k = S_k \cup \{\alpha_k^i\}$;
   caso contrário $\alpha_k^i = 0$ e $\tilde{\alpha}_{k+1}^i = \theta\tilde{\alpha}_k^i$, $V_k = V_k \cup \{y_k^i + \tilde{\alpha}_k^i p_k^i\}$, $S_k = S_k \cup \{\tilde{\alpha}_k^i\}$.
   Faça $y_k^{i+1} = y_k^i + \alpha_k^i p_k^i$.

3. Se $i < n$, faça $i = i + 1$ e volte para o passo 2.

4. Calcule $\alpha_k^{min} = \min_{\alpha\in S_k}\{\alpha\}$ e $\alpha_k^{max} = \max_{\alpha\in S_k}\{\alpha\}$.
   Se $\frac{\alpha_k^{max}}{\alpha_k^{min}} \leq c$, então calcule p_k^{n+1} tal que

$$p_k^{n+1} = \frac{v_k^{min} - v_k^{max}}{\xi_k},$$

   onde $v_k^{max} = \text{argmax}_{v\in V_k}\{f(v)\}$,
   $v_k^{min} = \text{argmin}_{v\in V_k}\{f(v)\}$, e
   $\xi_k \in [\alpha_k^{min}, \alpha_k^{max}]$;
   caso contrário faça

$$p_k^{n+1} = -\sum_{i=1}^{n} p_k^i.$$

5. Se $f(y_k^n + \tilde{\alpha}_k^{n+1} p_k^{n+1}) \leq f(y_k^n) - \gamma(\tilde{\alpha}_k^{n+1})^2$, então
   calcule α_k^{n+1} por **Procedimento LS**$(\tilde{\alpha}_k^{n+1}, y_k^n, p_k^{n+1}, \gamma, \delta)$

e faça $\tilde{\alpha}_{k+1}^{n+1} = \alpha_k^{n+1}$;
caso contrário $\alpha_k^{n+1} = 0$ e $\tilde{\alpha}_{k+1}^{n+1} = \theta\tilde{\alpha}_k^{n+1}$.
Faça $y_k^{n+1} = y_k^n + \alpha_k^{n+1} p_k^{n+1}$.

6. Encontre x_{k+1} tal que
$f(x_{k+1}) \leq f(y_k^{n+1})$,
faça $k = k + 1$, e volte para o passo 1.

Os passos 1 a 3 do algoritmo acima são essencialmente os mesmos do algoritmo 1. No passo 4, é verificado se os passos dados em cada direção são da mesma ordem de grandeza, o que nos diria se o conjunto de pontos visitados durante a iteração nos dá uma boa noção do comportamento local de f. Em caso afirmativo, uma direção p_k^{n+1} é tomada com a intenção de aproximar a direção de máxima descida. Caso contrário, p_k^{n+1} é tal que $\{p_k^1, p_k^2, \ldots, p_k^{n+1}\}$ gere positivamente o $\mathbb{R}^n$. Assim procedendo, teremos seqüências de direções $\{p_k^i\}_{i=1}^{n+1}$ que satisfazem C1.

O Passo 5 é como uma repetição do Passo 2, agora para a direção p_k^{n+1}. O Passo 6 representa, novamente, a possibilidade de extrapolação.

Para o Algoritmo 2 há o seguinte teorema, demonstrado em [8]:

Teorema 2 *Seja* $\{x_k\}$ *uma seqüência gerada pelo* Algoritmo 2. *Suponha que os vetores* $\{p_k^i\}$, *com* $i = 1, \ldots, n$, *são limitados e uniformemente linearmente independentes. Então o* Algoritmo 2 *está bem definido e temos*

$$\lim_{k\to\infty} ||\nabla f(x_k)|| = 0.$$

4 Outros Algoritmos

Nesta seção, descreveremos outros dois algoritmos de busca direta. São eles o Método de Nelder-Mead [12] e o Método das Direções Aleatórias [2].

4.1 Método de Nelder-Mead

O método de Nelder-Mead [12], publicado em 1965, é provavelmente o mais utilizado método de busca direta. A contribuição do algoritmo Nelder-Mead à classe dos métodos simplex reside na possibilidade de contração ou expansão do simplex, além da reflexão, como mostra a figura (4).

São necessários 4 coeficientes escalares no algoritmo de Nelder-Mead. São eles:

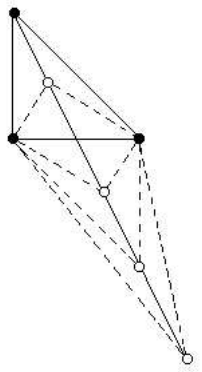

Figura 4: Possibilidades de atualização do simplex no método de Nelder-Mead.

- Coeficiente de reflexão: $\rho > 0$,
- Coeficiente de expansão: $\chi > 1$ com $\chi > \rho$,
- Coeficiente de contração: $0 < \gamma < 1$ e
- Coeficiente de redução (*shrink*): $0 < \sigma < 1$.

Para uma função $f(x) : I\!R^n \to I\!R$ e um simplex no $I\!R^n$, a iteração de Nelder-Mead se dá conforme o algoritmo abaixo:

1. *Ordenação:* Ordenar os vértices do simplex de maneira que $f(x_1) \leq f(x_2) \leq \cdots \leq f(x_{n+1})$. Calcular o centróide dos n melhores pontos, $\bar{x} = \sum_{i=1}^{n} x_i/n$.
2. *Reflexão:* Calcular o ponto de reflexão $x_r = (1+\rho)\bar{x} - \rho x_{n+1}$. Se $f(x_1) \leq f(x_r) < f(x_n)$, aceitar o ponto x_r e terminar a iteração.
3. *Expansão:* Se $f(x_r) < f(x_1)$, calcular o ponto de expansão $x_e = (1+\rho\chi)\bar{x} - \rho\chi x_{n+1}$. Se $f(x_e) < f(x_r)$, aceitar x_e e terminar a iteração, caso contrário ($f(x_e) \geq f(x_r)$) aceitar x_r e terminar a iteração.
4. *Contração:* Se $f(x_r) \geq f(x_n)$ fazer uma contração:
   (a) *Contração Externa:* Se $f(x_n) \leq f(x_r) < f(x_{n+1})$, calcular $x_c = (1+\rho\gamma)\bar{x} - \rho\gamma x_{n+1}$. Se $f(x_c) \leq f(x_r)$ aceitar x_c e terminar a iteração. Caso contrário, ir para o passo 5.

(b) *Contração Interna:* Se $f(x_r) \geq f(x_{n+1})$, calcular $x_c = (1-\gamma)\bar{x} + \gamma x_{n+1}$. Se $f(x_c) < f(x_{n+1})$, aceitar x_c e terminar a iteração. Caso contrário, ir para o passo 5.

5. *Redução (Shrink):* Calcular os vetores $v_i = x_1 + \sigma(x_i - x_1), i = 2, \ldots, n+1$. Os vértices (ainda fora de ordem) para a próxima iteração são $x_1, v_2, \ldots, v_{n+1}$.

No processo de se ordenar os vértices do simplex no início da iteração, podem ocorrer empates no valor da função em dois ou mais pontos. No artigo original de Nelder e Mead, não é mencionado como proceder em tal situação. Em [6], é sugerida uma regra de desempate que foi a utilizada durante a implementação computacional do algoritmo Nelder-Mead na próxima seção:

Iteração sem redução do simplex: Quando não há redução do simplex, apenas o vértice de índice $n+1$ é descartado e substituído por um vetor que chamaremos de v. O vetor v tomará a posição $j+1$ no simplex da iteração seguinte, onde $j = \max_{0 \leq m \leq n}\{m \mid f(v) < f(x_{m+1})\}$. Os outros vértices continuam na ordem que foi estabelecida antes de se iniciar a iteração.

Iteração com redução do simplex: Nesse caso, apenas o vértice x_1 será aproveitado na iteração seguinte. Somente uma regra de desempate é necessária: caso o vértice x_1 empate com um ou mais vetores como sendo o ponto de melhor valor da função no novo simplex. Caso isso ocorra, faremos $x_1^{k+1} = x_1^k$. Em qualquer outra situação, poderemos fazer uso de qualquer outra regra de desempate depois de uma redução.

Apesar de ser largamente empregado, o método de Nelder-Mead não tem garantias de convergência para dimensões superiores a 2. Existem exemplos onde o algoritmo converge a pontos não estacionários, fenômeno normalmente atribuído ao fato da direção de busca (ou seja, a direção definida pelo pior vértice e pelo centróide dos vértices restantes) se tornar numericamente ortogonal ao gradiente. Em [11], encontramos uma classe de funções estritamente convexas e diferenciáveis onde o método falha. Entretanto, o mau comportamento para essas funções pode ser evitado impondo, por exemplo, decréscimo suficiente e restrições sobre os ângulos internos do simplex, como em [17].

4.2 Método das Direções Aleatórias

O Método das Direções Aleatórias foi proposto em 2003 em um trabalho de M.A. Diniz-Ehrhardt, J.M. Martínez e M. Raydan [2]. Baseados em algumas técnicas *derivative-free* da literatura, os autores construíram um método globalmente convergente com estratégia de busca linear não monótona. Durante a execução do algoritmo, é permitido acréscimo na função, inclusive em direções de subida.

Inicialmente, escolhemos o parâmetro $M \in \mathbb{N}$. Em uma iteração k, definimos

$$\bar{f}_k = \max\{f(x_k), \ldots, f(x_{\max\{k-M+1,0\}})\}.$$

Um vetor x_{k+1} será aceito se satisfizer a inequação

$$f(x_{k+1}) = f(x_k + \alpha d) \leq \bar{f}_k + \eta_k - \alpha^2 \beta_k. \tag{2}$$

O parâmetro η_k deve ser escolhido de maneira que

$$\eta_k > 0 \ \ \forall \ k \in \mathbb{N} \qquad \text{e} \qquad \sum_{k=0}^{\infty} \eta_k < \infty.$$

Já o parâmetro β_k deve ser tal que $\{\beta_k\}$ seja uma seqüência limitada, ou seja, exista $C \in \mathbb{R}$ tal que

$$\beta_k < C \ \ \forall \ k \in \mathbb{N},$$

além disso é necessário que

$$\beta_k > 0 \ \ \forall \ k \in \mathbb{N},$$

e, para qualquer subconjunto infinito de índices $K \subset \mathbb{N}$,

$$\lim_{k \in K} \beta_k = 0 \Rightarrow \lim_{k \in K} \nabla f(x_k) = 0.$$

Observemos que a escolha $\beta_k \equiv 1$ é admissível.

Com o critério (2), eventualmente serão admitidos pontos que representem um acréscimo na função objetivo. Essa característica é devida ao parâmetro η_k e ao fato de utilizarmos $\bar{f}_k$ ao invés de $f(x_k)$. Já β_k tem um papel semelhante ao escalar γ do algoritmo de Lucidi e Sciandrone.

A inequação (2) é a base para um modelo de algoritmo globalmente convergente, e para mostrar a eficiência de se utilizar um critério não monótono, os autores

propõem um algoritmo onde as direções de busca são geradas aleatoriamente.

Mesmo utilizando direções aleatórias, o desempenho do algoritmo é satisfatório, apresentando bons resultados na grande maioria dos testes realizados. Além disso, o critério (2) de decréscimo suficiente apresenta algumas semelhanças com o utilizado por Lucidi e Sciandrone. Por esses motivos, surgiu a idéia de saber o que ocorre quando as direções de busca do algorimo de Lucidi e Sciandrone são geradas aleatoriamente.

É claro que, com direções aleatórias, não há garantias, por exemplo, de que o conjunto de direções satisfaz os critérios de independência linear, que são utilizados na demonstração de convergência. No entanto, o bom senso acusa que direções aleatoriamente geradas quase nunca são linearmente dependentes.

Outra questão relevante é uma certa incompatibilidade de idéias ao se usar direções geradas aleatoriamente em um algoritmo com a estrutura do de Lucidi e Sciandrone. Parece mais sábio utilizar direções de busca aleatórias em algoritmos que realizem busca em apenas uma direção a cada iteração, utilizando assim apenas α_k, como é o caso do algoritmo proposto em [2], ao invés de um conjunto de passos α_k^i. Além disso, outros critérios de parada talvez fossem mais sensatos, ao invés dos mais óbvios empregados em implementações do algoritmo de Lucidi e Sciandrone.

Entretanto, nenhum dos fatos acima nos tirou a curiosidade de testar os algoritmos da Seção 3 com direções aleatoriamente geradas. E os resultados foram interessantes, como mostraremos na Seção 5.1.2.

É importante observar que o algorimo de Direções Aleatórias proposto em [2] não é de busca direta segundo a definição da Seção 2, por utilizar o valor da função, por exemplo, em uma etapa que realiza interpolações quadráticas. Mas sem dúvidas é um método *derivative-free*, e os seus bons resultados teóricos e experimentais motivaram os testes com direções aleatoriamente geradas.

5 Experimentos Numéricos

Os Algoritmos de Lucidi e Sciandrone foram programados em Fortran 77 e rodados em um PC.

As seguintes direções de busca foram utilizadas, (com exceção para a Subseção

1.2):

Algoritmo 1:

$$p_k^i = \begin{cases} e^i & i = 1, \ldots, n \\ -e^{i-n} & i = n+1, \ldots, 2n \end{cases}$$

Algoritmo 2:

$$p_k^i = e^i \quad i = 1, \ldots, n$$

Para escolher os parâmetros do algoritmo, diversos valores foram experimentados. Testamos valores para θ e para δ no intervalo $[0.1, 0.9]$. Para c e para γ, empregamos valores da forma 10^k, com $k = -4, \ldots, 6$ para γ e $k = 0, \ldots, 4$ para c. Os valores que apresentaram melhores resultados foram selecionados para constituir o conjunto de parâmetros utilizados durante os experimentos desta Seção. Os valores adotados são:

$c = 10,$

$\delta = 0.5,$

$\gamma = 1,$

$\theta = 0.6.$

Como critério de parada, escolhemos uma tolerância τ tal que o algoritmo pára se em alguma iteração k

$$\max_{i=1,\ldots,r} \tilde{\alpha}_k^i \leq \tau.$$

A condição imposta pela tolerância τ procede, uma vez que a distância entre as aproximações $\{x_k\}$ diminui na mesma proporção que os coeficientes $\{\tilde{\alpha}_k\}$. Este critério de parada será referido como critério 1.

Diremos que o algoritmo fracassou em convergir quando fizer mais que *itmax* iterações (critério 2) ou realizar mais que *fevalmax* avaliações de função (critério 3).

Além disso, quando trabalhamos com problemas onde sabemos, a priori, que $f(x) \geq 0$ (fato verificado nos exemplos de Moré, Garbow e Hillstrom [3] por se tratar de soma de quadrados), adicionamos uma tolerância extra τ_f, de modo que a execução do algoritmo termina quando

$$f(x_k) \leq \tau_f,$$

condição que nomearemos critério 4.

Para os parâmetros de parada, foram utilizados os seguintes valores:

$\tau = 10^{-5}$,

$\tau_f = 10^{-7}$,

itmax = 50000,

fevalmax = 100000.

Comparamos primeiramente os algoritmos de Lucidi e Sciandrone com o algoritmo de Nelder-Mead. Os coeficientes escolhidos foram os mais comumente empregados:

$\rho = 1$,

$\sigma = 0.5$,

$\gamma = 0.5$ e

$\chi = 2$.

O simplex inicial foi calculado a partir do mesmo chute inicial utilizado nos Algoritmos 1 e 2. Dado um vetor x_0, escolhemos $v_{n+1} = x_0$, $v_i(j) = x_0$, $i = 1, \ldots, n$, $j \neq i$ e $v_i(i)$ da forma

$$v_i(i) = \begin{cases} 1.05x_0(i) & x_0(i) \neq 0, \\ 0.00025 & x_0(i) = 0. \end{cases}$$

Essa é uma interessante maneira de colocarmos o chute inicial que desejamos dentro de um simplex. Este é o mesmo procedimento que encontramos na rotina *fminsearch* do software `MATLAB` [4].

Como critério de convergência, após ordenarmos de forma que $v_1 \leq \cdots \leq v_{n+1}$, pediremos que

$f(v_{n+1}) - f(v_1) \leq 10^{-5}$ e

$\|v_i - v_{i+1}\|_\infty \leq 10^{-5}, i = 1, \ldots, n.$

O critério acima será denominado critério 1. Além desse, consideraremos os critérios 2, 3 e 4 como para os algoritmos de Lucidi e Sciandrone.

5.1 Exemplos de Moré, Garbow e Hillstrom

Primeiramente, testamos os algoritmos para a coletânea de problemas organizada por Moré, Garbow e Hillstrom [3]. Trata-se de um conjunto de problemas clássicos e diferenciáveis de minimização irrestrita. Problemas desse tipo correspondem corriqueiramente ao primeiro conjunto de testes pelo qual um algoritmo passa, mesmo sendo esse algoritmo de busca direta.

5.1.1 Direções Canônicas

Nesta subseção utilizaremos como direções de busca para o algoritmo de Lucidi e Sciandrone a base canônica, que é de fato o conjunto de direções que foi utilizado durante a maioria dos testes numéricos.

Para os problemas onde a dimensão n era variável, escolhemos primeiramente $n = 20$. A tabela a seguir mostra o número de iterações realizadas, a quantidade de cálculo de funções e o tipo de convergência encontrada (1, 2, 3 ou 4) para cada algoritmo.

		Algoritmo 1			Algoritmo 2			Nelder-Mead		
prob	n	it	feval	tipo	it	feval	tipo	it	feval	tipo
1	2	1905	13159	1	668	3335	4	78	147	4
2	2	89	538	1	68	316	1	68	131	1
3	2	1335	6990	1	2184	9543	1	156	284	4
4	2	63	306	1	63	228	1	140	265	4
5	2	104	665	4	35	157	4	46	91	4
6	2	31	145	1	40	160	1	44	86	1
7	3	1	12	4	114	770	1	129	234	4
8	3	2896	28870	1	13302	90762	1	132	242	1
9	3	17	105	4	30	130	1	12	28	4
10	3	10268	100008	3	13241	100002	3	1038	1798	1
11	3	10244	100009	3	14507	100002	3	297	515	4
12	3	6623	59675	4	14753	100004	3	271	496	1
13	4	657	8281	1	2416	20891	1	310	503	4
14	4	1	22	4	2549	21961	1	300	503	4
15	4	3668	46029	1	11537	98706	1	169	291	1

		Algoritmo 1			Algoritmo 2			Nelder-Mead		
prob	n	it	feval	tipo	it	feval	tipo	it	feval	tipo
16	4	103	1312	1	393	3326	1	211	368	1
17	5	5979	100013	3	1846	17431	1	620	978	1
18	6	5443	100012	3	8446	100005	3	620	983	1
19	11	1579	54181	1	4782	100001	3	2943	4054	1
20	9	3445	100006	3	1974	33153	1	1259	1844	1
21	20	1441	100031	3	1578	58236	1	16757	21268	1
22	20	807	51076	1	2622	100006	3	10887	13773	1
23	10	3308	100006	3	2914	61181	1	2830	4000	1
24	10	108	3027	1	142	1937	1	4646	6552	1
25	20	1748	100009	3	2840	100029	3	8175	10343	1
26	20	302	17627	1	516	19882	1	11062	13849	1
27	20	1813	100038	3	2682	100010	3	9871	12511	1
28	20	1496	100052	3	2618	98872	1	8538	10722	4
29	20	23	1034	4	124	4323	4	14720	18392	4
30	20	30	1481	4	111	3889	4	3118	3948	4
31	20	1388	100046	3	90	3028	1	7303	9180	1
32	20	33	1449	1	190	4158	1	23559	29484	1
33	20	35	1463	1	44	962	1	613	1002	1
34	20	34	1424	1	46	1017	1	637	1040	1
35	9	91	2571	4	622	10845	1	1151	1674	1

O Algoritmo 1 convergiu para menos problemas, falhando em 11 exemplos contra 8 do Algoritmo 2. Apesar disso, para muitos dos problemas onde ambos os algoritmos convergiram, o Algoritmo 1 realizou uma quantidade tão menor de iterações que o número de avaliações de função foi semelhante (muitas vezes até menor) que no Algoritmo 2, mesmo o primeiro fazendo ao menos $2n$ avaliações de função por iteração e o segundo ao menos $n + 1$.

O método de Nelder-Mead convergiu para todos os problemas e, na maior parte deles, realizou a menor quantidade de cálculos de funções entre os três algoritmos.

As figuras abaixo representam os gráficos do valor da função objetivo pelo número de avaliações de função. A linha contínua representa o comportamento do Algoritmo 1, a linha pontilhada é referente ao Algoritmo 2 e a tracejada ao algoritmo de Nelder-Mead. As figuras (5), (6), (7) e (8) correspondem, respectivamente, à Função de Rosenbrock, Função de Beale, Função da Equação Integral Discreta e Função Cheby-

quad, que correspondem aos problemas 1, 5, 29 e 35.

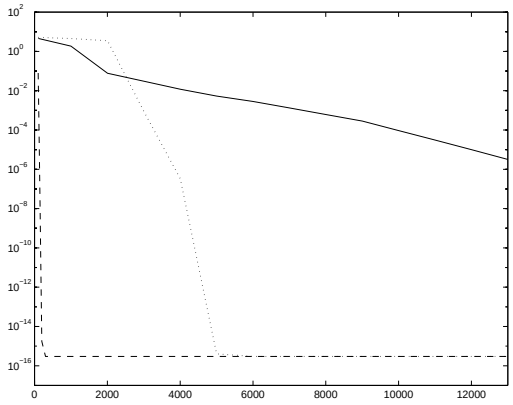

Figura 5: Função de Rosenbrock

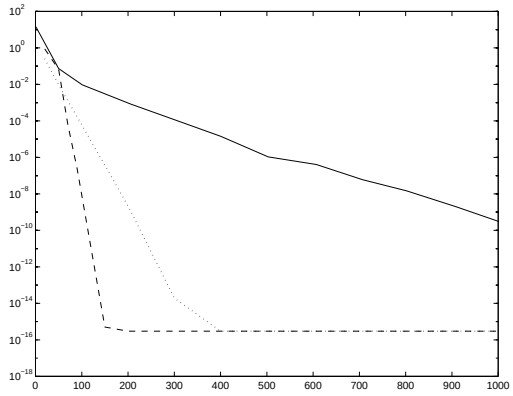

Figura 6: Função de Beale

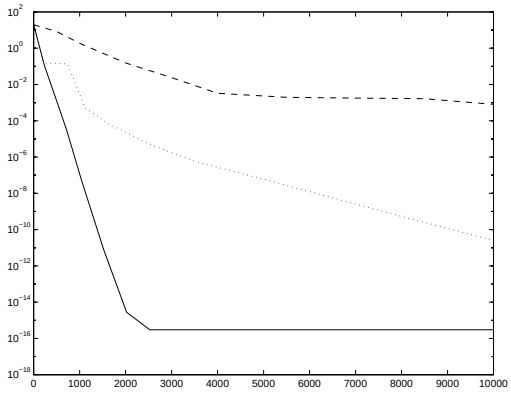

Figura 7: Função da Equação Integral Discreta

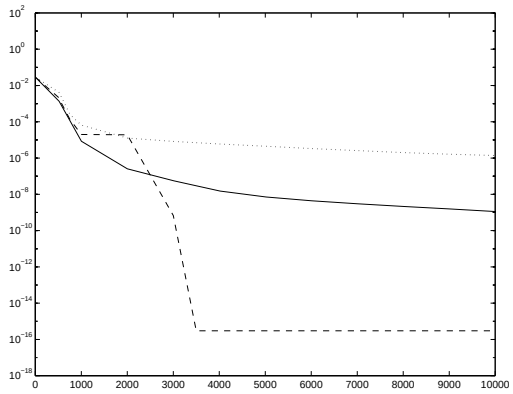

Figura 8: Função Chebyquad

Para as figuras (5), (6) e (8) o desempenho do método de Nelder-Mead é melhor, uma vez que o gráfico mostra que o valor da função é quase sempre menor se comparado com os obtidos pelos Algoritmos 1 e 2 para o mesmo número de avaliações de função. Apenas na figura (7) podemos ver um desempenho melhor para os Algoritmos de Lucidi e Sciandrone. Comparando os Algoritmos 1 e 2 entre si, vemos um melhor desempenho para 1 nas figuras (7) e (8).

Apesar do visivelmente melhor desempenho do método de Nelder-Mead para os exemplos até aqui vistos, a situação inverteu-se quando aumentamos o valor da

dimensão n. Para os problemas encontrados em [3] cuja dimensão é variável, utilizamos $n = 100$. Obtivemos os seguintes resultados:

	Algoritmo 1			Algoritmo 2			Nelder-Mead		
prob	it	feval	tipo	it	feval	tipo	it	feval	tipo
21	292	100201	3	620	100001	3	50000	54092	2
22	320	100010	3	594	100104	3	50000	53799	2
23	342	100277	3	536	100081	3	50000	53349	2
24	149	43051	1	280	51100	1	50000	53236	2
25	476	100005	3	885	100085	3	50000	53147	2
26	224	64323	1	481	87705	1	50000	53770	2
27	473	100098	3	888	100090	3	31844	39879	1
28	32	6581	1	29	2988	1	50000	54387	2
29	26	5812	4	261	47415	1	50000	53965	2
30	32	7513	4	417	77226	1	50000	56064	2
31	37	9224	1	434	79984	1	50000	76148	2
32	33	7249	1	981	100059	3	50000	53548	2
33	42	8508	1	57	5819	1	7651	10028	1
34	40	8106	1	58	5924	1	7365	9527	1
35	167	55232	1	145	24669	1	50000	54483	2

Como podemos observar, o método de Nelder-Mead convergiu apenas para 3 problemas. Já os Algoritmos 1 e 2 convergiram em 10 e 9 casos, respectivamente. É interessante observar principalmente a queda da qualidade do método de Nelder-Mead.

5.1.2 Direções Aleatórias

Durante a realização dos experimentos numéricos, notamos que a utilização da base canônica como direção de busca não torna os métodos muito eficientes. Seria mais interessante empregarmos direções que variassem a cada iteração. Com esse intuito, executamos os programas com direções aleatoriamente geradas, como já relatamos na Seção 4.2. Fizemos uso de um gerador de aleatórios para calcular cada uma das n direções usadas por iteração nos algoritmos. Para o algoritmo 1, calculamos as n direções p_k^i e depois utilizamos as direções $-p_k^i$. As direções foram computadas de maneira que $\|p_k^i\|_\infty \leq 1$. Obtivemos os seguintes resultados:

		Algoritmo 1			Algoritmo 2		
prob	n	it	feval	tipo	it	feval	tipo
1	2	318	2235	1	389	2034	1
2	2	334	2343	1	503	2625	1
3	2	26	112	1	29	96	1
4	2	13885	100003	3	13955	73920	1
5	2	148	1011	4	177	922	4
6	2	31	134	1	3725	20305	1
7	3	226	2355	1	349	2424	1
8	3	147	1496	1	7323	52980	1
9	3	27	195	4	28	144	4
10	3	46	379	1	30	140	1
11	3	9173	100007	3	13891	100002	3
12	3	9135	100004	3	12022	87392	1
13	4	558	7982	1	784	6960	1
14	4	1113	16120	1	1481	13193	1
15	4	3711	54031	1	3585	32509	1
16	4	69	852	1	104	837	1
17	5	539	9562	1	152	1516	1
18	6	4579	100001	3	4328	54935	1
19	11	2487	100035	3	4608	100017	3
20	9	3055	100013	3	1104	19629	1
21	20	1370	100054	3	2630	100021	3
22	20	1372	100063	3	2339	88748	1
23	10	2752	100019	3	2332	46530	1
24	10	31	731	1	40	583	1
25	20	201	13779	4	344	12605	1
26	20	139	9377	1	198	7155	1
27	20	804	57740	1	122	4232	1
28	20	1071	77837	1	564	21067	1
29	20	29	1459	4	47	1467	4
30	20	43	2394	4	73	2390	4
31	20	39	2084	4	77	2533	4
32	20	36	1852	1	58	1820	1
33	20	29	1190	1	30	675	1
34	20	28	1167	1	31	689	1
35	9	329	10500	1	679	12059	1

Com as direções geradas aleatoriamente, encontramos resultados melhores para os dois algoritmos. O Algoritmo 1 fracassou 9 vezes, ao invés das 11 falhas ocorridas quando utilizamos a base canônica. Já o algoritmo 2 apresentou uma melhora de 8 para 3 fracassos. Podemos observar a superioridade do Algoritmo 2 com direções aleatórias, pois este falhou para menos problemas e realizou menos avaliações de função para a maioria deles. O melhor desempenho para ambos os algoritmos se deve ao fato de estarmos fazendo buscas em um conjunto mais amplo de direções. Para o Algoritmo 2, vemos o quanto é interessante aproximarmos a direção de máxima descida.

Voltamos a frisar que, utilizando direções geradas aleatoriamente, não temos garantias de convergência, uma vez que não podemos afirmar que o conjunto de direções é linearmente independente. Porém, as possibilidades de que geremos um conjunto linearmente dependente são reduzidas.

Os resultados com direções aleatórias nos fazem crer que seja possível encontrar um conjunto de direções que seja fácil de se gerar e que possibilite bons resultados práticos para o algoritmo de Lucidi e Sciandrone.

5.2 Exemplos de McKinnon

O método de busca direta mais utilizado na prática é provavelmente o método de Nelder-Mead. Porém em [11] é apresentada uma classe de funções que inclui funções diferenciáveis estritamente convexas para as quais é demonstrado que o método de Nelder-Mead apresenta um comportamento indesejado. Se o método é aplicado à função

$$f(x) = \begin{cases} \theta\phi|x_1|^\tau + x_2 + x_2^2 & x_1 \leq 0 \\ \theta x_1^\tau + x_2 + x_2^2 & x_1 \geq 0 \end{cases}, \tag{3}$$

a seqüência gerada converge para $(0, 0)$, mesmo não sendo a origem um ponto estacionário. Para tanto, o simplex inicial deve ter os seguintes vértices:

$$\begin{array}{rcccc} v_1 = & (& 0 & , & 0 &), \\ v_2 = & (& 1 & , & 1 &), \\ v_3 = & (& \frac{1+\sqrt{33}}{8} & , & \frac{1-\sqrt{33}}{8} &). \end{array}$$

Além disso, os parâmetros τ, θ e ϕ devem satisfazer algumas condições. A função (3) tem derivadas contínuas se $\tau > 1$. Alguns conjuntos de parâmetros que provocam o mau comportamento do método são:

1. $(\tau, \theta, \phi) = (3, 6, 400)$,
2. $(\tau, \theta, \phi) = (2, 6, 60)$,
3. $(\tau, \theta, \phi) = (1, 15, 10)$.

Para esses valores, foram obtidos os seguintes resultados, tomando a origem como chute inicial para os Algoritmos 1 e 2 e o simplex acima para o Nelder-Mead:

	Algoritmo 1			Algoritmo 2			Nelder-Mead		
prob	it	feval	conv	it	feval	conv	it	feval	conv
1	28	118	OK!	28	90	OK!	67	137	X
2	28	118	OK!	33	110	OK!	67	137	X
3	28	118	OK!	23	70	X	83	169	X

Apesar de todos os exemplos terem convergido segundo o critério 1 (tamanhos dos passos se tornaram pequenos o bastante), no exemplo 3 o Algoritmo 2 também converge para a origem ao invés de encontrar a solução $(0, -0.5)$. Esse é um indício de que não será possível encontrar resultados de convergência para o Algoritmo 2 para funções não diferenciáveis. O Algoritmo 1 convergiu porque a direção $-e_2$, utilizada pelo algoritmo, é de descida. Como o esperado, o algoritmo de Nelder-Mead convergiu para a origem em todos os casos.

5.3 Problema da Conformação Molecular

Como último conjunto de experimentos, apresentamos um problema de minimização baseado em situações práticas. O problema da Conformação Molecular, advindo da Química, consiste em encontrar a disposição dos átomos em uma molécula, de maneira que haja consistência entre a solução encontrada e as expectativas provenientes do conhecimento em Química.

É possível utilizar elementos de otimização para se encontrar a configuração dos átomos em uma molécula de maneira que a energia do sistema seja mínima. Há inclusive artigos publicados que utilizam métodos de busca direta com esse intuito, ver [10]. Nesse caso, a dificuldade está na quantidade excessiva de minimizadores locais, que estima-se crescer exponencialmente com o quadrado do número de átomos do sistema.

Porém, o problema de Conformação que estudamos consiste em encontrar a disposição dos átomos de uma molécula de maneira que a distância entre alguns pares

de átomos seja coerente com dados de laboratório. O exemplo que abordaremos é teórico e encontra-se no software `Matlab,` de onde foram extraídos todos os dados, como chute inicial e distâncias desejadas entre os átomos.

Seja $z_j = (x_j, y_j)$, $j = 1, \ldots, n$ as coordenadas cartesianas dos n átomos que formam a molécula. A distância entre alguns (não necessariamente todos) átomos é conhecida, ou seja, sabemos qual é d_k tal que $\|z_{i_1(k)} - z_{i_2(k)}\| = d_k$, $k = 1, \ldots, m$ e $1 \leq m \leq C_{n,2}$, onde

$$C_{n,2} = \frac{n!}{2!(n-2)!}.$$

As funções $i_1(k)$ e $i_2(k)$ indicam os índices dos pares de átomos cujas distâncias são conhecidas. Por exemplo, num sistema de 4 átomos, se conhecemos as distâncias d_1 entre os átomos z_1 e z_2, d_2 entre os átomos z_1 e z_4 e d_3 entre z_2 e z_3, teremos $i_1 = \{1, 1, 4\}$ e $i_2 = \{2, 4, 3\}$.

O objetivo é minimizar a função erro quadrado $E(z_1, \ldots, z_n)$, dada por

$$E(z_1, \ldots, z_n) = \sum_{k=1}^{m} (z_{i_1(k)} - z_{i_2(k)})^2 - d_k^2.$$

É importante notar que a função acima é invariante, por exemplo, a translações. Por esse motivo, nos experimentos deixamos fixos 3 dos átomos, de maneira a evitar comportamentos indesejados.

Testamos o que ocorre com um exemplo fictício de 25 átomos. O chute inicial é uma conformação aleatória dos átomos como na figura (9).

Os pontos da figura representam os átomos. As linhas conectam os pares de átomos cujas distâncias são conhecidas a priori. Quanto mais escura é a linha entre dois átomos, mais discrepante é a distância entre estes, nesta configuração molecular, em relação à distância desejada.

No `Matlab` o problema é resolvido por um método do tipo Newton com região de confiança. O valor da função encontrado é $E(z_1, \ldots, z_{25}) = 0.0098$. O resultado é conseguido em poucas iterações, o que era esperado por se tratar de um método tão eficiente e que utiliza informações das derivadas de ordem 1 e 2 da função objetivo.

Resolvendo o problema com o Algoritmo 1, obtemos convergência avaliando em torno de 78 mil vezes a função f. O valor encontrado para o erro quadrado foi

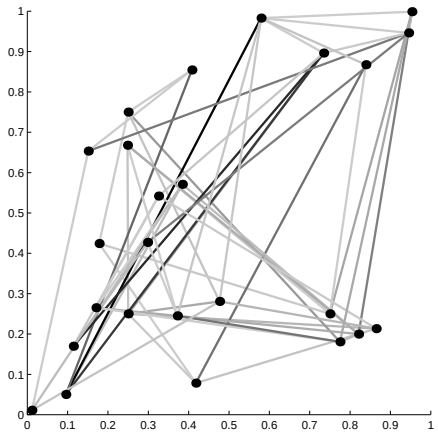

Figura 9: Chute inicial da conformação dos átomos.

$E(z_1, \ldots, z_{25}) = 0.0099$. A conformação molecular encontrada é a da figura (10).

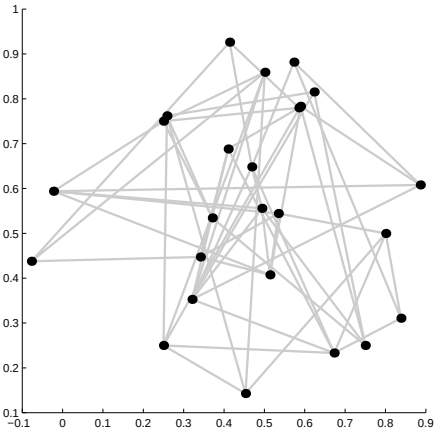

Figura 10: Conformação molecular para o Algoritmo 1.

O valor da função é bem próximo ao encontrado pelo método de Newton. Como as linhas da figura são todas claras, deduzimos que o erro de cada distância é diminuto.

Já o Algoritmo 2 não convergiu para o problema, por exceder 100000 avaliações de função. O valor final para a função E foi $E(z_1, \ldots, z_{25}) = 0.0146$. A conformação encontrada foi conforme a figura (11).

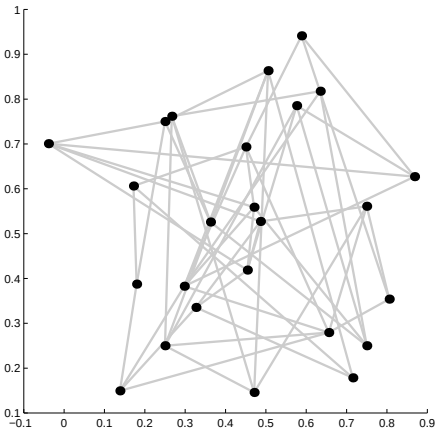

Figura 11: Conformação molecular para o Algoritmo 2.

É interessante notar as semelhanças entre as figuras (10) e (11). Mesmo um algoritmo tendo convergido e outro não, as soluções encontradas estão próximas.

Por fim, o algoritmo de Nelder-Mead também não convergiu, por realizar mais de 50000 iterações. O valor da função encontrado foi $E(z_1, \ldots, z_{25}) = 0.1959$. Ainda sim, o erro da distância entre cada par de átomos foi razoavelmente pequena, como podemos ver pelas linhas claras da figura (12).

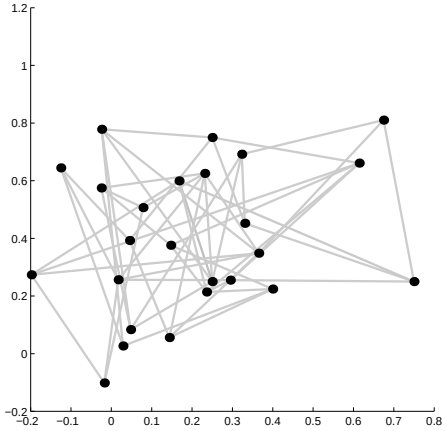

Figura 12: Conformação molecular para o Algoritmo Nelder-Mead.

Conclusões

Dentre os diversos métodos de busca direta, ressaltamos os métodos simplex, métodos de busca linear e métodos de busca padrão. O método simplex mais famoso é o de Nelder-Mead, que pode apresentar problemas no seu desempenho, mesmo para funções diferenciáveis e convexas. O algoritmo também pode ser insatisfatório para problemas de dimensões um pouco maiores. Métodos de busca padrão possuem resultados gerais de convergência global, e uma idéia para melhorar esse tipo de método é mesclá-lo com métodos de busca linear, para permitir maiores passos quando boas direções são encontradas. Essa é a essência do trabalho de Lucidi e Sciandrone, que propõe um esquema de algoritmos híbridos globalmente convergentes.

Quanto aos dois algoritmos de Lucidi e Sciandrone testados, verificamos que o Algoritmo 1 apresenta pequena vantagem em relação ao Algoritmo 2, se nos referirmos ao número de avaliações de função. Entretanto, o Algoritmo 2 converge para mais problemas, especialmente quando utilizamos direções geradas aleatoriamente. O bom resultado obtido mostra principalmente a qualidade do critério de decréscimo suficiente escolhido pelos autores. Provavelmente os resultados seriam melhores para ambos os algoritmos se as direções de busca fossem escolhidas com mais cuidado, por exemplo, explorando melhor os pontos visitados nas iterações anteriores . Mas, para tanto, seria necessário encontrar um conjunto de direções que satisfizesse a condição C1 e não apresentasse dificuldades em seu cálculo, para não aumentar demasiadamente o custo computacional dos algoritmos.

O Algoritmo 2 fracassou em testes realizados em uma função estritamente convexa e não diferenciável, o que indica que talvez não seja possível obter resultados teóricos de convergência para funções não diferenciáveis. O algoritmo de Nelder-Mead apresentou resultados indesejáveis para alguns problemas diferenciáveis, como já era previsto.

O problema da Conformação Molecular é um interessante tópico da Química que pode ser abordado pela otimização, em particular pelos métodos de busca direta. A semelhança entre os minimizadores encontrados para os Algoritmos 1 e 2, mesmo com um algoritmo acusando convergência e outro não, nos mostra desvantagens desses algoritmos que foram observadas durante todo o processo de experimentação numérica, como o comportamento local às vezes insatisfatório dos algoritmos e a ineficiência do critério de parada usado para detectar convergência. Fica o desejo de estudar o problema da Conformação Molecular com um aprofundamento nos

conceitos empíricos da Química e fazendo uso de Algoritmos mais eficientes, preferencialmente métodos *derivative-free.*

Referências

[1] Bazaraa, M. & Shetty, C.M., *Nonlinear programming - theory and algorithms*, New York, John Wiley and Sons, 1979.

[2] Diniz-Ehrhardt, M.A.; Martínez, J.M. & Raydan, M., *A derivative-free nonmonotone line search*, em preparação.

[3] Garbow, B.S.; Hillstrom, K.E. & Moré, J.J., *Testing unconstrained optimization software*, *ACM Transactions on Mathematical Software*, Vol. 7, No. 1, pp. 17 – 41, 1981.

[4] Hanselman, D. & Littlefield, B., *Mastering* `Matlab` *6: A Comprehensive Tutorial and Reference*, New Jersey, Prentice Hall, 2001.

[5] Hooke, R. & Jeeves, T.A., *Direct search solution of numerical and statistical problems*, *Journal of the Association for Computing Machinery* 8, pp. 212 – 229, 1961.

[6] Lagarias, J.C.; Reeds, J.A.; Wright, M.H. & Wright, P.E., *Convergence properties of the Nelder-Mead simplex algorithm in low dimensions*, *SIAM J. Opt.* 9, pp. 112 – 147, 1998.

[7] Lewis, R.M.; Torczon, V. & Trosset, M.W., *Direct search methods: then and now*, *ICASE Report* 2000-26, ICASE, NASA Langley Research Center, Hampton, VA, 2000.

[8] Lucidi, S. & Sciandrone, M., *On the global convergence of derivative-free methods for unconstrained optimization*, *SIAM J. Opt.* 13, pp. 97 – 116, 2002.

[9] Luenberger, D.G., *Linear and nonlinear programming*, 2ª edição, New York, Addison - Wesley Publishing Company, 1986.

[10] Martinez, M.L. & Meza, J.C., *On the Use of Direct Search Methods for the Molecular Conformation Problem*, *Journal of Computational Chemistry* 15, nº 6, pp. 627–632, 1994.

[11] McKinnon, K.I.M., *Convergence of the Nelder-Mead simplex method to a nonstationaty point*, *SIAM Journal on Optimization* 9, pp. 148–158, 1998.

[12] Nelder, J.A. & Mead, R., *A simplex method for function minimization*, *The Computer Journal* 7, pp. 308–313, 1965.

[13] Polak, E., *Computational Methods in Optimization: A Unified Approach*, Academic Press, New York, 1971.

[14] Powell, M.J.D., *Direct search algorithms for optimization and calculations*, *Acta Numerica*, pp. 287 – 336, 1998.

[15] Spendley, G.R., Hext, G.R. & Himsworth, F.R., *Sequential application of simplex designs in optimization and evolutionary operation*, *Technometrics* 4, pp. 441–461, 1962.

[16] Torczon, V., *On the convergence of pattern search algorithms*, *SIAM Journal on Optimization* 7, pp. 1–25, 1997.

[17] Tseng, P., *Fortified-descent simplicial search method: A general approach*, *SIAM Journal on Optimization* 10, pp. 269 – 288, 1999.