4
Fundamentos Teóricos

Neste capitulo são apresentados os fundamentos dos problemas de otimização matemática, abordando os diferentes conceitos e terminologias da otimização matemática, assim como seus métodos de solução.

4.1.
Otimização Matemática

Esta seção apresenta uma visão geral da teoria necessária para entender os conceitos dos problemas de otimização e de seus métodos de solução.

4.1.1.
Problema de Otimização Matemática

Um problema de otimização matemática, ou simplesmente *problema de otimização*, tem a forma:

$$\begin{aligned} &\underset{\boldsymbol{x} \in \Re^n}{min} && \{f(\boldsymbol{x})\} \\ &sujeito\ a && g_i(\boldsymbol{x}) \leq b_i,\ i = 1, \dots, m \end{aligned} \tag{4-1}$$

Onde:

- $\boldsymbol{x} = (x_1, \dots, x_n)$, é o vetor que representa as *variáveis de otimização* ou *variáveis de decisão* do problema, no espaço de busca n-dimensional ($\Re^n$),
- $f(\boldsymbol{x})$, é a *função objetivo*,
- $g_i(\boldsymbol{x}),\ i = 1, \dots, m$, são as *funções de restrição*,
- $b_i, \dots, b_m$, são os limites ou fronteiras para as restrições.

Um vetor $\boldsymbol{x}^*$ é denominado *ótimo* ou *solução ótima* do problema (4-1), se o valor da sua função objetivo é o menor entre todos aqueles que satisfazem as restrições $g_i(\boldsymbol{x})$, isto é, para qualquer $\boldsymbol{z} \in \Re^n$ com $g_1(\boldsymbol{z}) \leq b_1, \dots, g_m(\boldsymbol{z}) \leq b_m$, temos que $f(\boldsymbol{z}) \geq f(\boldsymbol{x}^*)$ (Boyd & Vandenberghe, 2009).

O espaço gerado pelas *variáveis de decisão* é denominado *espaço de busca* $\Re^n$, enquanto o espaço formado pelos valores da *função objetivo* é denominado *espaço de soluções*. Os problemas de otimização normalmente mapeiam o domínio de $\Re^n$ num *espaço de soluções* $\Re$.

Os conceitos apresentados daqui por diante são definidos para problemas de otimização sem restrições e que minimizem o valor da *função objetivo*.

4.1.2. Tipos de Problemas de Otimização

Geralmente os *problemas de otimização* são caracterizados por alguma condição na sua *função objetivo* ou nas suas *funções de restrição*. De acordo com essas características, os tipos de *problemas de otimização* podem ser classificados em:

- *Problemas de Programação Linear (LP),* se:
  1. $f(\boldsymbol{x}),\ g_i(\boldsymbol{x}),\ i = 1, \dots, m$; são todas funções lineares.
  2. $\boldsymbol{x} \in \Re^n$, onde todas as variáveis são continuas.
- *Problemas de Programação Quadrática (QLP), se:*
  1. $f(\boldsymbol{x})$, é uma função quadrática.
  2. $g_i(\boldsymbol{x}),\ i = 1, \dots, m$; são todas funções lineares.
  3. $\boldsymbol{x} \in \Re^n$, onde todas as variáveis são continuas.
- *Problemas de Programação Não Linear (NLP), se:*
  1. $f(\boldsymbol{x}),\ g_i(\boldsymbol{x}),\ i = 1, \dots, m$; têm pelo menos uma função não linear.
  2. $\boldsymbol{x} \in \Re^n$, onde todas as variáveis são continuas.
- *Problemas de Programação Inteira (IP)*, quando alguma parte do conjunto das variáveis é restrita a valores inteiros.

Em geral, se nenhuma restrição é especificada, o *problema de otimização* pode ser referido como um *problema de otimização sem restrições*, e se alguma função de restrição ou limite for estabelecido, então nos referimos como *problema de otimização restrito* (Yang, 2008).

4.1.3. Otimalidade

A definição de otimalidade pode ser apresentada através de alguns conceitos utilizados nos *problemas de otimização*:

- Um ponto $\boldsymbol{x}$ que satisfaça a todas as restrições é denominado como *ponto viável*, que por sua vez é uma *solução viável* do *problema de otimização*.
- O conjunto de *pontos viáveis* é denominado *região viável*.
- Um ponto $\boldsymbol{x_l}$ é denominado como *mínimo local* para o *problema de otimização* se $f(\boldsymbol{x})$ está definido numa vizinhança $V(\boldsymbol{x}, \delta)$ de tamanho δ, e satisfaz que $f(\boldsymbol{x_l}) \leq f(\boldsymbol{u})$ para $\forall\, \boldsymbol{u} \in V(\boldsymbol{x}, \delta)$ onde $\delta > 0$ e $\boldsymbol{u} \neq \boldsymbol{x_l}$.
- Um ponto $\boldsymbol{x}^*$ é denominado como *mínimo global* para o *problema de otimização* se $f(\boldsymbol{x})$ está definido no *espaço de busca* $\Re^n$, e satisfaz que $f(\boldsymbol{x}^*) \leq f(\boldsymbol{x})$ para $\forall \boldsymbol{x} \in \Re^n$ e $\boldsymbol{x} \neq \boldsymbol{x}^*$. $\boldsymbol{x}^*$ é também denominado como o *ótimo global* ou a *solução ótima.*

Na Figura 14 se ilustra a uma *função objetivo* f definida no domínio de $\boldsymbol{x}$, assim como também um dos *mínimos locais* e o *mínimo global* desta função.

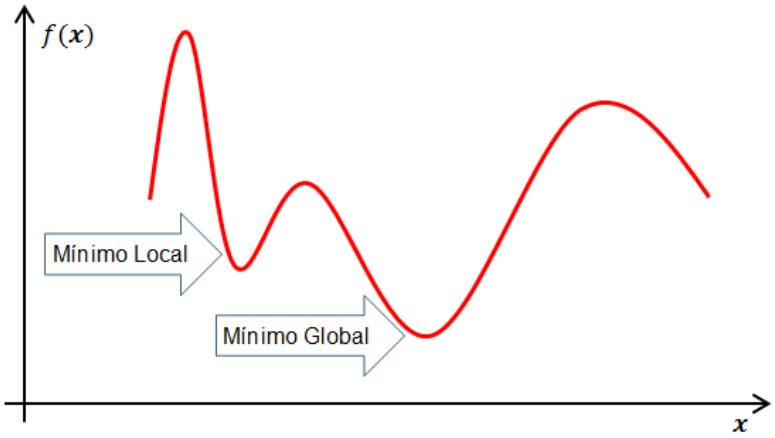

Figura 14. Mínimo Global e Local de $f(x)$

Um método de solução para um *problema de otimização* é um algoritmo que encontra a solução ótima ou sub-ótima para um determinado problema.

Antes de analisar os métodos de solução é necessário ressaltar que mesmo quando a *função objetivo* e as restrições são funções suavizadas (como as funções polinomiais), resolver o problema geral de otimização da Equação (4-1) pode ser complexo, com um tempo de processamento elevado ou mesmo com a possibilidade de não encontrar nenhuma solução satisfatória.

A seguir são apresentados sequencialmente os métodos de solução usados neste trabalho para resolver o *problema de otimização* dos parâmetros envolvidos no cálculo da vascularidade, conforme apresentado na Seção 3.3.1.

4.2. Evolução Diferencial

Segundo Storn & Price (Storn & Price, 1997), a Evolução Diferencial (*DE*, Differential Evolution) é um método para minimizar funções não lineares e não diferenciáveis num espaço contínuo de busca. Foi concebido visando atender às seguintes demandas:

1. Capacidade de lidar com *funções objetivo* não diferenciáveis, não lineares e multimodais.
2. Ser paralelizável, para suportar funções de elevada carga computacional.
3. Com facilidade de uso, ou seja, poucas variáveis de controle para definir o desempenho do método de minimização.
4. Boas propriedades de convergência.

O método de evolução diferencial se encontra no escopo dos algoritmos de estratégias evolutivas (ES) que se inspiram na teoria de evolução das espécies de Charles Darwin sobre como os seres vivos mais aptos para determinado problema são selecionados e persistem.

Em termos computacionais, um indivíduo representa uma solução potencial para um determinado problema e seus genes representam as variáveis ou parâmetros do *problema de otimização*. Neste trabalho, os genes são o conjunto de parâmetros utilizados no cálculo da vascularidade.

O conjunto de indivíduos forma uma população e evoluem do decorrer das gerações, isto é, vão sendo selecionados e modificados com o objetivo de encontrar soluções que minimizem (ou maximizem) a função objetivo.

A cada geração, a aptidão de cada indivíduo é avaliada a fim de determinar numericamente quão bom o indivíduo em questão é como solução do *problema de otimização*. A cada nova geração os indivíduos menos aptos são descartados, os mais aptos geram novos indivíduos através de diferentes operadores genéticos descritos nas secções 4.2.2 e 4.2.3 respectivamente, e dessa forma passam suas características para a próxima geração.

Ao final do processo o indivíduo mais apto é selecionado como solução aproximada do *problema de otimização*.

4.2.1. Descrição do Método

DE é um método de procura direta que utiliza P vetores de parâmetros n-dimensionais a cada geração G. Este vetor é representado como:

$$\boldsymbol{x}_{i,G}, i = 1, 2, \dots, P \tag{4-2}$$

Onde:

- $\boldsymbol{x}_{i,G} \in \Re^n$, representa o vetor i (denominado também indivíduo) formado por n genes; n é o numero de dimensões da *função objetivo*.
- P, é o tamanho da população.
- G, representa a geração atual do método.

A cada geração os indivíduos da população são escolhidos aleatoriamente e devem cobrir todo o *espaço de busca*. A formação de novos indivíduos é realizada através de dois passos subsequentes: mutação e cruzamento.

4.2.2. Mutação

A Figura 15 apresenta um exemplo de uma função bidimensional e ilustra os diferentes vetores que atuam na geração de $\boldsymbol{v}_{i,G+1}$. Nesta figura, o primeiro passo na criação dos novos indivíduos é o da mutação, realizado através da adição da diferença ponderada entre dois vetores da população atual com um terceiro vetor aleatoriamente escolhidos. Deste modo, para cada vetor $\boldsymbol{x}_{i,G}$, é gerado um vetor de mutação $\boldsymbol{v}_{i,G+1}$, através da formula descrita abaixo:

$$\boldsymbol{v}_{i,G+1} = \boldsymbol{x}_{r_1,G} + F \cdot \left(\boldsymbol{x}_{r_2,G} - \boldsymbol{x}_{r_3,G}\right) \tag{4-3}$$

Onde:

- $r_1, r_2, r_3 \in \{1, 2, \dots, P\}$, representam aos índices da população aleatoriamente escolhidos, diferentes entre si e diferentes também do índice atual i.
- $F \in [0, 2]$, é um valor real, e controla o efeito da variação diferencial $(\boldsymbol{x}_{r_2,G} - \boldsymbol{x}_{r_3,G})$.

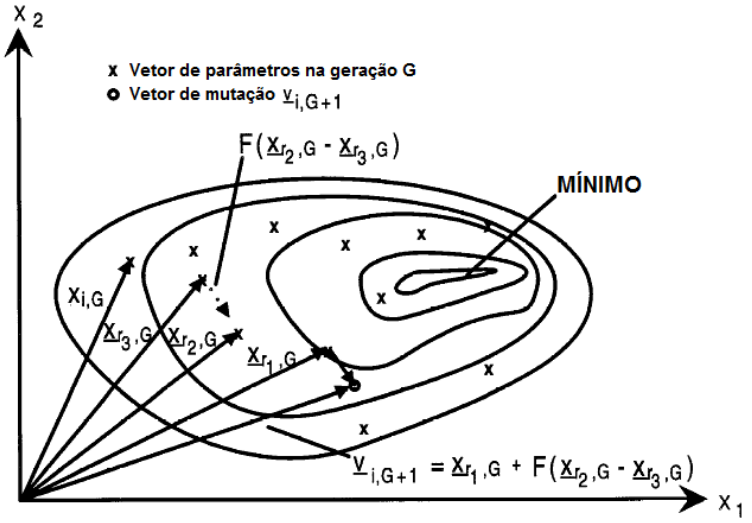

Figura 15. Linhas de contorno de uma *função objetivo* bidimensional, e o processo de mutação para gerar $\boldsymbol{v}_{i,G+1}$, (Storn & Price, 1997).

4.2.3. Cruzamento

Na Figura 16 se apresenta como exemplo o processo de cruzamento para um vetor 7-dimensional, procedimento no qual são utilizados dois vetores, o atual e de mutação, os quais são misturados para gerar o vetor de cruzamento. Este processo tem como objetivo incrementar a diversidade dos vetores no *espaço de busca*. Para isto, no cruzamento é gerado um vetor de prova $\boldsymbol{u}_{i,G+1}$, descrito por:

$$\boldsymbol{u}_{i,G+1} = \left(u_{1i,G+1}, u_{2i,G+1}, \dots, u_{ni,G+1}\right) \tag{4-4}$$

Onde cada $j\text{-}esima$ componente do vetor $\boldsymbol{u}_{i,G+1}$ é definida como:

$$u_{ji,G+1} = \begin{cases} v_{ji,G+1} & if\ (randb(j) \leq CR)\ or\ j = ranbr(i) \\ x_{ji,G} & if\ (randb(j) > CR)\ and\ j \neq ranbr(i) \end{cases} \quad \forall\ j = 1, 2, \dots, n \tag{4-5}$$

Onde:

- $randb(j)$, é um número aleatório entre $[0, 1]$, para cada *j-ésima* componente do vetor $\boldsymbol{u}_{i,G+1}$.
- $CR \in [0, 1]$, é a constante de crossover, definida pelo usuário.
- $ranbr(i) \in 1, 2, \dots, n$, é um índice escolhido aleatoriamente, garante que $\boldsymbol{u}_{i,G+1}$ tenha pelo menos um dos genes de $\boldsymbol{v}_{i,G+1}$.

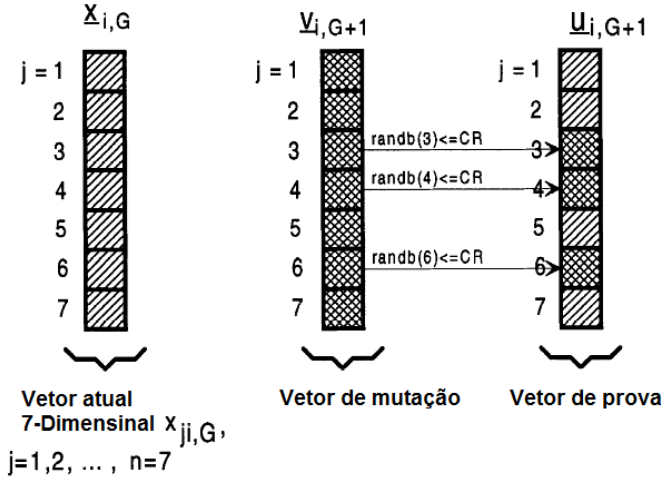

Figura 16. Processo de cruzamento para $n = 7$ parâmetros, (Storn & Price, 1997).

4.2.4. Seleção

Para decidir se o vetor de prova $\boldsymbol{u}_{i,G+1}$ será parte da população na geração $G + 1$, ele é comparado com o vetor atual $\boldsymbol{x}_{i,G}$, como ilustra a formulação abaixo:

$$\boldsymbol{x}_{i,G+1} = \begin{cases} \boldsymbol{u}_{i,G+1}, & if\ f(\boldsymbol{u}_{i,G+1}) < f(\boldsymbol{x}_{i,G}) \\ \boldsymbol{x}_{i,G}, & otherwise \end{cases} \tag{4-6}$$

Se o valor da *função objetivo* no vetor de prova é menor do que aquele produzido pelo vetor atual (referido também como vetor alvo), então o vetor de prova substituirá o vetor atual na geração seguinte.

Cada indivíduo deve atuar uma vez como vetor alvo, assim a cada geração são realizadas P avaliações da *função objetivo*. Este processo de seleção é iterativo e ao fim de G gerações o melhor indivíduo é selecionado.

4.3. Busca Generalizada de Padrões

A Busca Generalizada de Padrões (*GPS*, Generalized Pattern Search) é um algoritmo de Busca Direta (*DS*, Direct Search). Torczon (Torczon, 1997) define o *GPS* como um algoritmo iterativo para resolver *problemas de otimização* sem restrições e que não precisam do cálculo de derivadas em funções continuamente diferenciáveis. Mais recentemente, (MathWorks, 2012) indica que *GPS* também pode ser utilizado na solução de problemas onde a *função objetivo* não é contínua.

Segundo (Audet & Dennis, 2003), este algoritmo foi desenvolvido visando atender às seguintes demandas:

1. Ser suficientemente efetivo em *problemas de otimização* onde a avaliação da *função objetivo*, assim como o cálculo de suas derivadas, é muito custosa, ou não pode ser calculada.
2. Atingir um bom desempenho em *problemas de otimização* onde a *função objetivo* não é suavizada.

4.3.1. Fundamentos e Terminologia

O *GPS* é um algoritmo iterativo que a cada passo[3] busca na vizinhança de um ponto de análise, denominado *ponto atual*, um novo ponto cujo valor da *função objetivo* seja melhor[4] que o valor da função neste ponto. Este processo é realizado iterativamente até encontrar o *ponto ótimo*.

Para isto, o algoritmo cria um conjunto de pontos de prova, que resultam da adição do *ponto atual* com múltiplos escalares de um conjunto de vetores em direções de busca estabelecidas, denominadas *padrões* ou *direções* (MathWorks, 2012). Os *pontos de prova* junto com o *ponto atual* conformam um grupo de pontos de análise denominado como *malha atual.*

GPS é um algoritmo usado para minimizar um problema da forma:

$$\min_{\boldsymbol{x} \in \Omega} \quad \{f(\boldsymbol{x})\} \tag{4-7}$$

Onde:

- $f(\boldsymbol{x})$: $\Re^n \rightarrow \Re \cup \{\infty\}$, é a *função objetivo*.
- $\Omega = \{\boldsymbol{x} \in \Re^n : \Omega \neq 0 \subset \Re^n\}$, é a *região viável*; quando $\Omega = \Re^n$ trata-se de um problema sem restrições.

Quando $f(\boldsymbol{x})$ só existe dentro da *região viável* Ω, uma nova função $f_\Omega(\boldsymbol{x})$ é definida considerando-se esta região, deste modo:

$$f_\Omega(\boldsymbol{x}) = \begin{cases} f(\boldsymbol{x}) & if\ \boldsymbol{x} \in \Omega \\ \\ \infty & otherwise \end{cases} \tag{4-8}$$

[3] Entenda-se por passo como o número de iterações que o algoritmo leva avaliar a *função objetivo* em todos os *pontos de prova*.

[4] Entenda-se o termo "melhor" como uma referência àquele ponto que forneça o menor valor da *função objetivo*.

A cada iteração (o numero de iteração é denotado pelo índice k) um número finito de *pontos de prova é gerado* em torno do *ponto atual* $\boldsymbol{x}_k$, e o valor da *função objetivo* nestes pontos é comparado com o valor de $f_\Omega(\boldsymbol{x}_k)$ e o melhor candidato entre todos os pontos é escolhido. Na primeira iteração, o ponto atual é o ponto inicial $\boldsymbol{x}_o$, que deve ser definido pelo usuário.

O *ponto de prova* que minimize f_Ω é denominado *ponto de malha melhorado* e a iteração correspondente é denominada *iteração exitosa*.

O conjunto de direções "D" deve ser um conjunto de abrangência positivo; e cada direção[5] $d_j \in D$ (para $j = 1, 2, \dots, n_D$), deve ser o produto $G * z_j$ onde $G \in \Re^{n\times n}$ é uma matriz de geração não singular e fixa, e $z_j \in \mathbb{Z}^n$ é um vetor inteiro. O conjunto D é também visto como uma matriz real $n \times n_D$, de n_D direções cada uma com n variáveis. Assim, na iteração k, a malha M_k é definida como segue:

$$M_k = \left\{\boldsymbol{x}_k \pm \Delta_k^m d_j : d_j \in D\right\} \tag{4-9}$$

Os *pontos de prova* na *malha atual* estão situados em torno do *ponto atual* $\boldsymbol{x}_k$ nas direções e_j, com $e_j \in E$ e $E \subseteq D$, estas direções são escalonadas pelo *tamanho da malha* $\Delta_k^m \in \Re_+$. O conjunto dos *pontos de prova* é definido como:

$$P_k = \left\{\boldsymbol{x}_k \pm \Delta_k^m e_j : e_j \in E\right\} \tag{4-6}$$

A Figura 17 apresenta um exemplo de um conjunto de *pontos de prova* P_k ao redor de um ponto atual $\boldsymbol{x}_k$, onde a matriz de direções está definida por $D = \{d_1, d_2, d_3\}$ e o *tamanho da malha* é Δ_k.

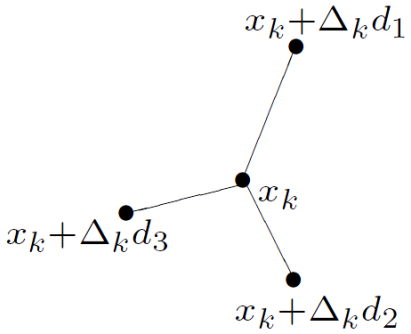

Figura 17. *Pontos de prova* ao redor de $\boldsymbol{x}_k$ (Abramsom, et al., 2003).

[5] A notação $d_j \in D$, indica que d_j representa à coluna j da matriz de direções D.

Uma vez determinados os pontos de vizinhança a serem analisados a cada passo, são realizadas mais duas etapas: *SEARCH,* que é opcional, e *POLL*, de busca local. Deste modo, a estrutura geral do algoritmo GPS pode ser descrita conforme o diagrama de fluxo apresentado na Figura 18.

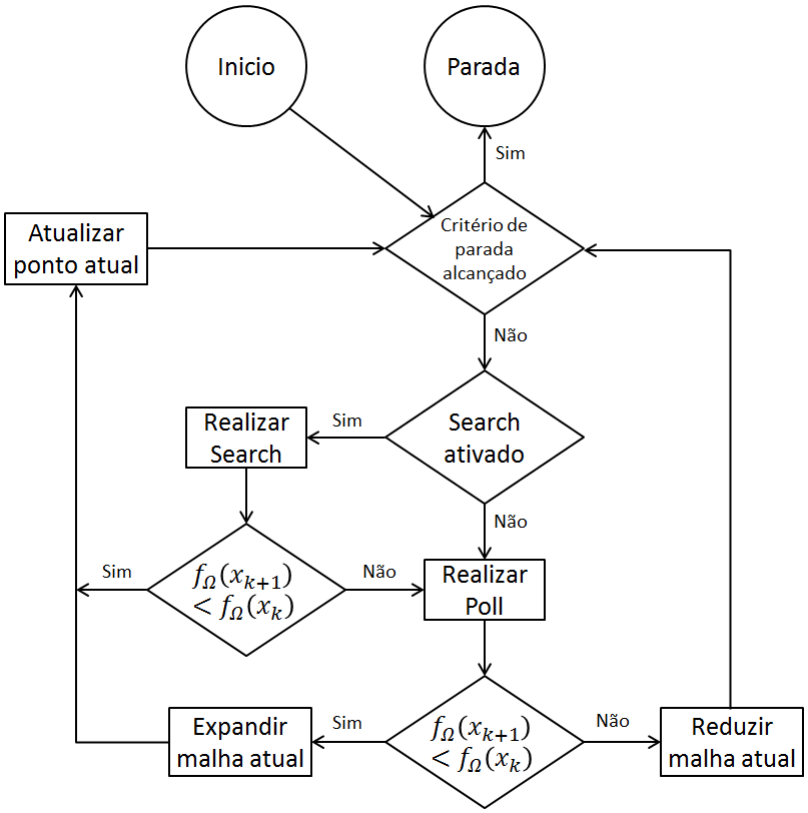

Figura 18. Estrutura geral do método *GPS* (MathWorks, 2012).

O algoritmo inicia com a definição de um *ponto atual*, a partir deste ponto é executada a etapa de *SEARCH* a fim de encontrar um ponto que seja melhor que o *ponto atual*, se esta etapa é exitosa o algoritmo atualiza o *ponto atual* e executa esta etapa novamente enquanto ela estiver ativada, caso contrário, a etapa de *POLL* é executada.

A etapa de *POLL* é utilizada como um método de busca local que se utiliza no controle do *tamanho da malha* para gerar os pontos de prova no passo em questão de forma a refinar a busca. O algoritmo é repetido até encontrar a solução ótima ou alcançar o número máximo de iterações.

As etapas de *SEARCH* e *POLL*, que definem o funcionamento do algoritmo *GPS*, são descritas em detalhes a seguir.

4.3.2.Etapa SEARCH

Esta etapa tenta localizar um ponto melhor do que o *ponto atual* $\boldsymbol{x}_k$ na *região viável* Ω. Caso seja encontrado, este ponto se torna o ponto atual no passo seguinte; caso contrário, a etapa de *POLL* é iniciada.

Existem duas razões principais para utilizar a etapa de *SEARCH*:

1. Para acelerar o processo de otimização.
2. Para obter uma melhor solução global.

Qualquer estratégia pode ser utilizada na geração de um numero finito de *pontos de prova* a serem avaliados nesta etapa. Nesta dissertação se utilizou dois algoritmos diferentes. O primeiro foi o algoritmo *DE* com uma população pequena com uma única geração, e o segundo foi um algoritmo de busca com uma distribuição uniforme na região viável, explicado a seguir.

4.3.2.1. Algoritmo de Busca com Distribuição Uniforme

O algoritmo de busca com distribuição uniforme (*UD*) distribui um conjunto de pontos de maneira uniforme na *região viável* Ω, a fim de encontrar um ponto dentro desta região que possa ser utilizado como ponto de partida na etapa de *POLL*. Esta busca é realizada em um conjunto de pontos P_{UD}, que resulta da combinação dos subpontos $u_{i,j}$ distribuídos ao longo de cada *região viável* Ω_i considerando-se $\Omega = \{\Omega_i \neq 0 \subset \Re: \min(\boldsymbol{x}_i) \leq \Omega_i \leq \max(\boldsymbol{x}_i)\,;\; x_i \in \Re\}\; \forall\, i = 1,..,n.$

O tamanho de Ω_i é definido por:

$$l_i = \max(\boldsymbol{x}_i) - \min(\boldsymbol{x}_i)\,, \forall\, i = 1, \dots, n \tag{4-11}$$

Os pontos distribuídos ao longo de Ω_i são determinados por:

$$u_{i,j} = \min(\boldsymbol{x}_i) + l_i * \frac{j}{4}, \forall\, j = 1,2,3 \tag{4-12}$$

$u_{i,j}$ representa o ponto j ao longo da região viável Ω_i (ver Figura 19). O número total de pontos que formam o conjunto P_{UD} é $n(P_{UD}) = 3^n$, onde n é o número de componentes do vetor $\boldsymbol{x}$.

Uma vez que o conjunto de pontos P_{UD} é gerado, calcula-se a função objetivo em cada ponto, e aquele que forneça o melhor resultado será o ponto utilizado na etapa de *POLL* como ponto inicial. Na Figura 19 se apresenta um exemplo do conjunto de pontos gerado por este algoritmo para um vetor de dois componentes.

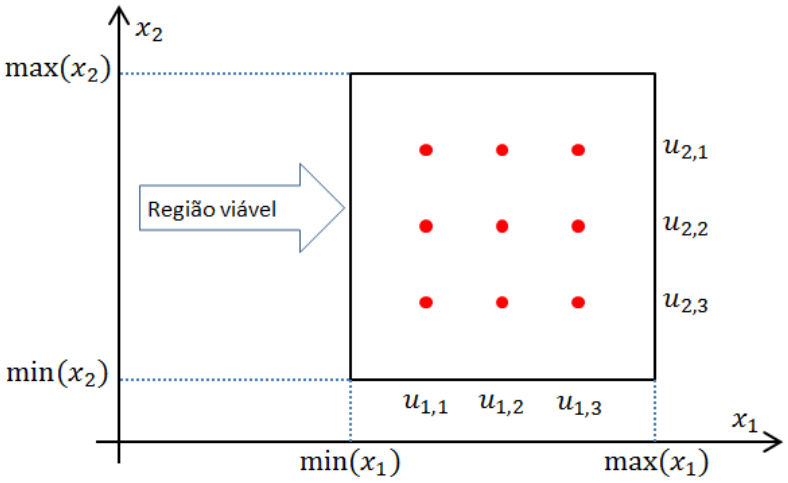

Figura 19. Pontos gerados no algoritmo *UD* para o caso bidimensional.

4.3.3. Etapa POLL

Na etapa de *POLL*, a *função objetivo* é avaliada nos *pontos de prova* que pertencem à *malha atual* a fim de analisar se algum deles fornece um valor da *função objetivo* melhor do que o ponto atual $\boldsymbol{x}_k$[6].

A forma como a etapa de *POLL* determina novos pontos a cada passo é definida por:

$$\boldsymbol{x}_{k+1} = \begin{cases} \boldsymbol{x}_{k+1} \;\; and \;\; \Delta^m_{k+1} = \tau^{w_k} \cdot \Delta^m_k, & if \; f_\Omega(\boldsymbol{x}_{k+1}) < f_\Omega(\boldsymbol{x}_k) \\ \boldsymbol{x}_k \;\; and \;\; \Delta^m_{k+1} = \tau^{r_k} \cdot \Delta^m_k, & otherwise \end{cases} \tag{4-13}$$

$$\tau > 1, \qquad 0 \le w_k < w^+, \qquad r^- < r_k \le -1$$

A equação (4-13) determina o comportamento da etapa de *POLL*, e estabelece que:

- Quando o algoritmo encontra um *ponto de prova* com valor da *função objetivo* melhor do que o *ponto atual*, ou seja: $f_\Omega(\boldsymbol{x}_{k+1}) < f_\Omega(\boldsymbol{x}_k)$, então o algoritmo torna esse ponto como o *ponto atual* no passo seguinte, e o *tamanho da malha* é aumentado num fator igual a $\tau^{w_k} \ge 1$, com w_k incrementado a cada passo até um valor máximo de w^+.
- Quando nenhum dos *pontos de prova* fornece um valor da *função objetivo* melhor do que o *ponto atual*, isto é $f_\Omega(\boldsymbol{x}_{k+1}) \ge f_\Omega(\boldsymbol{x}_k)$, o algoritmo conserva

[6] Nesta etapa $\boldsymbol{x}_k$ é ponto fornecido como resultado da etapa de *SEARCH*, quando ela for configurada, caso contrário, no inicio $\boldsymbol{x}_k$ será o ponto inicial $\boldsymbol{x}_0$ estabelecido pelo usuário.

o *ponto atual* no seguinte passo e o *tamanho da malha* é diminuido num fator igual a $\tau^{r_k} < 1$, com r_k decrementado a cada passo até um valor mínimo de r^-.

A implementação mais simples desta etapa é através de uma matriz de direções definida por $E = [I_n \quad -I_n]$, onde I_n representa a uma matriz identidade de n dimensões (Audet, et al., 2010); foi esta implementação aquela utilizada neste trabalho.

O critério de parada do método atende a uma das seguintes condições:

- O erro mínimo desejado é alcançado, este erro é a diferença entre dois valores consecutivos da *função objetivo*.
- O número máximo de iterações permitidas é alcançado.
- O tamanho da malha é menor do que um limiar pré-estabelecido.

4.4. Busca Direta de Malha Adaptativa

O algoritmo de Busca Direta de Malha Adaptativa, (*MADS, Mesh Adaptive Direct Search*), é também um dos algoritmos no escopo do método *DS* e é uma extensão do algoritmo *GPS*.

Foi desenvolvido para resolver problemas de otimização não lineares em funções não suavizadas. A vantagem principal do algoritmo *MADS* em relação ao *GPS*, é que a exploração dos *pontos de prova* na *região viável* Ω não está restrita a um número finito de *direções de busca*. (Audet & Dennis, 2006).

Da mesma maneira que o *GPS*, cada passo do *MADS* está dividido em duas etapas principais: *SEARCH,* que é opcional, e *POLL*, de busca local. Às duas etapas é sucedido um processo de atualização dos parâmetros que determinam o comportamento do algoritmo. Assim, o *tamanho da malha*, o *ponto atual* e o critério de parada são atualizados ou verificados ao final de cada passo (Audet, et al., 2008).

A etapa *SEARCH* no *MADS* tem a mesma flexibilidade que no *GPS*, permite realizar a avaliação da *função objetivo* em um número finito de *pontos de prova* na *região viável* Ω. Qualquer estratégia na geração destes pontos pode ser usada, levando-se em conta considerações iniciais acerca do comportamento do problema que permitam nos afastar das possíveis soluções locais, ou mesmo aproximações da *função objetivo* para minimizar o custo computacional do processo, quando este é reconhecidamente elevado (Audet, et al., 2010).

Da mesma forma que no *GPS*, na etapa de *SEARCH* se utilizou dois algoritmos, o primeiro foi o algoritmo *DE* com uma população pequena e uma única geração, e o segundo foi o algoritmo *UD* descrito na seção anterior.

A etapa de *POLL* no *MADS* difere da do *GPS* na forma como são escolhidas as *direções de busca*. A etapa de *POLL* é descrita em detalhes a seguir.

4.4.1. Etapa *POLL* no *MADS*

O *MADS* introduz um parâmetro Δ_k^p, denominado como *tamanho de busca* que permite determinar magnitudes de distância diferentes entre cada *ponto de prova* e o *ponto atual* $\boldsymbol{x}_k$.

No *GPS* é definido só o *tamanho de malha* é igual ao *tamanho de busca* ($\Delta_k^m = \Delta_k^p$). No caso do *MADS*, o *tamanho da malha* não sempre é igual ao *tamanho de busca*. A condição necessária para atualizar o *tamanho de busca* Δ_k^p, é que $\Delta_k^m \leq \Delta_k^p \; \forall \; k$; isto é, a cada passo o número de direções disponíveis deve ser maior do que aquelas direções que definirão a *malha atual*.

No *MADS* a definição do conjunto de *pontos de prova* é como segue:

$$P_k = \left\{ \boldsymbol{x}_k \pm \Delta_k^m d_j : d_j \in D \right\} \tag{4-14}$$

Esta definição permite escolher os *pontos de prova* em um espaço de direções maior D, que é função dos parâmetros: *tamanho de malha* e *tamanho de busca*. O *tamanho da busca* limita o tamanho máximo dos vetores de direção a cada passo, isto é, estabelece a amplitude máxima da *malha atual*. O *tamanho de malha* estabelece a definição espacial da malha atual sobre a qual serão definidos os vetores de direção e por consequência os pontos de busca. Assim, se Δ_k^m decresce mais rapidamente que Δ_k^p, a matriz de direções D usada para definir os *pontos de prova* não estará restrita a um conjunto finito de direções.

A seguir se apresentam duas figuras (Figura 20, Figura 21), a primeira ilustra o conjunto de *pontos de prova* para o *GPS*, e a segunda os possíveis pontos para o *MADS*, num *espaço de busca* definido em $\Re^2$.

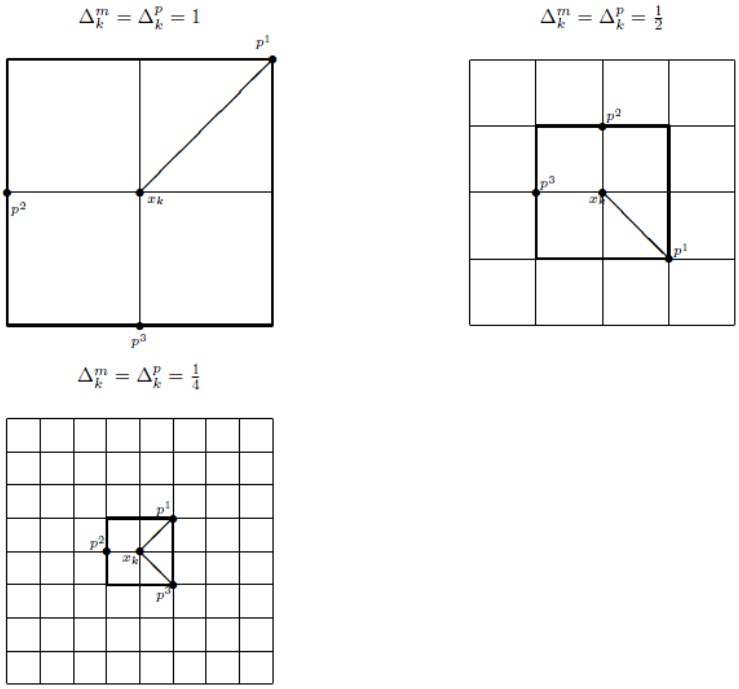

Figura 20. *GPS*: Exemplo de *pontos de prova* $P_k = \{\boldsymbol{x}_k \pm \Delta_k^m e_j : e_j \in E\} = \{p^1, p^2, p^3\}$ para diferentes valores de $\Delta_k^m = \Delta_k^p$ (Audet & Dennis, 2006).

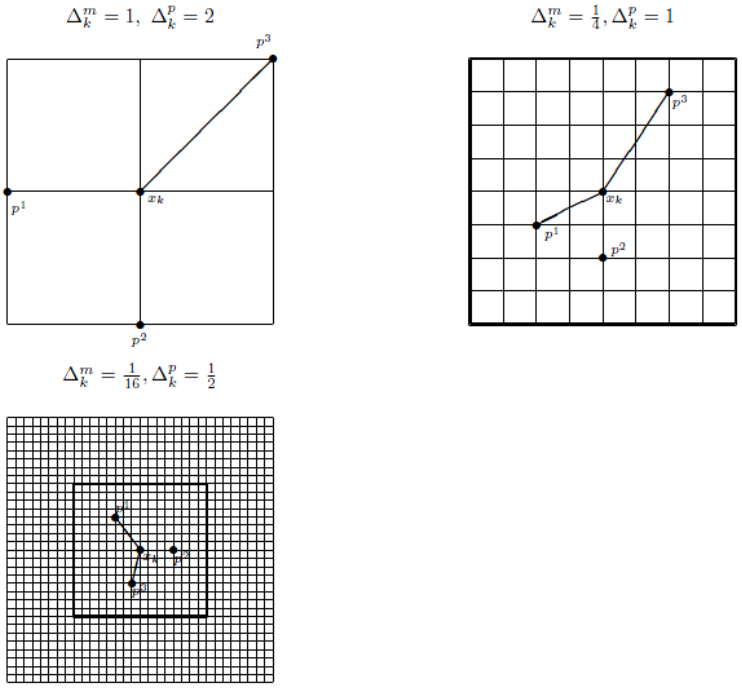

Figura 21. *MADS*: Exemplo de *pontos de prova* $P_k = \{\boldsymbol{x}_k \pm \Delta_k^m d_j : d_j \in D\} = \{p^1, p^2, p^3\}$ para diferentes valores de Δ_k^m e Δ_k^p (Audet & Dennis, 2006).

A Figura 20 e Figura 21 ilustram a influência dos parâmetros Δ_k^m e Δ_k^p têm sobre a definição das possíveis direções dos conjuntos de *pontos de prova* para cada caso, e pode ser verificado que no *MADS* a disponibilidade de direções é efetivamente maior e mais variável do que no *GPS*, onde a matriz de direções é menor e fixa durante todo o processo.

O LTMADS (Lower Triangle MADS) é uma implementação prática desta etapa e foi usada neste trabalho. No objetivo de gerar aleatoriamente as *direções de busca*, este algoritmo gera uma matriz triangular baixa não singular de números inteiros aleatórios, no intervalo aberto $\left[-1/\sqrt{\Delta_k^m} \quad 1/\sqrt{\Delta_k^m}\right]$, e depois aleatoriamente permuta as colunas desta matriz.

A matriz resultante, denominada B_k, forma uma base que serve para obter o conjunto positivo de direções $D = [B_k \quad -B_k]$. Define-se também que $\Delta_k^p = \sqrt{\Delta_k^m}$ como um parâmetro atualizado a cada passo.

O algoritmo LTMADS atualiza o *tamanho da malha* como segue:

$$\Delta_{k+1}^m = \begin{cases} \frac{\Delta_k^m}{4}, & if\ f_\Omega(\boldsymbol{x}_{k+1}) > f_\Omega(\boldsymbol{x}_k) \\ 4\Delta_k^m, & if\ f_\Omega(\boldsymbol{x}_{k+1}) < f_\Omega(\boldsymbol{x}_k)\ e\ \Delta_k^m \leq \frac{1}{4} \\ \Delta_k^m, & otherwise \end{cases} \tag{4-15}$$

A equação (4-15) determina o comportamento desta etapa e estabelece que:

- Quando o algoritmo encontra um *ponto de prova* com valor da *função objetivo* menor que no *ponto atual*, ou seja: $f_\Omega(\boldsymbol{x}_{k+1}) < f_\Omega(\boldsymbol{x}_k)$, o algoritmo torna esse ponto o *ponto atual* e o *tamanho da malha* é incrementado num fator de 4 no seguinte passo.
- Quando nenhum dos *pontos de prova* fornece um valor da *função objetivo* menor que no *ponto atual*, isto é $f_\Omega(\boldsymbol{x}_{k+1}) > f_\Omega(\boldsymbol{x}_k)$, o algoritmo conserva o *ponto atual* e o *tamanho da malha* é decrementado num fator de $1/4$.
- A atualização $\Delta_{k+1}^m = \Delta_k^m$ é definida para impedir que o tamanho da malha não exceda jamais o valor unitário.

A seleção do *ponto atual* e a atualização dos parâmetros Δ_k^m e Δ_k^p é um processo iterativo que é repetido até encontrar a *solução ótima* ou *sub-ótima*, ou até que o erro mínimo ou o número máximo de iterações seja atingido.

4.5.
Algoritmo Nelder – Mead

Nelder e Mead (Nelder & Mead, 1965) (NM) propuseram um método para minimização de uma função de n variáveis, que depende da comparação entre os valores da função em $(n+1)$ vértices de um simplex geral, seguida de um processo de substituição do vértice com o maior valor por outro ponto dentro do *espaço de busca* que forneça um menor valor da *função objetivo*. Este método é também parte dos algoritmos de *DS* e demonstrou ser preciso e computacionalmente eficiente (Luersen, et al., 2004).

O processo iterativo em que os vértices do simplex são substituídos até que se encontre o ponto ótimo é implementado através de operações de reflexão, expansão, contração e redução (Figura 22). O algoritmo termina a busca quando o valor da *função objetivo* nos seus vértices torna-se muito similar ou menor a um limiar pré-determinado, isto é, quando o método converge.

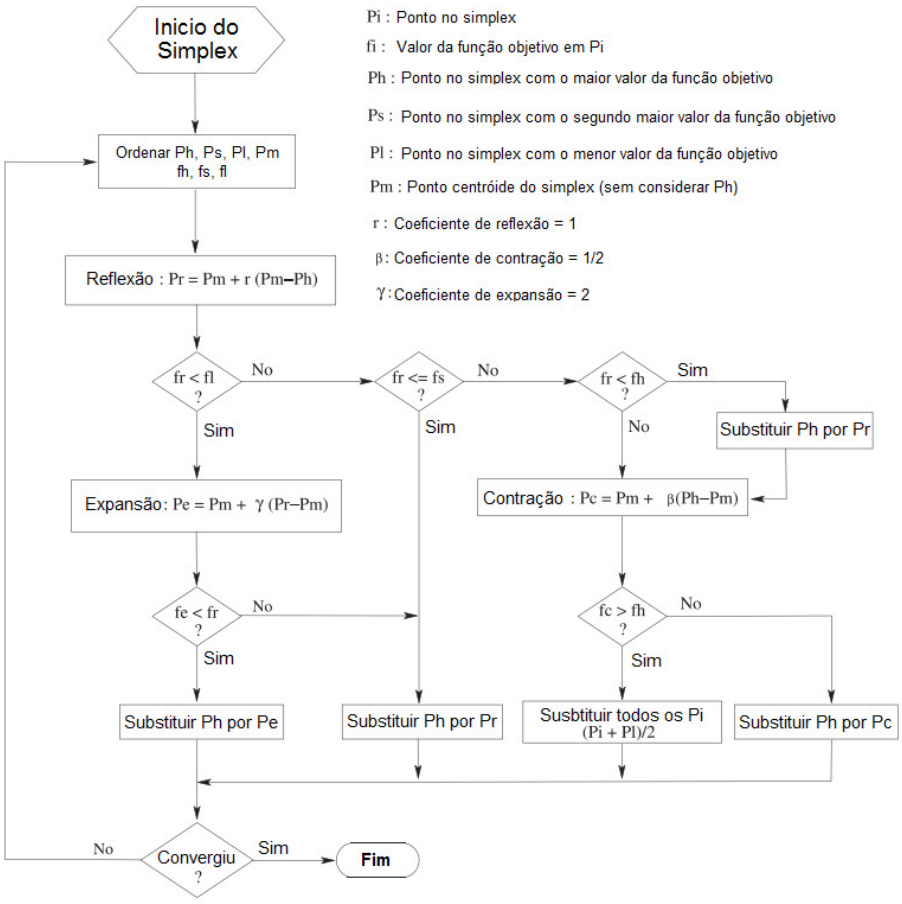

Figura 22. Descrição do algoritmo Nelder – Mead (Luersen, et al., 2004).

O algoritmo inicia ordenando os vértices do simplex em ordem crescente considerando-se a avaliação da *função objetivo* neles. Uma vez ordenados, são aplicados os processos de reflexão, expansão, contração e redução, conforme o mostrado na Figura 22, e de acordo com a descrição em pseudocódigo apresentada na Tabela 1. Esta tabela detalha as decisões lógicas que o método realiza a cada passo e que são repetidas durante todo o processo de otimização, e que serão detalhadas adiante.

SE $f_r < f_s$ ENTÃO Realiza Caso (1) {Reflexão ou Expansão} SENÃO Realiza Caso (2) {Contração ou Redução}	
BEGIN {Caso (1)} SE $f_l < f_r$ ENTÃO Substituir P_h com P_r SENÃO Calcula P_e e f_e SE $f_e < f_l$ ENTÃO Substituir P_h com P_e SENÃO Substituir P_h com P_r FIM-SE FIM-SE FIM {Caso (1)}	BEGIN {Caso (2)} SE $f_r < f_h$ ENTÃO Substituir P_h com P_r FIM-SE Calcula P_c SE $f_c < f_h$ ENTÃO Substituir P_h com P_c SENÃO Substituir P_h com P_{m1} Substituir P_s com P_m FIM-SE FIM {Caso (2)}

Tabela 1. Decisões lógicas para o algoritmo NM (Mathews & Fink, 2004).

Considera-se para fins de explicação o caso bidimensional onde o simplex é um triângulo.

4.5.1. Inicio do Simplex

Para começar, no caso bidimensional, deve-se determinar os três vértices do simplex: $P_i = (x_i, y_i),\ i = 1, 2, 3$, e também os valores da avaliação da *função objetivo* em cada vértice, $f_i = f(P_i)$. Estes valores devem então ser ordenados com $f_1 \leq f_2 \leq f_3$, e as seguintes nomenclaturas são estabelecidas:

$$P_l = (x_1, y_1),\ P_s = (x_2, y_2),\ P_h = (x_3, y_3) \tag{4-16}$$

Adicionalmente, o ponto médio do segmento que junta P_l e P_s é definido da seguinte maneira:

$$P_m = \frac{P_l + P_s}{2} \tag{4-17}$$

4.5.2.
Reflexão do Simplex

Supõe-se que o valor da *função objetivo* é minimizado na medida em que o ponto de avaliação P_r se afasta do ponto P_h na direção perpendicular ao segmento formado por P_s e P_l. Para implementar este comportamento define-se uma operação de reflexão que estabelece um ponto P_r, apresentado na Figura 23, e definido como:

$$P_r = P_m + r(P_m - P_h) \tag{4-18}$$

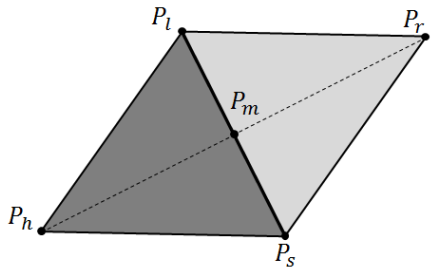

Figura 23. Processo de reflexão para P_r (Mathews & Fink, 2004).

4.5.3.
Expansão do Simplex

Se no ponto P_r o valor da função objetivo, f_r, é menor do que f_l, podemos supor que a direção escolhida é correta, e que o mínimo encontra-se mais adiante nessa mesma direção, e este comportamento de expansão é modelado através da definição de um ponto P_e nesta direção. Este ponto será melhor que P_r se a avaliação da *função objetivo* neste ponto é menor que em P_r. O ponto P_e, como apresentado na Figura 24, é definido como:

$$P_e = P_m + \gamma(P_r - P_m) \tag{4-19}$$

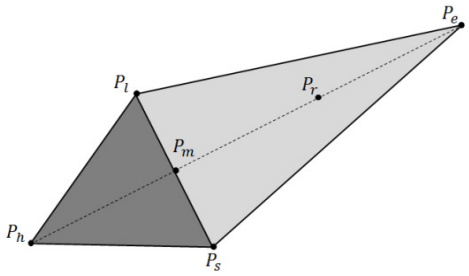

Figura 24. Processo de expansão para P_e (Mathews & Fink, 2004).

4.5.4.
Contração do Simplex

Se o valor de f_r não é menor do que f_s, mas sim menor ou igual a f_h, então se deve procurar por algum outro ponto, P_c, que minimize o valor da *função objetivo*. Este ponto é definido de maneira que se situe no segmento que une os pontos P_h e P_r, como apresentado na Figura 25, e definido como:

$$P_c = P_m + \beta(P_h - P_m) \tag{4-20}$$

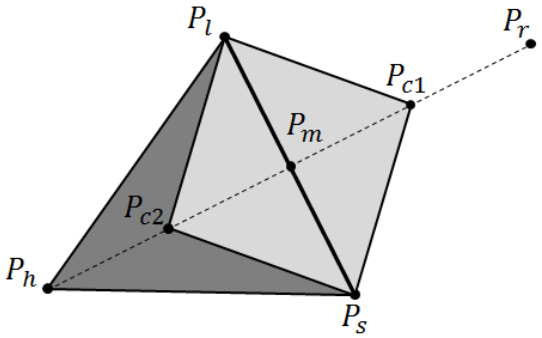

Figura 25. Processo de contração para P_c (Mathews & Fink, 2004).

O ponto P_{c1} na Figura 25 é definido quando o valor de f_r é menor do que o valor de f_h, caso contrário é definido o ponto P_{c2}. O ponto P_c é então definido como aquele que forneça o menor valor da função objetivo entre os dois casos.

4.5.5.
Redução do Simplex

Se o valor de f_c é maior que o valor de f_h, então os pontos P_s e P_h devem-se reduzir na sua metade na direção de P_l conforme apresentado na Figura 26.

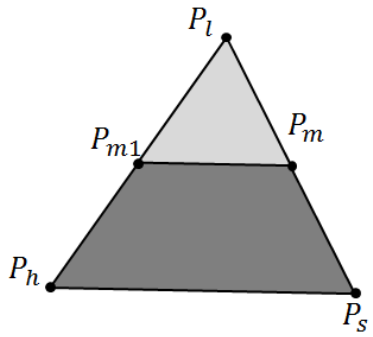

Figura 26. Processo de redução em direção de P_l (Mathews & Fink, 2004).

Estas operações visam obter um custo computacional eficiente para o processo de otimização já que o algoritmo avalia a *função objetivo* só quando necessário. A cada passo o algoritmo procura um ponto que minimize a *função objetivo* através da avaliação desta função nos vértices do simplex que é gerado a cada iteração. Quando um ponto melhor que P_h é encontrado, o algoritmo termina o passo atual e atualiza os vértices do simplex. Este procedimento é repetido iterativamente até encontrar a *solução ótima* ou até superar o número máximo de iterações permitidas.