UNIVERSIDADE ESTADUAL DE MARINGÁ
CENTRO DE CIÊNCIAS EXATAS
DEPARTAMENTO DE MATEMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM MATEMÁTICA
(Mestrado)

JESUS MARCOS CAMARGO

DETECÇÃO DE CURVAS EM IMAGENS COM AUXÍLIO DE FUNÇÕES ORDENADAS

Maringá-PR
2015

Jesus Marcos Camargo

Detecção de Curvas em Imagens com Auxílio de Funções Ordenadas

Dissertação apresentada ao Programa de Pós-Graduação em Matemática do Departamento de Matemática, Centro de Ciências Exatas da Universidade Estadual de Maringá, como requisito para obtenção do título de Mestre em Matemática.
Área de concentração: Matemática Aplicada.

Orientador: Emerson Vitor Castelani

Maringá
2015

Jesus Marcos Camargo

Detecção de Curvas em Imagens com Auxílio de Funções Ordenadas

Dissertação apresentada ao Programa de Pós-Graduação em Matemática do Departamento de Matemática, Centro de Ciências Exatas da Universidade Estadual de Maringá, como requisito para obtenção do título de Mestre em Matemática.

COMISSÃO JULGADORA

Prof. Dr. Emerson Vitor Castelani - Orientador
Universidade Estadual de Maringá (UEM)

Prof. Dr. Wesley Vagner Inês Shirabayashi
Universidade Estadual de Maringá (UEM)

Prof. Dr. André Luís Machado Martinez
Universidade Tecnológica Federal do Paraná (UTFPR)

Aprovada em: xx de xxxxx de 2015.

Local de defesa: Anfiteatro xxxxxx, Bloco F-67, *campus* da Universidade Estadual de Maringá.

Dedico este trabalho à minha família, hoje e sempre, pilar de minha existência.

Agradecimentos

A realização deste trabalho mostrou-se uma tarefa árdua e me exigiu um grande esforço. Mas, embora leve meu nome este trabalho está longe de ser só meu, por este motivo gostaria de registrar aqui minha gratidão.

Agradeço primeiramente a Deus que tem sido maravilhoso e me iluminado durante toda minha jornada pelos caminhos da ciência. Agradeço também a meus pais Neusa e Jair por todas as lições ensinadas. Graças a eles hoje tenho discernimento e sabedoria para realizar minhas escolhas. Agradeço ao meu irmão Anderson por sua companhia e amizade, que embora turbulenta, é sincera e verdadeira.

Agradeço a CAPES pelo apoio financeiro sem o qual este trabalho não seria possível.

Agradeço a todos os professores que contribuíram para minha formação, em especial ao meu orientador professor Emerson Castelani pela paciência e o tempo dedicado a esse projeto, foi uma bela parceria que deve permanecer por muito tempo.

Por fim agradeço profundamente aos meus amigos. À minha querida amiga Carla Melli Tambarussi que apesar da distância esteve sempre ao meu lado, dos melhores aos mais dramáticos momentos. À Priscila por ter sentado no lugar certo no primeiro dia de aula e assim deixado a vida em Maringá muito mais divertida e interessante. Junto a ela e aos amigos Laerte e Patricia vivi muitas histórias, muitos dias de estudo e alguns dias de festa.

Obrigado a todos os amigos de Maringá por tornar esses dois anos inesquecíveis. E ainda agradeço a todos aqueles que não foram mencionados mas de uma forma ou outra contribuíram fundamentalmente para essa jornada.

PRIMEIRO ELES O IGNORAM, DEPOIS RIEM DE VOCÊ,
EM SEGUIDA LUTAM COM VOCÊ, E ENTÃO VOCÊ GANHA.

Matthew Quick

Resumo

Neste trabalho apresentamos dois algoritmos para identificação de segmentos de retas e circunferências com raio fixo em imagens digitais. Para a primeira formulação usamos o método LOVO, que consiste em otimizar uma soma ordenada de um conjunto de funções. Na segunda formulação, usamos as funções ordenadas para construção de um sistema não linear. Trazemos a elaboração de alguns exemplos para teste, bem como seus resultados e avaliações numéricas.

Palavras chave: Detecção de Curvas, Hough; Otimização; Funções Ordenadas.

Abstract

In this work we present two algorithms for identification of line segments and circles with fixed radius in digital images. For the first formulation we used the LOVO method, which consists in optimizing an ordered sum of a set of functions. In the second formulation, we used ordained functions to construct a non-linear system. We bring the elaboration of some examples for testing and its results and numerical evaluations.

key words: Curves Detection; Hough Transform; Optimization; Ordained Functions.

Sumário

Introdução

O estudo de detecção de curvas vem sendo amplamente utilizado nas mais diversas áreas da ciência. Existem aplicações que vão das áreas médicas [1] à indústria de vestuário [8]. Dessa forma a Transformada de Hough, técnica mais comum para detecção de curvas, tem sido estudada e aprimorada desde sua criação.

Neste trabalhos apresentamos duas formulações diferenciadas para reconhecimento digital de curvas em imagens usando métodos de otimização e solução de sistemas não lineares. Este trabalho não contém comparativos entre as técnicas aqui desenvolvidas e os métodos que utilizam a transformada de Hough uma vez que não temos a pretensão de substituir tal técnica. Nossa intenção com este trabalho é explorar caminhos alternativos para os problemas, investigando os resultados afim de desenvolver um método satisfatórios para sua resolução.

O primeiro modelo de resolução aqui desenvolvido partiu da ideia apresentada por Martínez em [9], onde é sugerida a possibilidade de se trabalhar com a detecção de curvas utilizando o método LOVO. Dada essa sugestão inicial prosseguimos no desenvolvimento das funções utilizadas e algoritmos para soluções de tais problemas. Diante de algumas dificuldades recorrentes para os problemas testados optamos por desenvolver uma nova técnica, fornecendo deste modo uma nova formulação. Aproveitando as propriedades das funções de valor ordenado e algumas características específicas do problemas desenvolvemos o novo método utilizando a resolução de sistemas não lineares.

Assim o presente trabalho está organizado em cinco capítulos. No primeiro apresentamos as ideias de Hough, e estrutura básica de sua técnica de reconhecimento de retas. No segundo capítulo trazemos uma rápida revisão bibliográfica a cerca dos métodos de otimização utilizados na construção de nosso algoritmo. No terceiro capítulo é apresentada a

formulação LOVO. Neste capítulo além da teoria envolvida apresentamos também os testes realizados bem como resultados obtidos, tanto numéricos quanto gráficos. No capítulo quatro apresentamos a formulação SnL, justificando seu estudo e construção. Também, como no capítulo anterior, apresentamos os resultados numéricos e gráficos dos testes realizados. Por fim no último capítulo trazemos algumas considerações finais sobre o trabalho e questões que permanecem abertas para trabalhos futuros.

Capítulo 1

Detecção de Curvas

Nosso intuito é elaborar e desenvolver uma técnica para detecção de curvas em uma imagem. Vamos dar uma pequena ideia do funcionamento dos métodos existentes de modo a facilitar a compreensão do trabalho. Estamos interessados na detecção de retas e circunferências, assim apresentaremos o método mais comum, a Transformada de Hough, e embora existam muitas variações, descreveremos aqui a versão mais simples, apenas para dar apoio à leitura.

1.1 Transformada de Hough

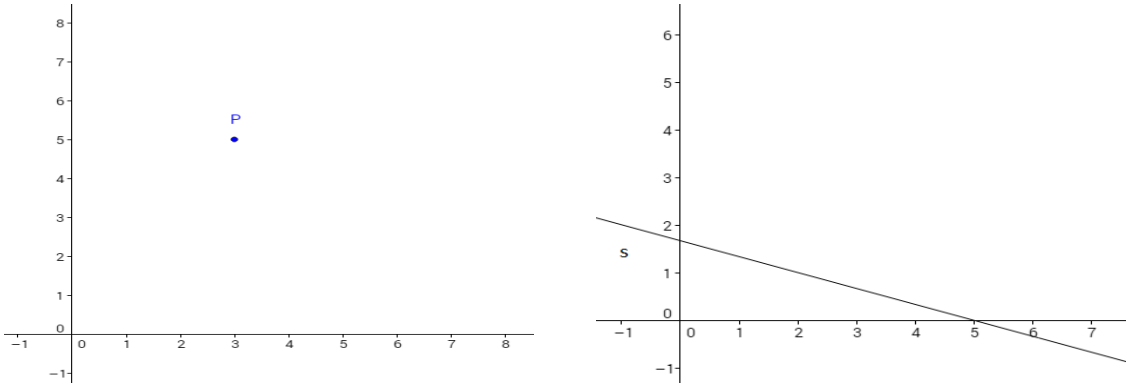
Idealizada por Paul Hough, na década de 60, conforme descrito em [7], o objetivo inicial do método era relacionado a identificação do trajeto de partículas subatômicas por meio da análise do "rastro"que essas deixavam. Tais "rastros"eram mapeados em imagens e então buscava-se identificar pontos colineares. Deste modo podemos entender que a técnica criada detectava retas nas imagens. Vamos entender o funcionamento deste modelo.

Uma reta pode ser descrita por uma equação do tipo

$$y = ax + b \tag{1.1}$$

onde a e b são sabidamente o coeficiente angular e o coeficiente linear, respectivamente. Assim Hough criou uma correspondência entre o "espaço imagem"($\mathscr{I}$) plano determinado pelos eixos coordenados x, y onde os pontos são identificados, e o "espaço transformado"($\mathscr{T}$) plano de eixos coordenados a, b onde são identificados os coeficientes angular e linear de uma reta em $\mathscr{I}$. Assim, dado um ponto $P = (x_1, y_1)$ no espaço imagem, e portanto fixado os

valores de (x, y) na equação 1.1, e variando os parâmetros (a, b) temos uma equação de reta em $\mathscr{T}$. Observe a figura 1.1.



(a) ponto no espaço imagem

(b) reta associada ao ponto P no espaço transformado

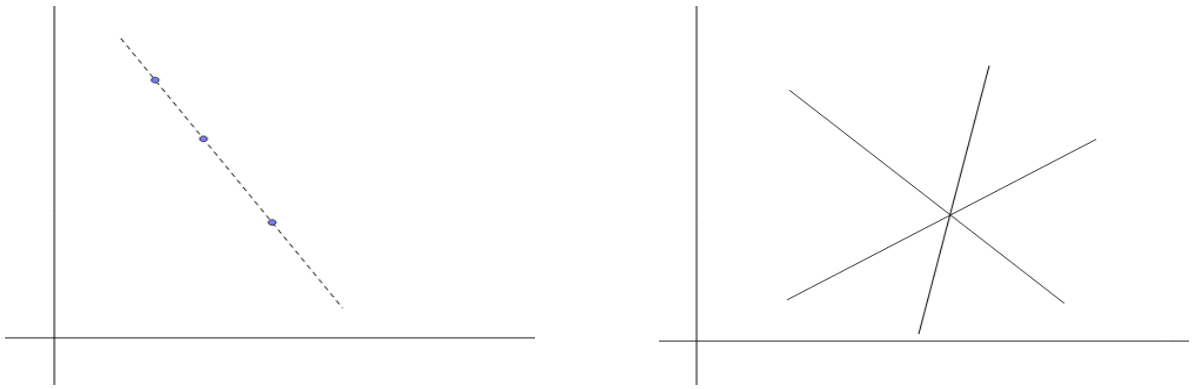
Figura 1.1:

Agora no espaço transformado temos uma reta s cujo conjunto de coordenadas de cada ponto descrevem todas as possíveis retas passando por P. Deste modo para um conjunto de pontos no espaço imagem temos um conjunto de retas no espaço transformado, mais que isso, dado um determinado conjunto de pontos $\{P_1, P_2, P_3 \ldots P_n\}$ de $\mathscr{I}$ tais que são todos colineares, no espaço transformado teremos uma intersecção no conjunto de retas que representam tais pontos. Essa intersecção será dada no ponto $Q = (a_1, b_1)$ que são coeficiente angular e linear, respectivamente, da reta que contém o conjunto de pontos. A argumentação é ilustrada na figura 1.2.

Desta maneira o que Hough fazia era transportar todos os pontos da imagem para o espaço transformado, associando-os à uma reta, e então buscar intersecções entre essas retas. Quanto maior o número de retas numa mesma intersecção, maior o número de pontos colineares, o que permitia identificar possíveis segmentos de retas na imagem.

1.2 Espaço Transformado (ρ, θ)

A técnica de identificação de pontos colineares, e portanto de retas, mostrou-se muito eficiente para o propósito de Hough. Contudo, de uma maneira geral este método



(a) três pontos colineares em $\mathscr{I}$ (b) intersecção das retas em $\mathscr{T}$

Figura 1.2:

possui alguns problemas, por exemplo o fato do espaço transformado não ser limitado gera um problema computacional, afinal torna-se inviável descrever todas possíveis retas passando por um determinado ponto. Além disso retas verticais não podem ser descritas pela equação 1.1. Essas observações foram feitas por Richard O. Duda e Peter E. Hart em [6], onde apresentaram uma método alternativo para a técnica de Hough. Este trabalho desenvolvido pelos dois pesquisadores, resultou no que hoje conhecemos como Transformada de Hough.

Ao identificar os problemas mencionados anteriormente no trabalho de Hough, Duda e Hart propuseram substituir a equação da reta por uma nova parametrização, nomeada por eles de parametrização normal. Observe na figura 1.3 que a reta está associada a dois parâmetros ρ e θ onde, o primeiro é o ângulo do vetor normal e o segundo a distância da reta até a origem. Do mesmo modo que a cada reta podia-se associar um único ponto no espaço transformado a, b agora podemos associar cada reta do espaço imagem a um único ponto no espaço transformado ρ, θ uma vez que restringimos o valor de θ ao intervalo $[0, \pi]$. Assim temos uma associação do espaço imagem à um espaço transformado limitado, o que computacionalmente é desejável para a estratégia que será explicada mais adiante. Além disso a parametrização dada pela equação

$$\rho = xcos\theta + ysen\theta$$

é capaz de descrever inclusive retas verticais ou com coeficiente linear com valor absoluto muito alto, tornando a detecção de retas completa.

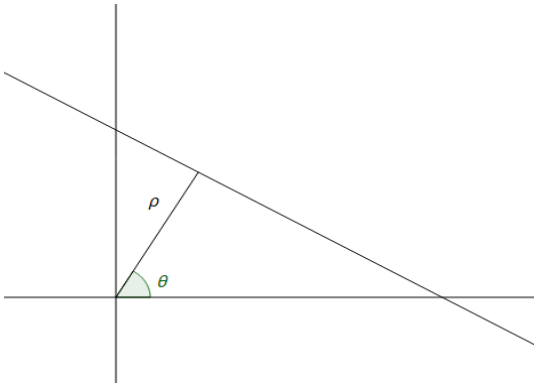


Figura 1.3:

Assim como acontecia no espaço transformado (x, y), em (ρ, θ) também associamos a cada ponto em $\mathscr{I}$ uma curva. Contudo, como podemos observar na equação 1.2, ao fixamos os valores de x e y e tomarmos como variáveis ρ e θ não teremos mais uma equação de reta, e sim uma senoidal que mantém as mesmas propriedades da elaboração original, ou seja, para cada ponto no espaço imagem atribui-se uma única senoidal no espaço transformado (lembre-se que limitamos o valor de θ), e cada ponto no espaço transformado corresponde a uma única reta no espaço imagem, deste modo dado um conjunto de pontos colineares em $\mathscr{I}$ a cada um deles temos uma senoidal associada em $\mathscr{T}$ de modo que existirá uma única intersecção entre elas, que se dá exatamente no ponto (ρ_0, θ_0) parâmetros que descrevem a reta que contem tais pontos na parametrização normal.

Vamos agora a técnica usada por Duda e Hart para identificar pontos colineares. Em primeiro lugar é necessário dizer que o trabalho foi realizado sobre imagens com um conjunto discreto de pontos pretos sobre um fundo branco. Além do parâmetro θ também se faz necessária uma limitação do parâmetro ρ, nesse caso tal parâmetro está limitado no intervalo $[-R, R]$, onde R é o tamanho da imagem (note que retas com $\mid \rho \mid \geq R$ não aparecem na imagem).Temos então um espaço transformado limitado em suas duas dimensões. Duda e Hart, assim como Hough definiram um erro aceitável para os parâmetros ρ e θ de modo a criar um espaço intervalado discreto, por meio deste espaço cria-se uma matriz de acumuladores de modo que cada ponto (x_i, y_i) no espaço imagem, a curva correspondente $\rho = x_i cos\theta + y_i sen\theta$ é descrita na matriz acumuladora incrementando cada célula de contagem por onde tal curva

passa, feito isso para todos os pontos, ao tomar as células com maiores valores, estamos basicamente detectando o maior número de intersecções das senoidais, assim o parâmetro (ρ, θ) atribuído a tal célula descreve a reta que contém os pontos colineares.

Vejamos um exemplo simples apenas para ilustrar como tal método funciona. Vamos supor que o espaço imagem contém somente 5 pontos conforme a figura 1.4. Por simplicidade vamos usar a formulação de Hough com equação de reta 1.1. Temos então em cada linha um valor para o parâmetro a e em cada coluna um valor para o parâmetro b. Em cada entrada a_{ij} da matriz, cujos parâmetros (a, b) correspondentes a linha e coluna são de uma reta que passa por um ponto da imagem, é somado o valor 1. Vejamos na tabela 1.1 a como fica a matriz acumuladora de pontos ao final do processo.

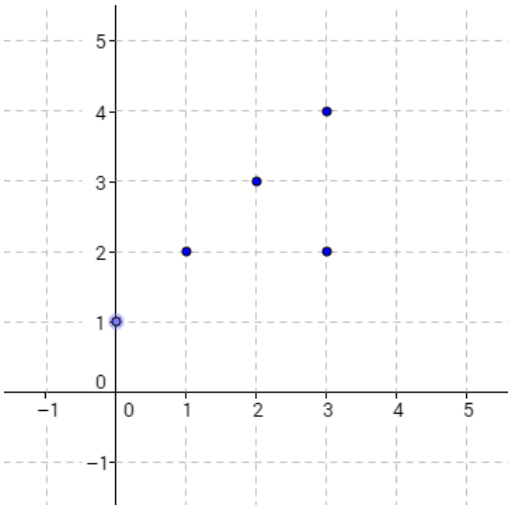


Figura 1.4: Pontos no espaço imagem

Além de aprimorar a técnica desenvolvida por Hough, Duda e Hart também propuseram que tal método poderia ser utilizado para detectar outras curvas, como por exemplo circunferências de modo que dado um ponto (x_0, y_0) poderia-se aplicar a mesma ideia para a equação $(x_0 - a)^2 + (y_0 - b)^2 = c^2$ criando um espaço transformado tridimensional (a, b, c), também deve ter três dimensões a matriz acumuladora.

a \ b	4	3	2	1	0	-1	-2	-3	-4	-5	-6	-7	-8	-9	-10	-11	-12	-13
0	1	1	2	1														
1				**4**		1												
2				1	1	1	1		1									
3				1		1		1		1		1						
4				1			1			1			1		1			
5				1				1				1				1		1

Tabela 1.1: Matriz de acumuladores

Capítulo 2

Otimização Irrestrita

O estudo de otimização é constituído pelo desenvolvimento de técnicas para localizar em um determinado conjunto Ω um valor x^* que torne o valor funcional mínimo ou máximo. Usualmente tratamos apenas da minimização de funções, uma vez que maximizar uma função f é equivalente a minimizar $-f$. Vamos agora apresentar a teoria de otimização de funções em $\Omega = \mathbb{R}^n$ na intenção de fornecer um suporte teórico para o desenvolvimento do problema abordado neste trabalho. O conteúdo aqui apresentado é resultado do estudo de [10] e [11], incluindo definições e demonstrações.

Antes de iniciar as definições e teoremas fazem-se necessárias algumas ressalvas. Primeiramente como já foi mencionado, trabalha-se em geral apenas com o conceito de minimização de funções. Além disso as funções que utilizadas no desenvolvimento trabalho são todas de classe C^∞, ainda que para a teoria apresentada seja suficiente supor C^1. Por padrão utilizaremos $\|\cdot\|$ para designar a norma euclidiano. Se durante o trabalho se fizer necessário o uso de outra norma, este será destacado, embora a maioria dos resultados independam da norma utilizada. Por fim, embora existam outros métodos de minimização irrestrita, como região de confiança apresentada no capítulo 4 de [11], aqui falaremos somente sobre métodos de busca linear.

2.1 Busca Linear

Ao estabelecer um algoritmo para minimização irrestrita de uma função, estamos, geralmente, desenvolvendo um processo iterativo de modo que dado um certo ponto x_n, onde $\nabla f(x_n) \neq 0$, seja possível determinar um ponto x_{n+1}, tal que $f(x_{n+1}) < f(x_n)$.

Na busca linear dado x_k procuramos o próximo ponto em uma direção d_k, que seja *direção de descida*, isto é, existe um certo $\varepsilon > 0$ tal que para todo $t \in (0, \varepsilon]$ temos $f(x_k + td_k) < f(x_k)$. Podemos ainda caracterizar tais funções utilizando o gradiente da função, veja o lema a seguir.

Lema 2.1. *Se $\nabla f(x)^T d < 0$ então para todo $t > 0$ suficientemente pequeno, $f(x+td) < f(x)$.*

Demonstração: Observe que $\nabla f(x)^T d = \lim_{t\to 0} \dfrac{f(x+td) - f(x)}{t}$. Como na hipótese temos que $\nabla f(x)^T d < 0$ então segue que para todo $t > 0$ suficientemente pequeno $f(x+td) < f(x)$. □

Observe que, usando a caracterização dada pelo lema anterior temos que $-\nabla f(x)$ é uma direção de descida (chamada *direção de máxima descida*). Deste modo é assegurado que dado um ponto x_k que não anula o gradiente, sempre é possível obter uma direção de descida, lembrando que estamos assumindo que a função é de classe C^1.

Infelizmente pedir somente que $f(x + td) < f(x)$ não é suficiente para garantir que depois de um determinado número de iterações atinja-se um ponto estacionário, isso porque não estamos impondo um controle sobre o tamanho do passo definido por t. Perceba que se tomarmos para t escolhas ruins isso pode gerar uma convergência muito lenta, ou ainda, o método pode ficar preso em um ponto não estacionário. Assim precisamos fazer escolhas inteligentes, de modo que t forneça um decréscimo suficiente. Para realizar essa escolha temos a chamada condição de Armijo que estabelece uma relação entre o tamanho do passo e a norma do vetor gradiente, esta condição é apresentada pelo teorema a seguir.

Teorema 2.2 (Condição de Armijo)**.** *Sejam $x, d \in \mathbb{R}^n$ tais que $\nabla f(x) \neq 0$, $\nabla f(x)^T d < 0$ e $\alpha \in (0, 1)$. Existe um $\varepsilon(\alpha) > 0$ tal que*

$$f(x + td) \leq f(x) + \alpha t \nabla f(x)^T d \qquad \forall t \in (0, t] \tag{2.1}$$

Demonstração: Como $\nabla f(x) \neq 0$ então temos que

$$\lim_{t\to 0} \frac{f(x+td) - f(x)}{t} = \nabla f(x)^T d \neq 0.$$

Deste modo podemos escrever

$$\lim_{t\to 0}\frac{f(x+td)-f(x)}{t\nabla f(x)^T d}=1.$$

Assim existe um certo $\varepsilon>0$ tal que $\forall t\in(0,\varepsilon]$ tem-se

$$\frac{f(x+td)-f(x)}{t\nabla f(x)^T d}\geq\alpha,$$

e portanto $f(x+td)\leq f(x)+\alpha t\nabla f(x)^T d$. □

Ainda que tenhamos agora determinado um controle sobre o tamanho do passo, esse não é completamente efetivo uma vez que a direção d também pode se tornar demasiadamente pequena, fazendo com que a sequência gerada na busca linear convirja novamente para um ponto não estacionário, este tipo de situação é contornada por meio da *condição* β que estabelece um controle sobre a norma de d_k a cada passo.

Dado um parâmetro $\beta>0$ estabelecemos que a cada iteração a direção d_k satisfaça

$$\|d_k\|\geq\beta\|\nabla f(x_k)\|. \tag{2.2}$$

Assim, controlamos a norma de d_k em função do gradiente da função no ponto x_k.

Essas duas condições são suficientes para garantir o controle dos passos dados em cada iteração. Como $f(x+td)<f(x)$ não era uma condição suficiente, também não é suficiente pedir somente que $\nabla f(x)^T d<0$. De fato, em algumas situação as direções d_k escolhidas em cada iteração podem gerar uma sequência convergindo para uma certa direção d ortogonal a $\nabla f(x)$. Como não desejamos que isso aconteça, vamos impor a *condição do ângulo*.

Sabemos da geometria analítica que dados dois vetores $\nabla f(x_k)$ e d_k vale a expressão

$$\frac{\nabla f(x_k)^T d_k}{\|\nabla f(x_k)\|\|d_k\|}=\cos\alpha$$

onde α representa o ângulo formado entre tais vetores. Nesse sentido como desejamos afastar as direções d_k de direções ortogonais ao gradiente no ponto x_k, impomos por meio de um parâmetro $\theta\in(0,1)$ que a cada iteração, a direção usada satisfaça

$$f(x_k)^T d_k\leq-\theta\|\nabla f(x_k)\|\|d_k\|. \tag{2.3}$$

Antes de estabelecermos um algoritmo conceitual, vamos destacar que embora tenhamos definido o decréscimo suficiente para a função f pela equação 2.1, não foi estabelecido como fazer a escolha de um t apropriado. Entre os métodos que podem ser escolhidos, optamos por usar *backtracking*, que consiste em, a cada passo, escolher o maior $t \in \{\frac{1}{2}, \frac{1}{4}, \cdots\}$ satisfazendo a condição de Armijo.

Algoritmo 2.3 (Busca linear com *backtracking*)**.** Defina $x_0 \in \mathbb{R}^n, \alpha \in (0,1), \beta > 0$ e $\theta \in (0,1)$ Dado x_k pela k-ésima iteração determine x_{k+1} da seguinte forma:

1. Se $\nabla f(x_k) = 0$, pare!
2. Escolha $d_k \in \mathbb{R}^n$ tal que
$$\|d_k\| \geq \beta \|\nabla f(x_k)\|$$
$$\nabla f(x_k)^T d_k \leq -\theta \|\nabla f(x_k)\| \|d_k\|$$
3. Defina $t = 1$
4. Enquanto $f(x_k + td_k) > f(x_k) + \alpha t \nabla f(x_k)^T d_k$ tome $t \leftarrow \bar{t} \in [0, 1t, 0.9t]$
5. Faça $x_{k+1} = x_k + td_k$

Na prática escolhemos valores bem pequenos para α, em geral $\alpha = 10^{-4}$ ([11]), e $\theta = 10^{-6}$ ([10]), entretanto a escolha de β é um tanto mais complexa e deve ser avaliada conforme o problema empregado, é sugerido que β assuma o inverso de uma cota superior para a norma da matriz hessiana, pois isso não inibe a aceitação das direções de Newton, que serão esclarecidas posteriormente.

Com base no desenvolvimento dado pelo texto acima, é fácil verificar que o algoritmo 2.3 está bem definido, além disso é possível provar a convergência global do algoritmo, ou seja, independente do ponto inicial escolhido. Esse resultado será demonstrado pelo teorema a seguir, tal teorema e demonstração dada aqui encontram-se [10].

Teorema 2.4 (Convergência global)**.** *Se x^* é ponto limite de uma sequência gerado pelo algoritmo 2.3 então $\nabla f(x^*) = 0$.*

Demonstração: Seja $s_k = x_{k+1} - x_k = td_k \forall k \in \mathbb{N}$. Seja K_1 um subconjunto infinito de $\mathbb{N}$, vamos dividir a demonstração em dois casos:

1. $\lim_{k \in K_1} \|s_k\| = 0$
2. Existe K_2 subconjunto de K_1 e $\varepsilon > 0$ tal que para todo $k \in K_2$ tem-se $\|s_k\| \geq \varepsilon$.

Supondo que vale o item (1) temos ainda duas outras possibilidades

1.1 Se existe uma subsequência de s_k, com índices em K_3 tal que $s_k = d_k$, então

$$\|\nabla f(x^*)\| = \lim_{k \in K_3} \|\nabla f(x_k)\| \leq \lim_{k \in K_3} \frac{\|d_k\|}{\beta} = \lim_{k \in K_3} \frac{\|s_k\|}{\beta} = 0;$$

1.2 Se para todo $k \in K_1$ com $k > k_0$ temos $t < 1$, então existe um $\overline{s_k}$ múltiplo de s_k tal que $\|\overline{s}_k\| \leq 10\|s_k\|$ e $f(x_k + \overline{s}_k) > f(x_k) + \alpha \nabla f(x_k)^T \overline{s}_k$.

Sabemos que $\lim_{k \in K_1} \|\overline{s}_k\| = 0$ e para todo $k \in K_1, k > k_0$ temos

$$\nabla f(x_k)^T \overline{s}_k \leq -\theta \|\nabla f(x_k)\| \|\overline{s}_k\|. \tag{2.4}$$

Seja v um ponto de acumulação da sequência $\dfrac{\overline{s}_k}{\|\overline{s}_k\|}$, então temos que $\|v\| = 1$. Passando a uma subsequência de índices em K_4 temos que $\lim \dfrac{\overline{s}_k}{\|\overline{s}_k\|} = v$.

Portanto,

$$\nabla f(x^*)^T v = \lim \nabla f(x_k)^T v = \lim \nabla f(x_k) \frac{\overline{s}_k}{\|\overline{s}_k\|}.$$

De 2.4 segue que

$$\nabla f(x_*)^T v \leq -\theta \lim \|f(x_k)\| \tag{2.5}$$

Agora para todo $k \in K_4$

$$f(x_k + \overline{s}_k) - f(x_k) = \nabla f(x_k + \xi_k \overline{s}_k)^T \overline{s}_k \qquad \xi_k \in (0, 1).$$

Veja que a condição de Armijo não foi satisfeita para $\overline{s}_k$, assim

$$\nabla f(x_k + \xi_k \overline{s}_k)^T \frac{\overline{s}_k}{\|\overline{s}_k\|} > \alpha \nabla f(x_k)^T \overline{s}_k \qquad \forall k \in K_4,$$

ou seja,

$$\nabla f(x_k + \xi_k \overline{s}_k)^T \frac{\overline{s}_k}{\|\overline{s}_k\|} > \alpha \nabla f(x_k)^T \frac{\overline{s}_k}{\|\overline{s}_k\|}.$$

Passando o limite temos

$$\nabla f(x^*)^T v \geq \alpha \nabla f(x^*)^T v$$

ou

$$(1-\alpha)\nabla f(x^*)^T v \geq 0$$

Logo $\nabla f(x^*)^T v \geq 0$ e por 2.5 segue que $\nabla f(x^*)^T v = 0$

Se $\nabla f(x_*) \neq 0$, novamente por 2.5, para k suficientemente grande

$$0 = \nabla f(x^*)^T v \leq -\theta \|\nabla f(x_k)\| < 0$$

contradição, portanto temos que $\nabla f(x^*) = 0$ □

Vamos agora considerar que é válido o item (2).

Como $\|s_k\| \geq \varepsilon \forall k \in K_4$, pela condição de Armijo temos que

$$\begin{aligned} f(x_k + s_k) &\leq f(x_k) + \alpha \nabla f(x_k)^T s_k \\ &\leq f(x_k) - \alpha\theta \|\nabla f(x_k)\| \|s_k\| \\ &\leq f(x_k) - \alpha\theta\varepsilon \|\nabla f(x_k)\| \quad \forall k \in K_4. \end{aligned}$$

Portanto,

$$\frac{f(x_k) - f(x_{k+1})}{\alpha\theta\varepsilon} \geq \|\nabla f(x_k)\| \geq 0.$$

Passando ao limite concluímos que $\nabla f(x_k) = 0$.

O método de busca linear apresentado fornece um algoritmo com convergência global. No entanto ele não define uma direção de descida a ser usada, por este motivo precisamos recorrer a outros métodos que forneçam uma escolha inteligente para tal direção. Existem uma série de métodos que podem ser utilizados nesse sentido, como por exemplo o método de máxima descida, neste, usa-se a cada iteração k a direção de menos o gradiente, ou seja, dado um x_k tomamos $d_k = -\nabla f(x_k)$. Nesse trabalho apresentaremos dois métodos, Newton e BFGS ambos dentro do contexto de busca linear que já foi apresentado.

2.2 O método de Newton

Umas das direções mais importantes, possivelmente, são as chamadas direções de Newton. Tais direções são derivadas da expansão da série de Taylor de segunda ordem [11] e são expressas pela seguinte equação $d_k = -(\nabla^2 f(x_k))^{-1}\nabla f(x_k)$, onde estamos assumindo que a hessiana $\nabla^2 f(x_k)$ é definida positiva. O método de Newton é uma variação do método usado para encontrar raízes de funções e posteriormente adaptado para resolver sistemas não lineares, conforme pode ser verificado em [10]. Para melhor compreendê-lo vamos fazer algumas considerações acerca de quadráticas.

Dada uma função quadrática da forma $g(x) = x^T Gx + b^T x + c$ onde G é uma matriz $n \times n$ definida positiva e $x \in \mathbb{R}^n$, sabemos do cálculo diferencial que esta função possui um mínimo global, mais que isso, partindo de qualquer ponto de $\mathbb{R}^n$ usando a direção de Newton em um passo atingimos esse mínimo, de fato temos $\nabla g(x) = Gx + b$ é o gradiente da função $g(x)$ assim é fácil ver que tomando a direção d_k zeramos o gradiente. Deste modo, temos direções que funcionam muito bem para quadráticas, assim para utilizar essa vantagem, tomamos uma aproximação local da função objetivo para uma forma quadrática, usando a fórmula de Taylor de segunda ordem.

É importante ressaltar que as direções fornecidas por Newton só podem ser assumidas como direção de descida se temos que a hessiana da função f é definida positiva (pág. 44 de [11]), entretanto para isso seria necessário pedir que a função f fosse convexa (ver convexidade em [10]), mas desejamos estabelecer um método de convergência global, mantendo assim a principal característica da busca linear. Assim usamos uma atualização da diagonal da matriz hessiana quando esta não for definida positiva. Salvo essas alterações temos seguinte algoritmo.

Algoritmo 2.5 (Newton com Busca Linear)**.** Defina $x_0 \in \mathbb{R}^n, \alpha \in (0,1), \beta > 0$ e $\theta \in (0,1)$ Dado x_k pela k-ésima iteração determine x_{k+1} da seguinte forma:

1. Se $\nabla f(x_k) = 0$, pare!

2. Obter fatoração Cholesky $\nabla^2 f(x_k) = LDL^T$;

3. Se 2 falhar defina $B_k = \nabla^2 f(x_k) + \mu I$ para $\mu > 0$ de modo que $B_k > 0$, então obtenha a fatoração Cholesky $B_k = LDL^T$;

4. Definir d_k resolvendo

$$Ly = -\nabla f(x_k) \text{ e } DL^T d_k = y;$$

5. Se $\nabla f(x_k)^T d_k > -\theta \|\nabla f(x_k)\| \|d_k\|$ faça $\mu \leftarrow \max\{2\mu, 10\}$ e volte em 3;

6. Se $\|d_k\| < \beta \|\nabla f(x_k)\|$, faça

$$d_k \leftarrow \beta \frac{\|\nabla f(x_k)\|}{\|d_k\|} d_k;$$

7. Obter t por *backtracking* que satisfaça

$$f(x_k + t d_k) \leq f(x_k) + \alpha t \nabla f(x_k)^T d_k;$$

8. Defina $x_{k+1} = x_k + t d_k$ e volte a 1.

2.3 A fórmula BFGS

O método de Newton possui boas propriedades de convergência [4]. Contudo, pode não ser apropriado em alguns casos por fazer-se necessário cálculo da hessiana da função. Este cálculo pode ser um tanto quanto inconveniente por diversos motivos: custo computacional, complexidade da função e mesmo a propensão a erros humanos, assim faz sentido desenvolver métodos que conservam de certo modo, as boas propriedades de convergência e no entanto não seja necessário o cálculo da hessiana. Essa é a ideia primordial dos chamados métodos de quase Newton, que buscam direções aproximadas das direções de Newton utilizando uma matriz B_k próxima de $\nabla^2 f(x_k)$ a cada iteração. Para fazer essas atualizações busca-se uma matriz B_k que satisfaça a equação secante

$$B_{k+1} s_k = y_k \text{ onde } s_k = x_{k+1} - x_k \text{ e } y_k = \nabla f(x_{k+1}) - \nabla f(x_k)$$

pois esta, em geral fornece, boas aproximações da hessiana [10].

Existem muitas fórmulas para a atualização de B_k. A fórmula que vamos tratar aqui impõe que B_k seja simétrica, e é dada pela seguinte expressão:

$$B_{k+1} = B_k + \frac{y_k y_k^T}{y_k^T s_k} - \frac{B_k s_k s_k^T B_k}{s_k^T B_k s_k}.$$

Conhecida como fórmula BFGS, desenvolvida independentemente por Broyden, Fletcher, Goldfarb e Shanno, em 1970 [10], esta é uma atualização de posto 2 que e possui uma propriedade conveniente a cerca da positividade da matriz B_{k+1} que será apresentada no teorema a seguir.

Teorema 2.6. *Na fórmula BFGS, se B_k é simétrica definida positiva e $s_k^T y_k > 0$, então B_{k+1} também é simétrica e definida positiva.*

Não apresentaremos aqui a demonstração deste teorema devido a sua simplicidade. Além disso pode ser encontrada na página 116 de [10]. Este teorema é fundamental, pois um dos requisitos para se obter direções de descida apresentado na seção anterior, era que a matriz hessiana fosse definida positiva, assim temos controle sobre a positividade em cada atualização. Uma vez que possa ser determinada a aproximação B_k da hessiana de $f(x_k)$ simétrica e definida positiva, sabemos por meio de um cálculo simples de $s_k^T y_k$ se a matriz B_{k+1} será também definida positiva. Deste modo para obter uma direção de descida d_k basta resolver o sistema linear

$$B_k d_k = -\nabla f(x_k).$$

Uma vez que temos B_k definida positiva, sabemos que o sistema acima possui solução única determinada por $d_k = -B_k^{-1}\nabla f(x_k)$. Assim seria conveniente que em vez de resolver o sistema linear, encontremos uma atualização para B_k^{-1}, como a fórmula BFGS é uma atualização de posto 2 a inversa de B_{k+1} já pode ser calculada como uma atualização da inversa de B_k, isso é feito por meio da seguinte formulação

$$B_{k+1}^{-1} = B_k^{-1} + \frac{(s_k - B_k^{-1}y_k)s_k^T + s_k(s_k - B^{-1}y_k)^T}{s_k^T y_k} - \frac{(s_k - B_k^{-1}y_k)^T y_k s_k s_k^T}{(s_k^T y_k)^2}$$

por uma questão de simplicidade de notação usaremos H_k para representar B_k^{-1}.

Agora que definimos uma forma adequada de encontrar direções de descida, podemos definir um algoritmo de busca linear usando as direções fornecidas com o cálculo da

fórmula BFGS. Apenas faz-se necessário uma correção para o caso em que a d_k não satisfaça a condição 2.3. Neste caso simplesmente descartaremos a direção fornecida por BFGS e usaremos a direção de máxima descida $-\nabla f(x_k)$, embora na prática essa situação seja rara.

Algoritmo 2.7. Defina $x_0 \in \mathbb{R}^n, \alpha \in (0,1), \beta > 0$ e $\theta \in (0,1)$ e uma matriz $H_0 \in \mathbb{R}^{n\times n}$ simétrica e definida positiva. Dado x_k pela k-ésima iteração determine x_{k+1} da seguinte forma:

1. Se $\nabla f(x_k) = 0$, pare!
2. Defina $d_k = -H_k \nabla f(x_k)$;
3. Se $\nabla f(x_k)^T d_k > -\theta \|\nabla f(x_k)\| \|d_k\|$ então

$$H_k \leftarrow I$$

$$d_k \leftarrow -\nabla f(x_k);$$

4. Se $\|d_k\| < \beta \dfrac{\|\nabla f(x_k)\|}{\|d_k\|}$ então corrija usando

$$d_k \leftarrow \beta \frac{\|\nabla f(x_k)\|}{\|d_k\|} d_k;$$

5. Obter t por *backtracking* que satisfaça

$$f(x_k + t d_k) \leq f(x_k) + \alpha t \nabla f(x_k)^T d_k;$$

6. Defina:

$$x_{k+1} = x_k + t d_k$$

$$s_k = x_{k+1} - x_k$$

$$y_k = \nabla f(x_{k+1}) - \nabla f(x_k);$$

7. Se $s_k^T y_k > 0$ defina

$$H_{k+1} = H_k + \frac{(s_k - H_k y_k) s_k^T + s_k (s_k - H_K y_k)^T}{s_k^T y_k} - \frac{(s_k - H_k y_k)^T y_k s_k s_k^T}{(s_k^T y_k)^2}$$

   caso contrário defina $H_{k+1} = H_k$;

8. Volte ao passo 1.

2.4 O Problema LOVO

Uma vez que temos estabelecidos os fundamentos básicos para o estudo de otimização com busca linear, vamos agora apresentar o problema de otimizar funções de menor valor ordenado conforme R. Andreani, J. M. Martínez, L. Martínez e F. Yano em [2].

Dado um conjunto finito de r funções reais $\{F_1, F_2, \dots F_r\}$ definidas em um conjunto $\Omega \subseteq \mathbb{R}^n$ e um inteiro $1 \leq p \leq r$, definimos a função de menor valor ordenado $S_p : \Omega \to \mathbb{R}$ por

$$S_p = \sum_{j=1}^{p} F_{i_j(x)}(x).$$

Tal função está definida para todo x em Ω onde $\{i_1(x), \dots, i_r(x)\}$ é uma reordenação dos índices das funções F_i em função de x de modo que

$$F_{i_1(x)}(x) \leq F_{i_2(x)}(x) \leq \dots \leq F_{i_r(x)}(x).$$

Note que a continuidade das funções F_i implicam diretamente na continuidade da função S_p, no entanto o mesmo não se pode afirmar sobre a diferenciabilidade, uma vez que, ainda que todas as F_i sejam diferenciáveis, a função S_p é geralmente não suave, salvo casos muitos particulares.

Agora que definimos S_p podemos apresentar o problema de otimização da função de menor valor ordenado - LOVO (*Low Order-Value Optimization*) como o seguinte

$$\min S_p(x)$$

$$\text{s. a. } x \in \Omega$$

Veja que existem algumas dificuldades ao lidarmos com funções de menor valor ordenado uma vez que não temos a diferenciabilidade requerida pelas teorias apresentadas até o momento. Para contornar este problemas, vamos utilizar uma reformulação do problema [2] de modo a aproveitar a diferenciabilidade das funções F_i. Vamos definir m como o número de combinações dos elementos do conjunto de índices de F_i com p elementos, assim teremos m subconjuntos $C_1, C_2, \dots, C_m$ cada um com cardinalidade p. Para todo $i = 1, \dots, m$ e $x \in \Omega$ defina

$$f_{min}(x) = \min\{f_1(x), \dots, f_m(x)\}.$$

Observe que $f_{min} = S_p$ para todo $x \in \Omega$ deste modo o problema LOVO pode ser escrito como

$$\min\ f_{min}(x)$$

$$\text{s. a. } x \in \Omega$$

a partir de agora esta será a formulação utilizada para o desenvolvimento apresentado neste trabalho, embora na prática não seja necessário o cálculo de todas as f_i para determinar f_{min} usaremos isso como recurso teórico. Começaremos por um lema, e posteriormente um teorema que relaciona os minimizadores de f_{min} e minimizadores de f_i para todo $i \in I_{min}(x)$ onde $I_{min}(x) = \{i \in \{1, \ldots, m\} \mid f_i(x) = f_{min}(x)\}$ [2].

Lema 2.8. *Seja $A \subseteq \Omega$ e $x \in A$. Se o ponto x_* é um minimizador global de $f_{min}(x)$ sujeito a $x \in A$, então x_* é um minimizador global de $f_i(x)$ sujeito a $x \in A$ para todo $i \in I_{min}(x_*)$. Em particular, se $A = \Omega$, um x_* minimizador global de $f_{min}(x)$ é minimizador global de $f_i(x)$ com $i \in I_{min}(x_*)$.*

Demonstração: Assuma que para algum $i \in I_{min}(x_*)$, x_* não é minimizador global de $f_i(x)$ sujeito a $x \in A$. Então existe um certo $y \in A$ tal que $f_i(y) < f_i(x_*)$. Deste modo temos, das definições de f_{min} e $I_{min}(x_*)$, que

$$f_{min}(y) \leq f_i(y) < f_i(x_*) = f_{min}(x_*)$$

logo x_* não é minimizador global de f_{min} sujeito a $x \in A$, contradição, portanto x_* é minimizador para toda f_i com $i \in I_{min}(x_*)$. □

O lema acima acerca de minimizadores globais será agora adaptado para minimizadores locais pelo seguinte teorema

Teorema 2.9. *Se x_* é minimizador local de $f_{min}(x)$ então $\forall i \in I_{min}(x_*)$ x_* é minimizador local de $f_i(x)$ sujeito a $x \in \Omega$.*

Demonstração: Seja $\varepsilon > 0$ tal que x_* é minimizador global de $f_{min}(x)$ sujeito a $x \in A$ onde $A - \{x \in \Omega \mid \|x - x_*\| < \varepsilon\}$, do lema anterior temos que x_* é minimizador global de $f_i(x)$ sujeito a $x \in A$ para todo $i \in I_{min}(x_*)$. Então, x_* é minimizador local de $f_i(x)$ sujeito a $x \in \Omega$ para todo $i \in I_{min}(x_*)$. □

Infelizmente a reciprocidade do teorema não é verdadeira, mesmo para exemplos muitos simples pode-se ver que ela falha (veja em [2]). No entanto o teorema a seguir mostra que se impormos algumas condições sobre a continuidade das f_i podemos contornar isso.

Teorema 2.10. *Assuma que x_* é um minimizador local de $f_i(x)$ para todo $i \in I_{min}(x_*)$ e que f_i é contínua em x_* para todo $i \notin I_{min}(x_*)$ então x_* é minimizador local de $f_{min}(x)$.*

Demonstração: Como $f_i(x)$ é contínua para todo $i \notin I_{min}(x_*)$ então existe $\delta_1 > 0$ tal que

$$f_i(x) > f_{min}(x_*) \qquad \forall i \notin I_{min}(x_*) \text{ e } x \in B_{\delta_1}(x_*).$$

Da hipótese existe $\delta_2 > 0$ tal que para todo $i \in I_{min}(x_*)$ temos que $f_i(x) \geq f_i(x_*) = f_{min}(x_*)$ sempre que $\|x - x_*\| \leq \delta_2$.

Defina $\delta = \min\{\delta_1, \delta_2\}$ então temos que para todo $x \in \Omega$ tal que $\|x - x_*\| \leq \delta$ e para todo $i = 1, \ldots, m$ temos

$$f_i(x) \geq f_{min}(x_*).$$

Portanto temos que $f_{min}(x) \geq f_{min}(x_*)$ para todo $x \in B_\delta(x_*) \cap \Omega$ como queríamos mostrar. □

Antes de tratar sobre o algoritmo de resolução do problema LOVO vamos apresentar um importante teorema a cerca das condições de otimalidade [2].

Teorema 2.11. *Seja $x_* \in \Omega$ um minimizador local de $f_{min}(x)$ onde todas as funções $f_i(x)$ são diferenciáveis em um aberto que contém Ω. Então, para todo $i \in I_{min}(x_*)$ x_* satisfaz as condições necessárias de otimalidade associadas ao problema*

$$min\ f_i(x)$$

$$s.\ a.\ x \in \Omega$$

Demonstração: Do teorema anterior temos que x_* é um minimizador local de $f_i(x)$ para todo $i \in I_{min}(x_*)$, então, x_* satisfaz as condições necessárias de otimalidade associadas a este problema. □

Os teoremas acima apresentados nos fornece uma importante informação a respeito dos minimizadores locais do problema LOVO na formulação que estamos utilizando (com as f_i). Assim vamos definir duas situações que podem acontecer em relação às condições de otimalidade das funções f_i. Se $x_* \in \Omega$ é ponto crítico para alguma $f_i(x)$ com $i \in I_{min}(x_*)$ dizemos então que x_* é fracamente crítico, se x_* é ponto crítico para todas as $f_i(x)$ tal que $i \in I_{min}$ então x_* é chamado de ponto fortemente crítico.

Como em nosso trabalho encontrar pontos fortemente críticos teria altíssimo custo computacional, e de uma maneira geral isso, na prática, não é usual, trabalharemos aqui apenas com o algoritmo pra encontrar pontos fracamente críticos. Além disso como será visto nos capítulos subsequentes trabalharemos apenas com o problema LOVO irrestrito portanto para um estudo do problema LOVO com restrições veja [3].

Como vimos nas seções anteriores os métodos de otimização apresentados são baseados em iterações que requerem o cálculo das derivadas de primeira e (no método de Newton) segunda ordem, mas essas derivadas podem sequer existir uma vez que a função S_p é geralmente não suave, o que faremos aqui é usar a formulação do problema com f_{min}, nesse caso aproveitaremos a diferenciabilidade das f_i, assim em um determinando ponto x_k tomaremos $\nabla f_{min}(x_k)$ como $\nabla f_i(x_k)$ para um certo $i \in I_{min}(x_k)$, note que o mesmo conceito pode ser entendido para derivadas segundas. Apesar de a princípio parecer uma estratégia duvidosa, vamos utilizá-la para definir o algoritmo e de pois mostrar que ela funciona bem para tal formulação.

O algoritmo que vamos apresentar agora pode ser usado para encontrar pontos fracamente críticos para o problema LOVO irrestrito, usando métodos de busca linear onde estamos assumindo que todas as f_i são diferenciáveis.

Algoritmo 2.12 (LOVO irrestrito com busca linear)**.** Defina $x_0 \in \mathbb{R}^n, \alpha \in (0,1), \beta > 0$ e $\theta \in (0,1)$ Dado x_k pela k-ésima iteração determine x_{k+1} da seguinte forma:

1. Escolha $r(k) \in I_{min}(x_k)$

2. Se $\nabla f_{r(k)}(x_k) = 0$, pare!

3. Escolha $d_k \in \mathbb{R}^n$ tal que

$$\|d_k\| \geq \beta \|\nabla f_{r(k)}(x_k)\|$$

$$\nabla f_{r(k)}(x_k)^T d_k \leq -\theta \|\nabla f_{r(k)}(x_k)\| \|d_k\|$$

4. Defina $t = 1$

5. Enquanto $f_{min}(x_k + td_k) > f_{min}(x_k) + \alpha t \nabla f_{r(k)}(x_k)^T d_k$ tome $t \leftarrow \bar{t} \in [0, 1t, 0.9t]$

6. Faça $x_{k+1} = x_k + td_k$

Vamos agora mostrar que tal algoritmo converge de fato para pontos fracamente críticos, a demonstração aqui apresentada é baseada na demonstração fornecida em [3].

Teorema 2.13. *O algoritmo 2.12 está bem definido e converge para x_k se, e somente se, x_k é um ponto fracamente crítico.*

Demonstração: Suponha por um momento que x_k não é um ponto fracamente crítico, então temos que $\nabla f_{r(k)}(x_k) \neq 0$ assim das condições 2.2 e 2.3 e da diferenciabilidade de $f_{r(k)}$ temos que

$$\lim_{t \to 0} \frac{f_{r(k)}(x_k + td_k) - f_{r(k)}(x_k)}{t} = \nabla f_{r(k)}(x_k)^T d_k < 0$$

ou ainda,

$$\lim_{t \to 0} \frac{f_{r(k)}(x_k + td_k) - f_{r(k)}(x_k)}{t \nabla f_{r(k)}(x_k)^T d_k} = 1$$

como sabemos que $\alpha < 1$ então para um t pequeno o suficiente temos

$$\frac{f_{r(k)}(x_k + td_k) - f_{r(k)}(x_k)}{t \nabla f_{r(k)}(x_k)^T d_k} \geq \alpha$$

como $\nabla f_{r(k)}(x_k)^T d_k < 0$ então

$$f_{r(k)}(x_k + td_k) \leq f_{r(k)}(x_k) + \alpha t \nabla f_{r(k)}(x_k)^T d_k$$

mas sabemos que $f_{min}(x_k + td_k) \leq f_{r(k)}(x_k + td_k)$ e $f_{min}(x_k) = f_{r(k)}(x_k)$ então

$$f_{min}(x_k + td_k) \leq f_{min}(x_k) + \alpha t \nabla f_{r(k)}(x_k)^T d_k.$$

Escolha um t_k por *backtraking* que satisfaça a equação anterior.

Isso prova que enquanto x_k não é um ponto fracamente crítico, o ponto x_{k+1} pode ser encontrado. □

Capítulo 3

Formulação LOVO para detecção de curvas

Uma vez que os capítulos anteriores forneceram uma base teórica para o estudo que desenvolvemos, seguimos agora para o problema que foi trabalhado utilizando a primeira formulação sob o ponto de vista da otimização.

Neste capítulo abordaremos apenas casos de otimização irrestrita $\Omega = \mathbb{R}^n$.

3.1 Construção de S_p

Sabemos do capítulo anterior que dado um conjunto $\mathscr{F}$ de r funções reais $F_i(x)$ definidas em $\Omega \subset \mathbb{R}^n$, o problema LOVO consiste em minimizar a função $S_p(x)$ dada pela soma das p funções de $\mathscr{F}$ com menor valor funcional em x. Agora partindo da ideia de que cada função $T(x)$ representa a lei de correspondência de um determinado fenômeno a ser modelado, e temos um conjunto $y_1, y_2, \ldots, y_r$ de observações desse fenômeno. Se tomarmos F_i como o erro quadrado da i-ésima observação, isto é, $F_i(x) = (T(x_i) - y_i)^2$ então o problema

$$\min S_r(x)$$

$(p = r)$, corresponde exatamente ao problema de quadrados mínimos, e nesse sentido pode-se dizer que LOVO é uma generalização dos quadrados mínimos [2]. Além disso ao escolhermos um parâmetro $p < r$ o problema LOVO descarta os maiores erros quadrados, ou seja, descarta os erros com maior influência no ajuste. Deste modo o LOVO é uma ferramenta para determinar funções de ajuste com descarte automático de *outliers*.

Como sabemos a identificações de curvas pela transformada de Hough se dá pela relação entre o espaço imagem e o espaço transformado de modo que para cada ponto no primeiro corresponda uma equação no segundo, cujos valores que satisfazem a equação, representam todas as possibilidades de curvas passando por tal ponto. Matematicamente dado um ponto P, associa-se a ele uma equação do tipo $h_P(x) = y$. No entanto o método proposto diferencia-se de Hough nesse ponto, pois não estamos mais interessados em encontrar intersecções entre as soluções do conjunto de equações, ao invés disso transformaremos as equações em funções e trabalharemos com otimização.

Antes de apresentar a formulação proposta é importante mencionar que aqui trabalhamos com imagens que sofreram um pré-processamento, foram submetidas a filtros detectores de borda, binarização e resgate de coordenadas dos pixels. Portanto uma imagem será aqui tratada como um conjunto de pontos do plano. O pré-processamento da imagem não será abordado nesse trabalho, pois usamos filtros prontos de *softwares* matemáticos, sem nenhuma contribuição especial. Aos interessados, sugerimos a leitura de [5] e [12].

Seja $P_1, P_2, \ldots, P_r$ um conjunto de pontos de uma imagem, assim temos para cada ponto uma equação $h_{P_i}(x) = y$ associada, de modo de que o ponto (x, y) no espaço transformado representa os parâmetros de uma curva passando sobre o ponto P_i no espaço imagem. Vamos agora definir para cada ponto P_i uma função $\varphi_{P_i}(x, y) = h_{P_i}(x) - y$. Veja que as raízes da função $\varphi_{P_i}(x, y)$ coincidem exatamente com as soluções da equação $h_{P_i}(x) = y$. Deste modo o conjunto de raízes de cada função representa todas as possíveis curvas do tipo procurado que passam pelo ponto associado. Temos a informação de todas as possíveis curvas do tipo procurado passando por cada ponto. Como estamos interessados em encontrar uma curva na imagem, buscamos o conjunto de pontos que descrevam tal curva, ou seja, buscamos um zero comum a todos os pontos descritos. Sem perda das propriedades características do problema podemos tomar para cada ponto uma nova função $F_i(x, y) = (\varphi_{P_i}(x, y))^2 = (h_{P_i}(x) - y)^2$, veja que o conjunto de raízes permanece inalterado. No entanto agora sabemos que os zeros de $F_i(x, y)$ são também minimizadores globais, além disso vale lembrar que buscamos reduzir ao máximo o valores de todos os pontos simultaneamente para encontrar pontos que sejam próximo de raízes comuns a todas as funções. Isso motiva o uso do problema LOVO, que descarta os pontos mais distantes da curva procurada.

3.2 Justificativa

Uma vez que fizemos a construção de S_p para cada curva que vamos identificar, vamos então justificar o uso da otimização.

Temos agora um conjunto de r funções onde cada $F_i(x, y)$ está associada a um ponto P_i no conjunto de pontos do espaço imagem. Além disso note que a os valores funcionais $F_i(x, y)$ estão diretamente relacionados com a distância entre a curva de parâmetros x, y e ponto P_i. Deste modo ao minimizar simultaneamente todas as funções encontramos a curva que melhor se ajusta ao conjunto de pontos, para isso vamos tirar proveito do fato de que cada função tem 0 como mínimo global. Logo minimizar simultaneamente todas as funções é equivalente a minimizar a soma, ou seja, a formulação do problema é dada por

$$\min S_r(x, y)$$

onde $S_r(x, y) = \sum_{i=1}^{r} F_i(x, y)$. Infelizmente na prática não é possível resolver tal problema com tamanha precisão por várias razões, como por exemplo erros numéricos, distorções da imagem, etc. Além disso, nem todos os pontos da imagem estão sobre a curva que buscamos, pois a imagem pode conter outros objetos além de ruídos, assim é necessário fornecer uma quantidade confiável de pontos que estão sobre a curva procurada, essa quantidade será o parâmetro p da formulação LOVO. Assim, dado um conjunto de r pontos extraídos de uma imagem, sabemos que p desses pontos estão sobre uma curva procurada, logo a solução de

$$\min S_p(x, y)$$

deverá fornecer o melhor ajuste descartando os $r - p$ pontos com pior ajuste.

Vamos agora apresentar o problema de encontrar retas, inicialmente tentamos usar a equação $ax + by = d$ para criar a função ajuste uma vez que tal equação é capaz de descrever qualquer reta no plano. Assim dado um conjuntos de pontos no espaço imagem $\{(x_i, y_i)\}_{i=1}^{r}$ o conjunto de funções associadas é dado por $F_i(a, b, d) = (ax_i + by_i - d)^2$. Deste modo se temos confiança de que p pontos estão sobre a reta, procurar o melhor ajuste de p pontos entre os r pontos disponíveis é equivalente a resolver

$$\min \sum_{j=1}^{p} ax_{i_j(a,b,d)} + by_{i_j(a,b,d)} - d)^2$$

onde, os índices $i_j(a,b,d)$ são definidos pela reordenação de cada $F(i)$. Veja que o vetor nulo, anula todas as funções $F_i(a,b,c)$ independente dos pontos usados na construção, deste modo o método sempre converge para $(0,0,0)$. Por este motivo concluímos que o melhor caminho é usar a equação 1.2 trabalhada no método de Hough, assim temos que o conjunto de funções associadas é dada por $F_i(\rho,\theta) = (x_i cos\theta + y_i sen\theta - \rho)^2$ e portanto encontrar a reta que melhor se ajusta a p pontos é equivalente a encontrar um mínimo global para o problema

$$\min \sum_{j=1}^{p} (x_{i_j(\rho,\theta)} cos\theta + y_{i_j(\rho,\theta)} sen\theta - \rho)^2.$$

De maneira análoga temos o problema de encontrar circunferências. Para este usamos a equação $(x-a)^2 + (y-b)^2 = r^2$. Veja que para descrever uma circunferência qualquer do plano precisamos de duas informações básicas, o centro da circunferência e seu raio. Com base nesta equação temos que (a,b) são as coordenadas do centro da circunferência e r o raio. Note que aqui assim como no problema de encontrar retas precisamos definir o parâmetro p que representa o número confiável de pontos que estão sobre a circunferência. Além disso, optamos por manter a formulação sugerida por [6] usando o raio como uma constante, assim o trabalho que apresentamos busca circunferências com raio determinado.

Para o caso de detectar uma circunferência de raio r vamos definir para cada ponto (x_i, y_i) uma função $F_i(a,b) = ((x_i - a)^2 + (y_i - b)^2 - r^2)^2$ assim o problema de detectar circunferências é equivalente a resolver o problema LOVO

$$\min \sum_{j=1}^{p} ((x_{i_j(a,b)} - a)^2 + (y_{i_j(a,b)} - b)^2 - r^2)^2.$$

Embora tenhamos mudado as equações das funções F_i em cada problema, as características teóricas permanecem as mesmas, isto é, se garantidamente um número p de pontos estão sobre a curva procurada, então as funções F_i associadas a tais p pontos terão o valor funcional zero em um ponto comum, cujas coordenadas são os parâmetros da curva que contém os p pontos, ou seja, os parâmetros procurados. Como foi dito anteriormente, na prática dificilmente encontraremos um ajuste perfeito, assim não teremos o zero como valor funcional. Contudo, veja que este valor funcional está relacionado com a distância entre o ponto associado a função e à curva descrita pelos parâmetros aplicados na função. Por exemplo, dado um ponto $P = (x_1, y_1)$ temos a ele associada a função $F_1(a,b) = ((x_1 -$

$a)^2 + (y_1 - b)^2 - r^2)^2$ cujo ponto (a, b) é o centro de uma circunferência c de raio r. Se P pertence a circunferência c, ou seja, a distância entre P e c dada por $d(P, c)$ é r teremos que $F_1(a, b) = 0$. Também, dados dois pontos (a_1, b_1) e (a_2, b_2), centro das circunferências c_1 e c_2 respectivamente, de modo que nenhuma contém P, sempre vale que se $d(P, c_1) > d(P, c_2)$ então $F_1(a_1, b_1) > F_1(a_2, b_2)$. De certa maneira tal relação vale para todos os ajustes, além disso $F_i(a, b) \geq 0$ para todas as funções i e para todo $(a, b) \in \mathbb{R}^2$. Por este motivo ao resolver o problema LOVO e determinar x^* minimizador, temos a garantia de que as coordenadas de x^* fornecem os parâmetros da curva com melhor ajuste a p pontos, no sentido de que a curva determinada tem a menor distância entre os p pontos usados no ajuste.

3.3 Testes

Nossos testes foram elaborados em 3 fases, primeiro geramos exemplos numéricos muito simples, com 20 pontos dos quais 15 estavam sobre a curva (reta ou circunferência) e 5 pontos aleatórios. O objetivo desse primeiro teste foi simplesmente avaliar o comportamento do algoritmo.

A fase seguinte foi realizada utilizando dois exemplos gerados numericamente, o primeiro contendo um total de 768 pontos dos quais eram: 61 pontos pertencente a uma circunferência de raio 2,7; 100 pontos pertencentes a uma circunferência de raio 4; 151 pontos pertencentes a uma circunferência não completa de raio 5; 67 pontos pertencente a uma reta horizontal, 37 pontos pertencentes a uma reta vertical, e outras três retas com inclinações variadas contendo respectivamente 39, 91 e 91 pontos cada. Os pontos restantes pertencem a uma elipse cuja finalidade será apresentada no capítulo seguintes, desta maneira não comentaremos aqui seus resultados.

O segundo exemplo numericamente gerado continha um total de 881 pontos dos quais 181 pertenciam a uma circunferência de raio 9, 100 pertenciam a uma circunferência de raio 5, 147 pertenciam a uma circunferência de raio 7.3, 31 pertenciam a uma circunferência de raio 1.5, 121 pertenciam a uma circunferência de raio 6 e 301 pertenciam a uma reta. Note que em ambos os exemplos não incluímos pontos aleatórios, no sentido de não pertencer a nenhuma curva, pois uma vez que um ponto não pertence a curva procurada, ele já faz esse

papel.

A terceira fase de testes foi realizada utilizando imagens reais. Nesta fase tomamos caminhos diferentes para a identificação de retas e de circunferências justificadas pelos resultados obtidos na segunda fase de testes. Essa escolha será esclarecida na seção posterior, onde serão comentados os resultados. Para as retas o teste efetivo foi realizada utilizando a imagem mostrada na figura 3.1, lembrando que trabalhamos com imagem depois de submetê-la a filtros de detecção detecção de borda e binarização, resgatando assim as coordenadas dos pixels que compõe seu contorno. Dessa imagem foram resgatados 1606 pontos.

Figura 3.1: Imagem usada para teste de identificação de retas em escala 1:1

Para o problema de detectar circunferências utilizamos a imagem mostrada na figura 3.2, criada em um editor de imagens, com raio determinado, nela são apresentadas um conjunto de 8 circunferências com raios de 10, 15, 20, 50, 54, 66, 100 e 150 pixels, também há uma elipse utilizada para um teste que será comentado no capítulo de conclusão. Nesta imagem foram resgatados 3531 pixels.

O algoritmo utilizado para realização dos testes foi programado em Julia[1], uma linguagem bastante nova no ramo de programação. Para execução dos testes, optamos por utilizar o JuliaBox[2], uma plataforma *on-line* para compilar códigos Julia, oferecendo assim uma maior imparcialidade para testes futuros.

O código dos algoritmos usados tanto na formulação LOVO, quando na SnL estão

[1]Mais informações disponíveis em http://julialang.org/

[2]https://www.juliabox.org/.

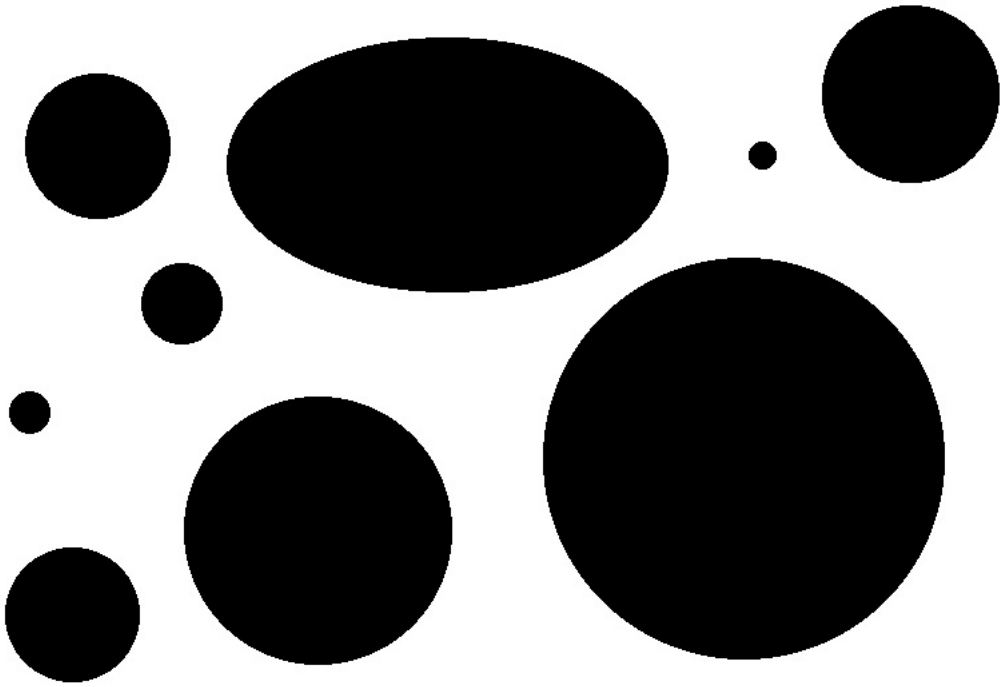


Figura 3.2: Imagem gerada para identificação de circunferências em escala 2:1

disponíveis na internet por meio da página *sites.google.com.br/emersonvitorcastelani/orientacoes*

3.4 Resultados

Apresentaremos nesta seção os resultados obtidos pelos testes da segunda e terceira fase. Vamos organizar os resultados em duas partes, uma para a identificação de retas e outra para identificação de circunferências

3.4.1 Retas

Os resultados obtidos a partir dos referidos testes estão dispostos na tabela 3.1. Para realização do teste usamos como ponto inicial o vetor $x_0 = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$ e como parâmetros do algoritmo os valores $\varepsilon = 1.0e-3$, $\alpha = 1.0e-5$, $\theta = 1.0e-7$ e $\beta = 0.25$.

Teste	p	Tempo (s)	Iterações	Valor de S_p	$\|\nabla S_p\|$
teste1	37	3.70087	19.0	0.0528863	0.000948258
teste2	39	6.54945	44.0	2.46851e-10	0.000581653
teste3	91	7.43751	17.0	1.29501e-10	0.000971384
teste4	300	64.31251496	31	1.2182052601673143e-12	0.000670981951974879
teste5	100	45.985863974	37	5.616828232866099	0.0005419004755245046
teste6	120	37.113710673	22	4.7902213040862955	0.0007342205397987788
teste7	160	53.999785067	22	52.43296267282826	0.0003853619298396294

Tabela 3.1: Tabela de resultados LOVO

A seguir apresentamos os resultados gráficos de cada teste nas figuras 3.3, 3.4 e 3.5.

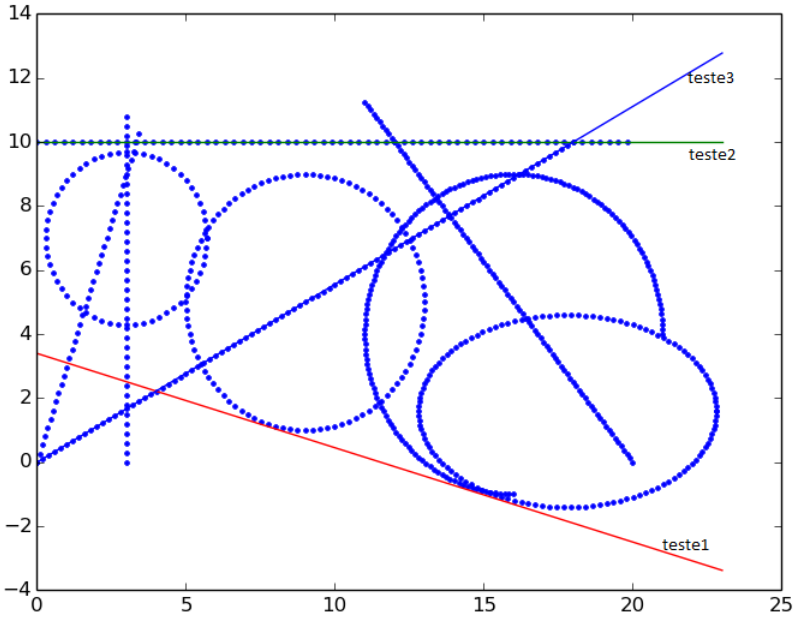


Figura 3.3: Exemplo artificial 1

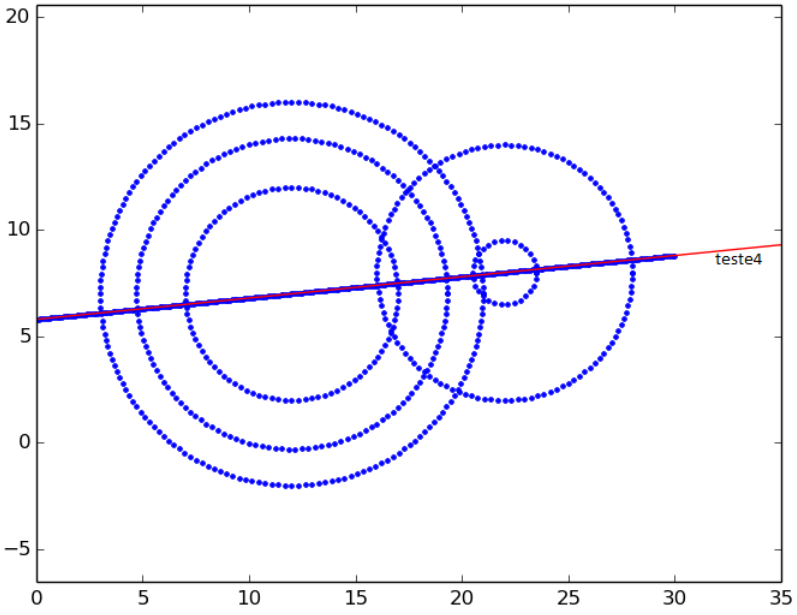


Figura 3.4: Exemplo artificial 2

3.4.2 Circunferências

Os resultados apresentados estão dispostos na tabela 3.2, onde utilizamos os seguintes valores de parâmetros para o algoritmo $x_0 = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$, $\varepsilon = 1.0e-3$, $\alpha = 1.0e-5$,

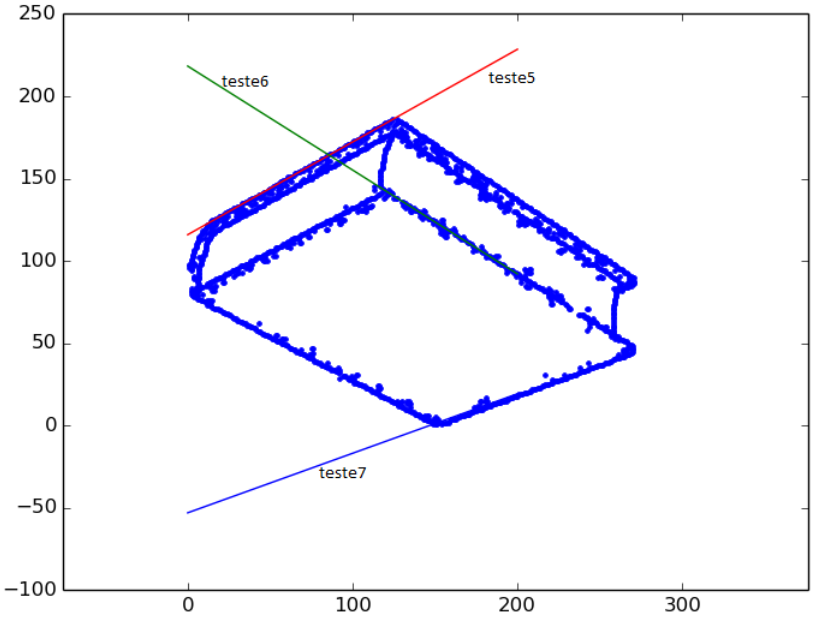


Figura 3.5: Teste com imagem real

$\theta = 1.0e - 7$ e $\beta = 0.25$. O algoritmo está limitado a 100 iterações.

As representações gráficas dos resultados são apresentadas em seguida pelas figuras 3.6, 3.7 e 3.8.

Teste	raio	p	Tempo (s)	Iterações	Valor de S_p	$\|\nabla S_p\|$
teste8	2.7	61	16.967391	59.0	39.072	0.000881
teste9	4.0	100	20.073376	35.0	166.837	0.000721
teste10	5.0	151	42.028513	48.0	1209.643	0.000806
teste11	9.0	180	98.447444	73	$6.692e^{-12}$	0.000881012
teste12	5.0	100	126.384504	100	5046.955	0.033721
teste13	7.3	140	70.306178	68	$2.031e^{-14}$	0.000842
teste14	1.5	30	3.168370	26	1.706	0.000812
teste15	6.0	120	17.306158	20	2007.534	0.000587
teste16	150	375	147.049353	21	0.0002	0.000576
teste17	100	250	38.281879	9	0.021	0.000261
teste18	66	165	136.608523	62	0.035	0.000978
teste19	54	135	13.019755	11	0.003	0.000444
teste20	20	30	0.224783	1	$8.079e^{-5}$	0.000440

Tabela 3.2: Tabela de resultados da identificação de circunferências com LOVO

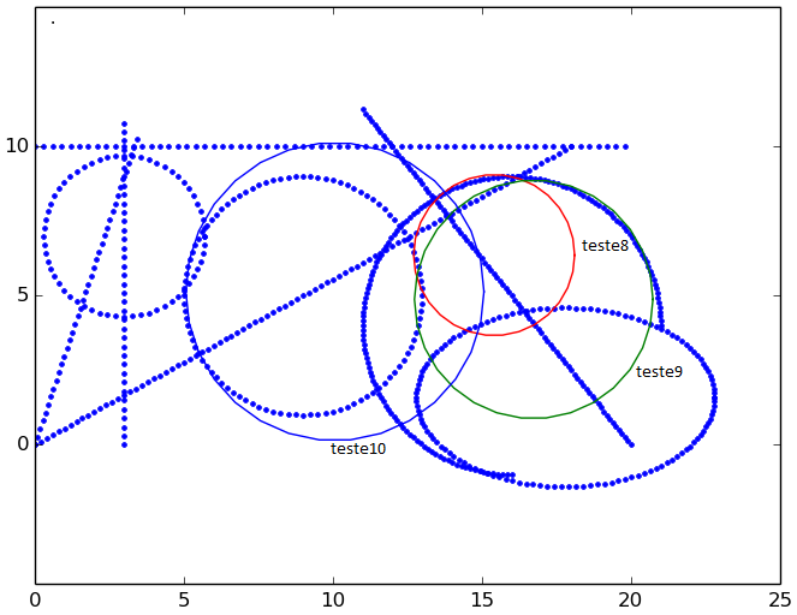


Figura 3.6: Detecção de circunferências no exemplo artificial 1

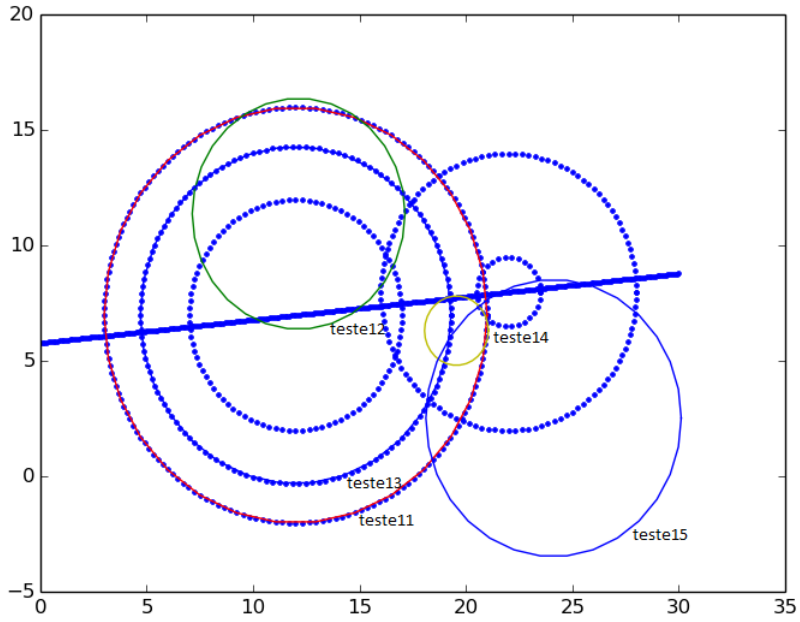


Figura 3.7: Detecção de circunferências no exemplo artificial 2

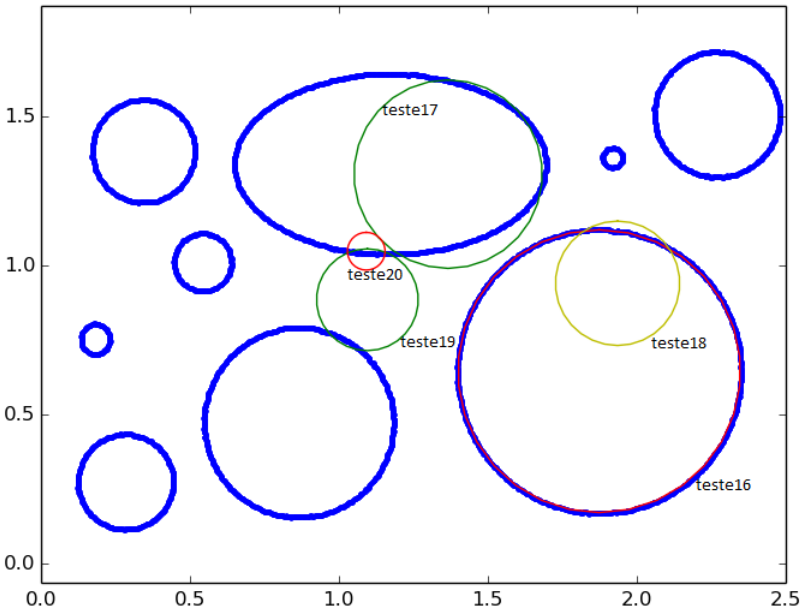


Figura 3.8: Detecção de circunferências em imagem real

Capítulo 4

Formulação SnL

Ao utilizar o método LOVO para detectar curvas em uma imagem identificamos alguns problemas recorrentes. Embora a lentidão no processamento não tenha sido a razão definitiva, também não foi satisfatório o resultado nesse sentido, mas a principal causa para buscar novos meios de resolução é a existência de uma expressiva quantidade de minimizadores locais. Todavia, percebemos que o modelo do problema tem muito potencial, uma vez encontrado um minimizador global, o algoritmo fornece ajustes muito satisfatórios. Desde modo remodelamos o problema para contornar tais dificuldades.

4.1 Justificativa

Veja que na elaboração do método LOVO para detecção de curvas de uma maneira geral, não aproveitamos uma propriedade particular deste problema. Como foi mencionado o desejável é localizar um minimizador global. Note que dada a formulação sabemos que o minimizador global tem valor funcional zero em um caso ideal. Além disso sabemos que o valor funcional de S_p está diretamente relacionado a distância entre os pontos considerados e a curva determinado pela solução do algoritmo. Deste modo, vamos apresentar uma definição e em seguida a nova formulação.

Definição 4.1. Seja $S_p = \sum_{j=1}^{p} F_{i_j(x)}(x)$ a função menor valor ordenado onde F_i são funções de ajuste para uma curva. Sem perda de generalidade, vamos assumir p par, uma vez que se p é ímpar basta excluir ou adicionar um ponto no ajuste conforme for o caso, assim definimos $S_p^i = \sum_{j=1}^{p-1} F_{k_{2j-1}(x)}(x)$ como a função menor valor ímpar-ordenado, que soma as

funções ordenadas com índice ímpar e menor que p, e a função $S_p^p = \sum_{j=1}^{p} F_{k_{2j}(x)}(x)$ a função menor valor par-ordenado, que soma as funções ordenadas de valor par menor que p.

Note que na elaboração do sistema foi necessário a divisão da equação $S_p(x) = 0$ em duas novas equações, compostas por somas parciais dos elementos que compõe S_p. Essa divisão a principio poderia ser qualquer, por exemplo, para a primeira equação somar as primeiras $\frac{p}{2}$ funções F_i e para a segunda somar as outras demais $\frac{p}{2}$ funções. Não obstante, o fato de tais funções terem sido ordenadas para a construção de S_p pode implicar em uma desuniformidade na convergência de cada equação. Uma vez que desejamos que a cada iteração ambas equações se aproximem cada vez mais de zero, a divisão sugerida acima, mostra-se arriscada pelo fato de que a soma das $\frac{p}{2}$ primeiras funções sempre será menor que a soma utilizada na segunda equação. Dito isso a construção por meio de índices pares e ímpares mostra-se bastante eficiente, uma vez que os valores das funções estão intercalados.

Dadas as funções conforme definição anterior, temos que se x_* é solução para o sistema

$$\begin{cases} S_p^i(x) = 0 \\ S_p^p(x) = 0 \end{cases}$$

então x_* fornece parâmetros para um curva perfeitamente ajustada na imagem.

4.2 Desenvolvimento

Para o desenvolvimento deste novo método escolhermos uma estratégia simples de Newton com aproximação da jacobiana para resolução de sistemas não lineares. Note que, como foi mencionado anteriormente, a função S_p não é suave. Assim para trabalhar com Newton aproveitamos a mesma estratégia usada na formulação LOVO, desse modo usamos as derivadas das funções ordenadas para caracterizar derivadas de S_p^p e S_p^i.

Este método foi apresentado sob o ponto de vista da otimização na seção 2.2 e sua formulação para solução de sistemas não lineares usa basicamente os mesmos princípios. Além disso o método de Newton é clássico na literatura, deste modo acreditamos não ser necessária uma apresentação mais formal sobre o assunto.

Como sabemos existe a possibilidade de a função S_p possuir muitos mínimos locais. Desse modo as jacobianas podem assumir valores muito pequenos e apresentar problemas de singularidade. A fim de contornar essa situação, embutimos no algoritmo um controle, de modo que, sempre a função se aproximar de um mínimo local será dado um passo grande na direção de Newton para a próxima iteração. Embora essa estratégia pareça arriscada, lembre-se que estamos aliando esse recurso com base nas funções que trabalhamos, e o mesmo pode não ter um funcionamento apropriado para outros problemas. É importante ressaltar também que essa estratégia contribui, mas não elimina um possível problema de singularidade. Por exemplo, não evita que linhas da jacobiana sejam múltiplas. Por essas razões não apresentamos provas de convergência. Contudo tais problemas não foram evidenciados na prática de forma que a estratégia funcionou bem.

4.3 Testes

Os testes realizados para essa versão com sistemas não lineares foram os mesmos realizados na versão LOVO incluindo os arquivos de dados, deste modo torna-se desnecessária uma nova apresentação dos mesmo.

4.4 Resultados

Os resultados dos testes seguirão a dinâmica apresentada no capítulo anterior.

4.4.1 Retas

Os resultados obtidos a partir dos referidos testes estão dispostos na tabela 4.1. Para realização do teste usamos como ponto inicial o vetor $x_0 = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$ e como parâmetros do algoritmo os valores $\varepsilon = 1.0e-3$.

Em seguida são apresentados os resultados gráficos nas figuras 4.1, 4.2 e 4.3.

Teste	p	Tempo (s)	Iterações	Valor de S_p	$\|\nabla S_p\|$
teste21	37	0.389853	41	0.000664	1.512421
teste22	67	1.180078	68	0.000865	2.723878
teste23	91	15.311152	640	0.000923	2.115991
teste24	300	3.615773	39	0.000828	0.720443
teste28	100	2.817537	44	0.001097	0.109172
teste29	120	27.297335	393	0.003330	0.715193
teste30	140	15.406302	215	0.006843	0.289092

Tabela 4.1: Tabela de resultados utilizando SnL pra identificação de retas

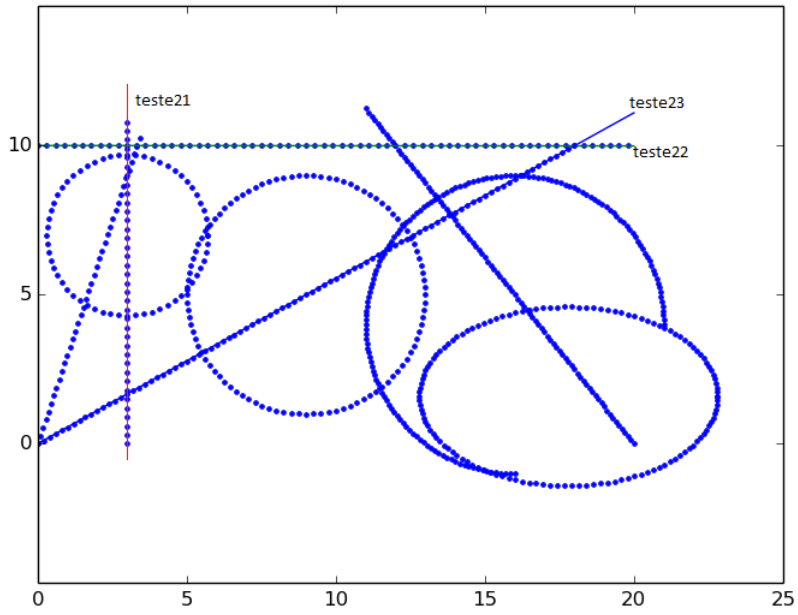


Figura 4.1: Resultado de SnL no exemplo artificial 1

4.4.2 Circunferências

Abaixo apresentamos na tabela 4.2 resultados obtidos pelo algoritmo SnL para identificação de circunferência. Os parâmetros utilizados foram os mesmo usados na identificação de retas. Os resultados gráficos são apresentados nas figuras 4.4, 4.5 e 4.6

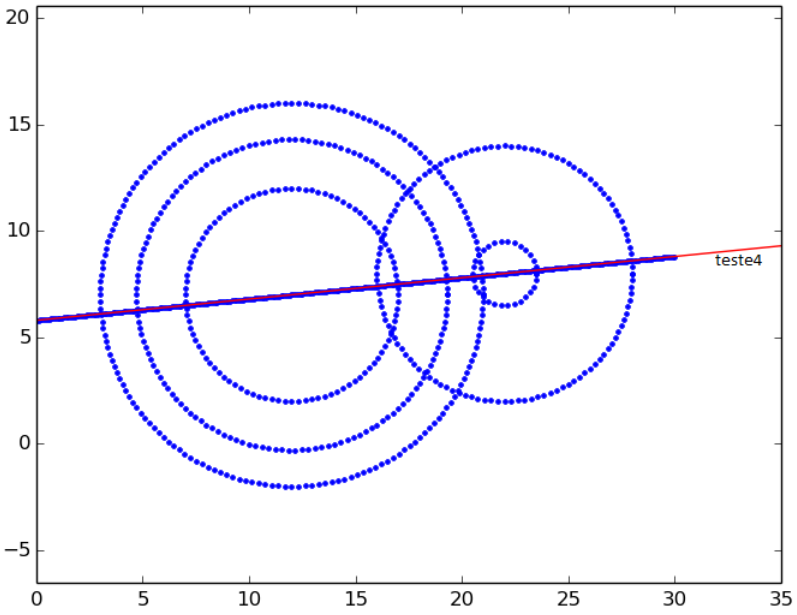


Figura 4.2: Resultado de SnL no exemplo artificial 2

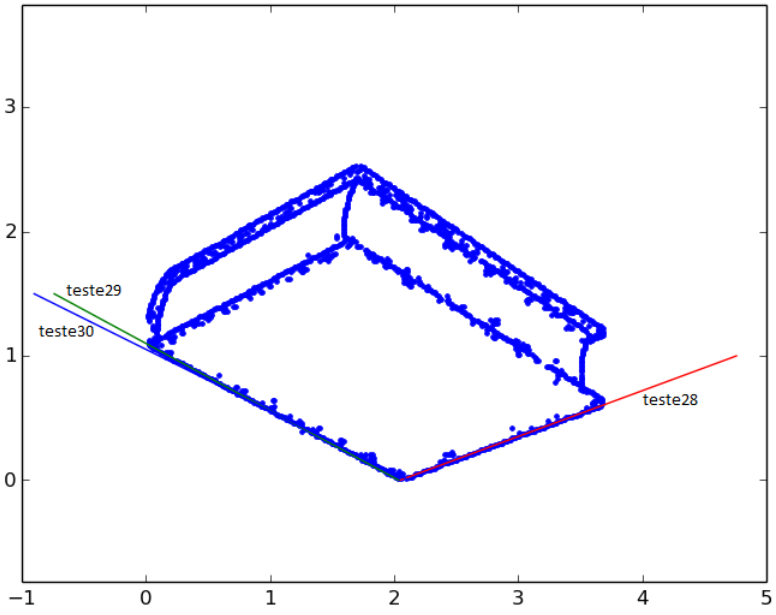


Figura 4.3: Resultado de SnL no exemplo real

Teste	Raio	p	Tempo (s)	Iterações	S_p	$\|S_p\|$
teste25	5	150	4.934564	129	0.001068	1.409059
teste26	2.7	60	2.458476	162	0.001072	0.507963
teste27	4	100	25.630755	1008	0.000968	0.890337
teste31	1.5	30	3.097754	315	0.016709	2.777113
teste32	9	180	97.470398	1879	0.0009636	10.596649
teste33	6	120	13.386819	393	0.000933	5.674524
teste34	7.3	140	69.139836	1756	0.000689	6.269649
teste35	5	100	76.224377	2695	0.000755	3.886099
teste36[1]	50	125	2.336976	16	$8.670463e^{-5}$	0.028359
teste37	100	250	145.867070	515	$9.574213e^{-5}$	0.038803
teste38	150	375	150.126476	323	0.000871	0.499632
teste39	15	37	0.559519	13	0.000145	0.007814
teste40	30	75	50.500607	555	$7.860890e^{-5}$	0.010835
teste41	10	25	0.402753	13	$8.315971e^{-5}$	0.007306
teste42	54	135	8.779627	61	0.000614	0.102243
teste43	66	165	48.597028	220	0.000102	0.043033

Tabela 4.2: Tabela de resultados de SnL na identificação de circunferêcias

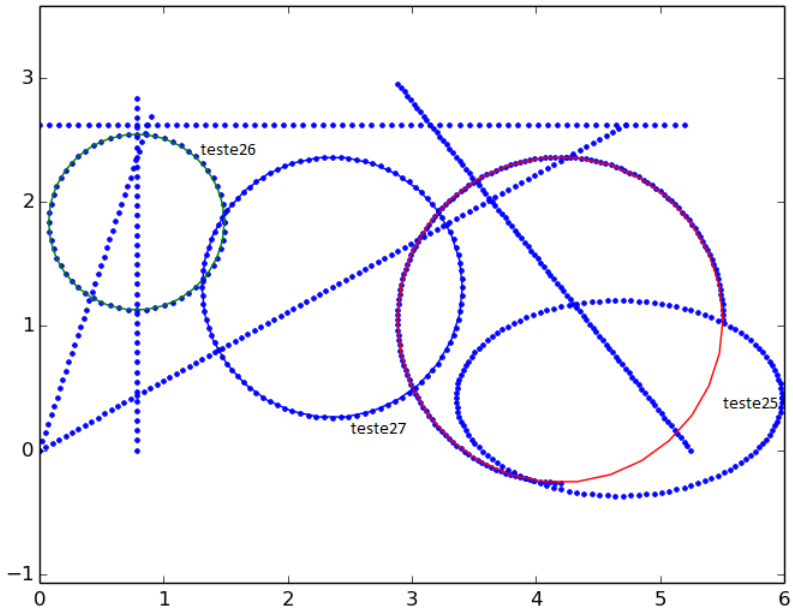


Figura 4.4: Resultado de SnL na identificação de circunferências no exemplo 1

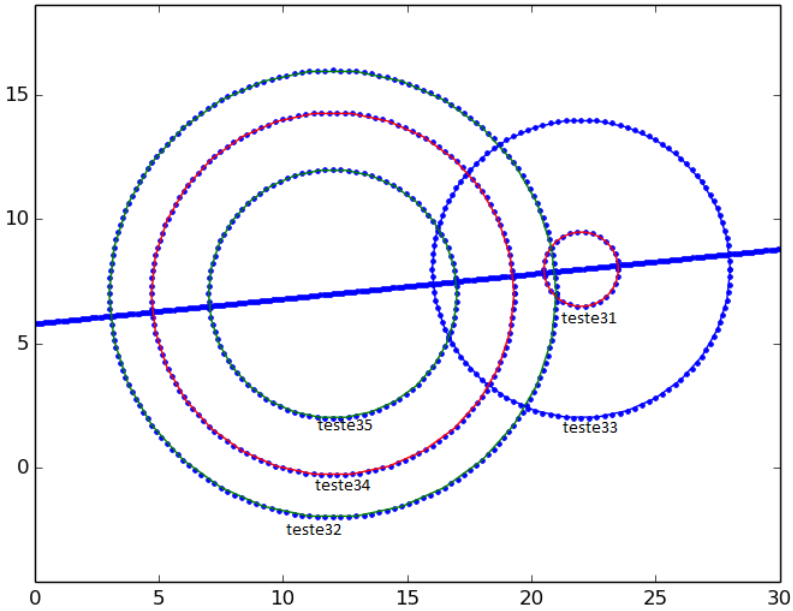


Figura 4.5: Resultado de SnL na identificação de circunferências no exemplo 2

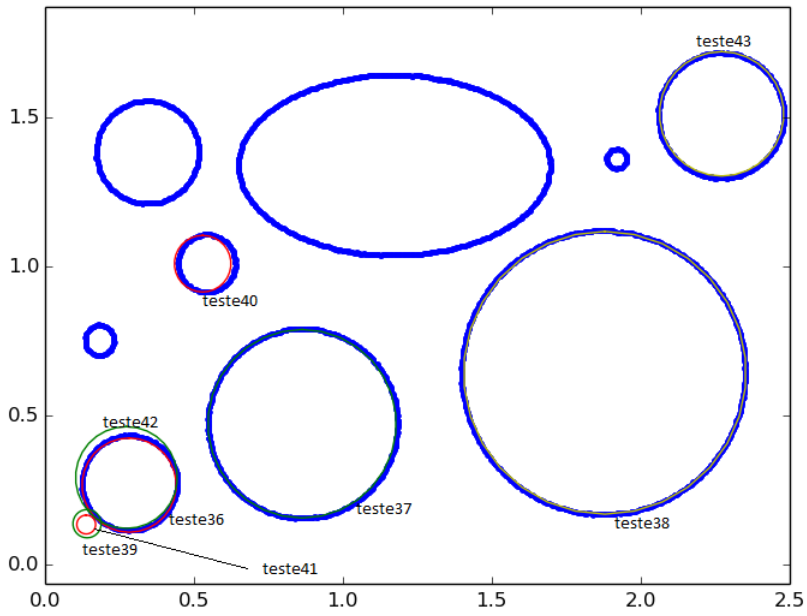


Figura 4.6: Resultado de SnL na identificação de circunferência no exemplo real

Capítulo 5

Considerações Finais

5.1 Comparações

Embora o desempenho do algoritmo de SnL mostrou-se muito superior em imagens geradas artificialmente, tanto numéricas quanto gráficas, ele apresenta algumas desvantagens (teóricas) em relação a formulação LOVO, uma vez que tal método não busca o melhor ajuste, mas sim uma vantajosa redução do valor funcional de S_p. Isso nos leva a acreditar que imagens com muito ruído e/ou baixa qualidade, podem fazer com o algoritmo não consiga resolver o sistema não linear, e sob tal ponto de vista, nos parece, o LOVO teria melhores condições de identificar uma curva. Conquanto como foi verificado nos testes realizado, o LOVO não é, como está, uma ferramenta confiável para realizar tais tarefas, especialmente curvas que gerem funções razoavelmente complexas, no sentido de que S_p tenha muitos mínimos locais, como é o caso da circunferência. Não obstante, um método de otimização que tenha certa predileção a encontrar minimizadores globais é capaz de contornar tal situação.

Note que no último teste usando imagens geradas em *software* de edição de imagens, o método SnL falhou na detecção das circunferências de raio menor, contudo, vale ressaltar que ao trabalhar com resgate de *pixels* obtemos coordenadas com valores altos, deste modo faz-se necessário reescalar o problema. Outros testes mostraram que ao reduzir o valor do parâmetro ε o algoritmo fornece ajustes melhores. Infelizmente isso afeta desempenho do método, tornando-o substancialmente mais lento. Tais testes não serão apresentados aqui, uma vez que a intenção é fazer comparativos entre ambos os métodos.

Sabemos que este trabalho é um ponto inicial de pesquisa e ainda há muito a ser estudado, aqui apenas apresentaremos algumas conclusões e comparativos a cerca do que

pudemos observar com nossos testes, a seguir uma tabela contento os resultados, tempo de processamento e qualidade de ajuste, lembrando que um valor funcional menor, implica em um ajuste melhor.

5.2 Automatização do parâmetro p

Uma das grandes dificuldades para se trabalhar com as funções menor valor ordenado, no nosso caso, foi uma escolha adequada para o parâmetro p, isso porque a escolha de um valor muito baixo dá a liberdade de um ajuste a um menor número de pontos, o que sob certas circunstâncias pode não gerar um resultado apropriado, por exemplo, ao invés de identificar uma circunferências o algoritmo elege p pontos aleatórios que fornecem um ajuste razoável, sem que tais pontos descrevam visualmente uma circunferência, por outro lado, como foi mencionado anteriormente eleger um valor alto para p pode ter complicações ainda mais graves uma vez que não tem existirá uma curva com essa quantidade de pontos.

Para fornecer uma escolha adequada, elaboramos alguns limitantes para p com base na geometria da imagem. Primeiramente é necessário mencionar que essa escolha que vamos sugerir é válida apenas para os casos mencionados, desse modo, para casos mais gerais ou específicos, a escolha deverá ser feita de maneira empírica, dito isso, vamos estabelecer uma relação entre as curvas trabalhadas, retas e circunferências, e o parâmetro p que deve ser adequado.

O caso de retas deve ser levado em consideração a proporção entre o comprimento do segmento a ser identificado e o tamanho da imagem aqui chamaremos ι, também devemos considerar a perda de informações devido a ruídos elegendo um percentual de confiabilidade dos pontos resgatados μ, tomado esses valores p é dado pela expressão $\lfloor \iota.\mu.\min\{h, p\} \rfloor$, onde $h \times p$ é a resolução da imagem e $\lfloor \cdot \rfloor$ é o operador menor valor inteiro, por exemplo, se buscamos um segmento e reta que percorre 50% de uma imagem com resolução 100×150 pixels com boa nitidez de modo que confiamos em 95% dos pontos extraídos, assim um escolha adequada de parâmetro é

$$p = \lfloor 0.5 * 0.95 * 100 \rfloor = \lfloor 47.5 \rfloor = 47$$

assim temos uma escolha adequada para identificar segmentos de reta que percorram até metade da image.

Para o caso de circunferências, essa automatização é indicada apenas para busca de circunferências completas, embora o método seja também capaz de identificar arcos, a quantidade de pontos será menor a medida que o arco seja menor, ou que a circunferência possua muitas falhas. depois de um estudo tomamos a seguinte formulação para a escolha de p, sendo r o raio da circunferência procurada, então $p = \lfloor 0.5 * 5r \rfloor$ onde 0.5 representa o ajuste numérico das coordenadas dos pixels usados, uma vez que as coordenadas são inteiras, percebemos que para um ajuste adequado devemos descartar metade dos pontos necessários para descrever tal curva que são aproximadamente $5r$.

5.3 Conclusão e trabalhos futuros

Dados os resultados dos testes, podemos concluir que ambos os métodos tem grande potencial para a função que foram desenvolvidos. O método LOVO embora tenha apresentado problemas na identificação das curvas, acreditamos que outras técnicas de minimização podem ser utilizadas de modo a aprimorá-lo. Por outro lado o método de sistemas não lineares ainda tem muito a ser explorado, quanto a suas características e propriedades teóricas. Um próximo passo natural é estudar, por exemplo, problemas relacionados a singularidade do sistema e como contorná-los.

Durante o processo de elaboração dos trabalhos que aqui constam foram realizados também alguns testes com elipses. Com intuito de avaliar a capacidade dos métodos ao lidar com outras curvas. Os resultados de tais testes não foram apresentados no texto, contudo mostrara-se muito promissores. Isso nos leva a acreditar que o trabalho não está restrito somente a detecção de retas e circunferências, mas ao longo do tempo, poderemos obter formas de detectar as mais diversas curvas.

Por fim, umas das questões mais importante para ser estudada futuramente, é o LOVO com restrições. Acreditamos que um método com restrições poderia resolver outras situações, como por exemplo, detectar circunferências com raio qualquer.

Bibliografia

[1] ALMEIDA, J. D. S. *Metodologia computacional para detecção automática de estrabismo em imagens digitais através do teste de Hirschberg.* Dissertação (Mestrado) - Curso de Pós-Graduação em Engenharia de Eletricidade, Universidade Federal do Maranhão, São Luís. 2010.

[2] ANDREANI, R., MARTÍNEZ, J. M., MARTÍNEZ, L., YANO, F. *Low order-valued optmization and applications.* Journal of Global Optimization, v.43, p.1-10, 2008.

[3] ANDREANI, R., MARTÍNEZ, J. M., MARTÍNEZ, L., YANO, F. *Low order-valued optmization and applications.* Technical report MCDO 051013, Department of Applied Mathematics, UNICAMP, Out, 2005

[4] BERTSEKAS, D. P. *Nonlinear programming* 2 ed.Massachusetts: Athena Scientific, 1999.

[5] CANNY, J. *A Computational Approach to Edge Detection* IEEE - Transactions on pattern analysis and machine inteligence, Vol. PAMI-8, Nº6, Nov, 1986

[6] DUDA, R., HART, P. *Use of the Hough transformation to detect lines and curves in the pictures.* Communication of the AMC, Vol. 15, Nº1, Jan, 1972

[7] HOUGH, P. V. C. *Method and means for recognizing complex patterns.* U.S Patent 3.069.654, Dez, 1962.

[8] MACEDO, M. M. G. *Uso da Transformada de Hough na Vetorização de Moldes e Outras Aplicações.* Dissertação (Mestrado em Ciência da Computação) - Universidade Federal Fluminense, Niterói. 2005.

[9] MARTÍNEZ, J. M. *Order-valued optmization and new applications.* Disponível em ¡http://ta.twi.tudelft.nl/wagm/users/rojas/COPT/Slides/talkdelft9.pdf¿ acesso Jan, 2015.

[10] MARTÍNEZ, J. M., SANTOS, S. A. *Métodos computacionais de otimização.* IMECC-UNICAMP, dez, 1998.

[11] NOCEDAL, J., WRIGHT, S. J. *Numerical optmization* 2 ed. New York: Springer Science+Business Media, 2000.

[12] WANGENHEIM, A., COMUNELLO, E., RICHA, R. *Encontrando a Linha Divisória: Detecção de Bordas* Seminário Introdução a visão computacional, PPGCC - INE - UFSC, disponível http://www.inf.ufsc.br/ visao/bordas.pdf acesso em: Dez, 2014.