ALGORITMOS DE PONTOS INTERIORES APLICADOS A FLUXO EM REDES [1]

Leonardo Nogueira Matos [2]

Departamento de Matemática Aplicada
IMECC - UNICAMP

Banca Examinadora :

Prof. Dr. Clovis Perin Filho [3]

Prof. Dr. Antonio Carlos Moretti [4]

Prof. Dr. Fernando Augusto Silva Marins [5]

Profa. Dra. Maria Aparecida Diniz Ehrhardt [6]

---

[1] Dissertação submetida ao Instituto de Matemática, Estatística e Ciência da Computação da Universidade Estadual de Campinas, para preenchimento dos pré-requisitos parciais para obtenção do Título de Mestre em Matemática Aplicada.

[2] O autor é Bacharel em Ciência da Computação pela Universidade Federal do Ceará.

[3] Professor do Departamento de Matemática Aplicada do IMECC - UNICAMP.

[4] Professor do Departamento de Matemática Aplicada do IMECC - UNICAMP.

[5] Professor do Departamento de Engenharia de Produção da Faculdade de Engenharia da UNESP - GUARATINGUETÁ.

[6] Professora do Departamento de Matemática Aplicada do IMECC - UNICAMP.

ALGORITMOS DE PONTOS INTERIORES APLICADOS A FLUXO EM REDES

Este exemplar corresponde à redação final da tese devidamente corrigida e defendida pelo Sr. Leonardo Nogueira Matos e aprovado pela Comissão Julgadora

Campinas, 1 de dezembro de 1993

Prof. Dr. Clovis Perin Filho

Perin Filho, Clovis

Dissertação submetida ao Instituto de Matemática, Estatística e Ciência da Computação da Universidade Estadual de Campinas, para preenchimento dos pré-requisitos parciais para obtenção do Título de Mestre em Matemática Aplicada.

Aos meus pais

e *in memoriam* de

Luis, Ticiano e Inah.

Agradecimentos

Durante a escrita desta tese, contei com o apoio de muitas pessoas. Gostaria de lembrar algumas delas.

Em primeiro lugar, devo meus agradecimentos ao prof. Clovis pela sua orientação e paciência.

Agradeço também à minha mãe, meu pai e irmãos; o apoio de minha família, mesmo estando distante, foi indispensável à conclusão deste trabalho. Neste sentido, lembro também os nomes dos tios Eduardo e Everardo e dos amigos Sidney e Francisco, pela acolhida em São Paulo e ajuda prestada durante todo este período.

Aos professores da área pelas sugestões e ajuda com o material bibliográfico.

Às pessoas com quem convivi nesta universidade, em especial a Patrícia e aos amigos da república, um grande abraço!

Ao Prof. Dr. Paulo França e ao funcionário Walcir, pela concessão de uma área na rede local de um dos laboratórios da Faculdade de Engenharia Elétrica.

Às instituições Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – CAPES e à fundação de Amparo à Pesquisa do Estado de São Paulo – FAPESP, pelo apoio financeiro.

Conteúdo

Lista de Figuras

Lista de Tabelas

Lista de Símbolos

x, y	Vetores em $\Re^n$ ou $\Re^m$
e	Vetor cujos elementos são todos iguais a 1
x_i^k	Supra-índices indicam o numero da iteração e subíndices a coordenada do vetor
x^{-1}	Vetor formado pelo inverso das componentes $x_1, \ldots, x_n$.
x^*	Solução ótima do problema primal.
X	Matriz diagonal cujos elementos são $x_1, \ldots, x_n$
A	Matriz
P_A	Projeção sobre $\mathcal{N}(A)$
$\|\cdot\|, \|\cdot\|_\infty, \|\cdot\|_1$	norma euclideana, norma-∞ e norma-1
ln, lg	Logaritmo neperiano e logaritmo na base 2
S, S^0	Região factível e seu interior
$\Re^n_+, \Re^n_{++}$	Vetores não negativos e positivos de $\Re^n$
$x \wedge y$	Vetor de coordenadas $(\min\{x_i, y_i\})_i$
$x \vee y$	Vetor de coordenadas $(\max\{x_i, y_i\})_i$

Capítulo 1

Introdução

1.1 Apresentação do Problema

Nos anos subseqüentes à Segunda Guerra Mundial ocorreu um grande avanço no campo científico com o desenvolvimento de diversos trabalhos iniciados durante o período de guerra. Estes trabalhos diziam respeito a um problema muito comum em nossas vidas: como fazer um melhor aproveitamento de recursos que são escassos.

Este pode ser considerado o marco do surgimento de uma importante ciência dos dias atuais: a Pesquisa Operacional. A Pesquisa Operacional (PO) compõe um conjunto de ferramentas científicas que auxiliam em problemas de tomada de decisão. Entre estas ferramentas se encontram a Programação Linear (PL) e a Programação de Fluxo em Redes, que como veremos adiante, trata-se de um caso particular da PL.

A palavra rede a que nos referimos diz respeito a uma representação de qualquer entidade caracterizada por possuir pontos de oferta e demanda de produtos e meios de transmissão destes produtos. Esta representação pode ser tão abrangente quanto se queira pois pode representar desde uma rede de telecomunicações até um chão de fábrica. Quanto à palavra fluxo, obviamente, ela diz respeito à circulação do produto na rede.

Diversos problemas da vida prática podem ser enquadrados como problemas de fluxo em redes, tais como: escalonamento de tarefas em máquinas, planejamento de circuitos VLSI, roteamento de veículos no transporte de cargas, planejamento e gerenciamento de redes elétricas, hidráulicas, gasodutos

e redes de telecomunicações e muitos outros.

Sob o ponto-de-vista acadêmico estes problemas recebem nomes clássicos na literatura, dentre os quais podemos destacar

- Problema de Fluxo Máximo - quando se deseja saber o maior fluxo possível que pode ser enviado de um nó a outro da rede.
- Problema do Caminho Mínimo - quando se deseja saber o menor caminho entre dois nós distintos da rede.
- Problema de Fluxo de Custo Mínimo - quando se deseja descobrir o fluxo de menor custo na rede, com este fluxo atendendo às restrições de oferta e demanda nos nós. No caso particular em que a rede é um grafo bipartido em conjuntos de nós de demanda e nós de oferta, o problema de fluxo de custo mínimo recebe a denominação especial de problema de transporte.
- Problema de Designação - Trata-se de um caso particular do problema de transporte. A formulação do problema de designação é semelhante à do problema de transporte, diferindo apenas no fato de que a demanda e oferta em cada nó é sempre igual a um.
- Problema de Circulação - É um caso particular do problema de fluxo de custo mínimo. Ocorre quando a demanda e oferta em cada nó é sempre igual a zero.

Existem na literatura diversos algoritmos específicos para resolver cada um dos problemas acima. Neste trabalho nos propomos a estudar o problema de fluxo de custo mínimo usando algoritmos não tradicionais. A abordagem que adotamos para resolver este problema é através de algoritmos de pontos interiores.

1.2 Histórico

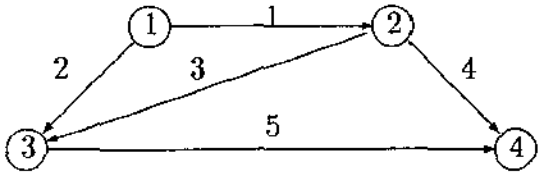
O problema de fluxo de custo mínimo, consiste em determinar um vetor $x^* \in \Re^n$ solução ótima de:

$$\begin{array}{lll} \min & c^t x & \\ s.a. & Ax = b & (P) \\ & 0 \leq x \leq u, & \end{array}$$

onde $A \in \Re^{m \times n}$ é a matriz de incidência da rede, c, u e x são vetores em $\Re^n$ que representam o custo, capacidade e intensidade do fluxo nos arcos, respectivamente, e $b \in \Re^m$ é o vetor de nós. A matriz de incidência obedece a seguinte regra de formação

$$a_{ij} = \begin{cases} 1 & \text{se o arco } j \text{ está saindo do nó } i, \\ -1 & \text{se o arco } j \text{ está chegando ao nó } i, \\ 0 & \text{se o arco } j \text{ não incide sobre o nó } i. \end{cases}$$

Na Figura 1.1 ilustramos uma rede com 4 nós e 5 arcos e uma matriz de incidência correspondente ao mesmo.



$$A = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 \\ -1 & 0 & 1 & 1 & 0 \\ 0 & -1 & -1 & 0 & 1 \\ 0 & 0 & 0 & -1 & -1 \end{bmatrix}$$

Figura 1.1: Exemplo de um grafo e sua matriz de incidência.

Por tratar-se de um problema de programação linear com uma matriz de restrições com uma estrutura especial, este problema é resolvido pelo método simplex especializado para redes [22]; extremamente rápido, eficiente e preciso pois executa apenas operações de soma, subtração e comparação.

Analogamente, existe uma versão do simplex dual [2] para redes, também construído de modo a aproveitar as características da matriz de incidência. Ambos os métodos possuem complexidade exponencial, o que é considerado não eficiente do ponto de vista teórico.

Em 1961 Fulkerson desenvolveu um método primal dual denominado o método out-of-kilter [22]. O método out-of-kilter tornou-se conhecido por ser específico para o problema de fluxo de custo mínimo e por lograr bons resultados práticos. Diversos trabalhos foram feitos comparando o método simplex e out-of-kilter sem que ainda se tenha uma definição de qual dos dois é o mais eficiente.O método out-of-kilter, assim como o simplex, também possui complexidade exponencial.

Os primeiros algoritmos com complexidade polinomial surgiram nos anos 70 com a idéia de aproximações sucessivas ou "scaling", proposta por Edmonds e Karp [9]. Métodos de aproximações sucessivas são métodos em que a cada iteração é resolvido um problema com um critério de otimalidade relaxado o qual se torna mais rígido à medida que as iterações se sucedem, até que na última iteração obtém-se uma solução ótima para o problema original. A abordagem utilizada para resolver o problema dentro de cada iteração classifica os diversos métodos de aproximações sucessivas; em todos estes métodos, cada etapa é resolvida em tempo polinomial. Diversos algoritmos com aproximações sucessivas foram desenvolvidos nestes últimos vinte anos; atualmente os trabalhos mais evoluídos devem-se a Goldberg e Tarjan [15] e [16] e Orlin [31], entre outros, que propuseram na segunda metade da década passada algoritmos fortemente polinomiais.

Paralelamente, nos últimos anos, o estudo de pontos interiores para a programação linear tomou um grande impulso. Algoritmos de pontos interiores para a programação linear surgiram com o esforço científico de desenvolver algoritmos polinomiais para a Programação Linear (PL). O primeiro algoritmo polinomial deve-se a Khachiyan [23], 1978. Ele provou que a complexidade do método dos elipsóides de Shor [34] para a PL é de $O(n^4L)$, onde L é o número de bits usados para armazenar os dados do problema. Apesar de ter boas propriedades teóricas, este método na prática é irremediavelmente lento e, portanto, não se mostra competitivo frente ao simplex. Em 1984, Karmarkar publicou seu método projetivo [21] com complexidade $O(n^{3.5}L)$ para o número de operações e $O(nL)$ para o número de iterações. Ao mesmo tempo, anunciou resultados superiores aos do método simplex para problemas com alguns milhares de variáveis. Este fato motivou muitos pesquisadores

ao estudo de pontos interiores. Em 1985 foram apresentados melhoramentos ao algoritmo de Karmarkar que originalmente utilizava um formato muito complicado e difícil. Uma simplificação radical deste algoritmo resultou no método primal afim, um método proposto em 1967 por Dikin [8] e que até aquele momento permanecera desconhecido, sendo redescoberto independentemente por Barnes [3] e Vanderbei, Meketon e Freedman [38]. Um estudo da complexidade deste método foi feito por Megiddo e Shub em [29] que concluiram que o método primal afim muito provavelmente não é polinomial.

Novos resultados foram obtidos a partir do estudo de trajetória central. A trajetória central foi inicialmente estudada por Bayer e Lagarias [4] e por Megiddo [28]. Renegar [32] publicou um algoritmo de trajetória central com complexidade de $O(n^{3.5}L)$ para o número de operações e $O(\sqrt{n}\ L)$ para o número de iterações, reduzindo assim a complexidade em relação ao algoritmo de Karmarkar. Vaidya [36], seguindo a metodologia de Renegar, reduziu a complexidade do número de operações para $O(n^3L)$. Gonzaga [18], sem utilizar a metodologia de Renegar, publicou um algoritmo de trajetória central com mesma complexidade que o de Vaidya. Em 1988 Gonzaga [18] [20] publicou um segundo algoritmo de trajetória central: algoritmo de barreira de passos longos, cuja complexidade é a mesma que a do anterior mas que permite que o tamanho dos passos entre iterações consecutivas seja maior.

Em 1989 foram publicados os primeiros algoritmos primais duais de pontos interiores. Inicialmente Megiddo em [28] fez um estudo relacionando condições de folgas complementares com a resolução simultânea do problema primal e dual através do uso de uma função barreira logaritmica. Baseado no trabalho de Megiddo, Kojima et al. [24] publicaram um primeiro algoritmo primal dual com complexidade da ordem $O(n^4L)$ operações aritméticas e $O(nL)$ iterações. Um melhoramento a este algoritmo foi proposto por Monteiro e Adler [30] que conseguiram reduzir a complexidade para $O(n^3L)$ operações e $O(\sqrt{n}\ L)$ iterações. O algoritmo de Monteiro e Adler valia-se da proximidade à trajetória central para assegurar a convergência em tempo polinomial; para isto, o tamanho do passo a cada iteração era mantido sempre dentro de um intervalo muito pequeno, o que resultava num baixo desempenho computacional. Resultados mais expressivos foram conseguidos por Lustig, Marsten e Shanno [27] com um algoritmo primal dual de passos longos.

1.3 Objetivos do Trabalho

Neste trabalho, estudaremos em detalhes os algoritmos de passos curtos e de passos longos de Gonzaga [20], [18]. Estudaremos com menor rigor o algoritmo primal afim [3] e [38] e primal dual de Lustig, Marsten e Shanno [27]. No capítulo 3 veremos as técnicas de resolução de sistemas lineares usadas e a adaptação que fizemos do problema de fluxo em redes à teoria desenvolvida por Gonzaga. O capítulo 4 consiste na apresentação dos resultados computacionais resultantes da implementação dos algoritmos estudados, e da apresentação de um algoritmo de pontos interiores baseado em modificações que fizemos ao algoritmo de passos longos de Gonzaga. Por fim, no último capítulo serão feitas conclusões gerais sobre o assunto tratado, citando linhas de pesquisa, contribuições e os aspectos bons e ruins da abordagem de pontos interiores ao problema de fluxo em redes.

1.4 Glossário de Notações

Ao longo deste trabalho nós utilizamos uma relação de notações semelhante a usada em [20]. As letras minúsculas em itálico representam vetores em $\Re^m$ ou $\Re^n$, com dimensão dada conforme o contexto. Particularmente, o vetor e corresponde ao vetor com todos os elementos iguais a um. O supra-índice nos vetores representam o número das iterações, o passo que os subíndices ao número da coordenada. Os vetores com expoente negativo, por exemplo, x^{-1}, representam vetores de coordenadas $(x_1^{-1}, \ldots, x_n^{-1})$. O vetor x^* representa a solução ótima do problema primal (P). As letras maiúsculas em itálico indicam matrizes, em particular a matriz X representa a matriz diagonal cujos elementos são $x_1, \ldots, x_n$. Usaremos o símbolo *diag* para denotar os elementos de uma matriz diagonal. A matriz P_A representa a matriz de projeção sobre o núcleo de A, isto é, $P_A = I - A^t(AA^t)^{-1}A$. O símbolo $\|\cdot\|$ indica a norma euclideana ou norma-2 de um vetor ou matriz. Usamos além da norma-2 a norma-∞, $\|\cdot\|_\infty$, e a norma-1, $\|\cdot\|_1$, onde são válidas as seguintes relações

$$\begin{array}{rcccl} \|\cdot\|_\infty & \leq & \|\cdot\| & \leq & \|\cdot\|_1, \\ \|x\| - \|y\| & \leq & \|x - y\| & \leq & \|x\| + \|y\|. \end{array}$$

Na definição da função barreira fazemos uso de uma função logarítmica.

Neste trabalho usamos o logaritmo neperiano que denotaremos por ln e o logaritmo na base 2 que denotaremos por lg.

O conjunto de pontos factíveis será representado por S, isto é,

$$S = \{x | Ax = b, 0 \leq x \leq u\},$$

e seu interior por S^0. Supomos que $S^0 \neq \emptyset$ ao longo de todo o trabalho, bem como, consideraremos também que o programa linear (P) é factível, limitado e possui apenas uma única solução ótima.

Os conjuntos $\Re^n_+$ e $\Re^n_{++}$ são os subconjuntos de $\Re^n$ dos vetores não negativos e positivos, respectivamente.

Nas manipulações algébricas com vetores consideramos que $x \geq y$ significa que todos os elementos de x são maiores ou iguais aos de y, coordenada a coordenada. O sentido de $x \leq y$, $x > y$ e $x < y$ é óbvio e deriva da relação $x \geq y$. A operação $x \wedge y$ corresponde a um vetor cujas componentes são $(x \wedge y)_i = \min\{x_i, y_i\}$. Analogamente, temos que $x \vee y$ corresponde ao vetor $(x \vee y)_i = \max\{x_i, y_i\}$.

Capítulo 2

Algoritmos de Pontos Interiores

2.1 Introdução

Neste capítulo faremos a apresentação dos algoritmos de pontos interiores. O foco principal de nossa atenção está voltado para os algoritmos de barreira de passos longos e algoritmo de barreira de passos curtos de Gonzaga. Para estes algoritmos faremos além da apresentação formal, uma análise completa da convergência e da complexidade de cada um.

Na seção 2.2 veremos o algoritmo primal afim segundo [20], [3] e [38]. Na seção 2.3 introduziremos o conceito de trajetória central e os fundamentos matemáticos básicos utilizados nas provas de convergência e complexidade dos algoritmos de Gonzaga. A seções 2.4 e 2.5 compõem, respectivamente, a apresentação do algoritmo de passos curtos e de passos longos de Gonzaga. Por fim, na seção 2.6 veremos um algoritmo primal dual de pontos interiores implementado em [27].

2.2 Algoritmo Primal Afim

Segundo Barnes [3] e Gonzaga [20], o algoritmo primal afim para a programação linear

$$\begin{array}{ll} \min & c^t x \\ s.a. & Ax = b \qquad (P) \\ & 0 \leq x \leq u, \end{array}$$

consiste em minimizar a função $c^t x$ em um elipsóide em S^0, isto é, procura-se minimizar $f(x) = c^t x$ em uma região menor, no caso um elipsóide contido no hiperplano $Ax = b$. Existem duas grandes vantagens em se trabalhar com tais regiões de confiança. Primeiro, a direção de descida aponta para "próximo" do ótimo, uma vez que o elipsóide é construído de modo que tende a assumir a forma da região factível. Segundo, os cálculos empregados na minimização nesta região de confiança são relativamente simples e rápidos. O problema, então, a cada iteração, k, torna-se

$$\begin{array}{ll} \min & c^t x \\ s.a. & Ax = b \qquad (PA) \\ & \|D^{-1}(x^k - x)\|^2 \leq R^2, \end{array}$$

com $R < 1$ e $D = diag(d_i)$ com $d_i > 0$, para que o elipsóide esteja contido em S^0. Em [38] Vanderbei et al. definem a matriz D como

$$D = diag\left(\frac{x_i^k(u_i - x_i^k)}{\sqrt{((x_i^k)^2 + (u_i - x_i^k)^2}}\right).$$

Para resolver (PA), fazemos uma mudança de escala que leva x a $y = D^{-1}x$ e transforma o elipsóide numa bola, o que equivale ao problema modificado

$$\begin{array}{ll} \min & (D^t c)^t y \\ s.a. & ADy = b \qquad (PA') \\ & \|y - D^{-1}x^k\|^2 \leq R^2. \end{array}$$

A direção que minimiza $c^t x$ em uma bola é a própria direção de máximo declive, $-c$. No problema modificado, a direção que minimiza (PA') é a direção de máximo declive feita uma mudança de escala, isto é, $-Dc$. Como esta direção deve pertencer ao núcleo de A, para que o próximo ponto permaneça factível, a direção de descida, h, é então calculada como $h = -DP_{AD}Dc$, com P_{AD} a matriz de projeção sobre o núcleo de AD.

A Figura 2.1 ilustra a razão de trabalharmos com este elipsóide. Abaixo descrevemos o programa linear associado a esta figura.

$$\begin{aligned} \min \quad & 2x_1 + 3x_2 \\ s.a. \quad & 0 \leq x \leq (8,4)^t, \end{aligned}$$

a aproximação inicial é o ponto $x^0 = (6,1)^t$. Observe que o elipsóide tende a assumir a forma da região factível na vizinhança de x^0, o que favorece a geração de direções que apontam para "próximo do ótimo".

Uma vez obtida a direção de descida, h, define-se o ponto seguinte como

$$x^{k+1} = x^k + \bar{\lambda}h, \qquad \bar{\lambda} \in \Re^*.$$

Para assegurar que $x^{k+1} \in S^0$ temos que definir $\bar{\lambda}$ de modo que

$$0 < (x + \bar{\lambda}h) < u,$$

o que equivale ao teste da razão

$$\lambda = \min\{\min\{x_i/h_i : h_i < 0\}, \min\{(u_i - x_i)/h_i : h_i > 0\}\}.$$

O valor de λ assim definido exprime o quanto podemos nos "afastar" de x^k na direção h até que encontremos a primeira fronteira. Como x^{k+1} deve pertencer ao interior de S, fazemos $\bar{\lambda} = \delta\lambda$, $\delta \in (0,1)$. Neste trabalho o valor de δ foi fixado empiricamente como 0.995.

Algoritmo 2.1 Algoritmo Primal Afim
Dados $x \in S^0$, $\delta \in (0,1)$

repita
$D = \text{diag}(x_i(u_i - x_i)/\sqrt{x_i^2 + (u_i - x_i)^2})$
Resolver $(AD^2A^t)y = AD^2c$
$h = -D(c - A^ty)$
$\lambda_1 = \min\{x_i/h_i : h_i < 0\}$
$\lambda_2 = \min\{(u_i - x_i)/h_i : h_i > 0\}$
$\bar{\lambda} = \delta \min\{\lambda_1, \lambda_2\}$
$x = x + \bar{\lambda}h$
até convergir

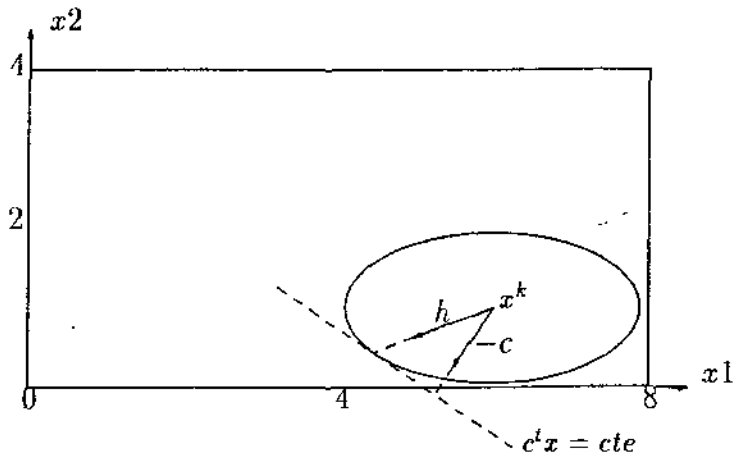


Figura 2.1: Iteração do Método Afim

2.2.1 Inicialização

O ponto inicial é encontrado segundo a metodologia de Vanderbei[37]. Consideramos o problema

$$\begin{aligned} \min \quad & \zeta \\ s.a. \quad & [A\ \rho] \begin{bmatrix} x \\ \zeta \end{bmatrix} = b \qquad (PI) \\ & 0 \leq x \leq u, \end{aligned}$$

onde

$$\rho = b - A\xi,$$

e ξ é um vetor que satisfaz $0 < \xi < u$. Vanderbei fez as seguintes observações:

(a) ζ é uma variável irrestrita.
(b) O ponto $x = \xi$, $\zeta = 1$ é factível para (PI).
(c) Qualquer ponto factível para (PI) com $\zeta = 0$ também satisfaz $Ax = b$.
(d) Se o mínimo de ζ é positivo, então (P) é infactível.

Ele utilizou (b) e (c) para justificar o emprego deste método. O algoritmo proposto por Vanderbei é:

Algoritmo 2.2 Algoritmo Fase 1
Dado x que satisfaz $0 < x < u$

repita
$y = u - x$
$D_x = diag(x \wedge y)$
$\rho = b - Ax$
$z = -D_x^2 A^t B\rho$ com $B = (AD_x^2 A^t)^{-1}$
$\gamma = \max\left(\frac{z}{x} \vee -\frac{z}{y}\right)$
$\delta = -\min\left(\frac{z}{x^2} \wedge -\frac{z}{y^2}\right)$
$M = \max(x \wedge y)$
se $\gamma + M\delta < \beta\varepsilon/n$ onde $\beta = \rho^t B\rho$
Pare! Problema Infactível
se $\gamma \geq \alpha$
$x = x - \frac{\alpha}{\gamma} z$
fim_processo = Falso
senão
$x = x - z$
fim_processo = Verdade
até fim_processo

2.2.2 Critério de Parada

Em [38] Vanderbei et al. utilizaram o seguinte critério de parada para provar a convergência do algoritmo.

Dado uma tolerância $\epsilon > 0$, defina

$$\begin{aligned} r_i(x) &= c - DA^t(AD^2A^t)^{-1}AD^2c, \\ \gamma(x) &= \max_i x_i r_i(x), \\ \delta(x) &= -\min_i r_i(x), \\ M(x) &= (1/n)e^t x, \end{aligned}$$

então o algoritmo pára na primeira iteração em que

$$\gamma(x^k) + \delta(x^h)M(x) \leq \epsilon/n.$$

2.3 A Trajetória Central

Na seção sobre o Método Afim trabalhamos com o problema na forma padrão canalizada (P). Nesta seção e nas seções 2.4 e 2.5, para melhor explicar o funcionamento dos algoritmos de passos longos e de passos curtos, trabalharemos com o problema considerando a canalização como parte integrante das restrições. O problema, portanto, torna-se

$$\begin{array}{ll} \min & c^t x \\ s.a. & Ax = b \\ & x + s = u \\ & x, s \geq 0, \end{array}$$

o que simplificaremos considerando o problema abaixo

$$\begin{array}{ll} \min & \hat{c}^t \hat{x} \\ s.a. & \hat{A}\hat{x} = \hat{b} \quad (P') \\ & \hat{x} \geq 0, \end{array}$$

onde

$$\hat{A} = \begin{bmatrix} A & 0 \\ I & I \end{bmatrix}, \hat{b} = \begin{bmatrix} b \\ u \end{bmatrix}, \hat{c} = \begin{bmatrix} c \\ 0 \end{bmatrix}, \hat{x} = \begin{bmatrix} x \\ s \end{bmatrix}.$$

Assim, definimos o conjunto das soluções associadas à restrição $\hat{A}\hat{x} = \hat{b}$ como $\hat{S}$ e seu interior como $\hat{S}^0$.

2.3.1 Definições

Alguns conceitos importantes devem ser introduzidos antes de apresentar a definição de trajetória central. Um dos mais relevantes é o da função barreira, a qual faremos referência durante todo o resto deste trabalho. Vimos anteriormente que o método afim gera pontos interiores por meio de um controle no tamanho do passo, isto é, fixando um parâmetro δ que indica uma

percentagem do deslocamento máximo permitido. Esta é uma técnica bastante rudimentar de se evitar a fronteira, pois o valor de δ é fixado de modo heurístico, sem garantia de que esta seja a melhor técnica a ser utilizada. Além disto, há fortes indícios de que o método afim não seja polinomial. O uso da função barreira surge, pois, como uma alternativa que garante a geração de pontos interiores de modo mais eficiente, assegurando a complexidade em tempo polinomial do método. Esta função é definida como:

$$p(\hat{x}) \;=\; -\sum_{j=1}^{2n} \ln \hat{x}_j.$$

A idéia dos métodos de barreira é trabalhar minimizando conjuntamente a função objetivo e a função barreira. Observe que para pontos próximos da fronteira o valor de $p(\hat{x})$ cresce indefinidamente. O algoritmo então é forçado a evitá-la, gerando uma sequência de pontos interiores.

O uso de métodos de barreira foi inicialmente proposto em 1955 por Frisch [14] e estudado posteriormente em [12] em 1967 para problemas genéricos de otimização. O emprego do mesmo para a PL deu-se somente em 1984 quando Karmarkar publicou seu algoritmo. Este algoritmo, como sabemos, logrou não só boas propriedades teóricas, como também, bons resultados práticos. A rigor, acredita-se que o algoritmo de Karmarkar opera melhor por que gera pontos mais distantes da fronteira, ao contrário do método afim, cujas iterações estão bem próximas dos limites da região factível. Atualmente o conceito de uma "boa região" onde processar-se as iterações tornou-se mais preciso com a definição de trajetória central. Resultados superiores aos apresentados por Karmarkar foram obtidos usando algoritmos de trajetória central.

A trajetória central é uma curva suave e contínua no interior de $\hat{S}$, formada por uma sequência de pontos centrais. A definição de centro à qual nos referimos é devida a Sonnevend [35] que introduziu o conceito de centro analítico de um politopo. Outros conceitos de centro tais como: centro do elipsóide de menor volume que contém $\hat{S}$ e centro volumétrico, foram utilizados anteriormente sem êxito computacional por Khachyian [23] e Vaidya [36], respectivamente.

Definição 2.1 O *centro analítico* de um politopo é o único ponto definido por

$$\tilde{x} \;=\; \arg\min \{p(\hat{x}) \;\mid\; \hat{x} \in \hat{S}^0\}.$$

Os *pontos centrais* são definidos por uma modificação do problema original pelo acréscimo de uma igualdade do tipo $\hat{c}^t\hat{x} = \text{constante}$, o que equivale ao subconjunto de $\hat{S}$ formado pelos pontos com custo constante e igual a P, que será representado como

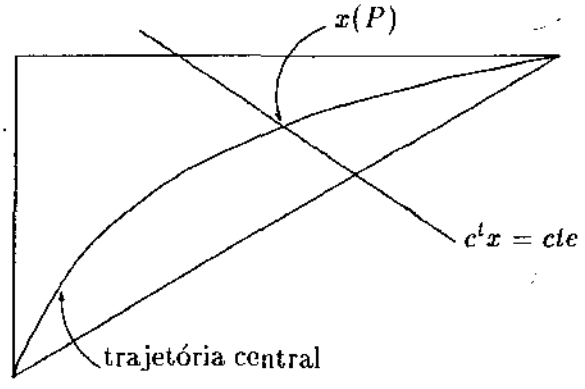


Figura 2.2: Trajetória Central

$$\hat{S}_P = \{\hat{x} \in S \mid \hat{c}^t\hat{x} = P\}.$$

Os pontos centrais, $x(P)$, são, portanto, centros analíticos dos subconjuntos $\hat{S}_P \subset \hat{S}$ com interior não vazio, Figura 2.2.

2.3.2 Função de Mérito

Como foi mencionado na seção anterior, a função barreira é combinada com a função objetivo de modo a gerar a sequência de pontos interiores convergente para $\hat{x}^*$, solução ótima de (P'). Esta função resultante é conhecida como função de mérito. Diversas destas funções foram propostas na literatura, a princípio para emprego em PNL (programação não linear) e agora utilizadas em PL. A função de mérito que usaremos neste trabalho é:

$$f_\epsilon(\hat{x}) = \epsilon\hat{c}^t\hat{x} + p(\hat{x}),$$

proposta por Frisch e conhecida por função penalizada.

Relação entre Trajetória Central e Função Penalizada

Vimos anteriormente que os pontos centrais são definidos como minimizadores de $\hat{c}^t\hat{x}$ nos subconjuntos $\hat{S}_P$, $\hat{S}_P \neq \emptyset$. Veremos agora uma maneira equivalente de definir pontos centrais, através da função penalizada, o que para nossos fins é mais atraente.

Lema 2.1 (Gonzaga [20]) Dado $\epsilon \in \Re, \epsilon > 0$, o ponto $x(\epsilon)$ definido como

$$x(\epsilon) = \arg\min\{f_\epsilon(\hat{x}) \in S^0\}, \tag{2.1}$$

é um ponto central.

O problema, portanto, a cada iteração torna-se resolver o *problema de centralização*,

$$\begin{array}{ll} \min & f_\epsilon(\hat{x}) \\ s.a. & \hat{A}\hat{x} = \hat{b} \quad (PC) \\ & \hat{x} \geq 0. \end{array}$$

Observe que à medida que ϵ tende a infinito, a solução de (PC) tende a $\hat{x}^*$, pois a função de mérito com valores grandes de ϵ prioriza a minimização do custo, ao passo que para pequenos valores de ϵ a solução de (PC) tende a $\bar{x}$ (centro analítico do politopo), pois prioriza-se a minimização da função barreira.

Propriedades da Função Barreira e Penalizada

A função barreira goza de propriedades matemáticas muito importantes; por conseguinte, as propriedades da função barreira são extensíveis à função penalizada, salvo pequenas modificações, uma vez que estas diferem entre si apenas por um termo linear. Ao longo desta seção faremos a apresentação de algumas destas propriedades da função barreira e penalizada. Optamos em não apresentar a demonstração das mesmas, visto que estas propriedades são simples de serem deduzidas e que pode-se encontrar as provas delas na referência citada.

Lema 2.2 (Gonzaga [20]) Considere um número real λ tal que $\mid \lambda \mid < 1$. Então

$$\ln(1+\lambda) \geq \lambda - \frac{\lambda^2}{2(1-|\lambda|)}. \tag{2.2}$$

A Função Barreira e Penalizada no Ponto e

A função barreira definida por

$$p(\hat{x}) = \sum_{j=1}^{2n} \ln \hat{x}_j.$$

Possui gradiente e hessiana iguais a

$$\begin{aligned} \nabla p(\hat{x}) &= -\hat{x}^{-1}, \\ \nabla^2 p(\hat{x}) &= \hat{X}^{-2}, \end{aligned}$$

onde $\hat{x}^{-1} = (\hat{x}_1^{-1}, \ldots, \hat{x}_{2n}^{-1})^t$ e $\hat{X}^{-2} = diag(\hat{x}_1^{-2}, \ldots, \hat{x}_{2n}^{-2})$

No ponto $e = (1, \ldots, 1)^t$ estes valores são

$$\begin{aligned} p(e) &= 0, \\ \nabla p(e) &= -e, \\ \nabla^2 p(e) &= I. \end{aligned}$$

Devido a esta característica, por assumir tais valores em e, estudaremos o comportamento da função barreira e penalizada neste ponto, o que facilita a evolução dos cálculos e deduções. Ademais, como veremos adiante, os resultados obtidos são extensíveis aos demais pontos através de uma mudança de escala conveniente.

Lema 2.3 (Gonzaga [20]) Dados $\lambda \in \Re$ e $h \in \Re^n$, onde $\lambda \in [0, 1)$ e $\|h\| = 1$, é válida a seguinte relação:

$$p(e + \lambda h) \geq -\lambda e^t h + \frac{\lambda^2}{2} \frac{1}{1 - |\lambda|}. \tag{2.3}$$

Lema 2.4 (Gonzaga [20]) Considere um vetor $d \in \Re^{2n}$, $\|d\|_\infty < 1$. Tem-se que

$$\nabla p(e + d) = \nabla p(e) + d + o(d), \tag{2.4}$$

onde $o(d) \in \Re^n$ é tal que

$$\|o(d)\| \leq \frac{\|d\|^2}{1 - \|d\|_\infty}.$$

Como a função barreira difere da função penalizada apenas por um termo linear, as mesmas propriedades vistas aqui para a função barreira são extensíveis também à função penalizada. Assim, de (2.3) e (2.4) derivam as seguintes relações, com $\|h\| = 1$, $\|d\|_\infty < 1$, e $\lambda \in [0,1)$:

$$f_\epsilon(e+\lambda h) \leq f_\epsilon(e) + \lambda \nabla f_\epsilon(e)^t h + \frac{\lambda^2}{2}\frac{1}{1-|\lambda|}, \tag{2.5}$$

$$\nabla f_\epsilon(e+d) = \nabla f_\epsilon(e) + d + o(d). \tag{2.6}$$

2.3.3 Propriedades da Trajetória Central

Sabemos que os algoritmos de trajetória central geram minimizadores da função $f_\epsilon(x)$ para diferentes valores de ϵ. Uma vez que é impossível determinar exatamente estes minimizadores em um computador digital, estudaremos nesta seção algumas propriedades da trajetória central que nos permitem resolver aproximadamente o problema de centralização. Esta aproximação envolve um critério de distância à trajetória central que será definido posteriormente. Estudaremos a princípio apenas as propriedades dos pontos próximos ao centro analítico do politopo, pois as conclusões tiradas podem ser naturalmente estendidas aos demais pontos centrais.

Suponha inicialmente que o centro analítico do politopo seja $\tilde{x} = e$. Temos que o valor da função barreira em torno de e pode ser escrita como

$$p(e+h) = p(e) + \nabla p(e)^t h + \frac{1}{2} h^t \nabla^2 p(e) h + o(h),$$

em e temos que

$$\begin{aligned} p(e) &= 0, \\ \nabla p(e) &= -e, \\ \nabla^2 p(e) &= I. \end{aligned}$$

Portanto

$$p(e+h) = \nabla p(e)^t h + \frac{1}{2}\|h\|^2 + o(h).$$

Como $\tilde{x} = e$ então $\nabla p(e)^t h = 0$, assim

$$p(e+h) = \frac{1}{2}\|h\|^2 + o(h). \tag{2.7}$$

Observe, pela equação acima, que para pequenos valores de h as curvas de nível da função barreira devem ser aproximadamente esféricas, pois $o(h)$ é pequeno. Isto implica que a aproximação quadrática é muito boa perto do centro. Na Figura 2.3 ilustramos as curvas de nível de $p(x)$ próximo ao centro analítico.

Mudança de ponto-de-vista

O problema como nos é apresentado é bem mais complexo que o do início desta exposição, o centro é desconhecido e certamente não é o ponto e. Ademais, uma mudança de escala que leve $\tilde{x}$ a e não poderia ser efetuada por que, obviamente, $\tilde{x}$ é desconhecido. Para sanar esta dificuldade, observemos o problema sob um ponto-de-vista diferente; suponha que desejamos saber se o ponto e está ou não na vizinhança de $\tilde{x}$. Os resultados deste estudo poderão ser naturalmente estendidos para os demais pontos; isto é, será sempre possível determinar se um ponto, x, é ou não aproximadamente central dado que, feito uma mudança de escala que leve x a e, e também é ou não aproximadamente central.

Pela equação (2.7) pode-se observar, intuitivamente, que uma boa estimativa da proximidade de x ao centro pode ser dada pela norma da direção de Newton, h_N, quando o centro é e. Da mesma forma que quando e está próximo do centro, a norma da direção de Newton continua sendo um bom parâmetro para estimar quão perto e está do centro, dado ao bom comportamento da função barreira próximo a e. Nas seções seguintes provaremos que a proximidade realmente pode ser expressa desta forma. Mostraremos que existe uma relação entre o módulo de h_N e a distância entre e e $\tilde{x}$ que nos assegura que para pequenos valores de $\|h_N\|$ temos que $\|e - \tilde{x}\|$ também é pequeno.

Direção de Newton e Proximidade

Nesta seção veremos como é determinada a direção de Newton-Raphson para o problema de centralização. Estaremos, portanto, tratando aqui de quaisquer pontos centrais e não apenas do centro analítico do politopo original como na exposição anterior.

Denotaremos por $h_N(\hat{x}, \epsilon)$ a direção de Newton-Raphson gerada a partir do ponto $\hat{x} \in \hat{S}^0$ para o problema $\{\min f_\epsilon(\hat{x}) \mid \hat{x} \in \hat{S}^0,\ \epsilon > 0\}$.

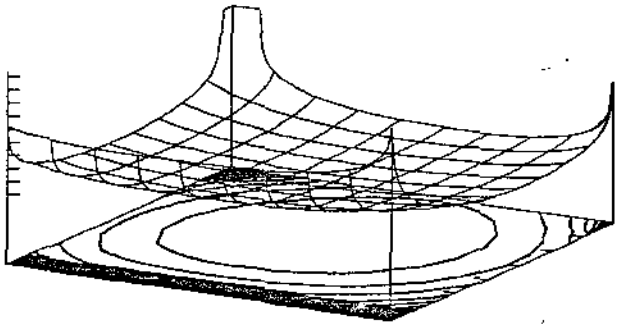


Figura 2.3: Curvas de nível da função barreira

Lema 2.5 (Gonzaga [20]) Considere o problema de centralização. A direção de Newton-Raphson é dada por:

$$\begin{array}{lrcl} (i) & h_N(\hat{x},\epsilon) & = & -P_{\hat{A}}\nabla f_\epsilon(e) \;=\; -P(\epsilon\hat{c}-e), \\ (ii) & h_N(\hat{x},\epsilon) & = & -\hat{X}P_{\hat{A}\hat{X}}\hat{X}\nabla f_\epsilon(\hat{x}) \;=\; -\hat{X}P_{\hat{A}\hat{X}}(\epsilon\hat{X}\hat{c}-e), \end{array}$$

com $\hat{X} = diag(\hat{x}_i)$

Demonstração. (i) A aproximação linear de ∇f_ϵ no ponto e é dada por:

$$\nabla f_\epsilon(e+h) \approx \nabla f_\epsilon(e) + (\nabla^2 f_\epsilon(e))^t h = \nabla f_\epsilon(e) + Ih.$$

O ponto que corresponde ao mínimo da aproximação quadrática de f sobre o núcleo de $\hat{A}$ anula a projeção de $\nabla f_\epsilon(e+h)$ portanto

$$P_{\hat{A}}\nabla f_\epsilon(e) + h = 0 \Rightarrow h = -P_{\hat{A}}\nabla f_\epsilon(e),$$

pois $h \in \mathcal{N}(\hat{A})$. Como $\nabla f_\epsilon(e) = \epsilon c - e$, segue-se $h_N(e,\epsilon) = -P(\epsilon c - e)$.

(ii) Considere

$$f_\epsilon(e) \;=\; \epsilon(\hat{X}\hat{c})^t e \;+\; p(e),$$

com

$$\nabla f_\epsilon(e) \;=\; \epsilon\hat{X}\hat{c} - e \;=\; \hat{X}\nabla f_\epsilon(x) \text{ e } A \;=\; \hat{A}\hat{X}.$$

Da primeira parte do lema temos que, a direção de Newton-Raphson após a mudança de escala é

$$\bar{h}_N(e,\epsilon) \;=\; -P_{\bar{A}}\nabla\bar{f}_\epsilon(e),$$

o que equivale a

$$\bar{h}_N(e,\epsilon) \;=\; -P_{\hat{A}\hat{X}}\hat{X}\nabla f_\epsilon(\hat{x}).$$

Voltando a escala original, tem-se

$$\begin{aligned} h_N(\hat{x},\epsilon) \;&=\; \hat{X}\bar{h}_N(e,\epsilon) \\ &=\; -\hat{X}P_{\hat{A}\hat{X}}\hat{X}\nabla f_\epsilon(\hat{X}) \\ &=\; -\hat{X}P_{\hat{A}\hat{X}}(\epsilon\hat{X}c \;-\; e). \end{aligned}$$

□

Vejamos agora a definição formal de proximidade que, como vimos, está intimamente relacionada com a norma do vetor $h_N(\hat{x},\epsilon)$.

Definição 2.2 Dados $\epsilon > 0$ e $\hat{x} \in \hat{S}^0$, a *proximidade* de $\hat{x}$ em relação a $x(\epsilon)$ será dada por

$$\delta(\hat{x},\epsilon) \;=\; \|\hat{X}^{-1}h_N(x,\epsilon)\|,$$

com $\hat{X} = diag(\hat{x}_i)$.

Lema 2.6 (Gonzaga [20]) Considere um ponto $\hat{x} \in \hat{S}^0$. Se $\delta(\hat{x},\epsilon) \;=\; \alpha \leq 0.1$ então $\|\hat{X}^{-1}(\hat{x} - x(\epsilon))\| \leq 1.15\alpha$.

Demonstração. Inicialmente, suponha que $\hat{x} = e$. Assim temos que por hipótese $\delta(e,\epsilon) = \alpha \leq 0.1$. Seja $h = (x(\epsilon) - e)/\|x(\epsilon) - e\|$, a direção normalizada entre e e $x(\epsilon)$. Seja $x(\epsilon) = e + \lambda h$, queremos provar que $\lambda \leq 1.15\alpha$.

Considerando que $\|\lambda h\| \leq 1.15\alpha$ e $\alpha \leq 0.1$, então temos que $\|\lambda h\| < 1$, o que implica em $\|\lambda h\|_\infty < 1$. Portanto, temos pela equação (2.6) que

$$\nabla f_\epsilon(e + \lambda h) \;=\; \nabla f_\epsilon(e) + \lambda h + o(\lambda h),$$

com

$$\|o(\lambda h)\| \leq \frac{\|\lambda h\|^2}{1 - \|\lambda h\|_\infty},$$

o que implica que

$$\|o(d)\| \leq \frac{\|\lambda h\|^2}{1 - \|\lambda h\|} = \frac{\lambda^2}{1 - \lambda},$$

pois $\|\cdot\|_\infty \leq \|\cdot\|$ e $\|\lambda h\| = \lambda$.

Temos então que

$$h^t \nabla f_\epsilon(e + \lambda h) = h^t \nabla f_\epsilon(e) + \lambda h^t h + h^t o(\lambda h),$$

como $h \in \mathcal{N}(\hat{A})$, então $h^t \nabla f_\epsilon(e) = h^t P_{\hat{A}} \nabla f_\epsilon(e)$, assim

$$\begin{aligned} h^t \nabla f_\epsilon(e) &= h^t P_{\hat{A}} \nabla f_\epsilon(e) \\ &\geq -\|h\| \, \|h_N(e, \epsilon)\| \\ &= -\alpha. \end{aligned}$$

O termo $\lambda \|h\|^2$ é obviamente igual a λ, o termo $h^t o(\lambda h)$ pode ser minorado observando que

$$\begin{aligned} h^t o(\lambda h) &\geq -\|h\| \, \|o(\lambda h)\| \\ &\geq -\frac{\lambda^2}{1 - \lambda}. \end{aligned}$$

Assim, temos que

$$h^t \nabla f_\epsilon(e + \lambda h) \geq -\alpha + \lambda - \frac{\lambda^2}{1 - \lambda}.$$

Por definição, $P_{\hat{A}} \nabla f_\epsilon(x(\epsilon)) = 0$, o que implica que

$$h^t \nabla f_\epsilon(x(\epsilon)) = h^t P_{\hat{A}} \nabla f_\epsilon(x(\epsilon)) = 0,$$

pois $h \in \mathcal{N}(A)$. Assim

$$0 = h^t P_{\hat{A}} \nabla f_\epsilon(x(\epsilon)) = h^t \nabla f_\epsilon(e + \lambda h) \quad \geq \quad -\alpha + \lambda - \frac{\lambda^2}{1-\lambda}.$$

Daí segue-se que

$$0 \quad \geq \quad -2\lambda^2 + (\alpha - 1)\lambda - \alpha. \tag{2.8}$$

Plotando a função para os valores de $\alpha \in [0, 0.1]$ pode-se observar que

$$\lambda \quad \leq \quad 1.15\alpha.$$

Como queríamos provar.

Para provar que para um ponto qualquer, $\hat{x}$, no interior de $\hat{S}$, $\delta(\hat{x}, \epsilon) = \alpha \leq \|\hat{X}^{-1}(\hat{x} - x(\epsilon))\| \leq 1.15\alpha$, basta que se observe que a mudança de de escala que leva $\hat{x}$ a e não afeta os valores de $\delta(e, \epsilon)$ e $\|\hat{X}^{-1}(\hat{x} - x(\epsilon))\|$. □

Este lema prova o que havia sido dito informalmente na seção 2.3.3. Provamos que a direção de Newton-Raphson, que deriva a proximidade, é de fato uma boa medida para determinar se um ponto está ou não próximo à trajetória central.

2.4 Algoritmo de Barreira de Passos Curtos

Este algoritmo consiste na geração de uma sequência de pontos $(\hat{x}^k)$, e de parâmetros de penalidade (ϵ_k) tais que os pontos $\hat{x}^k$ sejam aproximadamente centrais. A sequência $(\hat{x}^k)$ é gerada de tal modo que os pontos consecutivos $\hat{x}^k$ e $\hat{x}^{k+1}$ estejam próximos entre si, daí o nome algoritmo de passos curtos. A razão de se executar passos curtos é assegurar a proximidade à trajetória central; uma vez que, sendo $\hat{x}^k$ aproximadamente central, o ponto seguinte, $\hat{x}^{k+1}$, também será, desde que, distanciando-se apropriadamente em uma determinada direção, o tamanho do passo seja pequeno. O lema 2.8, como veremos a seguir, indica que direção deve ser tomada.

Lema 2.7 (Gonzaga [20]) Dado um ponto $z \in \Re^n_{++}$. Tem-se que se, $\|z - e\|_\infty \leq \alpha \leq 1$, então para qualquer $y \in \Re^n$

$$\|Z^{-1}d\| \leq \frac{1}{1-\alpha}\|d\|,$$

onde $Z^{-1} = diag(z_1^{-1}, \ldots, z_n^{-1})$.

Demonstração. Tem-se que $\|Z^{-1}d\| = \|[\frac{d_i}{z_i}]\|$, como $\|z - e\|_\infty \leq \alpha$ então.

$$|z_i - 1| \leq \alpha,$$

logo

$$z_i \geq 1 - \alpha,$$

o que implica que

$$\begin{aligned} \|Z^{-1}d\| &\leq \|[\frac{d_i}{1-\alpha}]\| \\ &= \frac{1}{1-\alpha}\|d\|. \end{aligned}$$

Completando assim a demonstração. □

Lema 2.8 (Gonzaga [20]) Dado um ponto $\hat{x} \in \hat{S}^0$ e um parâmetro de penalidade $\epsilon > 0$, tal que $\alpha = \delta(\hat{x}, \epsilon) \leq 0.1$, e seja $\hat{y} = \hat{x} + h_N(\hat{x}, \epsilon)$. Então

$$\delta(\hat{y}, \epsilon) \leq 1.12\alpha^2.$$

Demonstração. Assuma sem perda de generalidade que $\hat{x} = e$, consequentemente, $\hat{y} = e + h_N(e, \epsilon)$. Pelo lema 2.5

$$\begin{aligned} h_N(e,\epsilon) &= -P_{\hat{A}}\nabla f_\epsilon(e) \quad \text{e} \\ \alpha = \delta(e,\epsilon) &= \|h_N(e,\epsilon)\| = \|P_{\hat{A}}\nabla f_\epsilon(e)\|. \end{aligned}$$

Da equação (2.6) temos que

$$\nabla f_\epsilon(\hat{y}) = \nabla f_\epsilon(e + h_N(e,\epsilon)) = \nabla f_\epsilon(e) + h_N(e,\epsilon) + o(h_N(e,\epsilon)),$$

onde

$$\|o(h_N(e,\epsilon))\| \leq \frac{\|h_N(e,\epsilon)\|^2}{1-\|h_N(e,\epsilon)\|} = \frac{\alpha^2}{1-\alpha}.$$

Projetando $\nabla f_\epsilon(\hat{y})$ sobre o núcleo de $\hat{A}$, temos

$$\begin{aligned} P_{\hat{A}}\nabla f_\epsilon(\hat{y}) &= P_{\hat{A}}\nabla f_\epsilon(e) + h_N(e,\epsilon) + P_{\hat{A}}o(h_N(e,\epsilon) \\ &= P_{\hat{A}}o(h_N(e,\epsilon)), \end{aligned}$$

como $\|P_{\hat{A}}o(h_N(e,\epsilon))\| \leq \|o(h_N(e,\epsilon))\|$ então

$$\|P_{\hat{A}}\nabla f_\epsilon(\hat{y})\| \leq \frac{\alpha^2}{1-\alpha}.$$

Agora, tendo que $\|\hat{y} - e\|_\infty = \|h_N(e,\epsilon)\|_\infty \leq \|h_N(e,\epsilon)\| \leq 0.1$, usamos o lema 2.7 para obter

$$\begin{aligned} \delta(\hat{y},\epsilon) = \|\hat{Y}^{-1}P_{\hat{A}}\nabla f_\epsilon(\hat{y})\| &\leq \frac{1}{1-\alpha}\|P_{\hat{A}}\nabla f_\epsilon(\hat{y})\| \\ &\leq \frac{1}{1-\alpha}\frac{\alpha^2}{1-\alpha} \\ &\leq \alpha^2\frac{1+\alpha}{1-\alpha} \\ &\leq 0.012\alpha^2, \end{aligned}$$

como queríamos provar. □

A importância deste lema, como veremos a seguir, é fundamental no estudo do algoritmo de passos curtos, pois, dele deriva a idéia central do algoritmo.

Para obtermos as seqüências $(\hat{x}^k)$ e (ϵ_k) o algoritmo vale-se de duas prerrogativas básicas:

- $\delta(\hat{x}^k, \epsilon_k) \leq 0.015$
- $\delta(\hat{x}^k, \epsilon_{k+1}) \leq 0.1$

Essencialmente estas prerrogativas são os resultados derivados do lema 2.8 quando a direção de descida é a direção de Newton-Raphson. Torna-se mais claro, então, a idéia do funcionamento do algoritmo, veja Figura 2.4. A cada iteração, temos $\hat{x}^k$ e ϵ_k tais que $\delta(\hat{x}^k, \epsilon_k) \leq 0.015$, definindo em seguida o próximo parâmetro de penalidade ϵ_{k+1}, de tal modo que $\delta(\hat{x}^k, \epsilon_{k+1}) = 0.1$, e definindo o ponto seguinte como $\hat{x}^{k+1} = \hat{x}^k + h_N(\hat{x}^k, \epsilon_{k+1})$, pelo lema 2.8, $\hat{x}^{k+1}$ também será aproximadamente central, e $\delta(\hat{x}^{k+1}, \epsilon_{k+1}) \leq 0.015$. Este processo repete-se sucessivamente até que se obtenha $\hat{x}^k$ suficientemente próximo de $\hat{x}^*$. Resta-nos agora enunciar um meio de definir ϵ_{k+1} e provar que a sequência $(\hat{x}^k)$ gera custos decrescentes.

O próximo parâmetro de penalidade é obtido simplesmente pela resolução de uma equação do segundo grau. É fácil ver que $\delta(\hat{x}^k, \epsilon_{k+1}) = 0.1$ é na verdade uma equação do segundo grau em ϵ_{k+1}, pois δ deriva da norma euclideana.

$$\begin{aligned} 0.1 &= \delta(\hat{x}^k, \epsilon_{k+1}) \\ &= \|h_N(\hat{x}^k, \epsilon_{k+1})\| \\ &= \|\epsilon_{k+1}\hat{c}_p - e_p\| \\ &= \epsilon_{k+1}^2\|\hat{c}_p\|^2 + \epsilon_{k+1}\hat{c}_p^t e_p + \|e_p\|^2, \end{aligned}$$

com $\hat{c}_p = P_{AX}Xc$ e $e_p = P_{AX}e$ e $X = diag(\hat{X}_i)$.

Dentre as raizes da equação acima escolhe-se, obviamente, a de maior valor, pois a sequência (ϵ_k) deve ser estritamente crescente. Provemos agora que a sequência $(\hat{x}^k)$ gera custos decrescentes.

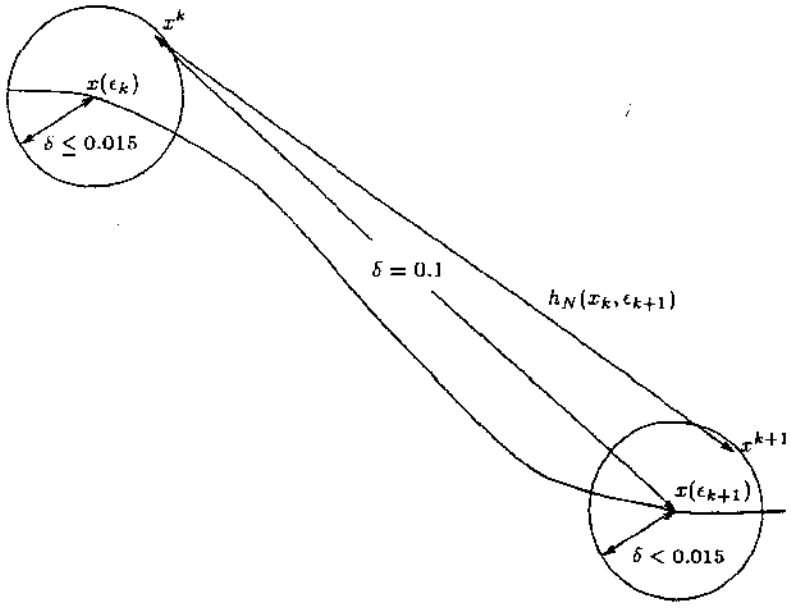


Figura 2.4: Iteração do algoritmo de passos curtos

Lema 2.9 (Gonzaga [20]) Considere a trajetória central $\epsilon > 0 \mapsto x(\epsilon) \in S^0$. A função $\epsilon > 0 \mapsto \hat{c}^t\hat{x}(\epsilon)$ é estritamente decrescente, e a função $\epsilon > 0 \mapsto v(\epsilon) = \hat{c}x(\epsilon) - \frac{2n}{\epsilon}$, é estritamente crescente.

Demonstração. Usando as condições necessárias e suficientes de otimalidade de primeira ordem para o problema de centralização tem-se em $x(\epsilon)$ que

$$P_{\hat{A}}\nabla f_\epsilon(x(\epsilon)) \;=\; 0,$$

o que implica que

$$P_{\hat{A}}(\epsilon\hat{c} - x(\epsilon)^{-1}) \;=\; 0. \tag{2.9}$$

Diferenciando em relação a ϵ, temos que

$$P_{\hat{A}}(\hat{c} - \frac{d}{d\epsilon}(x(\epsilon)^{-1})) \;=\; 0.$$

Quando $x(\epsilon) = e$, tem-se

$$P_{\hat{A}}(\hat{c} + I^{-2}\frac{d}{d\epsilon}x(\epsilon)) \;=\; 0.$$

O que implica em

$$\frac{d}{d\epsilon}x(\epsilon) \;=\; -P_{\hat{A}}\hat{c},$$

pois $\frac{d}{d\epsilon}x(\epsilon) \in \mathcal{N}(\hat{A})$. Neste sentido, se $x(\epsilon) = e$.

$$\tfrac{d}{d\epsilon}\hat{c}^t\hat{x}(\epsilon) \;=\; -\hat{c}^tP_{\hat{A}}\hat{c} \;=\; -\|P_{\hat{A}}\hat{c}\|^2 \;<\; 0.$$

Provamos, assim, que a função $\hat{c}^t\hat{x}(\epsilon)$ possui sempre derivada negativa em relação a ϵ para $\epsilon > 0$, logo a função $\epsilon > 0 \mapsto \hat{c}^t\hat{x}(\epsilon)$ é sempre decrescente em

$\hat{S}^0$. Para provar que $v(\epsilon)$ é sempre crescente, considerando $x = c$, observe que

$$\frac{d}{d\epsilon}v(\epsilon) = \frac{d}{\epsilon}(\hat{c}^t x(\epsilon) - \frac{2n}{\epsilon}),$$

segue-se então que

$$\frac{d}{d\epsilon}v(\epsilon) = \frac{d}{d\epsilon}\hat{c}x(\epsilon) + \frac{2n}{\epsilon^2} = -\|P_{\hat{A}}\hat{c}\|^2 + \frac{2n}{\epsilon^2}.$$

Resta-nos agora provar que $\|P_{\hat{A}}\hat{c}\|^2 < \frac{2n}{\epsilon^2}$. Considerando (2.9) temos que

$$\|P_{\hat{A}}\hat{c}\|^2 = \frac{\|P_{\hat{A}}e\|^2}{\epsilon^2},$$

logo

$$\|P_{\hat{A}}\hat{c}\|^2 < \frac{2n}{\epsilon^2},$$

pois, uma vez que $e \notin \mathcal{N}(\hat{A})$, então $\|P_{\hat{A}}e\| < \|e\| = 2n$. Assim, provamos que a derivada de $v(\epsilon)$ em relação a ϵ é sempre positiva, completando a demonstração. □

Algoritmo 2.3 Algoritmo de Passos Curtos
Dados $\hat{x} \in \hat{S}^0, \epsilon > 0$, tal que $\delta(\hat{x}, \epsilon) \leq 0.015$

repita
$\hat{X} = \text{diag}(\hat{x}_i)$
Resolver $(\hat{A}\hat{X}^2\hat{A}^t)y = \hat{A}\hat{X}^2\hat{c}$
$\bar{c}_p = \hat{X}c - \hat{X}\hat{A}^t y$
Resolver $(\hat{A}\hat{X}^2\hat{A}^t)y = \hat{A}\hat{X}^2 e$
$e_p = \hat{X}e - \hat{X}\hat{A}^t y$
Resolver equação do segundo grau em ϵ:
$\epsilon^2\|\bar{c}_p\|^2 - 2\epsilon\bar{c}_p^t e_p + \|e_p\|^2 = 0.01$
$h_N = -\epsilon\hat{c}_p + e_p$
$\hat{x} = \hat{x} + \hat{X}h_N$
até convergir

2.4.1 Inicialização

A inicialização será feita em duas etapas. Primeiro encontramos um ponto interior factível pelo algoritmo 2.2, em seguida, usamos o algoritmo de centralização, que será apresentado nesta seção, para encontrar o ponto inicial $\hat{x}^0$.

O Algoritmo de Centralização

Este algoritmo compõe o procedimento utilizado para resolver o problema de centralização

$$\begin{aligned} \min \quad & f_\epsilon(\hat{x}) \\ s.a. \quad & \hat{x} \in \hat{S}^0, \end{aligned}$$

onde o valor de ϵ é um parâmetro dado e deve ser maior do que zero. Por tratar-se de um problema com números reais, não possui solução exata calculada no computador. Resolveremos este problema aproximadamente dentro de uma tolerância estabelecida. O algoritmo nada mais é do que a aplicação do método de Newton-Raphson para minimizar a função penalizada; parte de um ponto interior factível dado, e segue iterativamente até que se encontre um ponto suficientemente próximo de $x(\epsilon)$ para a tolerância considerada.

Algoritmo 2.4 Algoritmo de Centralização
Dados $\hat{x} \in \hat{S}^0$, $\epsilon \geq 0$

repita
$X = diag(\hat{x}_i)$
Resolver $(\hat{A}X^2\hat{A}^t)y = \hat{A}X(\epsilon X\hat{c} - e)$
$h = -X(\epsilon X\hat{c} - e - X\hat{A}^t y)$
$\alpha = \|X^{-1}h\|$
Resolver aproximadamente $\bar{\lambda} = \arg\min\{f_\epsilon(x + \lambda h) \mid \lambda > 0\}$
$x = x + \bar{\lambda}h$
até $\alpha < 2^{-L}$

Tamanho do Passo A determinação do tamanho do passo é um procedimento simples, de complexidade $O(1)$, para resolver: $\bar{\lambda} = \arg\min\{f_\epsilon(x+\lambda h) \mid \lambda > 0\}$. Gonzaga em [20] valeu-se de uma propriedade da função penalizada apresentada no lema a seguir que garante o decréscimo da mesma para valores apropriados de um escalar λ. A idéia aqui é fazer $\bar{\lambda}$ igual a estes valores, pois estaremos asssim seguros do decréscimo da função penalizada.

Lema 2.10 (Gonzaga [20]) Dado um ponto $\hat{x} \in \hat{S}^0$, um parâmetro de penalidade ϵ e, sendo $h = h_N(\hat{x}, c)/\|h_N(\hat{x}, \epsilon)\|$, então.

(i) Se $\delta(\hat{x}, \epsilon) > 1$, então $f_\epsilon(\hat{x} + 0.5h) \leq f_\epsilon(\hat{x}) - 0.25\delta(\hat{x}, \epsilon)$.
(ii) Se $0.1 \leq \delta(\hat{x}, \epsilon) \leq 1$, então $f_\epsilon(\hat{x} + 0.5\delta(\hat{x}, \epsilon)h) \leq f_\epsilon(\hat{x}) - 0.25(\delta(\hat{x}, \epsilon))^2$.
(iii) Se $\delta(\hat{x}, c) \leq 0.1$, então $f_\epsilon(\hat{x} + \delta(\hat{x}, \epsilon)h) \leq f_\epsilon(\hat{x}) - 0.4(\delta(\hat{x}, \epsilon))^2$.

Demonstração. Suponha sem perda de generalidade que $\hat{x} = e$. De (2.5) temos que

$$f_\epsilon(e + \lambda h) \leq f_\epsilon(e) + \lambda(\nabla f_\epsilon(e))^t h + \frac{\lambda^2}{2}\frac{1}{1 - |\lambda|}, \tag{2.10}$$

como $h \in \mathcal{N}(\hat{A})$ então

$$f_\epsilon(e + \lambda h) \leq f_\epsilon(e) - \lambda\delta(e, \epsilon) + \frac{\lambda^2}{2}\frac{1}{1 - |\lambda|},$$

pois $(\nabla f_c(e))^t h = (P_{\hat{A}} \nabla f_\epsilon(e))^t h = -\delta(e, \epsilon)$.

Para obtermos os resultados de (i) a (iii), basta escolher valores apropriados para o tamanho do passo, de modo a termos satisfeita a relação acima.
(i) No primeiro caso temos que, fazendo $\lambda = 0.5$,

$$\begin{aligned} f_\epsilon(e + 0.5h) &\leq f_\epsilon(e) - 0.5\delta(e, \epsilon) + \frac{0.5^2}{2}\frac{1}{1 - |0.5|} \\ &= f_\epsilon(e) - 0.5\delta(e, c) + 0.25 \\ &= f_\epsilon(e) - 0.25(2\delta(e, \epsilon) - 1), \end{aligned}$$

como $2\delta(e,\epsilon) - 1 > \delta(e,\epsilon)$, então

$$f_\epsilon(c + 0.5h) \leq f_\epsilon(c) - 0.25\delta(c,\epsilon).$$

(ii) No caso em que $0.1 \leq \delta(e,\epsilon) \leq 1$, substituindo λ por $0.5\delta(e,\epsilon)$ temos

$$f_\epsilon(c + 0.5\delta(e,\epsilon)h) \leq f_\epsilon(e) - 0.5(\delta(e,\epsilon))^2 + \frac{(0.5\delta(e,\epsilon))^2}{2}\frac{1}{1 - |0.5\delta(e,\epsilon)|},$$

donde segue-se que

$$f_\epsilon(e + 0.5\delta(e,\epsilon)h) \leq f_\epsilon(e) + \left(-1 + \frac{1}{4(1 - 0.5\delta(e,\epsilon))}\right) 0.5(\delta(e,\epsilon))^2,$$

o que implica em

$$-1 + \frac{1}{4(1 - 0.5\delta(e,\epsilon))} \leq -\frac{1}{2},$$

logo

$$\delta(e,\epsilon) \leq 1,$$

como $\delta(e,\epsilon) \leq 1$ então fica provado o decréscimo da função penalizada para este valor de λ.

(iii) No caso em que $\delta(e,\epsilon) < 0.1$, temos que, substituindo o valor de λ por $\delta(e,\epsilon)$ encontramos

$$f_\epsilon(e + \delta(e,\epsilon)h) \leq f_\epsilon(e) - (\delta(e,\epsilon))^2 + \frac{\delta(e,\epsilon))^2}{2(1 - \delta(e,\epsilon))},$$

o que implica em

$$f_\epsilon(e + \delta(e,\epsilon)h)) \leq f_\epsilon(e) + \left(-1 + \frac{1}{2(1 - \delta(e,\epsilon))}\right)(\delta(e,\epsilon))^2,$$

então, devemos ter

$$-1 + \frac{1}{2(1-\delta(e,\epsilon))} \leq -\frac{2}{5},$$

o que implica que

$$\delta(e,\epsilon) \leq \frac{1}{6},$$

o que se verifica pois $\delta(e,\epsilon) \leq 0.1$. Completa-se assim a demonstração. □

A maior importância deste lema está em assegurar garantidamente o decréscimo da função penalizada fazendo-se uma escolha adequada do tamanho do passo, λ. Assim a busca linear torna-se um problema simples, o valor de $\bar{\lambda}$ é determinado como

$$\begin{array}{lrcll} (i) & \bar{\lambda} & = & 0.5 & se \quad \delta(\hat{x},\epsilon) > 1. \\ (ii) & \bar{\lambda} & = & 0.5\delta(\hat{x},\epsilon) & se \quad 0.1 \leq \delta(\hat{x},\epsilon) \leq 1. \\ (iii) & \bar{\lambda} & = & \delta(\hat{x},\epsilon) & se \quad \delta(\hat{x},\epsilon) < 0.1. \end{array}$$

Esta é uma maneira barata e segura de determinar o tamanho do passo. Certamente outras alternativas podem ser tomadas, pois o valor de $\bar{\lambda}$ sugerido por Gonzaga [20] não necessariamente representa o ótimo.

Critério de Parada do Algoritmo de Centralização O critério de parada é satisfeito quando a proximidade ao ponto central é menor do que a tolerância estabelecida. Neste trabalho a tolerância é dada por $2^{-(2n)L}$, a qual estudaremos mais detalhadamente na seção 2.4.2.

Complexidade do Algoritmo de Centralização A análise de complexidade será feita observando duas situações. Primeiro, quando o ponto $\hat{x}^k$ está distante do minimizador, $x(\epsilon)$; nos referimos aqui a situações em que $\delta(\hat{x}^k,\epsilon) \geq 0.1$. Segundo, quando x^k está próximo do minimizador, isto é, quando $\delta(\hat{x},\epsilon) \leq 0.1$. Faremos uma análise de complexidade separadamente para estas duas situações. A complexidade global será a soma dos resultados obtidos em cada uma delas.

Lema 2.11 (Gonzaga [20]) Considere o algoritmo 2.3. Dados $\hat{x}^0 \in \hat{S}^0$ e $\epsilon > 0$, tais que $\delta(\hat{x}^0, c) < 1$, o algoritmo pára no máximo em $400(f_\epsilon(\hat{x}^0) - f_\epsilon(x(\epsilon))) + \lg(L)$ iterações.

Demonstração. Dado o ponto $\hat{x}^0 \in \hat{S}^0$, o algoritmo de centralização operará de modo diferente conforme a distância ao minimizador, $x(\epsilon)$. De acordo com o lema 2.10, poderão ocorrer três situações distintas.

Temos pelo lema 2.10 que o decréscimo da função penalizada pode ser

$$\begin{array}{lrcll} \text{(i)} & f_\epsilon(x^{k+1}) & \leq & f_\epsilon(x^k) - 0.25 & \text{se } \delta(\hat{x}^k, \epsilon) > 1. \\ \text{(ii)} & f_\epsilon(x^{k+1}) & \leq & f_\epsilon(x^k) - 0.0025 & \text{se } 0.1 \leq \delta(\hat{x}^k, \epsilon) \leq 1. \end{array}$$

Assim, na pior situação, teríamos em todo ponto em que $\delta(\hat{x}^k, \epsilon) \geq 0.1$

$$f_\epsilon(x^{k+1}) \leq f_\epsilon(x^k) - 0.0025.$$

Após K iterações, teríamos

$$f_\epsilon(x^K) \leq f_\epsilon(\hat{x}^0) - \frac{K}{400}.$$

O que implica que para $K = 400(f_\epsilon(\hat{x}^c) - f_\epsilon(x(\epsilon)))$ o algoritmo pára, pois, para este valor de K teríamos $f_\epsilon(\hat{x}^K) \leq f_\epsilon(x(\epsilon))$.

A segunda parte recai na situação em que $\delta(\hat{x}^k, \epsilon) \leq 0.1$. Pelo lema 2.8 temos que

$$\delta(\hat{x}^{k+1}, \epsilon) \leq 1.12(\delta(\hat{x}^k, \epsilon))^2.$$

Supondo que a segunda fase do algoritmo inicia-se na l-ésima iteração, temos que após $K2$ iterações

$$\delta(\hat{x}^{K2}, \epsilon) \leq (1.12\delta(\hat{x}^{K2}, \epsilon))^{2^{K2-l}},$$

o que implica que

$$\lg\ \delta(\hat{x}^{K2}, c)\ \leq\ 2^{K2-l} \lg\ (0.112)\ \leq\ -2^{K2-l},$$

uma vez que $\lg\ (0.112) < -1$.

Fazendo $K2 - l \geq \lg\ L$ obtém-se $\delta(\hat{x}^{K2}, c) \leq 2^{-L}$. Desta forma estabelecemos o número máximo de iterações na segunda parte como $\lg\ L$. Somando os resultados obtidos encontramos o número máximo de iterações $(400(f_c(\hat{x}^0) - f_c(x(\epsilon))) + \lg\ L)$. □

2.4.2 Critério de Parada

Estudaremos nesta seção um critério de parada mais rigoroso. Este critério está intrinsecamente relacionado com L, o número de bits necessários para armazenar o problema. O algoritmo deve parar quando a diferença entre o custo da solução ótima e o custo da última iteração for menor do que 2^{-L}. Em Gonzaga [20] e Karmarkar [21] pode-se obter uma prova de que $\hat{x}^*$ é encontrado a partir desta última iteração sem que se gaste mais do que $O(n^3)$ operações, usando um algoritmo de "purificação". Esta prova foge ao objetivo principal deste trabalho e, portanto, não será incorporada ao mesmo. Serão deixadas como sugestão para o leitor mais interessado as referências acima, que abordam mais detalhadamente o problema de purificação.

Lema 2.12 (Gonzaga [20]) Considere $x \in S^0$ e $\epsilon \in \Re$ tais que $\delta(x, \epsilon) = \alpha \leq 0.1$, então

$$|\hat{c}^t \hat{x} - \hat{c}^t x(\epsilon)|\ \leq\ 1.1\alpha \frac{\sqrt{2n}}{\epsilon}. \tag{2.11}$$

Lema 2.13 (Gonzaga [20]) Considere $\epsilon > 0$ um parâmetro de penalidade e $x(\epsilon)$ o ponto central correspondente. Então

$$\hat{c}^t \hat{x}(\epsilon) - \hat{c}^t \hat{x}^*\ \leq\ \frac{2n}{\epsilon}.$$

Demonstração. Suponha que $\hat{x}^* = c$. Então

$$P_{\hat{A}} \nabla f_\epsilon(c) = 0,$$

daí segue-se que

$$P_{\hat{A}}(\epsilon c - e) = 0,$$

logo, temos que

$$\epsilon(e - \hat{x}^*)^t P_{\hat{A}} c = (c - \hat{x}^*)^t P_{\hat{A}} e.$$

Como $(e - x(\epsilon)) \in \mathcal{N}(A)$ então

$$\begin{aligned} \epsilon(c - x(\epsilon))^t c &= (e - x(\epsilon))^t e \\ &= \frac{2n - e^t x(\epsilon)}{\epsilon} \\ &\leq \frac{2n}{\epsilon}, \end{aligned}$$

pois $x(\epsilon) \geq 0$, donde segue-se que $e^t x(\epsilon) \geq 0$. □

Tendo em vista que o critério de parada adotado baseia-se na variação do valor das soluções nas iterações em relação ao valor da solução ótima e, como não dispomos a priori deste valor ótimo, necessitamos então de uma estimativa do valor da solução ótima. O lema acima fornece esta estimativa. Assim, podemos agora definir o critério de parada.

Por hipótese queremos que $\hat{c}^t \hat{x}^k - \hat{c}^t \hat{x}^* \leq 2^{-L}$, então, se fizermos $\epsilon_k > (2n)2^L$, por exemplo, $\epsilon_k \geq 2(2n)2^L$ teremos $\hat{c}^t x^k - \hat{c}^t \hat{x}^* \leq 2^{-L}$, pois pelos lemas 2.12 e 2.13 teremos

$$\begin{aligned} \hat{c}^t \hat{x}^k - \hat{c}^t \hat{x}^* &\leq \frac{2n}{\epsilon_k}(1 + \frac{0.1}{\sqrt{2n}}) \\ &\leq 1.2\frac{(2n)}{\epsilon_k} \\ &\leq 0.6\ 2^{-L}. \end{aligned}$$

Podemos então estabelecer $\epsilon_k \geq 4n2^L$ como critério de parada dos algoritmos de trajetória central de Gonzaga.

2.4.3 Complexidade

A análise de complexidade será feita tendo em vista o critério de parada definido na seção anterior. Apresentaremos nesta seção dois lemas que provam que o algoritmo pára em um número polinomial de iterações. A complexidade do número de operações é obtida por uma análise simples que faremos no final da seção.

Lema 2.14 (Gonzaga [20]) Considere uma aplicação do algoritmo 2.3; a variação no parâmetro de penalidade ϵ_k é tal que

$$\Delta\epsilon \;=\; \epsilon_{k+1} - \epsilon_k \geq 0.08\frac{\epsilon_k}{\sqrt{2n}}.$$

Demonstração. A idéia é provar que ao fixar o próximo valor de ϵ de modo que $\delta(\hat{x}^k, \epsilon_{k+1}) = 0.1$, deveremos ter ϵ_{k+1} tal que $\Delta\epsilon \geq 0.08\epsilon_k/\sqrt{2n}$.

Por hipótese temos que

$$\delta(\hat{x}^k, \epsilon_k) \;\leq\; 0.015,$$

o que implica que

$$\|h_N(\hat{x}^k, \epsilon_k)\| \;=\; \|P_{\hat{A}X} X \nabla f_\epsilon(\hat{x}^k)\| \;=\; \|P_{\bar{A}X} X(\epsilon c - (\hat{x}^k)^{-1}\| \;\leq\; 0.015.$$

Chamando $c_p = P_{\hat{A}X}\hat{c}$ e $e_p = P_{\hat{A}X}e$, temos que

$$\|\epsilon c_p - e_p\| \;\leq\; 0.015. \tag{2.12}$$

Como $\delta(\hat{x}^k, \epsilon_{k+1}) = 0.1$, por hipótese, então

$$\begin{aligned} 0.1 \;&=\; \|\epsilon_{k+1}c_p - e_p\| \\ &=\; \|\epsilon_{k+1}c_p - \epsilon_k c_p + \epsilon_k c_p - e_p\| \\ &=\; \|\Delta\epsilon c_p + \epsilon_k c_p - e_p\| \\ &\leq\; \|\Delta\epsilon c_p\| + \|\epsilon_k c_p - e_p\|. \end{aligned}$$

Substituindo (2.12) na expressão acima temos

$$\|\Delta e_p\| \geq 0.1 - 0.015 = 0.085,$$

o que implica que

$$\Delta\epsilon \geq \frac{0.085}{\|c_p\|}. \tag{2.13}$$

Resta-nos agora encontrar uma expressão que estabeleça um limitante superior para $\|c_p\|$. Observe que de (2.12) podemos ter que

$$\|\epsilon_k c_p\| - \|e_p\| \leq 0.015,$$

como $\|c_p\| \leq \|e\| = \sqrt{2n}$ então

$$\|c_p\| \leq \frac{0.015 + \sqrt{2n}}{\epsilon_k}.$$

Substituindo em (2.13) temos

$$\Delta\epsilon \geq \frac{0.085\epsilon_k}{0.015 + \sqrt{2n}},$$

como $\sqrt{2n} \geq 1$, então

$$\Delta\epsilon \geq \frac{0.085\epsilon_k}{0.015\sqrt{2n} + \sqrt{2n}},$$

o que implica que

$$\Delta\epsilon \geq 0.08\frac{\epsilon_k}{\sqrt{2n}},$$

como queríamos provar. □

Lema 2.15 (Gonzaga [20]) Considere uma aplicação do algoritmo de passos curtos. A partir de $\hat{x}^0 \in \hat{S}^0$ e $\epsilon_0 \geq 2^{-L}$ tais que $\delta(\hat{x}^0, \epsilon_0) \leq 0.015$, o algoritmo termina no máximo em $O(\sqrt{n}\ L)$ iterações.

Demonstração. Pelo Lema 2.14 temos que

$$\epsilon_{k+1} \geq \epsilon_k(1 + \frac{0.08}{\sqrt{2n}}).$$

Após K iterações teremos

$$\epsilon_K \geq \epsilon_0(1 + \frac{0.08}{\sqrt{2n}})^K,$$

donde segue-se que

$$\begin{aligned} \lg \epsilon_K &\geq \lg \epsilon_0(1 + \frac{0.08}{\sqrt{2n}})^K \\ &= \lg \epsilon_0 + K \lg(1 + \frac{0.08}{\sqrt{2n}}). \end{aligned} \tag{2.14}$$

Considere agora o lema 2.2. Para valores de $\lambda \in [0, 1]$ vale a seguinte expressão

$$\lg\ (1 + \lambda) \geq 0.9\lambda.$$

Fixando $\lambda = 0.08/\sqrt{2n} < 1$, pois $\sqrt{2n} \geq 1$, segue-se que

$$\lg(1 + \frac{0.08}{\sqrt{2n}}) \geq \frac{0.07}{\sqrt{2n}}.$$

Substituindo a expressão acima em (2.14) tem-se que

$$\lg \epsilon_K \geq \lg \epsilon_0 + \frac{0.07}{\sqrt{2n}}K,$$

o que implica que

$$\lg \epsilon_K \geq -L + \frac{0.07}{\sqrt{2n}} K,$$

pois, por hipótese $\epsilon_0 \geq 2^{-L}$. Fazendo

$$K = \frac{\sqrt{2n}}{0.07}(2L + 1 + \lg(2n)) = O(\sqrt{n}L),$$

segue-se que

$$\lg \epsilon_k \geq L + \lg(2n) + 1,$$

o que implica que

$$\epsilon_k \geq 2(2n)2^L.$$

Completando a demonstração. □

A complexidade do número de operações é obtida observando que a operação de maior ordem de complexidade, a cada iteração, corresponde à resolução de um sistema linear, que no caso geral é feito em $O(m^3)$ operações. Como são executadas $O(\sqrt{n}\ L)$ iterações, conclui-se que o algoritmo é da ordem de $O(m^3 n^{0.5} L)$ operações. Para o problema de fluxo em redes é possível reduzir a complexidade do número de operações usando o método do gradiente conjugado na resolução do sistema linear. Veremos no capítulo seguinte que o sistema linear é resolvido em $O(mn)$ operações, portanto, a complexidade do algoritmo para o problema que estamos lidando é de

- Complexidade de $O(\sqrt{n}\ L)$ iterações.
- Complexidade de $O(mn^{1.5}L)$ operações.

2.5 O Algoritmo de Barreira de Passos Longos

A idéia deste algoritmo é semelhante a do algoritmo de passos curtos; a cada iteração é fixado um novo valor para o parâmetro de penalidade e em seguida

aplica-se o método de Newton-Raphson para minimizar a função penalizada definida pelo novo ϵ. Essencialmente este algoritmo difere do anterior por permitir variações grandes no parâmetro ϵ, de modo que $\delta(x^k, \epsilon_{k+1})$ ultrapasse o valor 0.1. Isto faz com que sejam aplicadas várias iterações de Newton-Raphson até que se encontre $\hat{x}^{k+1} \simeq \arg\min\{f_{k+1}(\hat{x}) | \hat{x} \in \hat{S}^0\}$, Figura 2.5.

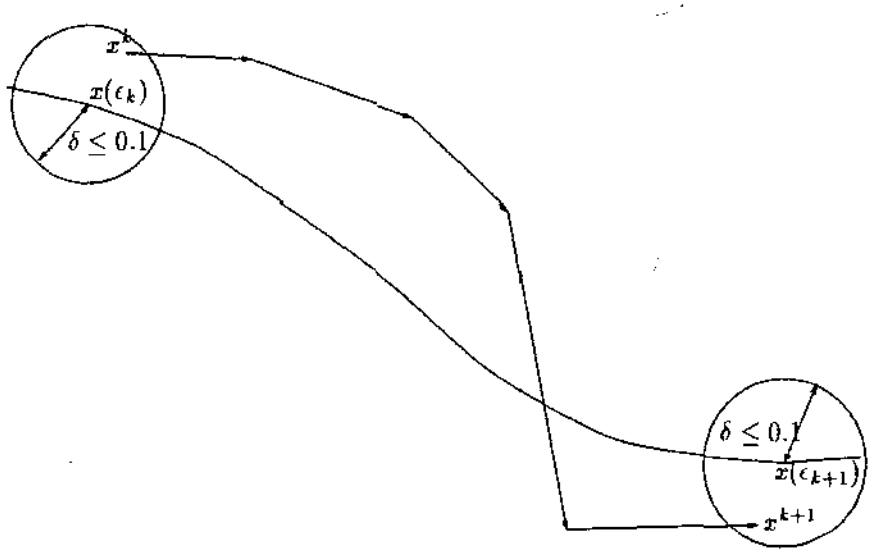


Figura 2.5: Iteração do algoritmo de passos longos

Em [20], Gonzaga define que a variação de $\Delta\epsilon$ é proporcional ao valor de ϵ_k, onde a constante de proporcionalidade é fixada no início do algoritmo. Definimos $\Delta\epsilon$ como

$$\Delta\epsilon = \frac{\nu}{\sqrt{2n}}\epsilon_k,$$

Como veremos a seguir esta formulação para $\Delta\epsilon$ será muito útil no estudo da complexidade do algoritmo. Em [20] o valor de ν é constante e deve estar no intervalo $(1, 2n)$.

O algoritmo de passos longos é então definido como:

Algoritmo 2.5 Algoritmo de Passos Longos
Dados $\hat{x} \in \hat{S}^0, \epsilon > 0$, tal que $\delta(\hat{x}, \epsilon) \leq 0.015$ e $\nu \in \Re$, tal que $1 < \nu < 2n$

repita

$\epsilon = \epsilon(1 + \frac{\nu}{\sqrt{2n}})$

enquanto $\delta(\hat{x}, \epsilon) > 0.1$

$\hat{X} = \text{diag}(\hat{x}_i)$

Resolver $(\hat{A}\hat{X}^2\hat{A}^t)y = \hat{A}(\epsilon\hat{X}^2\hat{c} - \hat{x})$

$h_N = -(\hat{X}^2\hat{c} - \hat{x} - \hat{X}^2A^ty)$

Resolver aproximadamente $\bar{\lambda} = \arg\min\{f_\epsilon(\hat{x} + \lambda h) | \lambda > 0\}$

$\hat{x} = \hat{x} + \bar{\lambda}h_N$

até convergir

2.5.1 Inicialização

A inicialização é feita conforme a apresentada no algoritmo de passos curtos. Usa-se o algoritmo de Vanderbei para encontrar um ponto inicial factível e em seguida aplica-se o algoritmo 2.4 para encontrar $\hat{x}^0$ tal que $\delta(\hat{x}^0, \epsilon_0) \leq 0.015$.

2.5.2 Tamanho do Passo

Assim como no algoritmo de centralização, a determinação do tamanho do passo é um procedimento de $O(1)$. Nos valemos aqui das mesmas propriedades do lema 2.10 usadas no algoritmo 2.4. De modo que $\bar{\lambda}$ é definido como

$$\begin{aligned} \bar{\lambda} &= 0.5 && \text{se } \delta(x, \epsilon) > 1. \\ \bar{\lambda} &= 0.5\delta(x, \epsilon) && \text{se } 0.1 \leq \delta(x, \epsilon) \leq 1. \\ \bar{\lambda} &= \delta(x, \epsilon) && \text{se } \delta(x, \epsilon) < 0.1. \end{aligned}$$

2.5.3 Complexidade

A análise da complexidade será feita observando o decréscimo da função penalizada a cada iteração do *loop* maior, isto é, a cada vez que se obtém $\hat{x}^k$ tal que $\delta(\hat{x}^k, \epsilon_k) \leq 0.1$. Para tanto faz-se necessário provar que o número de iterações do *loop* interno é limitado por uma constante.

O lema 2.10 apresenta uma condição necessária para a prova de que a variação de $f_\epsilon(\cdot)$ entre os pontos $\hat{x}^k$ e $\hat{x}^{k+1}$ está contida em um limite conhecido. O próximo lema deduz esta variação, o que será utilizado na prova do

resultado mais importante deste estudo de complexidade, anunciado no lema 2.17.

Lema 2.16 (Gonzaga [20]) Sejam x e $\bar{x} \in \Re^{2n}$, pontos gerados pelo algoritmo 2.5 e ϵ e $\bar{\epsilon} \in \Re$ parâmetros de penalidade tais que $\delta(x, c) = \alpha \leq 0.1$ e $\delta(\bar{x}, \bar{c}) \leq 0.1$, sendo $\bar{\epsilon} = (1 + \frac{\nu}{\sqrt{2n}})\epsilon$, e $1 \leq \nu \leq 2n$. Temos que

$$f_{\bar{\epsilon}}(x) \leq f_{\bar{\epsilon}}(x(\epsilon)) + 0.15\nu. \tag{2.15}$$

Demonstração. Suponha sem perda de generalidade que $x = e$. A demonstração é feita usando o fato de que $f_\epsilon(\cdot)$ é convexa, pois a matriz Hessiana de $f(\cdot)$ é definida positiva. Temos que

$$\begin{aligned} f_{\bar{\epsilon}}(x(\epsilon)) &\geq f_{\bar{\epsilon}}(e) + \nabla f_{\bar{\epsilon}}(e)^t(x(\epsilon) - e) \\ &= f_{\bar{\epsilon}}(e) + \frac{\nu}{\sqrt{n}}\epsilon\hat{c}^t(x(\epsilon) - c) + \nabla f_c(e)^t(x(\epsilon) - e), \end{aligned}$$

pois $\nabla f_{\bar{\epsilon}}(e) = \frac{\nu}{\sqrt{n}}\epsilon + \nabla f_\epsilon(e)$.

Isolando $f_{\bar{\epsilon}}(e)$ segue-se que

$$f_{\bar{\epsilon}}(e) \leq f_{\bar{\epsilon}}(x(\epsilon)) - \frac{\nu}{\sqrt{2n}}\epsilon\hat{c}^t(x(\epsilon) - e) - \nabla f_\epsilon(e)^t(x(\epsilon) - e). \tag{2.16}$$

Usando o lema 2.10

$$\begin{aligned} f_{\bar{\epsilon}}(e) &\leq f_{\bar{\epsilon}}(x(\epsilon)) - \nabla f_{\bar{\epsilon}}(e)^t(x(\epsilon) - e) + \frac{\nu}{\sqrt{2n}}1.1\alpha\sqrt{2n} \\ &= f_{\bar{\epsilon}}(x(c)) - \nabla f_{\bar{\epsilon}}(e)^t(x(\epsilon) - e) + 0.11\nu. \end{aligned}$$

Como por hipótese $\|P_{\bar{A}}\nabla f_\epsilon(e)\| = \delta(e, c) \leq 0.1$, e, pelo lema 2.12 $\|x(\epsilon) - e\| \leq 0.115$, então,

$$\begin{aligned} -(\nabla f_\epsilon(e))^t(x(\epsilon) - e) &= -(P_{\bar{A}}\nabla f_\epsilon(e))^t(x(\epsilon) - e) \\ &\leq -\|P_{\bar{A}}\nabla f_\epsilon(e)\| \, \|x(\epsilon) - e\| \\ &\leq 0.115 \times 0.1 = 0.0115. \end{aligned}$$

Assim, concluímos que

$$f_{\bar{\epsilon}}(e) \leq f_{\bar{\epsilon}}(x(\epsilon)) + 0.0115 + 0.11\nu,$$

e como $\nu \geq 1$ segue-se que

$$f_{\bar{\epsilon}}(e) \leq f_{\bar{\epsilon}}(x(\epsilon)) + 0.15\nu.$$

Como queríamos provar. □

Lema 2.17 (Gonzaga [20]) Seja J o número de passos do *loop* interno do algoritmo 2.5. Então

$$J \leq 400\nu(\nu + 1).$$

Demonstração. Chamemos de y^j os pontos gerados pelo *loop* interno do algoritmo 2.5 a partir da iteração k, portanto, pontos em que $\delta(y^j, \epsilon_{k+1}) > 0.1$ com exceção do último.

Como usamos o procedimento de busca linear descrito no lema 2.10 então,

$$f_{\epsilon_{k+1}}(y^{j+1}) \leq f_{\epsilon_{k+1}}(y^j) - \Delta,$$

com $\Delta = 0.0025$, pois por hipótese $\delta(y^j, \epsilon_{k+1}) > 0.1$. Na J-ésima iteração, quando finaliza-se o *loop* interno, teremos

$$f_{\epsilon_{k+1}}(\hat{x}^{k+1}) \leq f_{\epsilon_{k+1}}(\hat{x}^k) - J\Delta,$$

com $\hat{x}^{k+1} = y^J$. Usando (2.15) segue-se que

$$f_{\epsilon_{k+1}}(\hat{x}^{k+1}) \leq f_{\epsilon_{k+1}}(x(\epsilon_k)) - J\Delta + 0.15\nu,$$

mas, por definição, $f_\epsilon(\hat{x}) \geq f_\epsilon(x(\epsilon))$, então, subtraindo esta desigualdade da anterior, segue-se que

$$f_{\epsilon_{k+1}}(\hat{x}^{k+1}) - f_{\epsilon_k}(\hat{x}^{k+1}) \leq f_{\epsilon_{k+1}}(x(\epsilon_{k+1})) - f_{\epsilon_k}(x(\epsilon_k)) - J\Delta + 0.15\nu,$$

daí segue-se que

$$\frac{\nu}{\sqrt{2n}}\epsilon_k \hat{c}^t \hat{x}^{k+1} \leq \frac{\nu}{\sqrt{2n}}\epsilon_k \hat{c}^t x(\epsilon) - J\Delta + 0.15\nu, \tag{2.17}$$

pois $f_{\epsilon_{k+1}}(z) - f_{\epsilon_k}(z) = \frac{\nu}{\sqrt{2n}}\epsilon\hat{c}^t z$, para $\forall z \in \Re^{2n}_{++}$. Multiplicando (2.17) por $\frac{\sqrt{2n}}{\nu}$, chegamos a

$$\epsilon_k \hat{c}^t \hat{x}^{k+1} \leq \epsilon_k \hat{c}^t x(\epsilon_k) - \frac{\sqrt{2n}}{\nu}(J\Delta - 0.15\nu),$$

Pelo lema 2.9,

$$v(\epsilon_{k+1}) > v(\epsilon_k),$$

com $v(\epsilon) = \hat{c}^t x(\epsilon) - \frac{2n}{\epsilon}$, então

$$\begin{aligned}
\epsilon_k(\hat{c}^t \hat{x}^{k+1} - v(\epsilon_{k+1})) &\leq \epsilon_k \hat{c}^t(x(\epsilon_k) - v(\epsilon_k)) - \frac{\sqrt{2n}}{\nu}(J\Delta - 0.15\nu) \\
&= \epsilon_k(\hat{c}^t x(\epsilon_k) - \hat{c}^t x(\epsilon_k) + \frac{2n}{\epsilon_k}) - \frac{\sqrt{2n}}{\nu}(J\Delta - 0.15\nu) \\
&= 2n - \frac{\sqrt{2n}}{\nu}(J\Delta - 0.15\nu) \\
&= 2n(1 - \frac{1}{\nu\sqrt{2n}}(J\Delta - 0.15\nu)).
\end{aligned}$$

Multiplicando por $\frac{\epsilon_{k+1}}{\epsilon_k} = (1 + \frac{\nu}{\sqrt{2n}})$, obtemos

$$\epsilon_{k+1}(\hat{c}^t \hat{x}^{k+1} - v(\epsilon_{k+1})) \leq 2n(1 + \frac{\nu}{\sqrt{2n}})(1 - \frac{1}{\nu\sqrt{2n}}(J\Delta - 0.15\nu)).$$

Observe que $c_{k+1}(\hat{c}^t\hat{x}^{k+1} - v(c_{k+1})) = c_{k+1}(\hat{c}^t\hat{x}^k - \hat{c}^t\hat{x}^k(\epsilon_{k+1}) + \frac{2n}{c_{k+1}}) = c_{k+1}(\hat{c}^t\hat{x}^{k+1} - \hat{c}^t x(\epsilon_{k+1})) + 2n$, daí segue-se que

$$c_{k+1}(\hat{c}^t\hat{x}^{k+1} - \hat{c}^t\hat{x}(\epsilon_{k+1})) \leq$$
$$2n(1 + \frac{\nu}{\sqrt{2n}})(1 - \frac{1}{\nu\sqrt{2n}}(J\Delta - 0.15\nu)) - 2n.$$

Usando o lema 2.12, segue-se que

$$-0.11\sqrt{2n} \leq c_{k+1}(\hat{c}^t\hat{x}^{k+1} - \hat{c}^t x(\epsilon_{k+1}))$$

$$\leq 2n\left(1 + \frac{\nu}{\sqrt{2n}}\right)\left(1 - \frac{1}{\nu\sqrt{2n}}(J\Delta - 0.15\nu)\right) - 2n,$$

portanto

$$2n - 0.11\sqrt{2n} \leq 2n\left(1 + \frac{\nu}{\sqrt{2n}}\right)\left(1 - \frac{(J\Delta - 0.15\nu)}{\nu\sqrt{2n}}\right),$$

segue-se então que

$$1 - \frac{0.11}{\sqrt{2n}} \leq \left(1 + \frac{\nu}{\sqrt{2n}} - \frac{(J\Delta - 0.15\nu)}{\nu\sqrt{2n}} - \frac{(J\Delta - 0.15\nu)\nu}{\nu 2n}\right),$$

o que implica que

$$\nu 2n - 0.11\nu\sqrt{2n}$$
$$\leq \nu 2n + \nu^2\sqrt{2n} - (J\Delta - 0.15\nu)(\sqrt{2n} + \nu),$$

logo

$$\Rightarrow J\Delta - 0.15\nu \leq \frac{\nu\sqrt{2n}\,(\nu + 0.11)}{\sqrt{2n} + \nu}.$$

Como $\sqrt{2n} + \nu > \sqrt{2n}$, então

$$J\Delta - 0.15\nu \leq \frac{\nu\sqrt{2n}\,(\nu + 0.11)}{\sqrt{2n}},$$

o que implica que

$$J \leq \frac{1}{\Delta}\nu(\nu + 1).$$

Substituindo Δ por 0.0025 chegamos a

$$J \leq 400\nu(1 + \nu),$$

como queríamos provar. □

Por fim, a prova da complexidade geral do algoritmo é feita de forma semelhante à do algoritmo de passos curtos.

Lema 2.18 (Gonzaga [20]) Considere uma aplicação do algoritmo de passos longos a partir de uma aproximação inicial $\hat{x}^0 \in \hat{S}^0$ e $\epsilon_0 \geq 2^{-L}$ tais que $\delta(\hat{x}^0, \epsilon_0) \leq 0.015$, e um parâmetro, ν, $1 \leq \nu \leq 2n$. O algoritmo termina em $O(\sqrt{n}\,L)$ iterações.

Demonstração. Por hipótese temos

$$\epsilon_{k+1} \geq \epsilon_k(1 + \frac{\nu}{\sqrt{2n}}),$$

após K iterações do *loop* externo teremos

$$\epsilon_K \geq \epsilon_0(1 + \frac{\nu}{\sqrt{2n}})^K.$$

Tirando o logaritmo em ambos os lados teremos

$$\lg\, \epsilon_K = \lg\, \epsilon_0 + K \lg\,(1 + \frac{\nu}{\sqrt{2n}}).$$

Pelo lema 2.2, fazendo $\lambda = \frac{\nu}{\sqrt{2n}}$, temos que

$$\lg\ (1 + \frac{\nu}{\sqrt{2n}}) \ \geq \ 0.9\frac{\nu}{\sqrt{2n}}.$$

Assim

$$\begin{aligned} \lg\ \epsilon_k \ &\geq \ \lg\ \epsilon_0 + K\frac{0.9\nu}{\sqrt{2n}} \\ &\geq \ -L + K\frac{0.9\nu}{\sqrt{2n}}, \end{aligned}$$

pois $\epsilon_0 > 2^{-L}$.

Se fizermos

$$K \ = \ \frac{\sqrt{2n}}{0.9\nu}(2L + 1 + \lg\ 2n),$$

teremos

$$\lg\ \epsilon_k \ \geq \ L + \lg\ 2n + 1,$$

portanto,

$$\epsilon_k \ \geq \ 4n2^L,$$

completando assim a demonstração. □

O lema acima prova a complexidade polinomial do número de iterações e, conseqüentemente, a complexidade polinomial do número de operações, tendo em vista que o *loop* interno está limitado por uma constante, o que implica que, como no caso do algoritmo de passos curtos, o número de operações aritméticas é da ordem de $O(mn)$ a cada iteração, e um número total de $O(mn^{1.5}L)$ operações em todo algoritmo.

2.6 Algoritmos Primais Duais

Nesta seção estudaremos um algoritmo primal implementado por Lustig, Marsten e Shanno [27] para o programa linear (P), cujo dual é

$$\begin{array}{ll} \max & b^t y - w^t u \\ s.a. & A^t y - w + z = c \quad (D) \\ & z, w \geq 0. \end{array}$$

As condições de não-negatividade são eliminadas pelo uso de uma função barreira logaritmica, que em [27] é definida como

$$p(x) = -\mu \sum_{j=1}^{n} \ln\ x_j - \mu \sum_{j=1}^{n} \ln\ s_j,$$

onde μ é um parâmetro de centralização. Nosso problema então torna-se

$$\begin{array}{ll} \min & c^t x - p(x) \\ s.a. & Ax = b \\ & x + s = u \\ & x, s > 0. \end{array}$$

O lagrangeano deste problema é

$$\begin{aligned} L(x,s,y,w) = {} & c^t x - \mu \sum_{j=1}^{n} \ln\ x_j - \mu \sum_{j=1}^{n} \ln\ s_j \\ & -y^t(Ax - b) - w^t(x + s - u). \end{aligned} \tag{2.18}$$

Assim, as condições de otimalidade de primeira ordem de (2.18) são.

$$\left\{ \begin{array}{lcl} Ax - b & = & 0, \\ x + s - u & = & 0, \\ c - A^t y - w + \mu X^{-1} e & = & 0, \\ \mu S^{-1} e - w & = & 0, \end{array} \right. \tag{2.19}$$

onde $X = diag(x_1, \ldots, x_n)$ e $S = diag(s_1, \ldots, s_n)$. A partir de (2.19), definimos uma função $F(x,s,y,w,z)$, descrita como

$$F(x,s,y,w,z) \equiv \begin{pmatrix} Ax-b \\ x+s-u \\ A^t y - w + z - c \\ XZe - \mu e \\ SWe - \mu e \end{pmatrix},$$

onde $W = diag(w_1, \ldots, w_n)$ e $Z = diag(z_1, \ldots, z_n)$.

O algoritmo que apresentamos busca encontrar os zeros da função F usando o método de Newton, o que equivale a igualar o gradiente de (2.18) a zero e portanto obter as condições de otimalidade de primeira ordem. Este algoritmo parte de uma solução inicial $(x^0, s^0, y^0, w^0, z^0)$ interior primal e dual factível, isto é, $(x^0, s^0, y^0, w^0, z^0)$ satisfazem a (P) e (D) e $(x^0, s^0, y^0, w^0, z^0) > 0$. A cada iteração determina-se um novo valor para o parâmetro μ e aplica-se o método de Newton para F definida a partir deste novo valor.

O passo de Newton definido para resolver F é dado por

$$J(F(x,s,y,w,z))\Delta(x,s,y,w,z) = -F(x,s,y,w,z),$$

onde $J(F(x,s,y,w,z))$ é a matriz Jacobiana da função $F(x,s,y,w,z)$. Temos então o seguinte sistema

$$\begin{bmatrix} A & 0 & 0 & 0 & 0 \\ I & I & 0 & 0 & 0 \\ 0 & 0 & A^t & -I & I \\ Z & 0 & 0 & 0 & X \\ 0 & W & 0 & S & 0 \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta s \\ \Delta y \\ \Delta w \\ \Delta z \end{bmatrix} = \begin{bmatrix} b - Ax \\ u - x - s \\ -A^t y + w - z + c \\ \mu e - XZe \\ \mu e - WSe \end{bmatrix}.$$

O que equivale ao sistema

$$\begin{cases} A\Delta x = b - Ax, \\ \Delta x + \Delta s = u - x - s, \\ A^t \Delta y - \Delta w + \Delta z = -A^t y + w - z + c, \\ Z\Delta x + X\Delta z = \mu e - XZe, \\ W\Delta s + S\Delta w = \mu e - WSe. \end{cases}$$

Feitas algumas manipulações algébricas chegamos aos seguintes valores para $\Delta x, \Delta s, \Delta y, \Delta w$ e Δz.

$$
\begin{aligned}
\Delta y &= (A\Theta A^t)^{-1} A\Theta\rho(\mu), \\
\Delta x &= \Theta(A^t\Delta y - \rho(\mu)), \\
\Delta z &= \mu X^{-1}e - Ze - X^{-1}Z\Delta x, \\
\Delta w &= \mu S^{-1}e - We + S^{-1}W\Delta x, \\
\Delta s &= -\Delta x,
\end{aligned}
$$

onde $\Theta = (S^{-1}W + X^{-1}Z)^{-1}$ e $\rho(\mu) = \mu(S^{-1} - X^{-1})e - (W - Z)e$.

Neste algoritmo primal dual define-se um escalar α^k para determinar o tamanho do passo a cada iteração, a fim de garantir que x^{k+1} continue no interior da região factível. Em [27] Lustig et. al. definem α^k como

$$
\begin{aligned}
\alpha^k &= \min(1, 0.9995\hat{\alpha}^k), \\
\hat{\alpha}^k &= \frac{-1}{\min((X^k)^{-1}\Delta x^k, (S^k)^{-1}\Delta s^k, (W^k)^{-1}\Delta w^k, (Z^k)^{-1}\Delta z^k)}.
\end{aligned}
$$

O Parâmetro μ O parâmetro de penalização, μ, é usado para forçar as variáveis primais e duais para valores positivos, corrigindo a direção de descida para uma combinação linear entre a direção afim e de centralização. A escolha adequada deste parâmetro a cada iteração assegura a complexidade em tempo polinomial do método. Primeiramente, os resultados da convergência em tempo polinomial em função de μ foram provados para algoritmos primais e em seguida estendidos para algoritmos primais duais por Monteiro e Adler [30]. Neste último artigo, o valor de μ decresce de uma quantidade muito pequena a cada iteração, o que torna o algoritmo ineficiente do ponto-de-vista prático. Bons resultados computacionais foram obtidos quando a sequência (μ^k) tende a zero rapidamente. Em [27] Lustig et.al. escolheram μ como

$$
\mu = (c^t x - b^t y)/\varphi(n),
$$

onde

$$
\varphi(n) = \begin{cases} n^2 & n \leq 5000, \\ n\sqrt{n} & n > 5000. \end{cases}
$$

Algoritmo 2.6 Algoritmo Primal Dual
Dados: $0 < x < u$, $w > 0$, $z > 0$ e $y \in \Re^m$, tais que $Ax = b$, e $A^t y - w + z = c$,

Se $n < 5000$ **Então** $\varphi(n) = n^2$ **Senão** $\varphi(n) = n\sqrt{n}$
Repita
$X = diag(x_i)$
$Z = diag(z_i)$
$W = diag(w_i)$
$S = diag(u_i - x_i)$
$\Theta = (WS^{-1} + ZX^{-1})^{-1}$
$\mu = (c^t x - b^t y)/\varphi(n)$
$\rho(\mu) = \mu(S^{-1} - X^{-1})e - (W - Z)e$
Resolver $(A\Theta A^t)\Delta y = A\Theta\rho(\mu)$
$\Delta x = \Theta(A^t\Delta y - \rho(\mu))$
$\Delta z = \mu X^{-1}e - Ze - X^{-1}Z\Delta x$
$\Delta w = \mu S^{-1}e - We + S^{-1}W\Delta x$
$\hat{\alpha}^k = 1/\min\{X^{-1}\Delta x, W^{-1}\Delta w, Z^{-1}\Delta z, -S^{-1}\Delta x\}$
$\alpha^k = \min\{1, 0.9995\hat{\alpha}^k\}$
$x = x + \alpha^k\Delta x$
$y = y + \alpha^k\Delta y$
$z = z + \alpha^k\Delta z$
$w = w + \alpha^k\Delta w$
Até convergir

Critério de Parada Em [27] Lustig et. al. estabelecem como critério de parada o tamanho do gap dual relativo. O algoritmo pára quando

$$\frac{c^t x - b^t y + u^t w}{1 + |b^t y - u^t w|} < \epsilon.$$

2.6.1 Inicialização

A idéia aqui é acrescentar ao problema original variáveis artificiais com um peso elevado na função objetivo, como se faz no método tradicionalmente conhecido como big-M. O problema, com o acréscimo destas variáveis, torna-se

$$
\begin{array}{lrcll}
\min & c^t x + M_P x_{n+1} & & & \\
s/a & Ax + (b - Ax^0)x_{n+1} & = & b & \\
& x + s & = & u & (P') \\
& (A^t y^0 + z^0 - w^0 - c)^t x + x_{n+2} & = & M_D & \\
& x, s, x_{n+1}, x_{n+2} & \geq & 0, &
\end{array}
$$

e seu respectivo dual

$$
\begin{array}{lrcll}
\min & b^t y - u^t w + M_D y_{m+1} & & & \\
s/a & A^t y + (A^t y^0 + z^0 - w^0 - c)y_{m+1} + z - w & = & c & \\
& (b - Ax^0)^t y + z_{n+1} & = & M_P & (D') \\
& y_{m+1} + z_{n+2} & = & 0 & \\
& z, z_{n+1}, z_{n+2}, w & \geq & 0, &
\end{array}
$$

onde x^0, s^0, y^0, z^0, w^0 representam a aproximação inicial e x_{n+1}, x_{n+2},y_{m+1},z_{n+1} e $z_{n+2} \in \Re$ são variáveis artificiais. É fácil notar que para valores muito grandes de M_P e M_D a solução ótima de (P') e (D') é a mesma de (P) e (D).

Para efeito de simplicidade, denotaremos por d_P e d_D os vetores $b - Ax^0$ e $A^t y^0 - w^0 + z^0 - c$, respectivamente. Também, vamos nos referir a M_P e M_D genericamente por M. Em [26], Lustig provou que as direções de descida existem e podem ser calculadas quando o limite de M é infinito, com os seguintes valores para estas direções

$$
\begin{aligned}
\Delta y &= (A\Theta A^t)^{-1}[A\Theta(\rho(\mu) - z_{n+2}d_D) + x_{n+1}d_P], \\
\Delta x &= \Theta[A^t \Delta y - \rho(\mu) + z_{n+2}d_D], \\
\Delta z &= \mu X^{-1}e - Ze - X^{-1}Z\Delta x, \\
\Delta w &= \mu S^{-1}e - We - S^{-1}W\Delta x, \\
\Delta s &= -\Delta x,
\end{aligned}
$$

com $x_{n+1} = (b - Ax)/d_P$ e $z_{n+2} = (A^t y + w - z - c)/d_D$.

O Parâmetro μ A escolha do parâmetro μ para o caso em que não há factibilidade é definida segundo Lustig [26] como

$$
\mu = (c^t x + Mx_{n+1} - b^t y - My_{m+1})/\varphi(n),
$$

com $\varphi(n)$ definido como no algoritmo anterior e

$$M = \rho\varphi(n)\max\{\max\{|c_j|\},\max\{|b_i|\}\},$$

e ρ é um escalar que faz variar a aproximação inicial do valor de μ. Pode-se ver intuitivamente que quanto menor o valor de ρ, conseqüentemente menor é o valor inicial de μ, o que faz a direção de descida se aproximar mais da direção de máximo declive.

2.7 Conclusões

Neste capítulo fizemos a apresentação dos algoritmos como eles se encontram na literatura. Nesta apresentação, o aspecto mais importante que mencionamos diz respeito à ordem de complexidade; dissemos que, por tratar-se de um problema de fluxo em redes, o número de operações aritméticas envolvidas é da ordem $O(mn^{1.5}L)$ operações, o que representa um ganho sensível em relação à complexidade dos mesmos algoritmos para PL, que é da ordem $O(m^3n^{0.5}L)$. Detalharemos no capítulo seguinte os pontos que podem ser melhorados nestes algoritmos e que conduzem a este resultado.

Capítulo 3

Resolução de Sistemas Lineares

3.1 Introdução

Neste capítulo detalharemos alguns pontos em aberto deixados no capítulo anterior. Nos deteremos na resolução do sistema linear $A\Psi^2 A^t y = z$, comum em todos os algoritmos de pontos interiores, apenas com o vetor z e a matriz Ψ variando conforme o método. Estudaremos aqui técnicas de resolução de sistemas lineares com as características da matriz $A\Psi^2 A^t$.

Na seção 3.2 veremos um modo de simplificar os cálculos envolvidos na resolução do sistema $\hat{A}\hat{X}\hat{A}^t y = z$ que aparece nos algoritmos de passos longos e passos curtos. Este é um sistema de ordem $m+n$ para problemas de fluxo em redes. Veremos nesta seção que é possível reduzir o volume de contas envolvidas na resolução do mesmo, reduzindo-o a um sistema de ordem m. Também veremos que nos algoritmos de Gonzaga é possivel tratarmos as iterações, x^k, fazendo manipulaçôes com vetores em $\Re^n$ ou $\Re^m$. Na seção 3.3 estudaremos algumas particularidades da matriz de projeção que são de fundamental importância na resolução deste sistema. Os algoritmos de resolução de sistemas lineares são descritos na seção 3.4 onde é feita uma breve descrição do fundamento teórico e são traçados comentários citando vantagens e desvantagens de cada um.

3.2 Cálculos preliminares

3.2.1 Cálculo da Projeção sobre $\mathcal{N}(A)$

Em geral, as técnicas de resolução de um sistema linear de ordem m têm complexidade $O(m^3)$. Nos algoritmos 2.3, 2.4 e 2.5 o sistema $\hat{A}\hat{X}^2\hat{A}^t y = z$, como nos é apresentado, é de ordem $m+n$, onde m é o número de nós e n o número de arcos do grafo associado. Mostraremos que é possível obter a mesma solução resolvendo um sistema menor, cuja matriz é da ordem apenas do número de nós, m. Isto representa um ganho significativo, além do que pode ser feito de forma bastante simples.

Nos algoritmos apresentados no capítulo 2 existem três tipos de sistema de ordem $m+n$:

i)	$\hat{A}\hat{X}^2\hat{A}^t y = \hat{A}\hat{X}^2\hat{c}$		algoritmo de passos curtos.
ii)	$\hat{A}\hat{X}^2\hat{A}^t y = \hat{A}\hat{X}e$		algoritmo de passos curtos.
iii)	$\hat{A}\hat{X}^2\hat{A}^t y = \hat{A}\hat{X}^2(\epsilon\hat{c} - e)$		algoritmo de passos longos e de centralização.

i) Dividindo o sistema $\hat{A}\hat{X}^2\hat{A}^t y = \hat{A}\hat{X}^2\hat{c}$ em blocos obtemos

$$\begin{bmatrix} A & 0 \\ I & I \end{bmatrix}\begin{bmatrix} X^2 & 0 \\ 0 & S^2 \end{bmatrix}\begin{bmatrix} A^t & I \\ 0 & I \end{bmatrix}\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} A & 0 \\ I & I \end{bmatrix}\begin{bmatrix} X^2 & 0 \\ 0 & S^2 \end{bmatrix}\begin{bmatrix} c \\ 0 \end{bmatrix},$$

onde S é a matriz diagonal $S = diag(s_i) = diag(u_i - x_i)$. Assim temos que

$$\begin{bmatrix} AX^2A^t & AX^2 \\ X^2A^t & X^2 + S^2 \end{bmatrix}\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} AX^2 & 0 \\ X^2 & S^2 \end{bmatrix}\begin{bmatrix} c \\ 0 \end{bmatrix},$$

o que implica no sistema

$$\begin{cases} AX^2A^t y_1 + AX^2 y_2 = AX^2 c, \\ X^2A^t y_1 + (X^2 + S^2)y_2 = X^2 c. \end{cases} \tag{3.1}$$

Tirando o valor de y_2 na segunda equação encontramos

$$y_2 = (X^2 + S^2)^{-1}[X^2 c - X^2A^t y_1], \tag{3.2}$$

pois $(X^2 + S^2)$ é uma matriz diagonal com todos os elementos da diagonal principal não nulos, portanto, inversível.

Substituindo (3.2) em (3.1) temos

$$AX^2A^ty_1 + AX^2(X^2 + S^2)^{-1}[X^2c - X^2A^ty_1] = AX^2c,$$

daí segue-se que

$$AX^2A^ty_1 - AX^2(X^2 + S^2)^{-1}X^2A^ty_1 = AX^2c - AX^2(X^2 + S^2)^{-1}X^2c,$$

o que implica em

$$A[X^2 - X^2(X^2 + S^2)^{-1}X^2]A^ty_1 = A[X^2 - X^2(X^2 + S^2)^{-1}X^2]A^tc. \quad (3.3)$$

Observe que $X^2 - X^2(X^2 + S^2)^{-1}X^2$ é uma matriz diagonal cujos elementos da diagonal são $x_i^2s_i^2/(x_i^2 + s_i^2)$, para $i = 1 \ldots n$, assim temos que

$$X^2 - X^2(X^2 + S^2)^{-1}X^2 = D^2, \quad (3.4)$$

onde D é a matriz diagonal usada no método afim. Substituindo (3.4) em (3.3) temos

$$AD^2A^ty_1 = AD^2c.$$

A segunda parte do vetor y é encontrada fazendo os cálculos expressos em (3.2). O esforço computacional envolvido no cômputo de y_2 é de $O(mn)$ visto que já se possui o vetor y_1 e a inversão de $(X^2 + S^2)$ é imediata pois X e S são matrizes diagonais. Assim as únicas operações envolvidas são soma de vetores e multiplicação de matrizes por vetores.

ii) Para resolver $\hat{A}\hat{X}^2\hat{A}^ty = \hat{A}\hat{X}$ e usamos o mesmo raciocínio. Dividimos as matrizes em blocos e efetuando operações semelhantes chegamos ao sistema

$$\begin{aligned} AX^2A^ty_1 + AX^2y_2 &= AXc, \\ X^2A^ty_1 + (X^2+S^2)y_2 &= (X+S)c, \end{aligned} \tag{3.5}$$

onde y_1 e y_2 são, assim como no caso anterior, os sub-vetores de m coordenadas que compõe o vetor y. Tirando os valores de y_1 e y_2 do sistema acima chegamos a

$$\begin{cases} y_1 &= (AD^2A^t)^{-1}A[X - X^2(X^2+S^2)^{-1}(X+S)]e, \\ y_2 &= (X^2+S^2)^{-1}[(X+S)e - X^2A^ty_1], \end{cases}$$

ou simplificadamente

$$\begin{cases} y_1 &= (AD^2A^t)^{-1}AD'e, \\ y_2 &= (X^2+S^2)^{-1}[(X+S)e - X^2A^ty_1], \end{cases}$$

com D definida do mesmo modo que em (ii) e D' a matriz diagonal $D' = X - X^2(X^2+S^2)^{-1}(X+S) = diag((x_i^2s_i - x_is_i^2)/(x_i^2+s_i^2))$.

Assim como no caso anterior necessitamos resolver um sistema linear de ordem m para encontrar o valor de y_1. A segunda parte do vetor y é obtida substituindo y_1 em (3.5) e efetuando as operações da fórmula, o que requer um esforço computacional da ordem de $O(mn)$.

(iii) No último caso, para resolver o sistema $\hat{A}\hat{X}^2\hat{A}^t = \hat{A}\hat{X}(\hat{X}c - e)$, fazemos operações semelhantes às dos casos anteriores chegando ao seguinte sistema

$$\begin{aligned} y_1 &= (AD^2A^t)^{-1}[\epsilon AD^2c - AD'e], \\ y_2 &= (X^2+S^2)^{-1}[\epsilon X^2c - (X+S)e - X^2A^ty_1], \end{aligned} \tag{3.6}$$

com $D^2 = diag(x_i^2s_i^2/(x_i^2+s_i^2))$, e $D' = diag((x_i^2s_i - x_is_i^2)/(x_i^2+s_i^2))$,e y_1 e y_2 os dois subvetores resultantes da divisão do sistema original em blocos. Em resumo, o esforço total envolvido no cômputo de y nos casos (i), (ii) e (iii) é da ordem de $O(m^3 + mn) = O(m^3)$.

3.2.2 Cálculo da Direção $h_N(x, \epsilon)$

Mencionamos anteriormente que é possível considerar os pontos gerados pelos algoritmos de Gonzaga como vetores em $\Re^n$. De fato o vetor $\hat{x}$ que lidamos deriva de $x \in \Re^n$ por um cálculo simples e imediato, pois

$$\hat{x} = \begin{bmatrix} x \\ s \end{bmatrix} = \begin{bmatrix} x \\ u - x \end{bmatrix}.$$

Observe também que

$$\Delta\hat{x} = \begin{bmatrix} \Delta x \\ \Delta s \end{bmatrix} = \begin{bmatrix} \Delta x \\ -\Delta x \end{bmatrix}.$$

Isto nos motiva a determinar um meio de calcular apenas a direção Δx no vetor $\Delta\hat{x}$, visto que isto implicaria numa redução significativa do volume de cálculos e do espaço de memória alocado, pois passaríamos a tratar apenas com vetores em $\Re^n$.

Temos que

$$\begin{aligned} \Delta\hat{x} = \hat{X}h_N(\hat{x},\epsilon) &= -\hat{X}P_{\hat{A}\hat{X}}\hat{X}\bar{f}_\epsilon(\hat{x}) \\ &= -(\hat{X}^2(\epsilon\hat{c} - \hat{x}^{-1}) - \hat{X}^2(\hat{A}\hat{X}^2\hat{A}^t)^{-1}\hat{A}\hat{X}^2(\epsilon\hat{c} - \hat{x}^{-1})). \end{aligned}$$

Como vimos na seção anterior, o resultado de $(\hat{A}\hat{X}^2\hat{A}^t)^{-1}\hat{A}\hat{X}^2(\epsilon\hat{c} - \hat{x}^{-1})$ pode ser calculado como

$$(\hat{A}\hat{X}^2\hat{A}^t)^{-1}\hat{A}\hat{X}^2(\epsilon\hat{c} - \hat{x}^{-1}) = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix},$$

com y_1 e $y_2 \in \Re^n$ definidos como em (3.6). Assim temos que

$$\Delta\hat{x} = \begin{bmatrix} \Delta x \\ \Delta s \end{bmatrix} = -\begin{bmatrix} X & 0 \\ 0 & S \end{bmatrix} \left(\begin{bmatrix} X & 0 \\ 0 & S \end{bmatrix} \begin{bmatrix} \epsilon c - x^{-1} \\ -s^{-1} \end{bmatrix} - \begin{bmatrix} X & 0 \\ 0 & S \end{bmatrix} \begin{bmatrix} A^t & I \\ 0 & I \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} \right),$$

o que implica em

$$\Delta\hat{x} = \begin{bmatrix} \Delta x \\ \Delta s \end{bmatrix} = \begin{bmatrix} -\epsilon X^2 c + Xe + X^2 A^t y_1 + X^2 y_2 \\ Se + S^2 y_2 \end{bmatrix}.$$

Como só nos interessa calcular Δx, visto que $\Delta s = -\Delta x$, então, substituindo o valor de y_2 de (3.6) obtemos

$$\begin{aligned}\Delta x = & -\epsilon X^2 c + Xc + X^2 A^t y_1 \\ & + X^2(X^2 + S^2)^{-1}[\epsilon X^2 c - (X + S)e - X^2 A^t y_1],\end{aligned}$$

donde segue-se que

$$\begin{aligned}\Delta x &= -\epsilon[X^2 - X^2(X^2 + S^2)^{-1}X^2]c \\ &\quad +(X^2 - X^2(X^2 + S^2)^{-1}X^2]A^t y_1 \\ &\quad +[X - X^2(X + S)^{-1}(X + S)]e.\end{aligned}$$

De (3.4), substituindo $X^2 - X^2(X^2 + S^2)^{-1}X^2$ por D^2, e como $X - X^2(X + S)^{-1}(X + S) = D'$, então temos

$$\begin{aligned}\Delta x &= -\epsilon D^2 c + D'e + D^2 A^t y_1 \\ &= -\epsilon D^2 c + D'e + D^2 A^t (AD^2 A^t)^{-1}(\epsilon D^2 c - D'e).\end{aligned}$$

Se pusermos D^2 em evidência chegamos a

$$\Delta x = D^2(-\epsilon c + D'(D^2)^{-1}e) + D^2 A^t (AD^2 A^t)^{-1} D^2(\epsilon c - D'(D^2)^{-1}e),$$

efetuando algumas manipulações algébricas simples podemos escrever o termo $D'(D^2)^{-1}e$ como

$$D'(D^2)^{-1}e = s^{-1} - x^{-1},$$

assim, podemos simplificar a expressão que fornece Δx como

$$\Delta x = D^2(-\epsilon c - x^{-1} + s^{-1}) + D^2 A^t (AD^2 A^t)^{-1} D^2(\epsilon c + x^{-1} - s^{-1}).$$

Isto prova que $\Delta \hat{x}$ pode ser calculado apenas em função de vetores em $\Re^n$ e da matriz A.

3.3 A Matriz de Projeção

Como vimos, todos os métodos resolvem um sistema do tipo $A\Psi^2 A^t y = z$, sendo $\Psi = D$ ou $\Psi = \Theta$ a matriz diagonal definida nos algoritmos do capítulo anterior, e z o vetor a ser projetado sobre o núcleo de $A\Psi$. É importante, então, explorar as propriedades da matriz $A\Psi^2 A^t$ que venham a ser úteis na resolução deste sistema.

3.3.1 O Posto da Matriz de Incidência (Bazaraa [5])

Sabemos que cada linha da matriz de incidência de um grafo está associada a um nó e cada coluna a um arco. Os elementos de A são 0, -1 e 1 dispostos conforme a composição do grafo, como ilustrado na Figura 1.1. Observe que o posto desta matriz é claramente menor do que m pois $e^t A = 0$. Isto vale para toda matriz de incidência pois cada coluna possui exatamente dois elementos não nulos: $+1$, -1. Nesta seção provaremos, conforme o estudo desenvolvido em Bazaraa [5], que a matriz de incidência de um grafo conectado possui $m - 1$ vetores linearmente independentes, o que implicará que A tem posto $m - 1$. Para isto será preciso introduzir alguns conceitos de teoria dos grafos que facilitarão a apresentação desta prova.

Definição 3.1 Um *caminho* ou *cadeia* de um nó i_0 para i_p é um arco ou uma seqüência de arcos que une i_0 a i_p passando por nós intermediários distintos $\{i_1, \ldots, i_{p-1}\}$ como ilustra a Figura 3.1 (a).

Definição 3.2 Um *ciclo* é um caminho com dois ou mais arcos de um nó i_0 a i_p mais um arco que une i_p a i_0, o que está ilustrado na Figura 3.1(b).

Definição 3.3 Dizemos que um grafo é conectado quando existe um caminho para qualquer par de nós. Para o problema que estamos considerando todos os grafos são conectados.

Definição 3.4 Um grafo conectado é dito ser *próprio* se possui cardinalidades $|\mathcal{N}| \geq 2$ e $|\mathcal{A}| \geq 1$. Consideraremos também que no problema que estamos abordando os grafos sejam próprios.

Definição 3.5 A definição mais importante que apresentaremos neste contexto é a definição de árvore. Uma *árvore* é um grafo conectado sem nenhum

ciclo, Figura 3.1(c). dizemos que uma árvore T é *geradora* de um grafo G, se T for uma árvore contendo todos os nós de G, isto é, T é um subgrafo gerador de G que não possui ciclos.

Definição 3.6 Dizemos que um nó é *terminal* quando há apenas um ou nenhum arco incidente sobre este nó.

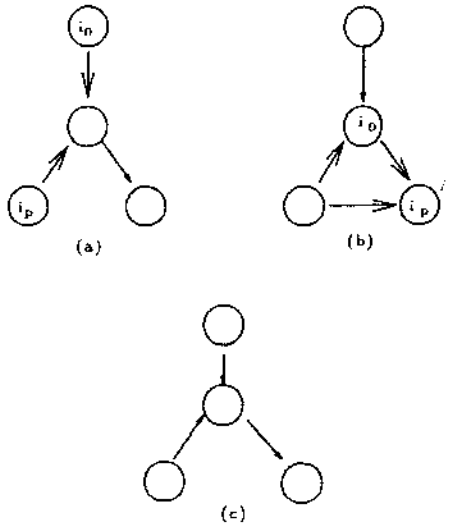


Figura 3.1: Exemplos de um caminho, ciclo e árvore

Proposição 3.1 Seja T uma árvore própria com $m \geq 2$ nós e (i, j) um arco de T. Então, desconectando (i, j) de T, isto é, removendo o arco (i, j) de T mas deixando os dois nós i e j em T, decompomos T em duas árvores T_1 e T_2.

Demonstração. Como T é uma árvore, ao remover o arco (i, j), desconecta-se T em duas partes T_1 e T_2, pois T não possui ciclos. Os grafos resultantes T_1 e T_2 são árvores pois como T originalmente não possuia ciclos, T_1 e T_2, que são subgrafos conectados de T, também não possuirão. □

Proposição 3.2 Uma árvore própria tem pelo menos dois nós terminais.

Demonstração. Para uma árvore com apenas dois nós a proposição é obviamente válida. Por indução, suponha que seja válido para uma árvore com $m-1$ nós. Provaremos que a proposição vale para uma árvore T com m nós ($m \geq 3$). Suponha que seja removido um arco (i,j) de T. Isto resultaria, pela proposição 3.1, na formação de duas árvores menores com no máximo $m-1$ nós. Pela hipótese de indução o somatório dos nós terminais das árvores resultantes seria igual a m, $m \geq 3$. No caso em que $m = 3$, quando uma das árvores é imprópria, ao colocar de volta o arco (i,j), perderia-se um dos nós terminais, o que implica que T possuiria no mínimo 2 nós terminais. Quando $m > 3$, caso em que nenhuma das árvores resultantes seriam impróprias, ao restabelecer o arco (i,j), no máximo dois nós terminais seriam perdidos, o que também implica que T possuiria no mínimo 2 nós terminais. □

Proposição 3.3 Uma árvore com m nós tem $m-1$ arcos.

Demonstração. Para uma árvore própria com 2 nós a proposição é obviamente válida. Provaremos por indução que o mesmo vale para árvores com mais de 2 nós. Assuma que seja válido para uma árvore com $m-1$ nós. Considerando T uma árvore com m nós, temos que, pela proposição 3.2, T possui um nó terminal. Desconectando de T o único arco incidente sobre este nó, teremos duas árvores T_1 e T_2, uma com um único nó e nenhum arco e outra com $m-1$ nós e, pela hipótese de indução, $m-2$ arcos. Assim o somatório dos arcos de T é $m-2+1 = m-1$, como queríamos provar. □

Proposição 3.4 Toda matriz de incidência associada a um grafo conectado próprio possui pelo menos $m-1$ colunas linearmente independentes.

Demonstração. Para provar que A possui $m-1$ colunas linearmente independentes, consideraremos T uma árvore geradora do grafo G, o que é sempre possível encontrar removendo de G os arcos que formam ciclos. Chamando A_T a submatriz de A associada à árvore geradora, tem-se pela proposição 3.3 que A_T é uma matriz de dimensão $m \times (m-1)$, pois T é uma árvore com m nós e, por ser uma árvore, possui $m-1$ arcos. Como T possui pelo menos um nó terminal, então existe uma linha em A_T em que todos os elementos são nulos, exceto aquele associado ao nó terminal. Fazendo uma permutação em A_T de modo a deixar este elemento na primeira posição, temos

$$A_T = \begin{bmatrix} \pm 1 & \vec{0} \\ \vec{p} & A_{T'} \end{bmatrix}.$$

Onde $A_{T'}$ é uma submatriz de A_T associada à árvore resultante da remoção de um nó terminal de T e o seu arco incidente. Repetindo o mesmo processo para $A_{T'}$ teremos

$$A_T = \begin{bmatrix} \pm 1 & 0 & \vec{0} \\ \vec{p} & \pm 1 & \vec{0} \\ & \vec{q} & A_{T''} \end{bmatrix}.$$

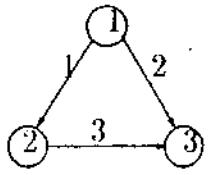
Com efeito, repetindo este processo $m-1$ vezes teremos no final uma matriz triangular inferior com diagonal unitária, portanto, uma matriz não singular. Completando assim a demonstração. □

Conclusão: Como a matriz A tem posto $m-1$, portanto incompleto, a matriz AD^2A^t é semipositiva definida. A solução encontrada para o sistema $AD^2A^ty = z$ é feita fixando $y_m = 0$ e resolvendo um sistema menor, $\bar{A}D^2\bar{A}^t\bar{y} = \bar{z}$, onde a matriz $\bar{A}$ é composta pelas $m-1$ primeiras linhas de A, assim como os vetores $\bar{y}$ e $\bar{z}$ são vetores com os primeiros $m-1$ elementos de y e z, respectivamente. A matriz $\bar{A}$ tem posto $m-1$, logo $\bar{A}D^2\bar{A}\bar{y} = \bar{z}$ é um sistema simétrico positivo definido e, portanto, pode ser resolvido com as técnicas que serão exibidas a seguir.

3.3.2 Operações com a Matriz de Incidência

O fato de trabalharmos com a matriz de incidência de um grafo favorece a geração de rotinas mais eficientes para efetuar operações de multiplicação com as matrizes A e A^t. Nesta seção descreveremos como são feitas estas operações.

Em nosso trabalho, o armazenamento desta matriz foi feito usando dois vetores de n elementos inteiros. São os vetores h (*head*) e t (*tail*) associados respectivamente à cabeça e à cauda dos arcos do grafo. Assim, se um elemento da matriz A, ou A^t, associado a um nó i do grafo, possui valor diferente de zero, haverá um arco j que chega ou sai deste nó, logo, a j-ésima



$$A = \begin{bmatrix} -1 & -1 & 0 \\ 1 & 0 & -1 \\ 0 & 1 & 1 \end{bmatrix}$$

h	2	3	3

t	1	1	2

Figura 3.2: Representação da matriz de incidência

coordenada do vetor h ou do vetor t, irá conter o valor i. A Figura 3.2 ilustra esta situação para um grafo com 3 nós e 3 arcos.

Por manipular a matriz apenas com vetores, as operações de multiplicação tornam-se mais simples. Assim, para efetuar a multiplicação com a matriz A, por exemplo, $u = Ax$, fazemos

para $i = 1, \ldots, m$
    u[i] = 0
para $j = 1, \ldots, n$
    u[t[j]] = u[t[j]] + x[j]
    u[h[j]] = u[h[j]] - x[j]

De modo análogo, para calcular $z = A^t y$, basta que se faça

para $j = 1, \ldots, n$
    z[j] = y[h[j]] - y[t[j]]

Assim, quando precisamos calcular o produto da matriz de projeção por um vetor, isto é, calcular $z = A\Psi A^t y$, como $\Psi = diag(\psi_i)$, fazemos

para $i = 1, \ldots, m$
    z[i] = 0
para $j = 1, \ldots, n$
    z[t[j]] = z[t[j]] + ψ[j] * (y[t[j]] - y[h[j]]);
    z[h[j]] = z[h[j]] + ψ[j] * (y[h[j]] - y[t[j]]);

Estas operações com as matrizes A e A^t são significativamente mais rápidas do que com matrizes cheias, além do que, usamos apenas um *loop*, o que reduz a complexidade destas operações para $O(n)$, contra o que no caso geral é feito em $O(m^2)$.

3.4 Técnicas de Resolução de Sistemas Lineares

Nesta seção apresentaremos as técnicas de resolução de sistemas lineares estudadas. Optamos em mostrar apenas a idéia de cada uma delas, sem nos deter em provas nem demonstrações, visto que este não é o foco principal deste trabalho, e que na literatura pode-se encontrar exaustivas publicações abordando este assunto. Em cada uma das seções colocamos as referências bibliográficas aonde pode-se encontrar uma abordagem mais completa tratando do assunto.

3.4.1 Método do Gradiente Conjugado

Consideramos o seguinte sistema

$$Hx = b, \tag{3.7}$$

onde $x, b \in \Re^m$ e $H \in \Re^{m \times m}$, sendo H simétrica e positiva definida. Resolver (3.7) é equivalente a resolver o problema de minimização irrestrita

$$\min f(x) = \frac{1}{2} x^t H x - b^t x, \tag{3.8}$$

pois $\nabla f(x) = Hx - b$, portanto, x^* que minimiza (3.8) satisfaz $\nabla f(x^*) = 0$, o que implica em $Hx^* - b = 0$. A idéia aqui é resolver o sistema linear (3.7) por um processo iterativo que minimiza (3.8), pois tratam-se de problemas equivalentes.

Definição 3.7 Dado uma matriz H, dois vetores d_1 e d_2 são ditos H-ortogonais ou conjugados em relação a H se $d_1^t H d_2 = 0$.

A idéia básica dos métodos de direções conjugadas parte do princípio de que x^* pode ser obtido como uma combinação linear de m vetores H-conjugados, isto é

$$x^* = \alpha_0 d_0 + \ldots + \alpha_k d_k.$$

O método de gradientes conjugados é apresentado como um processo iterativo que determina estas direções, d_k e os respectivos coeficientes, α_k, portando, minimiza $f(x) = \frac{1}{2}x^t Hx - b^t x$ em, no máximo, m iterações. Uma prova detalhada da idéia que apresentamos pode-se encontrar em várias publicações, sugerimos o capítulo 8 de Luenberger [25] para uma demonstração mais simples e completa.

Em Luenberger [25] as direções conjugadas são definidas como

$$d_{k+1} = -g_{k+1} + \beta_k d_k,$$

com

$$\begin{aligned} \beta_k &= \frac{g_{k+1}^t g_k}{g_k^t g_k}, \\ \alpha_k &= \frac{g_k^t g_k}{d_k^t H d_k}, \\ g_k &= Hx^k - b. \end{aligned}$$

Desta forma o método de gradientes conjugados é apresentado como

Algoritmo 3.1 Gradiente Conjugado
Dados $x \in \Re^n$, defina $d = -g = b - Hx$ e $\beta = 1$

repita
$\alpha = -\frac{g^t d}{d^t H d}$
$x = x + \alpha d$
$d = -g + \beta d$
$\beta = \frac{g^t H d}{d^t H d}$
$g = Hx - b$
até $g = \vec{0}$ ou $k = m$

Comentário: A complexidade do algoritmo acima é da ordem $O(m^3)$ para o caso geral, uma vez que para matrizes cheias os cálculos de α e β requerem $O(m^2)$ operações aritméticas. Para o problema de fluxo em redes a complexidade é reduzida para $O(mn)$, pois o cálculo de α e β pode ser feito em $O(n)$ operações, se as multiplicações envolvendo A e A^t forem feitas conforme o exposto na seção 3.3.2.

A segunda grande vantagem desta técnica reside no fato de que necessita-se apenas do armazenamento da matriz de incidência, ou seja, o processamento desta rotina não requer o cômputo de outras matrizes, o que é caro do ponto-de-vista de armazenamento. Como vimos anteriormente a matriz de incidência pode ser armazenada em apenas dois vetores, desta forma, a implementação desta rotina não requer mais do que $O(n)$ bytes para armazenar os resultados processados.

3.4.2 Gradiente Conjugado Pré-Condicionado

O algoritmo de gradiente conjugado pré-condicionado apresenta alguns melhoramentos ao algoritmo anterior, no sentido de acelerar a taxa de convergência e prover resultados mais seguros, isto é, reduzir o número de iterações necessárias para obter a convergência. Embora teoricamente o método de gradiente conjugado convirja em no máximo m iterações, se a matriz H for muito mal condicionada, o acúmulo de erros de arredondamento pode fazer com que na prática isto não ocorra. O pré-condicionamento surge, pois, como um artifício que minimiza o efeito do mal condicionamento da matriz acelerando a convergência.

A idéia é fazer uma modificação no sistema original multiplicando a matriz H por uma matriz simétrica não singular C^{-1}, resultando no seguinte sistema

$$\tilde{H}\tilde{x} = \tilde{b},$$

onde $\tilde{H} = C^{-1}HC^{-1}, \tilde{x} = Cx$ e $\tilde{b} = C^{-1}b$. A partir desta modificação, que serve para corrigir o número de condição de H, aplica-se o método do gradiente conjugado. O algoritmo resultante segundo Cunha [6] torna-se

Algoritmo 3.2 Gradiente conjugado pré-condicionado

Resolva $Mz = b$
$\alpha = z^t b / z^t Hz$
$x = \alpha z$
$s = b$
$r = s - \alpha Hz$
$t = z$
$k = 1$
enquanto $r \neq 0$ e $k \leq m$ **faça**
  Resolva $Mz = r$
  $\beta = z^t r / t^t s$
  $p = z + \beta p$
  $\alpha = z^t r / p^t Hp$
  $x = x + \alpha p$
  $s = r$
  $r = s - \alpha Hz$
  $t = z$
  $k = k + 1$

A matriz M chamada pré-condicionador, é definida como $M = C^2$. Bons pré-condicionadores podem conseguir convergência muito rápida, muitas vezes na ordem de $O(\sqrt{m})$ iterações.

Observe que o pré-condicionamento exige a cada iteração a resolução do sistema linear

$$Mz = r.$$

Por esta razão, alguma simplicidade é exigida na matriz M. Neste trabalho, segundo é sugerido em Golub [17], usamos como pré-condicionador a matriz formada pelos elementos da diagonal de H, isto é, fazemos $M = diag(h_{ii})$.

Comentário: O uso do pré-condicionamento torna-se obrigatório, pois o condicionamento da matriz $A\Psi A^t$ vai-se deteriorando gradativamente à medida que os iteratos aproximam-se do vértice ótimo. O mal condicionamento de $A\Psi A^t$ deve-se ao fato de que alguns dos elementos da diagonal de Ψ aproximam-se de zero, ao passo que outros tendem a valores positivos conforme as coordenadas de $\hat{x}^k$ vão se aproximando das variáveis básicas ou não

básicas. Ademais, a complexidade desta rotina é também da ordem $O(mn)$ operações aritméticas, e a memória alocada é da ordem de $O(n)$ bytes. Isto faz com que o uso do gradiente conjugado pré-condicionado seja a melhor dentre as técnicas estudadas, pois fornece resultados seguros com um baixo custo de memória e de tempo de processamento.

3.4.3 Fatoração de Cholesky

A fatoração de Cholesky é um caso particular da decomposição LU para matrizes simétricas e positiva definidas. A decomposição LU consiste em decompor uma matriz quadrada H em duas matrizes triangulares L e U, sendo L triangular inferior com diagonal unitária e U triangular superior. Particularmente, se H é simétrica e positiva definida então A pode ser decomposta em $A = GG^t$, sendo G triangular inferior. Para este caso particular temos a decomposição de Cholesky. Em Golub [17] e Forsythe e Mooler [13] pode-se encontrar a demonstração da existência da decomposição de Cholesky para matrizes simétricas e positivas definidas.

Para resolver o sistema $Hx = b$ então fazemos

$$\begin{aligned} H &= GG^t, \\ z &= G^{-1}b, \\ x &= (G^t)^{-1}z. \end{aligned}$$

Os cálculos dos vetores z e x, no sistema acima, são relativamente baratos pois a resolução de um sistema linear triangular requer apenas $O(m^2)$ operações aritméticas. A etapa mais cara é a decomposição de H em GG^t que requer $O(m^3)$ operações.

Algoritmo 3.3 Fatoração de Cholesky

$G = H$
Para $k = 1, \ldots, m$
  $g_{kk} = (g_{kk} - \sum_{p=1}^{k-1} g_{kp}^2)^{1/2}$
  Para $i = k+1, \ldots, m$
    $g_{ik} = (g_{ik} - \sum_{p=1}^{k-1} g_{ip}g_{kp})/g_{kk}$
Resolver $z = G^{-1}b$
Resolver $x = (G^t)^{-1}z$

Comentário: Dentre todas as técnicas testadas a que apresentou melhores resultados com relação à precisão foi a fatoração de Cholesky. Como poderá ser visto nas tabelas do capítulo seguinte, o uso da fatoração de Cholesky oferece resultados muito precisos na resolução do sistema linear independentemente de quão mal condicionada seja a matriz associada a este sistema.

Uma grande desvantagem da fatoração de Cholesky está no fato de que é necessário armazenar a matriz triangular G que, mesmo sendo triangular, envolve uma alocação da ordem de $O(m^2)$ bytes. A fatoração de Cholesky torna-se pior em relação ao método do gradiente conjugado e do gradiente conjugado pré-condicionado por não tirar maiores proveitos da estrutura da matriz de incidência. A complexidade do algoritmo 3.3 é da ordem $O(m^3)$.

3.4.4 Decomposição QR

A decomposição QR consiste em decompor a matriz em duas outras com estruturas especiais. Seja $H \in \Re^{m \times m}$, podemos escrever H como

$$H = QR,$$

onde $Q \in \Re^{m \times m}$ é ortogonal e $R \in \Re^{m \times m}$ é triangular superior. Uma vez tendo a decomposição QR, o sistema $Hx = b$ reduz-se a

$$Rx = Q^t b, \tag{3.9}$$

que é relativamente simples, pois envolve apenas a multiplicação de uma matriz por um vetor e a resolução de um sistema linear triangular. O esforço total para encontrar a solução de (3.9) é da ordem de $O(m^2)$, que é relativamente barato comparado com o esforço prévio de decompor a matriz H em QR, $O(m^3)$.

Do ponto-de-vista de armazenamento, não há um ganho significativo por que é necessário ter o conhecimento explícito da matriz R, o que é bastante caro, pois envolve uma alocação da ordem $O(m^2)$. Isto indica que não tiramos grande proveito da esparsidade do problema com a implementação que fizemos desta técnica de resolução de sistemas lineares.

A decomposição QR pode ser conseguida de diversas formas, em geral se usam transformações especiais como Givens ou Householder, sugerimos

Golub [17] para maiores detalhes sobre estas transformações. Em nosso trabalho, usamos as transformações de Givens. A matriz Q é na verdade uma série de transformações que operam sobre a matriz H. A figura abaixo ilustra como ocorre este processo para uma matriz $H \in \Re^{3\times 3}$. Na figura 3.3, os elementos não nulos da matriz H são simbolizados com a letra x e as transformações de Givens usadas para zerar o elemento $h_{i,j}$ pelas matrizes $G_{i,j}$.

$$H = \begin{bmatrix} x & x & x \\ x & x & x \\ x & x & x \end{bmatrix} G_{2,1} \begin{bmatrix} x & x & x \\ x & x & x \\ x & x & x \end{bmatrix} \rightarrow G_{3,1} \begin{bmatrix} x & x & x \\ 0 & x & x \\ x & x & x \end{bmatrix}$$

$$\rightarrow G_{3,2} \begin{bmatrix} x & x & x \\ 0 & x & x \\ 0 & x & x \end{bmatrix} = \begin{bmatrix} x & x & x \\ 0 & x & x \\ 0 & 0 & x \end{bmatrix}$$

Figura 3.3: Decomposição QR simbólica

Para uma matriz quadrada qualquer de ordem n, Q^t é definido como

$$Q^t = G_{nn-1}, \ldots, G_{n1}, \ldots, G_{31} G_{21}.$$

Para efeito de notação denotemos por $G(p, q, c, s)$ as matrizes de Givens usadas para zerar os elementos h_{pq} de H, sendo c e s uma solução do sistema

$$\begin{cases} c^2 + s^2 & = & 1, \\ ca_{pq} - sa_{pp} & = & 0. \end{cases}$$

Observe que não é necessário o conhecimento explícito da matriz Q. É suficiente que se façam as transformações de Givens simultaneamente no lado direito e na matriz A. Em Golub [17], o algoritmo é apresentado como

Algoritmo 3.4 Decomposição QR

$R = H$
$\hat{b} = b$
para $q = 2, \ldots, n$
  para $p = 1, \ldots, q-1$
    se $h_{pq} \neq 0$ **então**
      $c = h_{qq}/\sqrt{h_{qq}^2 + h_{pq}^2}$
      $s = h_{pq}/\sqrt{h_{qq}^2 + h_{pq}^2}$
      $R = G(p, q, c, s)R$
      $\hat{b} = G(p, q, c, s)\hat{b}$
Resolver $Rx = \hat{b}$

Comentário: Do ponto-de-vista teórico a decomposição QR é a menos atraente das técnicas estudadas. Assim como a fatoração de Cholesky, é necessário o armazenamento de uma matriz triangular, a matriz R, o que a torna cara do ponto-de-vista de armazenamento. Ademais, com relação à convergência, esta técnica apresenta resultados muito ruins. O teste $h_{pq} \neq 0$ é feito sob uma tolerância estabelecida, isto é, testamos de fato se $|h_{pq}| > \epsilon$. Ocorre que à medida que o algoritmo evolui alguns elementos da matriz Ψ aproximam-se rapidamente de zero, isto leva o algoritmo a não efetuar algumas operações que seriam necessárias, em virtude de ter $0 < |h_{pq}| < \epsilon$, o que torna o algoritmo impreciso. No capítulo seguinte mostramos algumas tabelas que ilustram este fato.

3.5 Conclusões

Neste capítulo descrevemos dois aspectos importantes na implementação das rotinas estudadas: a ordem de complexidade e a ordem da quantidade de memória necessária para o processamento das mesmas. Considerando estes dois aspectos podemos concluir que o método do gradiente conjugado e do gradiente conjugado pré-condicionado compõem as melhores técnicas para a resolução do sistema linear de interesse. De fato, como veremos no próximo capítulo, o gradiente conjugado pré-condicionado fornece resultados muito bons e na maioria dos casos superiores aos das demais técnicas. Entretanto, se nesta análise também considerarmos a precisão veremos que os métodos de gradiente conjugado tendem a se igualar aos demais. A rigor, o método

do gradiente conjugado sem pré-condicionamento é demasiado impreciso nas últimas iterações e, por isto, pode ser considerada a menos atraente das técnicas estudadas. Veremos no capítulo seguinte que, com base em resultados observados, a melhor maneira de resolver o sistema linear é através do uso da fatoração de Cholesky ou do gradiente conjugado pré-condicionado, pois estas rotinas fornecem em tempo hábil os resultados mais precisos.

Capítulo 4

Resultados Computacionais

4.1 Introdução

Neste capítulo tentaremos verificar empiricamente, com base em resultados observados, quais dos métodos de pontos interiores e técnicas de resolução de sistemas lineares estudados são mais eficientes para o problema de fluxo de custo mínimo. Os algoritmos foram implementados em Pascal e executados em uma estação de trabalho SUN, modelo SPARC 1+, com 12.5 Mips, 1.4 Mflops, 8Mbytes de memória principal e 2 extensões de memória principal de 4Mbytes. O sistema operacional usado foi o sistema SunOS versão 4.1.

Nos capítulos anteriores estudamos quatro algoritmos de pontos interiores que combinados com as quatro técnicas de resolução de sistemas lineares abordadas geram dezesseis implementações diferentes. A fim de elaborar um estudo organizado evitando expor resultados irrelevantes, os testes foram sendo feitos de forma seletiva, fazendo gradualmente a exclusão dos métodos e técnicas que apresentavam pior desempenho. Por esta razão, o foco principal deste estudo está concentrado sobre os algoritmos: primal afim e passos longos, da fatoração de Cholesky e do método do gradiente conjugado pré-condicionado. O algoritmo primal dual apresentou resultados intermediários e, portanto, os testes com os mesmos não foram exaustivos como com os citados acima. O algoritmo de barreira de passos curtos e a decomposição QR, como veremos a seguir, mostraram-se irremediavelmente lentos, e a resolução do sistema linear por gradiente conjugado imprecisa, o que fez com que nosso interesse sobre os mesmos ficasse restrito apenas ao campo da teoria.

4.2 Sobre a Decomposição QR

Usando a Decomposição QR executamos ao todo 10 redes distintas de 300 nós e 4000 arcos para os algoritmos primal afim e passos longos. Para a geração das redes usamos um gerador de redes desenvolvido pelo grupo de estudo de fluxo em redes do Laboratório de Matemática Aplicada do IMECC - UNICAMP. Com relação ao tempo de CPU, podemos ver pela Tabela 4.1, que os resultados obtidos pela decomposição QR são muito ruins se comparados às demais técnicas. Isto nos motivou a abandoná-la nos testes com redes maiores.

Na construção das tabelas usamos para denotar o método do gradiente conjugado as iniciais GC, bem como denotamos o método do gradiente conjugado pré-condicionado por GCPC. As lacunas contendo um traço horizontal (—) indicam que não houve convergência do algoritmo e da técnica para a massa de testes utilizada; onde não convergir significa ter o processo iterativo interrompido por erro de arredondamento, pelo estouro no número de iterações ou a convergência para um ponto diferente do ótimo. Para obter a solução ótima destes problemas usamos uma implementação do método simplex desenvolvido no Laboratório de Matemática Aplicada do IMECC - UNICAMP.

	Algoritmos	
Técnicas	Primal afim	Passos longos
GCPC	—	$4'$
GC	—	$8'$
Cholesky	$11'$	$16'$
Decomposição QR	$36'$	$46'$

Tabela 4.1: Tempo médio de CPU

Outro fato importante na análise que estamos fazendo diz respeito a precisão dos resultados. Na Tabela 4.2 podemos ver o percentual de sucesso na execução dos algoritmos primal afim e de passos longos, onde sucesso neste caso significa atingir aproximadamente uma solução ótima obtida pelo método simplex. Nesta tabela constatamos o quão impreciso é o uso do

Técnicas	Algoritmos	
	Primal afim	Passos longos
GCPC	0%	90%
GC	0%	40%
Cholesky	90%	90%
Decomposição QR	60%	90%

Tabela 4.2: Percentual de sucesso

gradiente conjugado sem o pré-condicionamento. O mesmo fato pode ser observado na Tabela 4.3, onde exibimos a variação percentual média entre o valor da solução ótima obtida pelo método simplex e da solução obtida pelos métodos de pontos interiores.

Técnicas	Algoritmos	
	Primal afim	Passos longos
GCPC	—	0.61%
GC	—	4.27%
Cholesky	0.04%	0.04%
Decomposição QR	0.05%	0.07%

Tabela 4.3: Variação percentual média

4.3 Sobre o Algoritmo de Barreira de Passos Curtos

A fim de controlar a duração máxima dos testes, foi imposto o limite de 200 iterações como o máximo de iterações permitidas, evitando assim que determinado algoritmo dispendesse um tempo excessivo de processamento. Na Tabela 4.4 ilustramos o desempenho dos algoritmos de pontos interiores com redes de 300 nós e 4000 arcos, usando como técnica de resolução do sistema linear a fatoração de Cholesky. Para esta tabela foram rodadas 10

redes, observamos então que em nenhuma destas execuções o algoritmo de passos curtos chegou a uma solução ótima, tendo em todos os casos o processo finalizado pelo estouro no número de iterações permitidas. Usando outra técnica para resolver o sistema linear, o resultado é igualmente desanimador. Na Tabela 4.5 ilustramos o desempenho do algoritmo de passos curtos testado com as demais técnicas de resolução de sistemas lineares; em nenhum dos casos o algoritmo convergiu, tendo interrompido o processamento ou pelo estouro no número de iterações ou por erro de arredondamento.

Algoritmos	CPU	Iterações
Primal afim	$10'$	30
Passos longos	$16'$	31
Primal dual	$26'$	67
Passos curtos	$2h\ 55'$	200

Tabela 4.4: Valores médios de CPU e No. iterações

Algoritmos	CPU	Iterações
GCPC	$39'$	200
GC	—	—
Decomposição QR	—	—
Cholesky	$2h\ 39'$	200

Tabela 4.5: Implementações do algoritmo de passos curtos

A razão pela qual o algoritmo de passos curtos possui um elevado número de iterações pode ser entendida através da visualização geométrica da trajetória deste algoritmo.Na Figura 4.1 ilustramos a trajetória do algoritmo para um problema de PL com 2 variáveis e 3 restrições de desigualdade. Podemos ver que o algoritmo de passos curtos acompanha a própria trajetória central desde o primeiro ponto x^c em que $\delta(x^c, \epsilon_0) < 0.0015$ até a solução ótima, x^*. Para manter-se sempre na região central o algoritmo requer que sejam dados passos muito pequenos, conseqüentemente ocorre um grande

aumento no número de iterações, o que o deixa excessivamente lento. Deste modo um estudo computacional deste algoritmo para redes maiores torna-se impraticável.

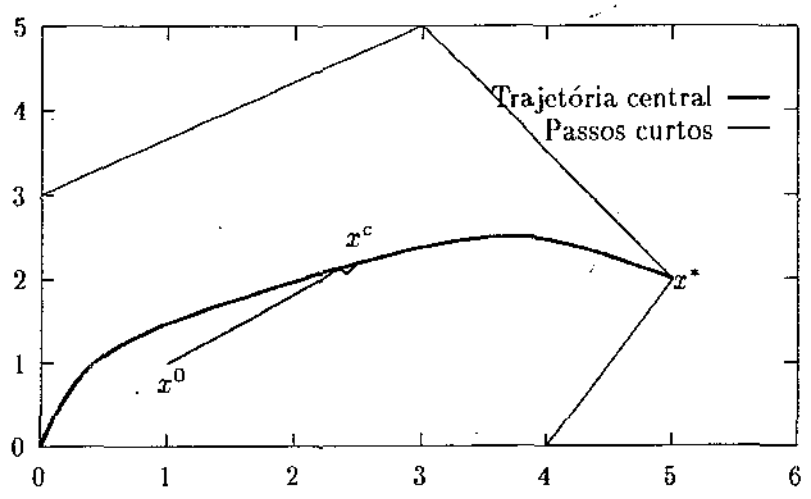


Figura 4.1: Trajetória do algoritmo de passos curtos

4.4 Sobre o Algoritmo de Passos Longos

O algoritmo de passos longos que implementamos difere um pouco do apresentado no capítulo 2. Fizemos algumas modificações que o tornaram mais rápido. Estas modificações foram feitas na tentativa de corrigir dois problemas que constatamos no algoritmo original.

1. O tamanho do passo no algoritmo de passos longos é geralmente "pequeno".
2. O critério que estabelece a proximidade à trajetória central ($\delta(x, \epsilon) \leq 0.1$) é muito rígido; pode ser relaxado.

Com base nestas observações, fizemos as alterações que buscam corrigir de imediato estas duas limitações tentando conservar as demais características do algoritmo original.

Para aumentar o tamanho do passo, implementamos uma rotina de busca linear exata. Com isto permite-se dar passos maiores. O tamanho do passo torna-se ainda muito maior se $\delta(x^k, \epsilon_{k+1}) \gg 0.1$. Daí a necessidade de relaxarmos o critério de proximidade à trajetória central. Uma vez que com a busca linear exata, o algoritmo caminha a passos largos em direção ao ótimo, consideramos então que todos os pontos gerados pelo algoritmo desde a aproximação inicial x^0, tal que $\delta(x^0, \epsilon_0) \leq 0.1$, sejam pontos centrais.

A figura 4.2 ilustra o comportamento do algoritmo modificado, com busca linear exata e critério de proximidade relaxado, contra o algoritmo original, com o tamanho do passo definido pelo lema 2.10 e com o controle rígido da proximidade à trajetória central. Observe que os tamanhos dos passos do algoritmo modificado são muito maiores que os do algoritmo original, e que cada iteração do mesmo compõe muitas iterações do segundo.

O algoritmo modificado é descrito como segue abaixo

Algoritmo 4.1 Algoritmo de Passos Longos Modificado
Dados $\hat{x} \in \hat{S}^0, \epsilon > 0$, tal que $\delta(\hat{x}, \epsilon) \leq 0.1$ e $\nu \in \Re$

repita
$\epsilon = \epsilon(1 + \frac{\nu}{\sqrt{2n}})$
$\hat{X} = \text{diag}(\hat{x}_i)$
Resolver $(\hat{A}\hat{X}^2\hat{A}^t)\hat{y} = \hat{A}(\epsilon\hat{X}^2\hat{c} - \hat{x})$
$h_N = -(\hat{X}^2\hat{c} - \hat{x} - \hat{X}^2A^t\hat{y})$
Resolver "exatamente" $\bar{\lambda} = \arg\min\{f_\epsilon(\hat{x} + \lambda h) | \lambda > 0\}$
$\hat{x} = \hat{x} + \bar{\lambda}h_N$
até convergir

A Tabela 4.6 mostra um quadro comparativo entre os dois algoritmos de passos longos. Nesta tabela, nas colunas onde mostramos o número de iterações do algoritmo de passos longos original, ilustramos, além do número de iterações do *loop* mais externo do algoritmo, o número total de iterações, o que envolve as iterações intermediárias do *loop* mais interno. Observe que na

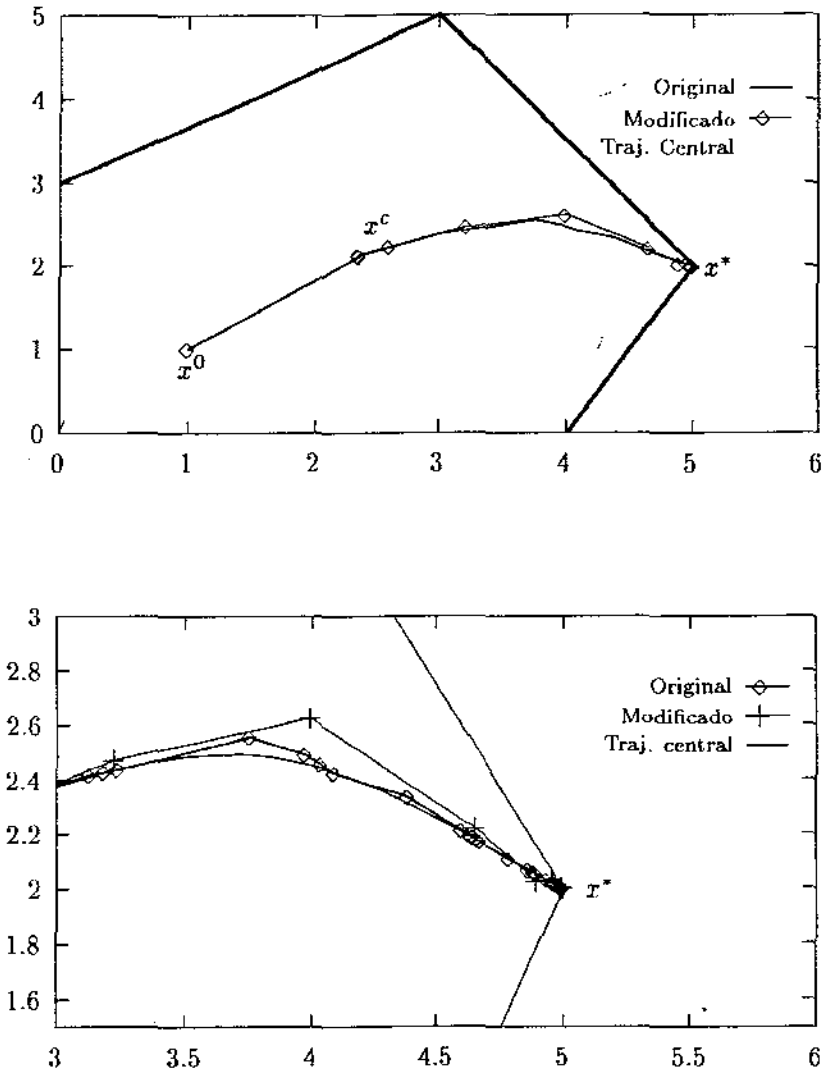


Figura 4.2: Iterações do algoritmo de passos longos

prática o algoritmo original executa muitas iterações a mais que o algoritmo modificado, o que o torna mais lento. Rodamos neste exemplo redes de 300 nós e 4000 arcos e 500 nós e 5000 arcos. Estes resultados nos motivaram a abandonar o algoritmo de passos longos original nos testes com as demais redes.

	300/4000		500/5000	
Algoritmos	CPU	Iterações	CPU	Iterações
Original	$12h29'$	21/1706	2dias $13h$	22/1858
Modificado	$16'$	30	$1h6'$	42

Tabela 4.6: Tempo médio de CPU

4.4.1 A Busca Linear

Na seção 2.5.2 apresentamos um algoritmo que usa um procedimento de $O(1)$ para determinar o tamanho do passo a cada iteração. Para tanto faz-se uma busca linear inexata com a única garantia de que o tamanho do passo faz decrescer o valor da função penalizada. Na implementação que fizemos empregamos uma busca linear que fornece o valor exato do tamanho do passo que minimiza

$$f_\epsilon(\hat{x}) \;=\; \epsilon\hat{c}^t\hat{x} + = p(\hat{x}),$$

na direção Δx, dentro de uma tolerância estabelecida. Estas modificações, o emprego da busca linear e a relaxação da proximidade à trajetória central, devem alterar a complexidade do algoritmo fazendo-o perder a complexidade de tempo polinomial.

Baseado em testes realizados, constatamos que o custo da rotina de busca linear é relativamente barato, na Tabela 4.7 ilustramos o percentual de tempo de CPU que esta rotina consome no cômputo de cada iteração. Para resolver o sistema empregamos o método do gradiente conjugado pré-condicionado, uma vez que este é o método que apresenta melhores resultados com relação ao tempo de CPU. Empregando outras técnicas, certamente o percentual de

tempo consumido na resolução do sistema seria maior, o que seria pouco ilustrativo para esta tabela.

Para implementar a busca linear exata usamos uma idéia sugerida em Gonzaga [20]. Observamos que a função $f_\epsilon(\cdot)$ é convexa, contínua e com valores tendendo a infinito nos pontos próximos à fronteira, logo, o único ponto que minimiza $f_\epsilon(\cdot)$ na reta Δx pode ser encontrado com o emprego do método da bissecção. Como a derivada de $f_\epsilon(\cdot)$ é mais barata de ser calculada pois não envolve cálculo com logaritmo, usamos o método da bissecção aplicado à função derivada de $f_\epsilon(\cdot)$ na reta Δx,

$$\frac{d}{d_\lambda} f_\epsilon(x + \lambda\Delta x) \;=\; (\nabla f_\epsilon(x + \lambda\Delta x))^t \Delta x.$$

Que é calculada como

$$\frac{d}{d_\lambda} f_\epsilon(x + \lambda\Delta x) \;=\; \sum_{j=1}^{n} \Delta x_j (\epsilon c_j - \frac{1}{x_j + \lambda\Delta x_j} + \frac{1}{u_j - x_j - \lambda\Delta x_j}). \quad (4.1)$$

O valor de λ retornado por esta busca linear é tal que o valor de $(\nabla f_\epsilon(x + \lambda\Delta x))^t\Delta$ calculado por (4.1) seja suficientemente próximo de zero para uma tolerância estabelecida.

	Redes(nós/arcos)			
Rotinas	300/4000	400/4000	7000/9000	1500/15000
Resolução sistema	85.1%	90.4%	90.6%	91.5%
Busca linear	11.1%	7.1%	8.0%	7.9%
Outros processamentos	3.6%	2.3%	1.2%	0.4%

Tabela 4.7: Percentual de CPU consumido pela busca linear

4.4.2 O Parâmetro ν

A escolha do parâmetro ν foi feita com base em resultados de testes observados. Constatamos que, usando a fatoração de Cholesky, quando $e_k \simeq 10^5$

o algoritmo 2.5 pára, também vimos que o algoritmo primal afim realizou em média pouco mais de trinta iterações na maioria dos problemas testados e, em todos eles, o valor retornado era muito próximo do ótimo. Estas observações nos levou a estipular o parâmetro ν de tal modo que $c_{30} \simeq 10^5$, uma vez que com isto estaríamos fixando um limitante para o número de iterações que muito provavelmente faria o algoritmo convergir para um valor confiável. Fizemos, então, o seguinte cálculo

$$c_0(1 + \frac{\nu}{\sqrt{2n}})^{30} = 10^5,$$

o que implica que

$$\nu \simeq 0.6883\sqrt{2n}.$$

4.5 Resultados Gerais

Nesta seção fazemos mais algumas comparações entre as técnicas de resolução do sistema linear estudadas e comparamos o algoritmo de passos longos ao método simplex e out-of-kilter, desenvolvidos no Laboratório de Matemática Aplicada do IMECC - UNICAMP, no processamento de redes com até 1500 nós.

No gráfico da Figura 4.3 mostramos o tempo de CPU de cada técnica ao longo das iterações no processamento de uma rede com 500 nós e 5000 arcos. Na construção desta figura, da Tablea 4.8 e da Figura 4.4 usamos o algoritmo de passos longos modificado. O gráfico da Figura 4.3 é bastante ilustrativo no sentido de mostrar o efeito do mal-condicionamento da matriz sobre a resolução do sistema por cada técnica em particular. Observe como evoluem os métodos de gradiente conjugado e como é significativo o ganho em tempo de CPU quando se usa o pré-condicionamento.

Resultados mais expressivos podem ser observados na Tabela 4.8 onde mostramos a média dos tempos de processamento de cada uma das técnicas para redes de diversos tamanhos. A partir desta tabela construímos o gráfico da Figura 4.4. Esta figura oferece uma perspectiva dos tempos de CPU das técnicas para diversos exemplos de problemas de fluxo de custo mínimo. Observe que no caso médio a fatoração de Cholesky e a decomposição QR

tendem a aproximar-se de uma curva com inclinação superior à dos métodos de gradiente. Isto justifica o melhor desempenho dos métodos de gradiente conjugado, quando comparamos o tempo de CPU.

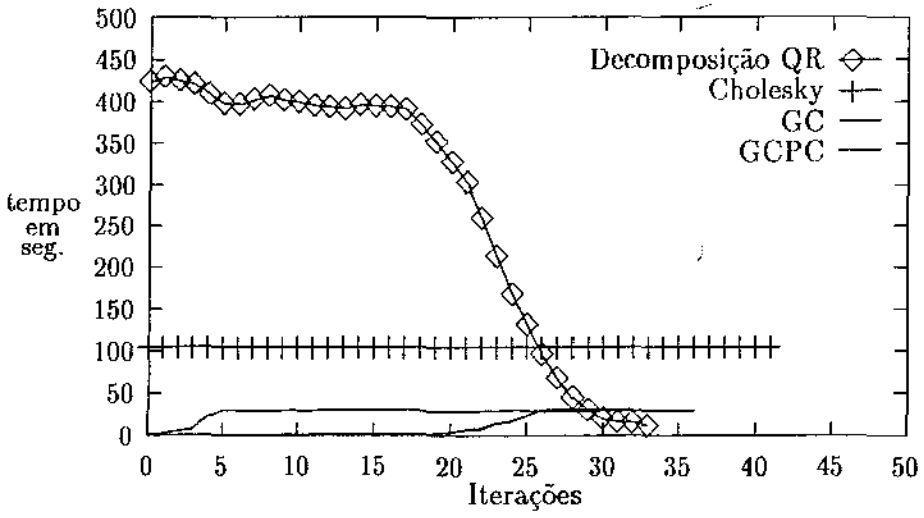


Figura 4.3: Tempos de CPU ao longo de cada iteração

Técnicas	Número de nós/arcos				
	100/600	150/1500	300/4000	400/1000	500/5000
GCPC	0.4"	0.8"	5"	6"	9"
GC	0.5"	1"	8"	12"	26"
Cholesky	0.6"	2"	16"	39"	104"
Decomposição QR	1"	4"	50"	103"	281"

Tabela 4.8: Tempos médio de CPU

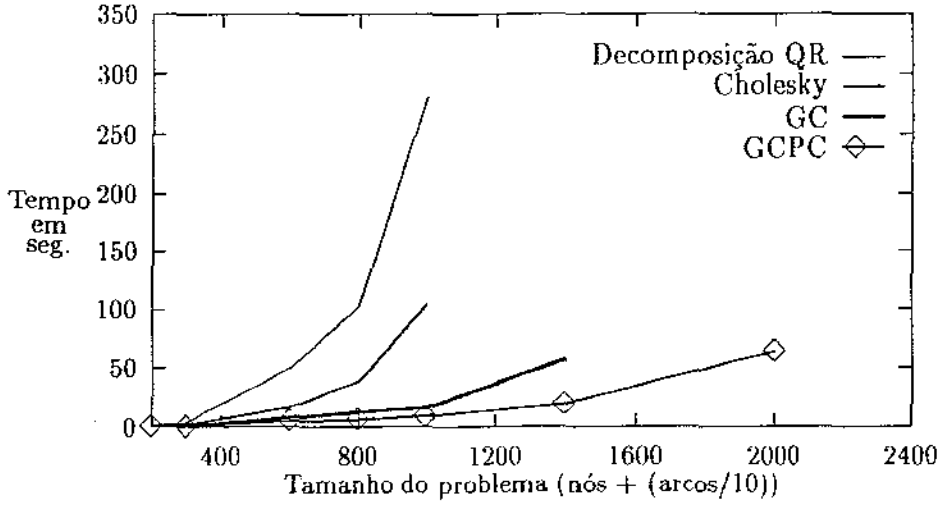


Figura 4.4: Desempenho das técnicas com relação ao tempo de CPU

4.6 Conclusões

Este capítulo fornece subsídios para tirarmos várias conclusões. Como fôra mencionado na introdução, podemos comprovar que de fato a decomposição

QR e o método do gradiente conjugado sem pré-condicionamento não são recomendáveis na resolução do sistema linear para nenhum dos algoritmos. Também podemos concluir que o algoritmo de passos curtos é demasiadamente lento, independentemente da técnica usada para resolver o sistema e, portanto, não possui interesse prático.

Com relação ao algoritmo primal dual, com base no que traz a literatura a seu respeito, podemos concluir que é possível melhorá-lo ainda bastante. O mesmo pode-se dizer do método do gradiente conjugado pré-condicionado, pois muito provavelmente existem pré-condicionadores mais eficientes para o problema de fluxo em redes.

Também podemos observar outros aspectos positivos resultantes deste trabalho. Estes resultados dizem respeito ao algoritmo de passos longos modificado e o primal afim. Acreditamos que quando aplicados a problemas genéricos de programação linear de grande porte estes algoritmos devam lograr grande êxito computacional. Para os exemplos de problemas de fluxo de custo mínimo que abordamos, estes algoritmos foram pouco eficientes, em alguns testes que fizemos comparando-os ao método simplex e out-of-kilter, os algoritmos tradicionais chegam em média a ser trinta vezes mais velozes. Isto deve-se ao fato de que o problema de fluxo de custo mínimo possui uma estrutura muito simplificada e já exaustivamente estudada na literatura. A aplicação mais adequada dos métodos que estudamos seria, então, para problemas de grande porte de programação linear, como cita a literatura a este respeito, ou até mesmo para problemas de grande porte de fluxo de custo mínimo para redes generalizadas, pois este problema não possui uma estrutura tão simplificada quanto a do problema de fluxo de custo mínimo.

Capítulo 5

Conclusões

Diversas conclusões podem ser tiradas sobre tópicos específicos dentro de cada capítulo. Como achamos que não seria conveniente incorporá-las todas nesta única seção, optamos em colocar no final de cada capítulo uma seção onde tecemos tais conclusões. Deixamos para este capítulo apenas as conclusões mais gerais.

Com base nos testes que fizemos, podemos concluir que os métodos de pontos interiores, tal como implementamos, não consistem numa boa opção para resolver problemas de fluxo de custo mínimo. Para termos algoritmos de pontos interiores competitivos frente ao simplex e ao método out-of-kilter, seria necessário tentar aperfeiçoá-los ainda mais, em especial, aperfeiçoar as rotinas de resolução de sistemas lineares, de modo que se assegurasse resultados mais precisos sob um custo mais reduzido de memória e de tempo de processamento. Algumas sugestões neste sentido são apresentadas na seção 5.2.

5.1 Contribuições

Podemos destacar três contribuições principais resultantes deste estudo.

- A principal destas é a apresentação do algoritmo de passos longos modificado. Este algoritmo, conforme visto no capítulo 4, fornece resultados muito bons chegando muitas vezes a superar o algoritmo primal afim e os algoritmos primais duais.

- Também podemos citar como contribuição a abordagem que usamos para adaptar os algoritmos de trajetória central de Gonzaga para problemas canalizados, pois as referências bibliográficas que utilizamos não tratam diretamente o problema canalizado.

- Por fim, devemos citar também a análise comparativa que fizemos entre métodos de pontos interiores, o que foi uma motivação inicial para este trabalho.

5.2 Sugestões para Desenvolvimentos Futuros

Consideramos a resolução do sistema linear como o principal ponto a ser aperfeiçoado nos algoritmos. Nossos esforços então devem se voltar especialmente no sentido de como construir rotinas mais precisas, rápidas e que tenham baixo custo de armazenamento.

Com relação à precisão podemos ver pela Tabela 4.2 que, exceto à fatoração de Cholesky, todas as outras rotinas apresentam resultados imprecisos, especialmente para o algoritmo primal afim. O estudo de melhores pré-condicionadores, então, surge como uma primeira sugestão para desenvolvimentos futuros, pois o uso do gradiente conjugado pré-condicionado fornece bons resultados com relação ao tempo de processamento e pode ainda ser aprimorado para gerar resultados mais precisos. Em [33], Rezende e Veiga sugerem o uso de um pré-condicionador diferente cujo desempenho, segundo eles, é melhor que o usado neste trabalho.

Em se tratando da fatoração de Cholesky, pode-se aprimorar a rotina que trabalhamos, usando a fatoração simbólica apresentada em [1] e [10]. Com o uso da fatoração simbólica é possível reduzir o tempo de CPU e o espaço de memória alocado sem com isto perder a precisão da rotina original.

Por fim, também podemos usar, para aumentar a velocidade de convergência, indicadores de variáveis nulas como sugerem Tapia et al. [11]. Indicadores de variáveis nulas são funções que, baseados em iterações anteriores, indicam se uma variável tornar-se-á ou não nula ao término do processo iterativo. Isto é feito de modo seguro e eficiente, pois é possível fazer esta identificação muitas iterações antes do término do algoritmo. Isto permite

antecipar a convergência pois sabendo de antemão se uma variável se anulará, em alguns casos, é possível reduzir o número de iterações.

Bibliografia

[1] I. Adler, N. Karmarkar, M. Resende, and G. Veiga. Data structures and programming techniques for the implementation of Karmarkar's algorithm. *ORSA Journal on Computing*, 1:84 – 106, 1989.

[2] A. I. Ali, R. Padman, and H. Thiagarajan. Dual algorithm for pure network problem. *Operations Research*, 37(1):159 – 171, 1989.

[3] D.A. Barnes. A variation on Karmarkar's algorithm for solving linear programming problems. *Mathematical Programming*, 36(2):174 – 182, 1986.

[4] D. A. Bayer and J. C. Lagarias. The non-linear geometry of linear programming I. Affine and projective scaling trajectories. *Transactions of the American Mathematical Society*, 314(2):499 – 526, 1989.

[5] M. S. Bazaraa and J. J. Jarvis. *Linear Programming and Network Flows.* The John Wiley and Sons, 1977.

[6] C. Cunha. *Métodos numéricos para as engenharias e ciências aplicadas.* Editora da UNICAMP, Campinas, S.P., 1993.

[7] D. den Hertog and C. Roos. A survey of search directions in interior point methods for linear programming. *Mathematical Programming*, 52(3):481 – 509, 1991.

[8] I. I. Dikin. Iterative solution of problems of linear and quadratic programming. *Soviet Mathematics Doklady*, 8(3):674 – 675, 1967.

[9] J. Edmonds and R. M. Karp. Theoretical improvement in algorithmic efficiency for network flow problems. *Journal of the ACM*, 19(2):248 – 264, 1972.

[10] S. C. Eisat and M. H. Schultz. Algorithms and data structures for sparse and symetric gausian elimination. *SIAM J. Sci. and Stat. Comput.*, 2(2):225 - 237, 1981.

[11] A. S. El-Bakry, R. A. Tapia, and Y. Zhang. A study of indicators for identifying zero variables in interior point methods. Technical report, Rice University, Huston, Texas 77252 - 1892, December 1992.

[12] A. Fiacco and G. McCormick. *Nonlinear Programming: Sequential Unconstrained Minimization Techniques.* John Wiley and Sons, New York, N.Y., 1968.

[13] G. E. Forsythe and C. B. Mooler. *Computer Solution of Linear Algebraic System.* Prentice-Hall, Inc, 1967.

[14] K. R. Frisch. The logarithmic potential method of convex programming. Memorandum, University Institute of Economics, Oslo, Norway, 1955.

[15] A. V. Goldberg and R. E. Tarjan. Finding minimum-cost circulation by canceling negative cicles. *Journal of the ACM*, 36(4):873 - 886, 1989.

[16] A. V. Goldberg and R. E. Tarjan. Finding minimum-cost circulation problems by successive approximation. *Mathematics of Operations Research*, 15(3):430 - 466, 1990.

[17] G. H. Golub and C. F. Van Loan. *Matrix Computations.* The John Hopkins University Press, Baltimore, Maryland, 1983.

[18] C. C. Gonzaga. An algorithm for solving linear programming problems in $O(n^3\mathrm{L})$ operations. In *Progress in Mathematical Programming - Interior Point and Related Methods*, chapter 1. Springer Verlag, Berlin, 1989.

[19] C. C. Gonzaga. Path following methods for linear programming. *SIAM REVIEW*, 34(2):167 - 224, 1992.

[20] C.C. Gonzaga. *Algoritmos de pontos interiores para programação linear.* XVII Colóquio Brasileiro de Matemática - IMPA, Rio de Janeiro, Brasil, 1988.

[21] N. Karmarkar. A new polynomial time algorithm for linear programming. *Combinatorica*, 4:373 - 395, 1984.

[22] J. Kennington and R. Helgason. *Algorithms for network programming.* John Wiley & Sons, New York, 1978.

[23] L. G. Khachiyan. A polynomial algorithm in linear programming. *Soviet Mathematics Doklady*, 20(1):191 - 194, 1979.

[24] M. Kojima, S. Mizuno, and A. Yoshise. A primal-dual interior point algorithm for linear programming. In *Progress in Mathematical Programming - Interior Point and Related Methods*, chapter 2. Springer Verlag, Berlin, 1989.

[25] D. G. Luenberger. *Introduction to Linear and Nonlinear Programming.* Addison Wesley, 1984.

[26] I. J. Lustig. Feasibility issues in a primal-dual interior-point method for linear programming. *Mathematical Programming*, 49(2):145 - 162, 1991.

[27] I. J. Lustig, R. E. Marsten, and D. F. Shanno. Computational experience with a primal-dual interior-point method for linear programming. *Linear Algebra and its Applications*, 152:191 - 222, 1989.

[28] N. Megiddo. Pathways to the optimal set in linear programming. In *Progress in Mathematical Programming - Interior Point and Related Methods*, chapter 8. Springer Verlag, Berlin, 1989.

[29] N. Megiddo and M. Shub. Boundary behaviour of interior point algorithms in linear programming. *Mathematics of Operations Research*, 14(1):97 - 146, 1989.

[30] R. D. C. Monteiro and I. Adler. Interior path following primal-dual algorithms. Part I: linear programming. *Mathematical Programming*, 44(1):27 - 41, 1989.

[31] J. B. Orlin. A faster strongly polinomial minimum cost flow algorithm. *Operations Research*, 41(2):338 - 350, 1993.

[32] J. Renegar. A polynomial-time algorithm based on Newton's method for linear programming. *Mathematical Programming*, 40(1):59 - 94, 1988.

[33] M. Resende and G. Veiga. An implementation of the dual affine scaling algorithm for minimum cost flow on bipartite uncapacitated networks. Technical report, AT&T Bell Laboratories, Murray Hill, NJ, 1990.

[34] N. Shor. Utilization of the operation of space dilatation in the minimization of convex functions. *Kibernetika*, 1:6 – 12, 1970. English translation: Cybernetics 6(1), 7 - 15.

[35] G. Sonnevend. An analytical centre for polyhedrons and new classes of global algorithms for linear (smooth, convex) programming. In *Lecture Notes in Control and Information Sciences 84*, pages 866–876. Springer Verlag, New York, NY, 1985.

[36] P. M. Vaidya. An algorithm for linear programming wich requires $O(((m+n)n^2 + (m+n)^{1.5}n)L)$ arithmetic operations. *Mathematical Programming*, 47(2):175 – 202, 1990.

[37] R. J. Vanderbei. Afine-scaling for linear programs with free variables. *Mathematical Programming*, 43(1):31 – 44, 1989.

[38] R. J. Vanderbei, M. S. Meketon, and B. A. Freedman. A modification of Karmarkar's linear programming algorithm. *Algorithmica*, 1(4):395 – 407, 1986.