UNICAMP

Projeção Direta de Vetores

Fernando Rocha Villas Bôas

IMECC – UNICAMP

1995

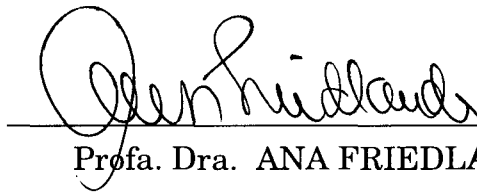
Este exemplar corresponde à redação final da tese devidamente corrigida e defendida pelo Sr. Fernando Rocha Villas Bôas e aprovada pela Comissão Julgadora

Campinas, 14 de março de 1995

Prof. Dr. _____
Clovis Perin Filho

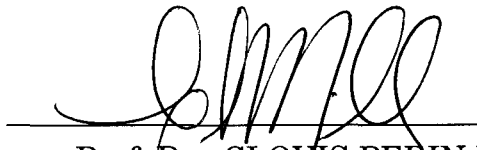
Dissertação apresentada ao Instituto de Matemática, Estatística e Ciências da Computação, UNICAMP, como requisito parcial para obtenção do título de Mestre em Matemática Aplicada.

Tese defendida e aprovada em 14 de março de 1995 pela Banca Examinadora composta pelos Profs. Drs.



Profa. Dra. ANA FRIEDLANDER

Prof. Dr. PETRONIO PULINO



Prof. Dr. CLOVIS PERIN FILHO

Resumo

Neste trabalho analisamos o problema de determinar a projeção de um vetor c no núcleo de uma matriz A. Apresentamos um método direto que permite o tratamento simultâneo da deficiência de posto e da esparsidade da matriz A. A relação entre o método proposto e os métodos de pontos interiores para programação linear recebe especial atenção.

Abstract

In this work we consider the problem of computing the projection of a vector c into the null space of a matrix A. We present a direct method that permits the treatment of both the numerical rank deficiency and the sparsity of the matrix A. Special attention is given to it's relation to interior points methods for linear programming.

SUMÁRIO

1. Introdução 5
2. Métodos de projeção 13
3. Fundamento do método 17
4. Interpretação geométrica 18
5. Cálculo e armazenamento das rotações 18
6. Descrição do método 20
7. Tratamento de colunas densas 21
8. Uma estrutura sem fill-in 22
9. Conclusão 24
10. Desenvolvimentos futuros 26
11. Apêndice – complexidade 33
12. Notas 35
13. Referências bibliográficas 36

1. INTRODUÇÃO

1.1. Motivação histórica[1]

A programação linear trata de problemas de minimizar ou maximizar uma função linear na presença de restrições lineares. Desde o desenvolvimento do método simplex por George B. Dantzig em 1947, a programação linear tem sido usada extensivamente em setores militares, industriais, governamentais e de planejamento urbano entre outros. A popularidade da programação linear pode ser atribuída a muitos fatores, entre eles sua capacidade de modelar problemas grandes e complexos, e de possibilitar aos usuários resolver problemas grandes em um tempo razoável.

Entretanto, essa possibilidade de se modelar e resolver problemas de grande porte mostrou-se não estar sempre disponível. A resolução de certos problemas grandes, originários de modelagens de problemas reais e importantes, não é satisfatória usando o método simplex.

A família de problemas que melhor ilustra este ponto trata da determinação da melhor composição de diversos produtos, onde cada um deles pode ser feito escolhendo-se alguns entre vários insumos possíveis. Se os produtos não têm nenhum insumo em comum o problema pode ser decomposto em problemas menores, e assim problemas muito grandes são resolvidos rapidamente. Entretanto, quando os diversos produtos têm vários insumos em comum, que são escassos, o problema passa a exigir um tempo muito grande para ser resolvido, e esse tempo aumenta exponencialmente à medida que aumenta o número de insumos em comum.

---

[1] Parte dessa subseção foi baseada em Gill *et al.* [5], Gonzaga [8], Bazaraa e Jarvis [2] e Goldfarb e Mehotra [6].

O tempo necessário para obter a solução desse tipo de problema é excessivo e ocorre em qualquer variante especializada do método simplex, de modo independente dos detalhes de implementação. Simplificadamente, o que ocorre é que os insumos em comum, usualmente chamados de *restrições de acoplamento*, estão presentes em quase todas as bases durante as iterações do método simplex, o que faz com que a inversa da base fique densa.

Por esse motivo vários problemas tiveram que ser simplificados de modo insatisfatório para que pudessem ser resolvidos, o que gerou a motivação e a necessidade para que se prosseguisse na pesquisa de métodos com melhor desempenho para certos problemas grandes. Não teria havido motivação real para isso se, *na prática*, o desempenho do método simplex tivesse sido satisfatório para resolver todos os problemas grandes que surgiam.

Dessa motivação e necessidade surgiram os métodos de pontos interiores.

Muitos pesquisadores, começando pelo próprio Dantzig, observaram a característica aparentemente insatisfatória do método simplex de caminhar ao longo da *fronteira* da região viável. Foram feitas tentativas de se desenvolver métodos práticos de programação linear que cruzassem o *interior* da região viável — por exemplo Neuman (1947), Hoffman *et al.* (1953), Thompkins (1955, 1957) e Frisch (1957). Tais métodos envolveram algumas vezes a aplicação de técnicas *não-lineares* à programação linear. Entretanto, nenhum desses métodos foi considerado, nem mesmo pelos próprio autores, como sendo competitivo em rapidez com o método simplex para a programação linear em geral.

Do lado teórico, a questão que se colocava era se existiria algum algoritmo polinomial para o problema da programação linear. Isso foi respondido em 1978 por Khachiyan [10], [11], que aplicou o método de Shor [18] e o mé-

todo elipsóide de Yudin e Nemirovski [22] ao problema da programação linear, baseado na geometria não-linear de elipsóides que se reduzem, e demonstrou a existência de uma limitação superior ao número de operações aritméticas necessárias para se obter uma solução ótima. A limitação[2], $O(n^4L)$, depende do número L, que é o número total de bits usados na descrição dos dados do problema. O método de Khachiyan, entretanto, mostrou-se não ser prático porque o número de iterações necessárias é quase sempre muito grande, próximo à limitação fornecida pela análise do pior caso, e porque os erros numéricos de arredondamento tendem a se acumular a cada iteração.

Em 1984 Karmakar [9] apresentou um novo algoritmo com tempo polinomial para a programação linear. O método de Karmakar se baseia em três pontos principais.

- Através de uma *transformação projetiva* de $\Re^n$ em $\Re^{n+1}$, a região viável em $\Re^n$, dada pela interseção de um espaço afim $Ax = b$ com o ortante positivo de $\Re^n$, é transformada em uma região em $\Re^{n+1}$, dada pela interseção de um subespaço vetorial (imagem do espaço afim através da transformação projetiva) com um *simplex* (imagem do ortante positivo de $\Re^n$ pela transformação projetiva). Essa transformação projetiva leva um ponto inicial do interior da região viável em $\Re^n$ no centro do simplex de $\Re^{n+1}$.
- A partir do centro do simplex e na direção da projeção do gradiente na região viável, é dado um passo cujo tamanho é limitado por uma *função potencial* que impede que o novo ponto fique próximo da fronteira.
- A cada iteração, um *outro tipo* de transformação projetiva, desta vez de $\Re^{n+1}$ em $\Re^{n+1}$, transforma o problema da iteração interior, levando novamente o ponto que foi calculado antes para o centro do simplex.

Karmakar demonstrou que seu método básico tem a mesma complexidade do método de Khachiyan, e apre-

[2] Ver nota 1 a respeito dessa complexidade e do número L.

sentou também uma variante, usando *atualizações de posto um*, que tem complexidade $O(n^{3.5}L)$.

Muito mais importante do que uma melhora na complexidade é o fato de que o método de Karmakar forneceu um enfoque novo, original e prático para resolver problemas de programação linear. O método foi bastante divulgado pela mídia e nos meios acadêmicos, nem tanto por seus méritos teóricos — inegáveis —, mas mais pela alegação de que era capaz de resolver problemas reais de grande porte muito mais rapidamente do que o método simplex. Tentativas iniciais para verificar essa alegação confirmaram que o método de Karmakar e métodos relacionados obtêm uma solução em um número muito pequeno de iterações. Esse número usualmente é menor que 50 e tende a crescer muito lentamente à medida que n, o tamanho do problema, cresce.

Entretanto, verificar as alegações a respeito do *tempo computacional* foi muito mais difícil. Os primeiros resultados publicados nessa direção foram os de Gill *et al.* [5] e Adler *et al.* [1], que indicaram que métodos baseados no enfoque de Karmakar eram viáveis.

Essa aparente discrepância entre um pequeno número de iterações *versus* um grande tempo computacional, observada nas primeiras implementações, se deve ao fato de que a operação de maior complexidade nos métodos de ponto interior é a projeção a cada iteração de um vetor no núcleo de uma matriz, e a menos que sejam tomados cuidados especiais com relação à preservação da esparsidade, projetar no núcleo de uma matriz esparsa pode ser equivalente a projetar no núcleo de uma matriz densa.

O trabalho de Karmakar suscitou de imediato uma reavaliação dos métodos *não-lineares* que já tinham sido tentados antes — por exemplo os métodos de função de penalização suave, que estavam então "fora de moda" devido ao mal-condicionamento que com freqüência ocor-

re próximo a uma solução ótima —, e foi mostrada por Gill *et al.* (op. cit.) a conexão entre o trabalho de Karmakar e o método da função de barreira logarítmica de Frisch (1955), assim como foi discutida sua relação com o método de centros de Huard (1967), por Gill *et al.* (op. cit.) e por Todd e Burrel [20].

Os métodos de pontos interiores são hoje bastante variados na sua essência e não é consensual ainda qual deles é *o melhor*, se é que isso de fato é possível. O trabalho de Gonzaga [8] é uma excelente apresentação de muitos desses métodos, especialmente daqueles onde os pontos interiores caminham ao longo de uma trajetória (vide Megiddo [13]), apesar de seu trabalho não tratar das aproximações da trajetória central por séries de potências (ver a esse respeito Monteiro *et al.* [15]).

O problema de projetar um vetor no núcleo de uma matriz tem aplicações em várias áreas e é parte importante de vários algoritmos. Entretanto, neste trabalho estaremos tratando apenas de suas aplicações na programação linear, e em especial em métodos de pontos interiores.

1.2. Projeções

Em cada passo dos métodos de pontos interiores para programação linear é necessário projetar um vetor c no núcleo de uma matriz AD, onde A é uma matriz mxn com $m<n$ de posto completo e D é uma matriz diagonal com elementos estritamente positivos e que muda a cada iteração. Apesar de alguns desses métodos não serem enunciados explicitamente em termos de projeções, isso sempre pode — e *deve* — ser feito, primeiro porque o método mais adequado de projeção varia de um problema para outro, e segundo porque as longas fórmulas usualmente apresentadas nos métodos de pontos interiores podem esconder a natureza de transformação

linear que a projeção tem — junto com as vantagens que se pode tirar de uma transformação linear.

Os métodos atualmente usados para determinar a projeção têm enfrentado alguns problemas sérios.

O primeiro está relacionado com a perda da esparsidade original comumente encontrada em programação linear, e a conseqüência é uma perda considerável no desempenho.

O segundo problema ocorre quando a seqüência de pontos interiores converge para um ponto para o qual a matriz diagonal D, associada a cada ponto interior, faz com que a matriz escalada AD se torne numericamente deficiente em posto.

Neste trabalho estudamos esses problemas e propomos um método estável para projetar vetores em núcleos de matrizes que permite também um tratamento adequado da esparsidade. O método não resolve nem é capaz de resolver nenhum problema intermediário de quadrados mínimos.

Mostraremos que a projeção de um vetor no núcleo de uma matriz, usualmente feita resolvendo-se um sistema linear, pode ser feita usando-se uma transformação ortogonal Q — e o que é mais importante — de um modo viável computacionalmente. Isso traz de volta ao debate essa família de métodos, e essa é talvez a contribuição mais importante desse trabalho.

1.3. Definição do problema

O *núcleo* de uma matriz A, definido por

$$N(A) = \{x \in \Re^n \mid Ax = 0\}$$

e a *imagem* de A^T, definida por

$$I(A^T) = \{x \in \Re^n \mid \exists y \in \Re^m : x = A^T y\},$$

formam subespaços vetoriais de $\Re^n$ que são ortogonais e cuja soma das dimensões é n. Assim, qualquer vetor de $\Re^n$ pode ser escrito de forma única como soma de dois

vetores ortogonais, um pertencente ao núcleo de A e o outro pertencente à imagem de A^T, o que nos permite definir as *projeções ortogonais* de um vetor c no núcleo de A e na imagem de A^T.

Definição: Dois vetores c_p e c_i são as *projeções ortogonais* de um vetor c respectivamente no núcleo de uma matriz A e na imagem de sua transposta A^T se

- $Ac_p = 0$ (isto é, c_p pertence ao núcleo de A),
- $c_i = A^T y$ para algum y (c_i pertence à imagem de A^T).
- $c = c_p + c_i$ (a soma dos dois é c).

A definição de projeção feita dessa forma permite estabelecer um critério de precisão numérica para as projeções ortogonais, associado a um dado ε, que não depende do método de projeção usado.

Definição: Dois vetores c_p e c_i são projeções ortogonais de um vetor c no núcleo de A e na imagem de A^T *com uma precisão* ε se

- $\|Ac_p\| < \varepsilon$,
- $\exists y: \|c_i - A^T y\| < \varepsilon$ e
- $\|c_p + c_i - c\| < \varepsilon$.

Quando uma definição é apresentada de uma determinada maneira é natural indagar porquê definir desse modo particular. Em nosso caso, o propósito é ter um critério que permita avaliar a precisão numérica de diferentes métodos de cálculo da projeção, quando usados em diferentes problemas.

Há outra pergunta que também surge naturalmente: *para que* serve a projeção?

A projeção no núcleo é crucial em métodos de pontos interiores. Vejamos por que.

Seja f uma função de valores reais definida em um espaço vetorial E e que queiramos minimizar em um

subconjunto. Chamaremos um vetor c de *direção de melhoria* em um ponto x se existir um intervalo real aberto $(0,\alpha)$ tal que para todo $\lambda \in (0,\alpha)$ tenhamos $f(x+\lambda c) < f(x)$.

O subconjunto a que essa minimização está restrita é chamado de conjunto das soluções viáveis ou simplesmente *conjunto viável*. Na forma padrão da programação linear ele é definido pelas soluções de um sistema linear em variáveis não negativas do tipo

$$Ax = b, \quad x \geq 0.$$

O conjunto viável é portanto um subconjunto das soluções do sistema $Ax = b$, que é um *espaço afim*.

Todo espaço vetorial E com produto interno define de modo natural uma norma de vetores, que por sua vez define uma métrica ou distância entre vetores, a qual finalmente define uma topologia em E. Enquanto subconjunto desse espaço topológico o espaço afim forma também um espaço topológico com a topologia induzida. O conjunto viável, por ser um subconjunto do espaço afim, tem portanto seu interior sempre bem definido, mesmo que eventualmente seja vazio.

Se x é viável e c' é uma direção de melhoria associada a um intervalo real aberto $(0,\alpha')$ chamaremos esse tipo de direção de *direção viável de melhoria* se pontos do tipo $(x + \lambda c')$ com $\lambda \in (0,\alpha')$ forem também pontos viáveis.

Vejamos em que isso implica. Se x é um ponto *viável e interior* e $x + \lambda c'$ é um ponto viável então $A(x + \lambda c') = b$. Como $Ax = b$ (pois x também é viável) concluímos que $Ac' = 0$, isto é, c' pertence ao núcleo de A. Em outras palavras toda direção viável de melhoria pertence necessariamente ao núcleo de A. Além disso, se existe um intervalo real aberto $(0,\alpha)$ para o qual c é direção viável de melhoria, e como x é um ponto interior em um espaço métrico, existe sempre um subintervalo $(0,\alpha')$ contido em $(0,\alpha)$ para o qual c' também é direção viável de melhoria em relação ao novo intervalo $(0, \alpha)$. Mas mais importante ainda, ago-

ra todo ponto do tipo ($x + \lambda c'$) não só é melhor que x como também é ponto interior do conjunto viável.

É isso que ocorre na minimização de uma função linear sujeita a restrições lineares. O gradiente da função linear $f(x) = cx$ fornece o vetor de melhoria $-c$ e a projeção ortogonal de $-c$ no núcleo de A fornece a direção viável de melhoria.

2. MÉTODOS DE PROJEÇÃO

Os métodos de projeção ortogonal de um vetor c no núcleo de uma matriz A podem ser divididos em três tipos: os que resolvem um sistema simétrico positivo-definido, os que resolvem um sistema triangular obtido por meio de transformações ortogonais, e os que evitam a etapa intermediária de resolução de sistemas, utilizando para isso apenas uma transformação ortogonal Q. Designaremos esses últimos por *métodos diretos*.

Faremos aqui uma descrição simplificada desses métodos. A descrição inicial de nosso método direto será sumária, para somente mais tarde detalharmos como torná-lo operacional.

2.1. Sistemas simétricos positivo-definidos

A fórmula para a projeção de c pode ser obtida diretamente das equações normais[3] como

$$c_p = \left[I - A^{\mathrm{T}}\left(AA^{\mathrm{T}}\right)^{-1}A\right]c$$

e computada resolvendo-se primeiramente o sistema simétrico positivo-definido $AA^{\mathrm{T}}y = Ac$ (usualmente pelo método de Cholesky) e depois calculando-se $c_p = c - A^{\mathrm{T}}y$.

[3] Ver [5] e [10] como referência para problemas de quadrados mínimos, e para transformações ortogonais de Givens, Householder e reflexões planas.

Entretanto devemos considerar três pontos interrelacionados.

Primeiro, mesmo quando A é esparsa a matriz AA^T pode não ser. Permutações nas linhas de A podem eventualmente aumentar a esparsidade do fator de Cholesky de AA^T.

Segundo, se A tiver alguma coluna densa a matriz AA^T será inevitavelmente densa e essa é a razão pela qual colunas densas são usualmente removidas de A para serem tratadas separadamente, usando por exemplo a fórmula de Sherman-Morrison-Woodbury [7].

Terceiro, essa remoção de colunas densas de A com freqüência leva a um decréscimo de posto, gerando instabilidade.

Além disso tudo, o número de condição de AA^T é o quadrado do de A, o que prejudica a resolução do sistema $AA^T y = c$.

2.2. Sistema triangular obtido com transformações ortogonais

Consideremos o problema de quadrados mínimos de determinar o vetor que minimiza a norma euclidiana

$$\|A^T y - c\|.$$

Se y^* é a solução desse problema, podemos calcular a projeção como $c_p = c - A^T y^*$.

Uma vez que

$$\|A^T y - c\| = \|Q(A^T y - c)\|$$

para qualquer matriz ortogonal Q, podemos aplicar uma seqüência de transformações ortogonais (Householder, Givens ou reflexões planas[4]) para formar uma matriz tri-

[4] Ver nota 2 a respeito da distincão reflexões *versus* rotações, e destas *versus* reflexões de Householder.

angular superior R em A^{T} e assim obter Q como produto (composição) dessas transformações.

À medida que aplicamos cada transformação a A^{T} também a aplicamos a c, obtendo assim $QA^{\mathrm{T}} = \left[\begin{smallmatrix} R \\ 0 \end{smallmatrix}\right]$ e $Qc = \left[\begin{smallmatrix} c_R \\ c_0 \end{smallmatrix}\right]$, onde R é uma matriz triangular superior, 0 é a matriz zero com dimensão compatível e o vetor Qc está separado dimensionalmente em duas partes c_R e c_0, de modo consistente com R e 0 respectivamente.

O problema de quadrados mínimos fica então transformado em minimizar

$$\left\|A^{\mathrm{T}}y - c\right\| = \left\|Q\left(A^{\mathrm{T}}y - c\right)\right\| = \left\|\left[\begin{smallmatrix} R \\ 0 \end{smallmatrix}\right]y - \left[\begin{smallmatrix} c_R \\ c_0 \end{smallmatrix}\right]\right\| = \left\|\begin{smallmatrix} Ry - c_R \\ -c_0 \end{smallmatrix}\right\|$$

e esse mínimo é alcançado com a solução do sistema $Ry^* = c_R$ com o valor $\left\|c_0\right\|$.

Apesar desse segundo método ser essencialmente diferente do primeiro, que resolve um sistema simétrico positivo-definido — e portanto não ter seu número de condição elevado ao quadrado —, mesmo assim ele sofre dos mesmos problemas de instabilidade numérica durante a resolução do sistema triangular. Isso ocorre porque a matriz triangular superior formada através da fatoração QR é igual ao fator de Cholesky da AA^{T}, uma vez fixados os sinais dos elementos na diagonal.

A razão básica pela qual ambos os métodos enfrentam problemas numéricos é que a matriz triangular R é usada para resolver um sistema linear — mas essa matriz freqüentemente se torna mal-condicionada. Nos métodos de pontos interiores isso ocorre por exemplo quando A é o resultado do escalamento de uma matriz fixa A' por uma matriz diagonal D, associada a um ponto interior, e a convergência desses pontos faz com que a matriz $A'D$ fique numericamente deficiente em posto (no que diz respeito à resolução de sistemas lineares).

2.3. Métodos diretos

Designamos por *métodos diretos de projeção* os métodos que calculam a projeção de um vetor no núcleo de uma matriz A sem resolver um sistema linear como etapa intermediária. Com isso não são calculados os *coeficientes* da combinação linear da colunas de A^T que fornecem a projeção no espaço ortogonal ao núcleo de A.

Os métodos diretos já são conhecidos, mas geralmente não são considerados práticos já que a matriz Q é quase sempre densa — de uma certa forma independentemente da esparsidade da matriz A — e com isso esses métodos são mencionados como uma curiosidade aplicável apenas a matrizes densas.

Entretanto, uma matriz ortogonal densa Q pode ser decomposta na chamada forma produto, isto é, como produto de transformações ortogonais mais simples. Tenta-se nesse caso obter a decomposição com um mínimo de transformações procurando reduzir o número total de operações aritméticas envolvidas.

Apresentaremos aqui um método desse tipo, fazendo a decomposição da matriz Q como produto de rotações de Givens e armazenando cada rotação dentro da própria estrutura de dados que fornece a matriz esparsa A.

Isso nem sempre é imediatamente possível — e no caso geral são necessárias permutações de linhas e colunas de A. Mostraremos, no entanto, que para uma família importante de estruturas esparsas podemos armazenar as rotações dentro da própria estrutura esparsa de A.

Além disso, mostraremos que a presença simultânea de linhas *e* colunas densas nesse tipo de estrutura não constitui problema. Assim, obtemos uma precisão numérica ainda maior pois não é necessário tratar separadamente as colunas densas.

3. FUNDAMENTO DO MÉTODO

Definimos

$$w = \begin{bmatrix} 0 \\ c_0 \end{bmatrix}.$$

Modificando um pouco o procedimento da fatoração QR temos

$$c_p = c - A^{\mathrm{T}} y^* = Q^{\mathrm{T}} Q\left(c - A^{\mathrm{T}} y^*\right) =$$
$$Q^{\mathrm{T}}\left(\begin{bmatrix} c_R \\ c_0 \end{bmatrix} - \begin{bmatrix} R \\ 0 \end{bmatrix} y^*\right) = Q^{\mathrm{T}} \begin{bmatrix} 0 \\ c_0 \end{bmatrix} = Q^{\mathrm{T}} w \qquad (1)$$

e na última igualdade R e y^* não aparecem.

Isso significa que para obter a projeção precisamos calcular w além de gerar e armazenar Q^{T} de modo eficiente sob a forma de produto de transformações mais simples. Significa também que não necessitamos nem de R nem de y^*.

Na fórmula (1) está implícito que $Ry = c_R$ admite teoricamente uma solução, isto é, que R não é singular.

No caso de projeções usadas em métodos de pontos interiores isso está sempre garantido, já que a matriz que está sendo fatorada é uma matriz escalada do tipo AD, com A de posto completo e D diagonal com elementos estritamente positivos. Desse modo a matriz triangular superior R é não singular — mas pode ser mal-condicionada. A condição de R, entretanto, em nosso caso não afeta a precisão numérica[5] com que os cálculos são feitos, já que R não é usada para resolver qualquer sistema linear. Tiraremos proveito disso em nosso método, pois não apenas não usaremos a matriz R como também sequer a calcularemos.

[5] A questão da precisão numérica *versus* mal-condicionamento é tratada na seção 10.2.

4. INTERPRETAÇÃO GEOMÉTRICA

Considere a descrição de $\Re^3$ pelos eixos ortogonais x, y e z. Para projetar um vetor c no plano xy tudo que precisamos fazer é igualar a zero a coordenada z de c.

Estendendo essa idéia para $\Re^n$, para projetar c em qualquer subespaço vetorial de $\Re^n$ de dimensão m, do tipo $E = \{x \in \Re^n \mid x_i = 0, i > m\}$, basta igualar a zero as últimas $(n-m)$ coordenadas de c. Em outras palavras, projetar em um subespaço desse tipo é trivial.

Suponhamos agora que o subespaço não seja desse tipo especial. Podemos transformar o subespaço original em um subespaço do mesmo tipo especial descrito antes, aplicando uma seqüência adequada de rotações, representada pela transformação ortogonal Q, e que é aplicada ao mesmo tempo ao vetor c. Projetar agora é novamente trivial nesse espaço transformado, isto é, igualamos a zero as $(n-m)$ últimas coordenadas de Qc, obtendo de imediato a projeção w de Qc neste novo subespaço.

Finalmente, para obter a projeção desejada c_p, transformamos o subespaço de volta ao subespaço original, aplicando a inversa de Q à projeção transformada w. Em termos de rotações, o que fazemos é aplicar a w a inversa de cada uma das rotações que gerou Q, e na ordem inversa em que elas foram geradas.

É exatamente isso que faremos.

5. CÁLCULO E ARMAZENAMENTO DAS ROTAÇÕES

Para tornar operacional nosso método precisamos poder calcular Q de modo eficiente, já que a matriz Q é

densa. Isso será feito representando e armazenando Q na forma de produto de rotações de Givens, isto é,

$$Q = \theta_1\theta_2\theta_3 \ldots \theta_k,$$

onde cada θ_i representa uma rotação de Givens dada na forma matricial por

$$\theta = \begin{pmatrix} 1 & & & & & \\ & \ddots & & & & \\ & & \cos(\alpha) & & \operatorname{sen}(\alpha) & \\ & & & \ddots & & \\ & & -\operatorname{sen}(\alpha) & & \cos(\alpha) & \\ & & & & & \ddots \\ & & & & & 1 \end{pmatrix}$$

para algum ângulo α.

Se θ é uma rotação que atua sobre as linhas i e k de A^T de modo a cancelar um elemento $A^T_{k,j}$, usualmente definimos

$$p = A^T_{i,j}, \qquad q = A^T_{k,j}$$

e computamos

$$\cos(\alpha) = \frac{p}{\sqrt{(p^2+q^2)}}, \qquad \operatorname{sen}(\alpha) = \frac{q}{\sqrt{(p^2+q^2)}}.$$

Esses cálculos podem ser instáveis quando $(p^2 + q^2)$ é pequeno. Ao invés disso definimos

$$f(t) = \frac{1}{\sqrt{(1+t^2)}}$$

e computamos

se $|p| < |q|$ então

$$t = \frac{p}{q}, \quad \operatorname{sen}(\alpha) = f(t), \quad \cos(\alpha) = t * f(t)$$

caso contrário

$$t = \frac{q}{p}, \quad \cos(\alpha) = f(t), \quad \operatorname{sen}(\alpha) = t * f(t).$$

Observe que

$$t = \frac{\operatorname{sen}(\alpha)}{\cos(\alpha)} \quad \text{ou} \quad t = \frac{\cos(\alpha)}{\operatorname{sen}(\alpha)}.$$

Em qualquer das duas possibilidades todas as operações aritméticas são efetuadas sem divisão por um número pequeno (em relação ao numerador).

Além disso, se acrescentarmos a cláusula:
{se $\cos(\alpha) < 0$ então $\cos(\alpha) := -\cos(\alpha)$ e $\operatorname{sen}(\alpha) := -\operatorname{sen}(\alpha)$},
a rotação preserva seu efeito de cancelamento e garantimos ainda que $\cos(\alpha)$ é não negativo — o que equivale a efetuar uma rotação de um ângulo $(\alpha+\pi)$.

Finalmente, podemos agora representar cada rotação de um modo mais simples, armazenando apenas $\operatorname{sen}(\alpha)$ na posição do elemento que foi cancelado. Mais tarde o co-seno pode ser reconstituído como

$$\cos(\alpha) = +\sqrt{1-\operatorname{sen}^2(\alpha)}.$$

Observe que $Q^{-1} = Q^{\mathrm{T}}$ já que Q é ortogonal, e que $Q^{\mathrm{T}} = \theta_k^{-1}\theta_{k-1}^{-1}\ldots\theta_1^{-1}$ e assim se mantivermos um registro da ordem em que cada rotação foi aplicada podemos mais tarde reconstituir completamente Q^{T}, já que

$$\theta^{-1} = \theta^{\mathrm{T}} = \begin{pmatrix} 1 & & & & & \\ & \ddots & & & & \\ & & \cos(\alpha) & & -\operatorname{sen}(\alpha) & \\ & & & \ddots & & \\ & & \operatorname{sen}(\alpha) & & \cos(\alpha) & \\ & & & & & \ddots \\ & & & & & 1 \end{pmatrix}.$$

6. Descrição do método

1. Forme um triângulo superior em A^{T} usando rotações de Givens mas

- compute e armazene cada rotação conforme descrito na seção 5,
- aplique as rotações ao vetor c à medida que cada uma for calculada, e

- ao final dessa etapa o vetor c está transformado em $\begin{bmatrix} c_R \\ c_0 \end{bmatrix}$.

2. Aplique Q^T em $w = \begin{bmatrix} 0 \\ c_0 \end{bmatrix}$, isto é, recupere cada θ_i^{-1} e aplique ao vetor w na ordem inversa em que cada rotação θ_i foi aplicada para gerar Q.

O vetor resultante $Q^T w$ é a projeção de c no núcleo de A.

7. TRATAMENTO DE COLUNAS DENSAS

Suponhamos que uma rotação $\theta_{i,j}$, que afeta apenas as linha i e j de A^T, esteja sendo usada para cancelar um elemento $A^T_{j,k}$. Chamaremos o elemento $A^T_{i,k}$ de *pivô* desse cancelamento.

Supondo que já tenham sido feitas permutações de linhas e colunas de A de modo a reduzir o "fill-in" intermediário, podemos supor que as colunas densas de A são as últimas k colunas.

O algoritmo usual para se formar um triângulo superior em A^T, por meio de rotações de Givens, consiste em usar $A^T_{1,1}$ como pivô para cancelar todos os elementos da primeira coluna, depois usar $A^T_{2,2}$ como pivô e assim por diante.

Faremos quase exatamente assim. Para cada i os cancelamentos são feitos em três etapas:

1. Use $A^T_{i,i}$ como pivô para cancelar os elementos da coluna i, exceto os das k últimas linhas densas.

2. Use $A^T_{n-k+1,i}$ como pivô para cancelar os elementos na coluna i das k últimas linhas densas.

3. Use $A^T_{i,i}$ para cancelar $A^T_{n-k+1,i}$ mas não calcule a nova linha i porque ela não será mais necessária. Qualquer que seja a estrutura de esparsidade que a linha i tinha

antes desse passo, apenas esses elementos esparsos serão necessários para calcular a nova linha n-k+1.

Ao final obtemos um armazenamento completo e econômico da Q em termos das rotações θ_i mas não obtemos a matriz triangular superior R, o que não tem problema já que essa matriz não é necessária, conforme foi mostrado, — e mais, a matriz R seria uma matriz densa.

8. UMA ESTRUTURA SEM FILL-IN

Apresentaremos agora uma estrutura que não gera "fill-in" adicional no método apresentado, isto é, o número de operações aritméticas envolvidas é proporcional ao número de elementos não nulos na estrutura, e as rotações podem ser armazenadas na própria estrutura de esparsidade dada.

Essa estrutura é não trivial e corresponde à modelagem em programação linear de problemas de otimização da produção de vários itens que compartilhem recursos limitados (problemas de multi-produto com restrições de estoque). Ela é conhecida como estrutura bloco-angular com banda-linha e banda-coluna densas.

8.1. Descrição da estrutura

Uma matriz A tem estrutura *bloco-angular* com *banda-linha* e *banda-coluna* densas se existirem partições
$\{(1,i_1),(i_1+1,i_2),\ldots,(i_k+1,m)\}$ e
$\{(1,j_1),(j_1+1,j_2),\ldots,(j_k+1,n)\}$
tais que $A_{i,j} \neq 0$ somente quando uma das três condições abaixo ocorrer:

- $\{(i_{p-1}+1 \leq i \leq i_p)$ e $(j_{p-1}+1 \leq j \leq j_p)\}$ para algum $p \leq k$ (nesse caso $A_{i,j}$ pertence ao bloco p) ou

- $i_k + 1 \leq i \leq m$ (nesse caso $A_{i,j}$ pertence à banda-linha) ou
- $j_k + 1 \leq j \leq n$ (nesse caso $A_{i,j}$ pertence à banda-coluna).

8.2. Especialização do algoritmo

Para projetar um vetor c no núcleo de uma matriz A com estrutura bloco-angular com banda-linha e banda-coluna densas usaremos um procedimento muito parecido com o que foi desenvolvido na seção 7. Para projetar procedemos assim:

para cada um dos k blocos de A^{T}

para cada coluna envolvida nesse bloco

1. use o elemento da diagonal do bloco como pivô de cancelamento dos demais elementos dessa coluna;
2. use o primeiro elemento da banda-linha densa para cancelar os demais elementos dessa coluna;
3. use o pivô de (1.) para cancelar o pivô de (2.) mas não calcule a nova linha correspondente ao pivô de (1.) porque ela não será mais necessária. Somente a antiga linha desse pivô será necessária para calcular a nova linha correspondente ao pivô de (2.).

Lembre-se que

- a cada cancelamento efetuado, a correspondente rotação é armazenada no elemento cancelado sob forma de um número real, conforme descrito na seção 5;
- cada rotação usada em um cancelamento é aplicada, logo que for calculada, tanto ao vetor c quanto às colunas de A^{T} correspondentes à banda-linha de A.

Ao final dessa etapa as rotações usadas geraram implicitamente a transformação ortogonal Q, que satisfaz $QA^{\mathrm{T}} = \begin{bmatrix} R \\ 0 \end{bmatrix}$ (apesar de R não ser calculada), e o vetor c

é transformado no vetor $Qc = \begin{bmatrix} c_R \\ c_0 \end{bmatrix}$. Recordando que tínhamos definido $w = \begin{bmatrix} 0 \\ c_0 \end{bmatrix}$, devemos agora aplicar Q^T a w.

Como a ordem em que as rotações foram aplicadas esteve sob controle, podemos aplicar o passo (2.) da seção 6 na seqüência correta, obtendo finalmente a projeção.

Apresentamos no apêndice uma análise de complexidade de nosso método para essa estrutura.

8.3. Generalização

Em 8.2 supusemos que cada bloco é denso. Podemos generalizar essa estrutura admitindo que cada bloco por sua vez tem também a mesma estrutura bloco-angular com banda-linha e banda-coluna densas. Essa nova estrutura também não gera "fill-in". O argumento é uma repetição do que acabou de ser exposto.

Essa nova estrutura tampouco é trivial e corresponde à modelagem em programação linear de problemas de otimização da produção de vários itens que compartilham recursos limitados, só que agora otimizando por vários períodos — o chamado multi-produto por multi-período.

9. Conclusão

Apresentamos um método de projeção de um vetor no núcleo de uma matriz que é direto e viável computacionalmente, já que com ele é possível tratar uma classe importante de estruturas sem que haja perda de esparsidade.

No contexto de projeções o método apresentado é mais estável que o método de Cholesky, uma vez que depende do número de condição da matriz A e não de seu quadrado, além de não resolver sistemas lineares

intermediários; é também mais estável que a fatoração QR, já que a efetua essencialmente as mesmas operações aritméticas porém não resolve sistemas lineares.

Entretanto é crucial fazer uma ressalva. Os argumentos apresentados aqui *não significam* que nosso método seja preferível ao método de Cholesky ou ao método do gradiente conjugado. Estrutura e esparsidade continuam sendo os pontos principais para se optar por um ou outro método.

Como contra-exemplo imediato, quando m, o número de linhas de uma matriz esparsa A, é muito menor que o número n de colunas, o tempo de resolução do sistema $AA^{T}y = Ac$ depende apenas de m^3, e a própria formação da matriz AA^{T} pode ser feita com eficiência usando-se a estrutura de esparsidade de A. Além disso, as heurísticas já existentes para reordenação de linhas — e correspondente aumento da esparsidade do fator de Cholesky R —, não podem nem devem ser desprezadas.

Prosseguindo ainda mais nessa linha, em problemas de fluxos em redes usualmente o número de linhas da matriz de incidência A é de $O(m)$, enquanto que o número de colunas é de $O(m^2)$, o que nos dá uma diferença de *ordem de grandeza* entre o número de linhas e o de colunas. Além disso a esparsidade em uma rede é tão grande que formação de AA^{T} é muito "barata" computacionalmente. Tudo isso leva a um tempo computacional para a projeção que praticamente descarta o uso de nosso método nesses problemas.

Mais ainda, em problemas de multi-fluxo em redes mesmo o método de Cholesky não é tão competitivo quanto o método do gradiente conjugado, novamente por razões ligadas à estrutura e à esparsidade.

O que os exemplos (a estrutura sem fill-in) e contra-exemplos (matrizes com o número de linhas muito menor que o número de colunas) apresentados indicam é

que não existe um método de projeção que seja *o melhor* método de um modo geral, apesar de alguns métodos serem mais estáveis que outros. Isso sugere que uma implementação mais detalhada de um método de pontos interiores deve dispor de vários métodos de projeção, cuja escolha seja feita conforme o problema que estiver sendo resolvido.

10. DESENVOLVIMENTOS FUTUROS

Apresentaremos agora alguns tópicos promissores que foram abordados durante a elaboração desse trabalho e que indicam possíveis linhas a serem melhor investigadas.

10.1. Uma escolha melhor de μ

Logo na introdução enfatizamos que a natureza de transformação linear que a projeção tem muitas vezes passa despercebida na prática, talvez devido às complexas fórmulas utilizadas para projetar vetores quando isso é feito através da resolução de sistemas lineares. Essa linha surgiu quando, analisando o método de Monteiro e Adler [14] e re-escrevendo as fórmulas apresentadas em termos do operador projeção, observamos que a escolha do parâmetro μ — crucial nesse método e em outros métodos primal-dual de pontos interiores — pode ser feita de um modo melhor, não heurístico, isolando esse parâmetro real μ usando o fato de que a projeção em um núcleo é uma transformação linear.

Isso sugere seria oportuno aplicar a mesma idéia em outros métodos similares.

10.2. Precisão numérica *versus* mal-condicionamento

Desenvolvemos, nas seções 5, 6 e 7, um método numericamente preciso para projetar um vetor no núcleo de uma matriz — o que pode não ocorrer se usarmos o método de Cholesky ou a fatoração QR.

Entretanto, esse resultado preciso pode mascarar um mal-condicionamento *do problema da projeção*. Podemos obter com precisão a projeção de um vetor no núcleo de uma matriz — e no entanto essa projeção pode não ter significado algum!

Conforme vimos na introdução, a precisão com que calculamos as projeções ortogonais c_p e c_i de um vetor c no núcleo de A e na imagem de A^T é dada pelo menor ε que satisfizer

- $\|Ac_p\| < \varepsilon$,
- $\exists y: \|c_i - A^T y\| < \varepsilon$ e
- $\|c_p + c_i - c\| < \varepsilon$.

Entretanto, a precisão numérica em si não é suficiente para avaliarmos o quão adequadas numericamente são as projeções ortogonais. Podemos obter projeções ortogonais c_p e c_i de um vetor c, que satisfazem as condições acima, mas que são soluções para um problema de projeção que é numericamente instável por si mesmo, ou *mal-condicionado*, isto é, um problema no qual uma pequena variação nos dados do problema gera uma grande variação nos resultados.

Aqui é importante enfatizar uma diferença conceptual. Precisão numérica é atributo de um *método*, enquanto que mal-condicionamento é atributo de um *problema*.

Para tratar da questão do condicionamento de um problema de projeção de um vetor no núcleo de uma matriz necessitaremos de algumas definições.

Vamos convencionar a projeção de c no núcleo de A por $c_p = P_{N(A)}c$.

- Definimos $\overline{A} = A + \delta_1$ e $\bar{c} = c + \delta_2$ como *perturbações de norma* δ de A e c respectivamente, se A e δ_1 são matrizes mxn, c e δ_2 são vetores de dimensão n, e δ é um número real positivo tal que $\|\delta_1\| < \delta$ e $\|\delta_2\| < \delta$.
- Se $c_p = P_{N(A)}c$, $\bar{c}_p = P_{N(\overline{A})}\bar{c}$ e $c_p \neq 0$ então *o erro relativo* associado a essas perturbações de norma δ é o número $E_\delta = \dfrac{\|\bar{c}_p - c_p\|}{\|c_p\|}$.
- Designemos por Ψ_A^δ e Ψ_c^δ os conjuntos formados por *todas* as perturbações de norma δ de A e c respectivamente. Definimos o δ-*número de condição* associado à projeção de c no núcleo de A como sendo o número

$$\aleph_A^c(\delta) =$$

$$\sup\left\{\frac{\|\bar{c}_p - c_p\|}{\|c_p\|} \;\middle|\; \bar{c} \in \Psi_c^\delta, \overline{A} \in \Psi_A^\delta, \bar{c}_p = P_{N(\overline{A})}\bar{c}, c_p = P_{N(A)}c\right\}.$$

As definições acima generalizam a definição de erro relativo apresentada no trabalho de Ding [3], que fixa *a priori* a direção das perturbações.

10.3. A arbitragem da perda de posto

A perda estritamente numérica de posto (para fins de resolução de sistemas) da matriz A tem freqüentemente levado a métodos heurísticos que estipulam, no decorrer de um algoritmo de ponto interior, o decréscimo no posto de A, quando instabilidades numéricas chegam ao ponto de impedir a resolução do sistema linear.

Essa arbitragem pode ser — ou não — adequada.

Mostramos aqui que a resolução de sistemas lineares é apenas uma etapa intermediária — dispensável —

para projetar um vetor. Isso invalida critérios de arbitragem para decréscimo numérico no posto de A que se baseiam na dificuldade de resolver um sistema linear.

Entretanto, a projeção de um vetor no núcleo de uma matriz pode ser um problema mal-condicionado. Uma perturbação de norma δ, associada exclusivamente à precisão do computador onde estivermos implementando o método de projeção, pode ser tal que o correspondente erro relativo seja grande. Nessas condições a arbitragem da perda de posto é necessária já que a projeção deixa de ter significado.

Existe uma outra razão — intrigante — pela qual a arbitragem da perda de posto pode ser indispensável.

Se o operador P, que a cada par (A,c) associa a projeção $P_{N(A)}c$ de c no núcleo de A, for contínuo com a topologia métrica usual, então em uma vizinhança de A toda matriz tem o mesmo posto de A. Entretanto a continuidade de P é uma condição *necessária* — e não apenas suficiente. Denotando por $r(A)$ o posto de A, temos que se $A_k \to A$ e $c_k \to c$, então $r(A_k) = r(A)$ para k suficientemente grande *se e somente se* $P_{N(A_k)}c \to P_{N(A)}c$. Esse resultado não trivial é corolário do teorema famoso, primeiro apresentado por Penrose [16] e depois por Stewart [19], que estabelece as condições necessárias e suficientes para a continuidade da projeção de um vetor em um espaço afim (vide Ding [3]).

Como aplicação imediata desse resultado, consideremos a resolução do problema de programação linear

$$\min cx$$
$$\text{s.a. } Ax = b,\ x \geq 0$$

utilizando um método de ponto interior com escalamento primal-dual e suponhamos que o ótimo desse problema seja único e primal degenerado. A seqüência de matrizes diagonais D_k, associada ao escalamento primal-dual a

cada iteração, converge para a matriz diagonal D, correspondente ao escalamento primal-dual associado *ao ponto ótimo*. Como o ótimo é único e primal degenerado, a matriz AD tem posto menor que o de A. Assim, *se* a seqüência de vetores, formada pelas projeções de cada um dos vetores escalados $D_k c$ no núcleo da matriz AD_k, *convergir* para a projeção do vetor Dc no núcleo de AD (o que é usual), *então* o posto da matriz AD_k *necessariamente decresce* a partir de uma determinada iteração K, isto é,

- $AD_k \rightarrow AD$
- $D_k c \rightarrow Dc$

logo

- $P_{N(AD_k)} D_k c \rightarrow P_{N(AD)} Dc$

se e somente se

- $\exists K: \quad k > K \Rightarrow r(AD_k) = r(AD) < r(A).$

Assim, ficam abertos para pesquisa posterior os seguintes tópicos:

- em que condições é realmente necessária a convergência de $P_{N(AD_k)} D_k c$ para $P_{N(AD)} Dc$;
- que critério usar para a arbitragem da perda de posto quando isso for necessário;
- a arbitragem "precoce" da perda de posto acelera a convergência?

10.4. Transformações ortogonais pela direita

Se y^* é a solução do sistema $Ry = c_R$ e Ω é qualquer transformação ortogonal aplicável a y^* então podemos definir $y' = \Omega y^*$ e assim, construindo Q de modo que $QA^{\mathrm{T}}\Omega^{\mathrm{T}} = \begin{bmatrix} R \\ 0 \end{bmatrix}$, e modificando a fórmula (1), temos que

$$c_p = c - A^{\mathrm{T}} y' = Q^{\mathrm{T}} Q \left(c - A^{\mathrm{T}} \Omega^{\mathrm{T}} y^* \right)$$
$$= Q^{\mathrm{T}} \left(\begin{bmatrix} c_R \\ c_0 \end{bmatrix} - \begin{bmatrix} R \\ 0 \end{bmatrix} y' \right) = Q^{\mathrm{T}} \begin{bmatrix} 0 \\ c_0 \end{bmatrix} = Q^{\mathrm{T}} w.$$

Isso significa que a aplicação de transformações ortogonais à direita de A^{T} não prejudica o esquema de projeção

direta. Por outro lado, a transformação Ω pode ser construída, como *pré-processamento*, de modo a *aumentar* a esparsidade de A^T, por exemplo no caso de matrizes bloco-angulares com banda-linha e banda-coluna densas. Como estamos interessados em métodos de pontos interiores, conseguimos desse modo que a aplicação de uma única transformação ortogonal à direita reduza o número de operações aritméticas em todas as demais projeções em núcleo.

10.5. Permutações

Suponhamos que A seja uma matriz esparsa e consideremos o problema de obter permutações de linhas e colunas de A que reduzam o número de operações aritméticas em nosso método, mas de um modo que seja computacionalmente viável. Escolhemos propositadamente o termo "viável", para permitir tanto heurísticas como métodos polinomiais exatos.

Analisemos dois problemas aparentemente similares.

Primeiro, considere o problema clássico de minimizar o "fill-in" do fator de Cholesky R de AA^T. Após remover colunas densas para que sejam mais tarde tratadas à parte (apesar de que quando as colunas densas são removidas de A a matriz restante freqüentemente tem posto menor), devemos encontrar uma permutação de linhas de A que minimize o "fill-in" de R.

Esse problema é NP-completo, como foi demonstrado por Yannakakis [21], mas há alguns métodos heurísticos de ordenação como o *minimum degree* e o *minimun local fill-in* que funcionam bem na prática (ver a esse respeito Rose [17]).

Segundo, quando a fatoração QR clássica é usada, temos que efetuar também permutações de colunas de A, a fim de minimizar o "fill-in" intermediário, agravando ainda mais esse problema que já era NP-completo. O

trabalho de A. George e E. Ng [4] fornece um mecanismo para estudar a conexão entre uma boa ordenação de linhas e uma boa ordenação de colunas que sugere algumas direções para se criar heurísticas promissoras. Apesar deles fornecerem também um algoritmo de dissecção aninhada, esse algoritmo é muito específico para alguns tipos de estruturas que são encontradas com mais freqüência em problemas de quadrados mínimos.

Nosso problema, entretanto, no que diz respeito a uma boa ordenação de linhas e colunas, é diferente em essência dos dois problemas previamente discutidos — e é difícil enfatizar o quão diferente ele é.

Quando tentamos aumentar a esparsidade do fator de Cholesky R de AA^{T}, ou quando tentamos *também* minimizar o fill-in intermediário durante a fatoração QR clássica, estamos interessados na eficiência em resolver o sistema linear associado com R.

Em nosso método estamos interessados exclusivamente com o fill-in intermediário na parte *inferior* de A^{T} abaixo da matriz triangular R que será formada, sem qualquer consideração pela esparsidade de R. Essa é uma diferença de essência e não de detalhes.

Para ilustrar ainda mais esse ponto, não conhecemos ainda qualquer prova ou sugestão de que *esse* problema seja NP-completo. Infelizmente, por outro lado tampouco existe qualquer heurística especializada para esse problema para o caso geral de uma matriz esparsa.

O modo com que as colunas densas foram tratadas na seção 7, assim como o modo como a banda-coluna densa de A foi cancelada, mostram como essa é uma diferença em essência.

Dessa forma, é possível que uma boa (ou ótima) ordenação de linhas e colunas possa ser encontrada de um

modo viável — mesmo quando não for possível aumentar a esparsidade de R.

APÊNDICE – COMPLEXIDADE

Para avaliarmos a complexidade de nosso método, especificamente para a estrutura descrita em 8.2, vamos supor simplificadamente que

- cada bloco denso tem p linhas e $2p$ colunas,
- existem p blocos,
- a banda-linha tem p linhas, e
- a banda-coluna tem p colunas.

Dessa forma, o número total de elementos não nulos da estrutura descrita é de $5p^3 + p^2$.

Na fatoração de Cholesky da matriz simétrica positivo-definida associada a essa estrutura, se não tratássemos à parte a banda-coluna, a complexidade da fatoração seria de $O(p^6)$, já que estaríamos trabalhando com uma matriz densa de $O(p^2)$ linhas por $O(p^2)$ colunas. Supondo que a banda-coluna é tratada à parte, é necessário obter a fatoração de cada um dos p blocos, o que exige um esforço computacional de $O(p^3)$ para cada bloco e assim a complexidade da fatoração dos p blocos é de $O(p^4)$; a mesma complexidade é necessária para calcular a parte do fator de Cholesky gerado pela banda-linha. Com isso a complexidade da fatoração de Cholesky para essa estrutura é de $O(p^4)$.

Em nosso método e para essa estrutura, na formação da matriz triangular superior R_k de cada um dos k blocos densos de A^T são necessários $2p - j$ cancelamen-

tos por coluna, $1 \leq j \leq p$ (estamos trabalhando com a matriz transposta), e para cada cancelamento é necessário efetuar a rotação de duas linhas com $p-j+1$ elementos cada. Como existem p blocos, isso nos leva a um número de total de operações[6] de

$$p\sum_{j=1}^{p}(2p-j)(p-j+1)=\frac{5p^4+3p^3-2p^2}{6}.$$

Devemos considerar ainda as operações relacionadas à banda-linha e à banda-coluna de A.

- Para cada cancelamento foi necessário efetuar a rotação de duas linhas da banda-coluna de A^{T} com p elementos cada, o que nos dá um adicional de $2p^2$ rotações por bloco ou $2p^3$ rotações considerando todos os blocos. Na banda-linha de A^{T} , para cada bloco k e cada coluna j temos que efetuar p cancelamentos, cada um envolvendo a rotação de duas linhas de $(p-j)$ elementos, correspondente ao bloco k, e a rotação de duas linhas de $(p-k)p$ elementos, correspondentes aos demais blocos. Assim, temos que adicionar mais

$$p\sum_{j=1}^{p}(p-j)+p\sum_{k=1}^{p}p(p-k)=\frac{p^4-p^2}{2}\ \text{operações.}$$

O total de operações aritméticas envolvidas, portanto, é de $O(p^4)$, igual à da fatoração de Cholesky.

[6] Estamos considerando tanto o cálculo dos números cos(α) e sen(α) para cada rotação, como a aplicação de uma rotação a dois números reais, como sendo uma única operação.

NOTAS

1. A rigor, as complexidade dos algoritmos de Khachiyan e de Karmakar são respectivamente $O(n^4L^2 \text{ lnL lnlnL})$ e $O(n^{3.5}L^2 \text{ lnL lnlnL})$. Além disso, L é definido como

$$L = \ln(1 + |D_{max}|) + \ln(1 + \alpha), \quad \text{onde}$$

$$D_{max} = \max \{|\det(X)| : X \text{ é submatriz quadrada de } A\} \quad \text{e}$$

$$\alpha = \max \{|c_i|, |b_i| \ i = 1, \ldots, n\}.$$

   Entretanto L é majorado pelo número total de bits que descrevem os dados do problema (Karmakar [7]).

2. As reflexões planas, assim como as rotações planas (ou de Givens) — e ao contrário das transformações de Householder —, atuam apenas sobre duas linhas de uma matriz A. As representações matriciais da reflexão e da rotação são dadas respectivamente por

$$\begin{pmatrix} \cos(\alpha) & \text{sen}(\alpha) \\ \text{sen}(\alpha) & -\cos(\alpha) \end{pmatrix} \quad \text{e} \quad \begin{pmatrix} \cos(\alpha) & \text{sen}(\alpha) \\ -\text{sen}(\alpha) & \cos(\alpha) \end{pmatrix}.$$

   Cabe observar também que uma rotação plana e uma reflexão plana requerem exatamente o mesmo esforço computacional; entretanto, uma permutação (que também é uma transformação ortogonal) de duas linhas de uma matriz é uma caso particular de uma reflexão plana, e não de uma rotação.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] I. Adler, M. Resende, G. Veiga and N. Karmakar, "An implementation of Karmakar's algorithm for linear programming", *Mathematical Programming* 44 (1989) 297–335. Errata in *Mathematical Programming* 50 (1991) 415.

[2] M.S. Bazaraa and J.J. Jarvis, *Linear programming and network flows*, John Wiley & Sons, 1977.

[3] J. Ding, "Perturbation analysis for the projection of a point to an affine set", *Linear Algebra and its applications* 191 (1993) 199–212, Elsevier Science Publishing.

[4] J.A. George and E.G.Y. Ng, "On row and column orderings for sparse least square problems", *SIAM Journal of Numerical Analysis* 20-2 (1983) 326–344. Erratum in 20-4 (1983) 872.

[5] P.E. Gill, W. Murray, M.A. Saunders, J.A. Tomlin and M.H. Wright, "On projected Newton barrier methods for linear programming and an equivalence to Karmakar's projective method", *Mathematical Programming* 36 (1986) 183-209.

[6] D. Goldfarb and S. Mehotra, "A relaxed version of Karmakar's method", *Mathematical Programming* 40 (1988) 289–315.

[7] G.H. Golub and C.F. Van Loan, *Matrix Computations*, 2nd ed., The Johns Hopkins University Press, 1989.

[8] C.C. Gonzaga, "Path-following methods for linear programming", *SIAM Review* 34-2 (1992) 167–224.

[9] N. Karmakar, "A new polynomial time algorithm for linear programming", *Combinatorica* 4 (1984) 373–395.

[10] L.G. Khachiyan, "A polynomial algorithm in linear programming", *Soviet Mathematics Doklady* 20 (1979) 191–194.

[11] L.G. Khachiyan, "Polynomial algorithms in linear programming", *USSR Computational Mathematics and Mathematical Phisics* 20 (1980) 53–72.

[12] C.L. Lawson and R.J. Hanson, *Solving Least Squares Problems*, Prentice-Hall, 1974.

[13] N. Megiddo, "Pathways the optimal set in linear programming" in: N. Megiddo, ed., *Progress in Mathematical Programming* (Springer, New York, 1988) pp. 131–158.

[14] R. Monteiro and I. Adler, "Interior path-following primal-dual algorithms, part I: Linear programming", *Mathematical Programming* 44 (1989) 27–41.

[15] R. Monteiro, I. Adler and M. Resende, "A polynomial-time primal-dual affine scaling algorithm for linear and convex quadratic programming and it's power series extension", *Mathematics of Operations Research* 15-2 (1990) 191–213.

[16] R. Penrose, "A generalized inverse for matrices", *Proc. Cambridge Philos. Soc.,* 51 (1955) pp. 406–413.

[17] D. J. Rose, "A graph-theoretic study of the numerical solution of sparse positive definite systems of linear equations" in R. C. Read, ed. *Graph theory and computing*, Academic Press, New York, 1973.

[18] N. Shor, "Utilization of the operation of space dilation in the minimization of convex functions", *Cybernetics* 6 (1970) 7–15.

[19] G.W. Stewart, "On the continuity of generalized inverse", *SIAM Journal of Applied Mathematics,* 17 (1963) 33–45.

[20] J.M. Todd and B.P. Burrel, "An extension of Karmakar's algorithm for linear programming using dual variables", *Algorithmica* 1 (1986) 409–424.

[21] M. Yannakakis, "Computing the minimum fill-in is NP-complete", *SIAM Journal of Algorithms and Discrete Methods,* 1 (1981) 77–99.

[22] D. Yudin and A. Nemirovskii, "Informational complexity and efficient methods for the solution of convex extremal problems", *Matekon* 13-2 (1976) 3–25.