Implementação do Método Simplex

Bruno Sesso 8536002
Gustavo Estrela de Matos 8536051

15 de Junho de 2015

1 Introdução

1.1 Apresentação do problema

Os problemas de Programação Linear (PL) são casos específicos de otimização combinatória em que a função objetivo e as restrições são ambos lineares. Portanto a função objetivo é da forma $c^T x$ e as restrições são da forma $a_i^T x \geq b_i$ ou $a_i^T x \leq b_i$, com $c, x, a_i \in \mathbb{R}^n$ e $b_i \in \mathbb{R}$.

Multiplicando por -1 todas as restrições da forma $a_i^T x \geq b_i$, podemos escrever qualquer PL como:

$$\begin{array}{rl} \text{minimizar} & c^T x \\ \text{sujeito a} & Ax \leq b, \\ & A \in R^{m \times n} \text{ e } b \in \mathbb{R}^m. \end{array}$$

Também é possível mostrar que qualquer PL pode ser escrito na forma:

$$\begin{array}{rl} \text{minimizar} & c^T x \\ \text{sujeito a} & Ax = b, \\ & x \geq 0 \text{ [1].} \end{array}$$

Se for escrito dessa maneira, dizemos que o problema está no formato padrão. Adotaremos esse formato durante todo o trabalho.

Se vale que $Ax^1 = b$ e $x^1 \geq 0$ dizemos que x^1 é um ponto viável. O conjunto $P = \{x | Ax = b, x \geq 0\}$ de todos os pontos viáveis é chamado conjunto viável.

Uma solução ótima do problema é um ponto $x^1 \in P$ que minimiza [1] a função objetivo c. Se x^1 existe, dizemos que o custo ótimo é $c^T x$. Se x^1 não existe, ou não existem pontos viáveis ($P = \emptyset$), ou podemos diminuir o custo o quanto quisermos e dizemos que o custo ótimo é $-\infty$.

1.2 Objetivos do trabalho

Neste trabalho, temos o objetivo de desenvolver, na linguagem Octave, o algoritmo simplex para resolver problemas de Programação Linear.

[1] Se o interesse for maximizar $c^T x$, podemos simplismente conseguir um problema equivalente em que o objetivo seja minimizar $-c^T x$.

2 Conceitos fundamentais

Antes de introduzirmos o funcionamento do nosso algoritmo, precisamos definir alguns conceitos que são fundamentais para garantir sua corretude.

Seja o nosso problema de Programação Linear o seguinte:

$$\begin{aligned} \text{minimizar} \quad & c^T x \\ \text{sujeito a} \quad & Ax = b \\ & x \geq 0 \\ \text{com} \quad & c, x \in \mathbb{R}^n,\ A \in \mathbb{R}^{m \times n} \text{ e } b \in \mathbb{R}^m. \end{aligned}$$

Além disso, vamos usar a notação a_i para a i-ésima linha de A e A_i para a i-ésima coluna de A.

2.1 Restrições e degenerecência

Uma restrição $a_i^T x \geq b_i$ (ou $a_i^T x \leq b_i$), com $a_i \in \mathbb{R}^n$ e $b_i \in \mathbb{R}$, é uma *restrição ativa* em um ponto $x^1 \in \mathbb{R}^n$ se $a_i^T x^1 = b_i$. Uma restrição de igualdade é sempre ativa. Um conjunto de restrições será dito LI se os vetores a_i correspondentes forem LI.

Diremos que x^1 uma solução viavel básica é *degenerada* se existem mais de n restrições ativas LI nesse ponto. Como as m restrições de igualdade são sempre cumpridas, temos que as soluções básicas degeneradas possuem mais do que $n - m$ componentes nulas, enquanto que as não degeneradas possuem exatamente $n - m$.

2.2 Soluções Viáveis Básicas

Dizemos que um ponto $x \in \mathbb{R}^n$ do conjunto viável P é uma *solução viável básica*, se existem n restrições ativas em x que são LI. Note que para problemas no formato padrão, existem sempre m restrições ativas LI vindas de $Ax = b$, e as outras $n - m$ vem, necessariamente de $x \geq 0$. Portanto, uma solução viável básica possui ao menos $n-m$ componentes nulas.

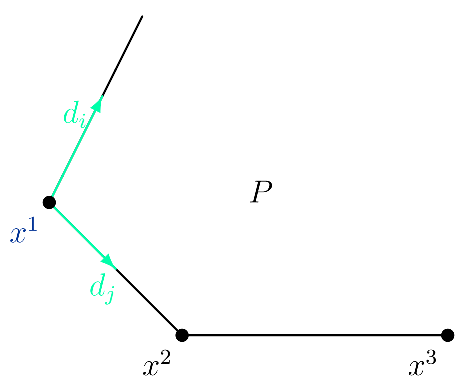
Se x^1 é uma solução básica não degenerada e seja $B(1), ..., B(m)$ os índices das componentes não nulas de x. A matriz $B = \left[A_{B(1)}, ..., A_{B(m)}\right]$ é chamada *matriz básica* associada a x^1.

Se o conjunto P tem uma solução viável básica, então ou o custo ótimo é $-\inf$ ou existe $x^1 \in P$ solução viável básica que é ótimo, ou seja, o custo de qualquer ponto do conjunto viável é maior ou igual do que o custo de x^1. Portanto, na solução de um PL com ao menos uma solução viável básica, podemos limitar a esses elementos a nossa busca por um ponto de custo ótimo [1].

2.3 Direções básicas

Se x^1 é um solução viável básica de P, com índices básicos $B(1), ..., B(m)$. Dizemos que $d \in \mathbb{R}^n$, tal que $d_j = 1$, $Ad = 0$ $(A(x + \theta d) = b)$ e $d_i = 0$ para todo $i \notin \{B(1), ..., B(m)\}$, é a j-ésima direção básica partindo de x^1. Seja $d_B = \left[d_{B(1)}, ..., d_{B(m)}\right]$, como $A(x+\theta d) = b$, temos que $d_B = -B^{-1}A_j$. Usaremos $u = -d_B = B^{-1}A_j$ por facilidade de notação, durante o trabalho.

A figura 2.3 dá um exemplo de direções básicas, di e dj a partir de uma solução viável básica x^1. Note que o poliedro pode ou não limitar um θ tal que o ponto $y = x^1 + \theta * d$ (d direção básica) seja viável, e como veremos em 2.5 isso pode implicar em custo $-\infty$ se nessa direção o custo diminui.



2.4 Custos reduzidos

Seja x^1 uma solução viável básica, B a matriz básica associada e $c_B = \left[c_{B(1)}, ..., c_{B(m)}\right]$. Definimos, para cada $j \in \{1, ..., n\}$ o custo reduzido:

$$\overline{c}_j = c_j - c_B^T B^{-1} A_j.$$

Seja x^1 uma solução viável básica e $\overline{c}$ o vetor de custos reduzidos correspondente. Sabemos que se $\overline{c} \geq 0$, então x^1 é ótimo. Além disso, se x^1 for ótimo e não degenerado, então $\overline{c} \geq 0$ [1]. Portanto, se estivermos em uma solução viável básica e $\overline{c} \geq 0$, então estamos em um ponto ótimo.

Note que ao escolhermos uma direção viável básica, o custo de um ponto $y = x^1 + \theta d_j$ é $c^T(x^1 + \theta d_j) = c^T x^1 + \theta(c_j - c_B^T B^{-1} A_j) = c^T x^1 + \theta \overline{c}$. Portanto, podemos dizer que o custo reduzido representa a variação do custo ao percorrer uma direção básica.

2.5 Soluções Viáveis Básicas adjacentes

Seja x^1 uma solução viável básica com índices básicos $B(1), ..., B(m)$. Uma solução viável básica é *adjacente* a x^1 se compartilha $m - 1$ índices com x^1. Para achar uma solução viável básica adjacente, vamos usar as direções básicas, pois elas forçam o crescimento de uma variável j não-básica, mantendo $Ax = b$ e $x \geq 0$. Veremos que para um $\theta \geq 0$, o ponto $x^1 + \theta d_j$ é solução viável básica adjacente a x^1, com d_j como foi definido em 2.4.

Vamos tomar $\theta = \min_{i=1,...,m|u_i>0} \{x_{B(i)}/u_i\}$ e ver que $x^2 = x^1 + \theta d_j$ é de fato uma solução viável básica adjacente a x^1. Caso todas as componentes de u_i sejam menores ou igual a zero e o custo reduzido na direção j menor do que zero teremos que o problema tem custo ótimo $-\infty$, como será explicado a seguir.

Se θ definido acima não existe, temos que todas as componentes de u_i são menores ou igual a zero ($d \geq 0$), logo qualquer ponto $x^2 = x^1 + \theta d$ é viável com $\theta \geq 0$, pois a restrição $Ax^2 = b$ é verificada (por construção), e $x_j^2 = x_j^1 + \theta \geq x_j^1 \geq 0$, e para i básico $x_j^2 = x_j^1 + \theta d_j \geq x_j^1 \geq 0$. Se ainda tivermos que o custo diminui nessa direção, poderemos diminuir o custo o quanto quisermos e a solução do problema será $-\infty$.

Se $\theta \in \mathbb{R}$, como $d_i = 0 \; \forall i \in \{B(1), ..., B(m)\}, i \neq j$, temos que para essas mesmas componentes x^2 é nulo. Logo, temos $n-1$ restrições ativas LI em x^2. Suponha que para $l \in \{1, .., m\}$ vale que $\theta = x_{B(l)}/u_l$, então $x_{B(l)}^2 = x_{B(l)}^1 + (-x_B^1(l)/d_{B(l)}) * d_{B(l)} = 0$ (diremos que $B(l)$ sai da base), logo existem n restrições ativas LI em x^2. Além disso, por construção, vale que $Ax = b$ e $x \geq 0$ para variáveis não básicas e para $x_B(l)$. Para $B(k)$ básico diferente de $B(l)$, temos que $x_B^2(k) \geq x_{B(k)}^1 + (-x_B^1(k)/d_{B(k)}) * d_{B(k)} = 0$.

Portanto x^2 é solução viável básica adjacente a x^1 e, como a base de x^2 é $\{B(1), ..., B(l-1), j, B(l+1), ..., B(m)\}$, x^2 é adjacente a x^1.

3 Fase 1 do simplex

Para iniciar a fase 2 do algoritmo simplex precisamos de uma solução viável básica x^1, a sua base associada e a inversa da matriz básica. Nesta seção explicaremos o funcionamento da fase 1 do método simplex, responsável por descobrir estes parâmetros.

Para descobrir esses parâmetros vamos criar um problema auxiliar de programação linear:

$$\begin{array}{rl} \text{minimizar} & \sum_{i=1}^{m} y_i \\ \text{sujeito a} & [\ A \mid I\ ] \begin{bmatrix} x \\ \hline y \end{bmatrix} \leq b, \\ \text{com} & A \in R^{m\times n}; b, y \in \mathbb{R}^m; I \in \mathbb{R}^{m\times m} \\ & x, y \geq 0 \end{array}$$

Chamaremos as variáveis de y aritificiais, e a matriz $[\ A \mid I\ ]$ de $\overline{A}$.

Note que o ponto $[\overbrace{0, ..., 0}^{n} | b]$ é uma solução viável básica do problema auxiliar, assumindo $b \geq 0$, com índices básicos $n+1, ..., n+m$ e matrix básica $B = I$. Portanto, temos os parametros necessários para embalar a fase 2 do simplex para o problema auxiliar.

Assumimos no último parágrafo que $b \geq 0$. É fácil verificar que isso não implica em uma perda de generalidade, pois, se $b_i < 0$, basta multiplicar a i-ésima restrição por -1.

3.1 Custo ótimo do problema auxiliar e viabilidade do problema original

Observe que qualquer ponto viável do problema tem custo maior ou igual a zero. Além disso, se existe uma solução viável x^1 do problema original, o ponto $[x^1|0, ..., 0]$ é uma solução viável com custo zero, portanto com custo ótimo. Sendo assim, temos que se o problema original possui solução viável, então o problema auxiliar tem custo ótimo igual a zero. Na contra-positiva, se acharmos uma solução ótima para o problema auxiliar com custo diferente de zero, então o problema original é inviável.

3.2 Solução do problema auxiliar

Como o custo do problema auxiliar é sempre maior ou igual a zero, o custo ótimo nunca será $-\infty$, portanto a fase 2 sempre descobrirá alguma solução viável básica ótima. Se o custo dessa solução for maior que zero, o problema original é inviável, como explicado na seção 4.1. Se o custo for zero, a solução do problema auxiliar $[x|y] = [x|0, ..., 0]$ também será uma solução do problema original.

Porém, mesmo que $[x|0, ..., 0]$ seja uma solução viável básica do problema original, é possível que a base dada pela fase 2 do problema auxiliar tenha índices das variáveis artificiais, que serão eliminadas para iniciar a fase 2 para o problema original. Veremos na proxima seção como remover os índices ariticiais da base.

3.3 Remoção de índices artificiais da base

Suponha que temos um índice $l > m$ aritificial que esteja na base. Queremos tirá-lo da base e adicionar um índice $j \leq m$ na base. Veremos agora, como descobrir tal índice e como mudar a base.

Suponha, sem perca de generalidade, que apenas os $k < m$ primeiros indices básicos não são de variáveis artificiais. Para o índice j na base, devemos garantir que A_j é LI com $\{A_{B(1)}, ..., A_{B(k)}\}$. Suponha que a matriz básica é B e que queremos remover a l-ésima variável da base, por ser atificial. Veja que $B^{-1}A_{B(i)} = e_i$ para $1 \leq i \leq k$, portanto, basta escolher j não artificial tal que $(B^{-1}A_j)_l = A_j \neq 0$ e teremos garantia de que estamos escolhendo uma coluna LI para formar a base.

Se não ouver j como descrito acima, teremos que $(B^{-1}A_j)l = 0$ para todo $1 \leq j \leq m$. Veja que se g^T é a l-ésima linha de B^{-1}, então $g^T A = 0$, ou seja, $g_1 a_1 + ... + g_m a_m = 0$, portanto as linhas de A são LD e podemos então, eliminar a l-ésima linha de A do nosso problema [1].

Veja abaixo o pseudocódigo para remoção de índices artificiais da base:

function $RemoveAritificiais$(A, I, B^{-1}, m, n)
  for all $\{l \in \{B(1), ..., B(m)\} | l > n\}$ **do**
    $candidates \leftarrow \{i \in I.n | i < n\}$
    $x \leftarrow length(candidates)$
    $k \leftarrow 1$;

```
        while k ≤ x and B_l^{-1} A_{I.n(candidates(k))} do
            k ← k + 1
        end while
        if k > x then
            m ← m − 1
            remova l-ésima linha de A
            remova a l-ésima entrada de I.b
            remova a l-ésima entrada de b
        else
            u ← B^{-1} A_k
            atualizaBase(I, invB, u, l, k, m)
        end if
    end for
end function
```

4 Pseudocódigo do algoritmo simplex

```
function Simplex(A, b, c, m, n)
    for all i|b_i < 0 do
        b_i ← b_i * −1
    end for
    A ← [A|I], I ∈ ℝ^{m×m}
    x ← [0⃗, b], 0⃗ ∈ ℝ^n
    c1 ← [0⃗, ac⃗], ac⃗ = [1, ..., 1] ∈ ℝ^m
    I.b ← [n + 1, ..., n + m]
    I.n ← [1, ..., n]
    B^{-1} = I, I ∈ ℝ^{m×m}
    (ind, x, d, I, B^−1) = fase2(A, b, c1, m, n + m, x, I, B^{-1})
    if c1^T x > 0 then
        return (1, NIL, NIL)                          ▷ Problema inviável
    end if
    (I, A, invB, m) ← removeArtificials(A, I, invB, m, n, b)
    x ← [x_1, ..., x_n]
    (ind, x, d, I, B^−1) = fase2(A, b, c, m, n, x, I, B^{-1})
    if ind = −1 then
        return (−1, x, d)                             ▷ Custo ótimo = −∞
    else
```

return $(0, x, d)$ ▷ Solução ótima encontrada
end if
end function

5 O algoritmo

5.1 Objetivo

Nossa motivação é solucionar um problema de programação linear (*PL*) que consiste em minimizar uma função linear (função de custos) sujeita a restrições lineares. Agora que temos os conceitos necessários, vamos desenvolver um algoritmo para solucionar tal problema.

5.2 O que temos a princípio

Dado o problema:

$$\begin{array}{rl} \text{minimizar} & c^T x \\ \text{sujeito a} & Ax = b \\ & x \geq 0 \end{array}$$

Queremos achar o x que satisfaça todas as restrições e minimize $c^T x$. Portanto temos como informação inicial:

A: Matriz $m \times n$ de restrições

b: Vetor de dimensão m

c: Vetor de custos de dimensão n

m: Número de restrições de igualdade

n: Número de variáveis

Logo nossa função que calcula x terá a seguinte forma:

```
simplex(A, b, c, m, n)
```

5.3 O que recebemos no final

Como já vimos antes, temos três possíveis cenários de resultados da nossa função:

1. O problema é inviável, portanto não existe solução ótima
2. Existe solução ótima, logo o custo ótimo é um valor finito
3. O custo ótimo é $-\infty$, e não existe solução ótima

Logo, nossa função deve retornar informação suficiente para sabermos em qual dos três casos estamos e os valores relevantes para cada um. Com isso em mente, nossa função tem o seguinte esqueleto:

```
[ind x d] = simplex(A, b, c, m, n)
```

Onde:

Caso	ind	x	d
1	1	vetor vazio	vetor vazio
2	0	solução ótima	vetor vazio
3	-1	ultima svb computada	direção que vai a $-\infty$

5.4 Pseudocódigo

Nos preocupando somente com o caso onde o problema tem solução, nosso algortimo seria algo próximo do seguinte:

function $simplex(A, b, c, m, n)$

  `multiplica restrições com` $b < 0$ `por -1`

  $A' \leftarrow \left[\ A \,\middle|\, I\ \right]$

  $x' \leftarrow \begin{bmatrix} 0 \\ \hline b \end{bmatrix}$

  $c' \leftarrow \begin{bmatrix} 0 \\ \hline 1 \end{bmatrix}$

  $B \leftarrow \begin{bmatrix} A_{B(1)} & A_{B(2)} & \dots & A_{B(m)} \end{bmatrix}$

  $x \leftarrow \texttt{fase2}(A', b, c', m, m+n, x')$

  $x \leftarrow \texttt{fase2}(A, b, c, m, n, x)$

return x
end function

Nota-se que o algoritmo é dividido em 3 partes:

1. Modificar problema inicial, adicionando m variáveis artificiais
2. Encontrar solução inicial para o problema primal resolvendo o modificado
3. Resolver o problema primal

Podemos resolver os dois problemas, dado uma solução inicial, graças a função `fase2` que será explicada mais a frente. Imaginando que já temos essa função, nosso algoritmo final em octave é:

```
function [ind, x, d] = simplex(A, b, c, m, n)
    % Arruma restricoes para que b > 0
    for i = 1 : m
        if (b < 0)
            b *= -1;
            A(i, :)  *= -1;
        end
    end

    % Cria problema auxiliar para achar primeira solucao
    % para o problema primal
    A = [A, eye(m)];
    x = [zeros(n, 1); b];
    c1 = [zeros(n, 1); ones(m, 1)];
    I = struct('b', [n + 1 : n + m], 'n', [1 : n]);
    invB = eye(m);

    % Resolve problema auxiliar
    [ind, x, d, I, invB] = fase2(A, b, c1, m, n + m, x, I, invB);

    if x(n + 1 : n + m) != 0
        % Problema Inviavel
        ind = 1;
        x = [];
        d = [];
        return;
    end

```

```
    % Remove vaiaveis artificiais da base. (nao altera x)
    [I, A, invB, m] = removeArtificials(A, I, invB, m, n, b);
    invB = inverse(A(:, I.b));
    x = x(1 : n);


    % Resolve problema primal, com solucao encontrada
    [ind, x, d, I, invB] = fase2(A, b, c, m, n, x, I, invB);
```

Nessa implementação, é clara a correspondência entre as principais etapas vistas quando apresentamos o pseudocódigo.

5.5 Pseudocódigo da fase 2 do simplex

Na implementação do nosso algoritmo simplex, utilizamos uma estrutura chamada de I que guarda os índices básicos e não básicos, representados respectivamente pelos vetores $I.b$ e $I.n$.

5.5.1 Função custoDirecao

A função custoDirecao é responsável por escolher uma direção básica j com custo reduzido menor do que zero.

function $custoDirecao$(A, B^{-1}, c, n, m, I)
    $j \leftarrow 1$
    while $j \leq n - m$ **do**
        $redc = c(I.n(j)) - c_B^T A_{I.n(j)}$
        if $redc < 0$ **then**
            $ij \leftarrow j$
            $u \leftarrow B^{-1} A_{I.n(ij)}$
            return $(redc, u, ij)$
        end if
        $j \leftarrow j + 1$
    end while
    return $(0, -1, NIL)$
end function

5.5.2 Função calculaTheta

A função calculaTheta é responsável por calcular θ como foi explicado na seção 2.5.

function $calculaTheta(x, u, I)$
  $imin = -1$
  $theta = \infty$
  for $i = 1$ to m **do**
    if $u_i > 0$ **then**
      $t = x_{I.b(i)}/u_i$
      if $t < theta$ **then**
        $theta = t$
        $imin = i$
      end if
    end if
  end for
  return $(imin, teta)$
end function

5.5.3 Atualização da base

Atualizações nas bases ocorrem quando tiramos um índice da base e adicionamos outro. Esta operação ocorre tanto na fase 1 quanto na fase 2 do método simplex.

O cálculo da inversa de uma matriz é uma operação muito cara, portanto devemos investigar uma maneira de atualizar a inversa de B ao invés de recalculá-la a todo momento. Seja B uma base, e $\overline{B}$ a base depois de uma atualização, tirando o l-ésimo indice básico e adicionando o índice j a base. Note que:

$$B^{-1}\overline{B} = \begin{pmatrix} B^{-1}A_{B(1)} & \cdots & B^{-1}A_{B(l-1)} & B^{-1}A_j & \cdots & B^{-1}A_{B(m)} \end{pmatrix}$$
$$= \begin{pmatrix} e_1 & \cdots & e_{l-1} & u & \cdots & e_m \end{pmatrix}$$

Portanto, se pré-multiplicarmos B^{-1} por matrizes fazendo com que, no lado direito da equação, u se torne e_l, teremos que B^{-1} pré-multiplicada pelas mesmas matrizes será igual a $\overline{B^{-1}}$.

O pseudo-código:

function $atualizaBase(I, B^{-1}1, u, imin, ij, m)$
  $(I.b(imin), I.b(ij)) \leftarrow (I.n(ij), I.b(imin))$
  for $i = 1$ to m **do**

if $i \neq imin$ **then**
$B^{-1}_{i,j} \leftarrow B^{-1}_{i,j} - (u_i/u_{imin}) * B^{-1}_{imin,j}$ for $j = 1, ..., n$
end if
end for
$B^{-1}_{i,j} \leftarrow B^{-1}_{i,j}/u(imin)$ for $j = 1, ..., n$
end function

5.5.4 A função fase2

function $fase2$(A, b, c, m, n, x, I, B^{-1})
$(redc, u, ij) \leftarrow custoDirecao(A, B^{-1}, c, n, m, I)$
while $redc < 0$ **do**
$(imin, teta) \leftarrow calculaTheta(x, u, I)$
if $imin = -1$ **then**
return $(-1, u)$ ▷ custo ótimo $= -\infty$
end if
$x \leftarrow atualizax(x, teta, u, I.n(ij), I)$
$(I, B^{-1}) = atualizaBase(I, B^{-1}, u, imin, ij, m)$
$(redc, u, ij) \leftarrow custoDirecao(A, B^{-1}, c, n, m, I)$
end while
return $(0, x)$
end function

Referências

[1] Dimitris Bertsimas, John N. Tsitsiklis. Introduction to Linear Optimization. 1997.