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.

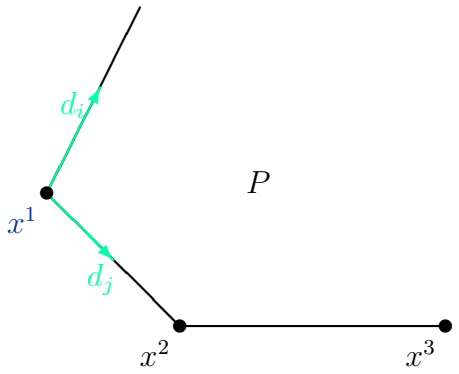
Se x^1 é uma solução básica não degenerada e seja $B(1), ..., B(m)$ os índices das componentes não nulas de x. A matriz $B = \left[A_{B(1)}, ..., A_{B(m)}\right]$ é chamada *matriz básica* associada a x^1.

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:

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

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 3.1. 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, $g_1a_1 + ... + g_ma_m = 0$, 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 $k \leq x$ and $B_l^{-1} A_{I.n(candidates(k))}$ **do**
$k \leftarrow k + 1$
end while
if $k > x$ **then**
$m \leftarrow m - 1$
remova l-ésima linha e l-ésima coluna de B
remova l-ésima linha de A
remova a l-ésima entrada de $I.b$
remova a l-ésima entrada de b
else
$u \leftarrow B^{-1} A_k$
$atualizaBase(I, invB, u, l, k, m)$
end if
end for
end function

4 O algoritmo

4.1 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.

4.2 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)
```

4.3 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 $-\infty$

4.4 A função simplex

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`
$A' \leftarrow [\ A \mid I\ ]$
$x' \leftarrow \begin{bmatrix} 0 \\ \hline b \end{bmatrix}$
$c' \leftarrow \begin{bmatrix} 0 \\ \hline 1 \end{bmatrix}$
$B \leftarrow \begin{bmatrix} A_{B(1)} & A_{B(2)} & \dots & A_{B(m)} \end{bmatrix}$
$x \leftarrow$ `fase2`$(A', b, c', m, m+n, x')$

$x \leftarrow$ `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 inicial resolvendo o modificado

3. Resolver o problema inicial

Podemos resolver os dois problemas, dado uma solução inicial, graças a função `fase2` que resolve problemas de programação linear a partir de uma solução viável básica inicial. 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 inicial
    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);

    for artific = n + 1 : n + m
        if abs(x(artific)) > 1e-10
            % Problema Inviavel
            ind = 1;
            x = [];
            d = [];
            printf("\n\nO problema e inviavel\n");
            return;
        end
    end

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

```
35
36
37      % Resolve problema inicial, com solucao encontrada
38      [ind, x, d, I, invB] = fase2(A, b, c, m, n, x, I, invB);
```

Utilizamos uma estrutura chamada de I que guarda os índices básicos e não básicos, representados respectivamente pelos vetores $I.b$ e $I.n$. Nessa implementação, é fácil ver a correspondência entre as principais etapas vistas quando apresentamos o pseudocódigo. Entre as linhas `3` e `16` modificamos o problema original, nas linhas de `19` a `31` encontramos uma solução inicial e na linha `35` resolvemos o problema.

Uma das melhorias feita para reduzir o espaço utilizado foi armazenar A' e x' (do pseudocódigo) em A e x respectivamente. Ainda assim, há certo desperdício de espaço utilizado em funções, pois o octave não envia argumentos por referência. Ou seja, quando enviamos $invB$ para `fase2`, por exemplo, ao modifica-la dentro da função, o octave cria uma cópia e então altera seus valores. Poderiamos ter feito alterações para que não haja esse tipo de desperdício, mas optamos por um código mais legível e modularizado.

Visando melhorar nosso desempenho, passamos e retornamos alguns argumentos extras nas funções `fase2` e `removeArtificials`. Dessa forma, evitamos cálculos desnecessários de B^{-1} e dos indices básicos associados à solução x. Nos poupar o cálculo de B^{-1} é uma melhoria significativa, visto que isso tem complexidade $O(n^3)$.

4.5 fase2

A função `fase2` vista na seção anterior, é de grande importância para nosso algoritmo. Dada uma solução viável básica inicial, essa função ”anda”pelas soluções adjacentes, buscando uma na qual o custo calculado seja menor que na atual. Tendo a nova solução, voltamos a situação inicial e o algoritmo é executado novamente. Fazemos isso, tomando o cuidado para não andarmos em ciclos, até encontrar uma solução viável básica na qual todas as soluções adjacentes tem custo maior. Assim concluimos que essa solução é ótima. Tendo os conceitos da seção 2, produzimos o seguinte pseudocódigo:

function $fase2(A, b, c, m, n, x)$
  $\overline{c}^T \leftarrow c^T - (c_B^T B^{-1})A_j$

while $!(\bar{c} \geq 0)$ **do**
$j \leftarrow j \notin B(1), \ldots, B(m), t.q.\ \bar{c}_j < 0$ ▷ Usando regra anti-ciclagem
$u \leftarrow B^{-1}A_j$
if $u \leq 0$ **then**
return $(-1, d)$ ▷ Custo ótimo é $-\infty$
end if
$\theta \leftarrow \min_{u_l > 0} \frac{x_{B(l)}}{u_l}, l = 1, \ldots, m$
$x \leftarrow x + \theta d$
$B(l) \leftarrow j$
$atualiza(invB)$
$\bar{c}^T \leftarrow c^T - (c_B^T B^{-1})A_j$
end while
return $(0, x)$
end function

De forma muito similar, temos nosso algoritmo implementado em octave:

```
function [ind, x, d, I, invB] = fase2(A, b, c, m, n, x, I, invB)
    [redc, u, ij] = custoDirecao(A, invB, c, n, m, I);
    while redc < 0  % se essa condicao falha, x e otimo
        [imin, teta] = calculaTeta(x, u, I);
        if imin == -1  % custo otimo e -inf e u tem a direcao
            ind = -1;
            d = u2d(u, I.n(ij), I);
            return;
        end

        % atualiza
        x = atualizax(x, teta, u, I.n(ij), I);
        [I, invB] = atualizaBase(I, invB, u, imin, ij, m);

        [redc, u, ij] = custoDirecao(A, invB, c, n, m, I);
    end

    d = [];
    ind = 0;
end
```

4.6 Complexidade

A complexidade da função `simplex` depende basicamente da complexidade de `removeArtificials` e `fase2`. Primeiramente analisaremos a complexidade das funções auxiliares para então analisar as principais.

4.6.1 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

Apesar de estar dentro do loop principal, $u \leftarrow B^{-1} A_{I.n(ij)}$ é executada somente uma vez, pois assim que executada, logo em seguida uma chamada de retorno também é. Portanto essa linha leva $O(m^2)$, pois é multiplicação de uma matrix $m \times m$ por um vetor de dimensão m. Já a primeira linha de dentro do loop, leva $O(m)$, pois é produto de dois vetores de tamanho m. No entanto, essa linha é executada, no pior caso, $(n - m)$ vezes, isto é $O((n - m)m) = O(nm)$. Como $n > m$, a função como um todo tem complexidade $O(nm)$.

4.6.2 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

O loop itera m vezes executando operações que levam $O(1)$, portanto `calculaTheta` tem complexidade $O(m)$.

4.6.3 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 $atualizaBase(I, B^-1, u, imin, ij, m)$
$(I.b(imin), I.b(ij)) \leftarrow (I.n(ij), I.b(imin))$
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

Essa função executa o loop m vezes. A cada iteração, uma operação de soma de vetores de tamanho n é feita. Portanto a complexidade de `atualizaBase` é $O(nm)$.

4.6.4 Função atualizaX

Para atualizar x, temos que fazer $x + \theta d$. No entanto, não há a necessidade de calcular d a partir de u:

function $atualizax(x, t, u, j, I)$
$x_{I.b} \leftarrow x_{I.b} - t * u$
$x_j \leftarrow t$
return x
end function

Claramente, por ser uma subtração de vetores de tamanho m, a complexidade é $O(m)$.

4.6.5 A função fase2

Finalmente podemos analisar uma das funções que determinam a complexidade da função `simplex`. A cada iteração do loop principal dessa função, temos chamadas às seguintes funções: `calculaTeta`, `atualizax`, `atualizaBase` e `custoDirecao`. As duas primeiras são lineares, $O(m)$. Já as duas últimas, são $O(nm)$. Portanto cada iteração da `fase2`, temos uma complexidade $O(mn)$.

4.6.6 A função removeArtificials

Para a função `removeArtificials`, a cada iteração do loop principal, temos uma possível chamada de `atuzliaBase`, que é $O(nm)$. Logo no pior caso, ela é chamada em todas as iterações e no melhor ela nunca é chamada. O loop principal depende de quantas variáveis artificiais existem na base. Podem haver no máximo m dessas variáveis. Logo o pior

dos cenários é quando temos m variáveis artificiais e atualizamos a base para todas elas. Nesse caso o algoritmo leva $O(nm^2)$. No melhor dos casos, não há variáveis artificiais dentro da base e isso leva $O(1)$.

4.6.7 Conclusão da complexidade

Seja α o número de iterações da primeira chamada de `fase2` em `simplex` e β o da segunda chamada. Então na primeira delas, temos complexidade $O(\alpha nm)$ e na outra $O(\beta nm)$. Se $\alpha + \beta > m$, então as duas juntas são $\Omega(nm^2)$ e nesse caso essa é a complexidade do `simplex`. Caso contrário, `removeArtificials` continuará sendo $O(nm^2)$ e a função `simplex` também será.

5 Exemplos

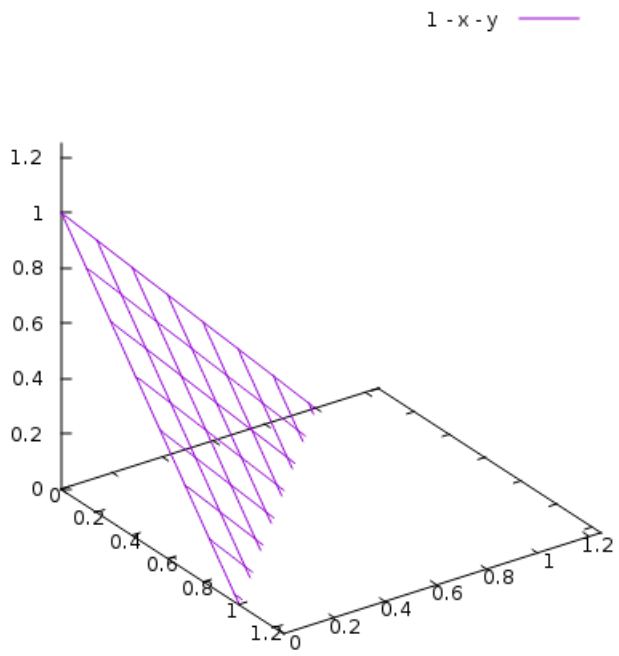
Nessa seção apresentaremos 3 exemplos do algoritmo implementado em octave sendo executado.

5.1 Solução Ótima

Consideremos o seguinte problema:

$$\begin{aligned} \text{minimizar} \quad & x_1 + x_2 + x_3 \\ \text{sujeito a} \quad & x_1 + x_2 + x_3 \leq 1 \\ & x \geq 0 \end{aligned}$$

Podemos visualizar o problema com a seguinte imagem, onde a região viável está representada abaixo do plano desenhado e acima dos eixos:



Como esse problema não está no formato padrão, temos que reformulá-lo adicionando uma variável de folga:

$$\begin{array}{rl} \text{minimizar} & x_1 + x_2 + x_3 \\ \text{sujeito a} & x_1 + x_2 + x_3 + x_4 = 1 \\ & x \geq 0 \end{array}$$

Para esse problema, temos:

- $A = \begin{pmatrix} 1 & 1 & 1 & 1 \end{pmatrix}$
- $b = \begin{pmatrix} 1 \end{pmatrix}$
- $c = \begin{pmatrix} 1 & 1 & 1 & 0 \end{pmatrix}$
- $m = 1$
- $n = 4$

Portanto, podemos aplicar nosso algoritmo:

```
octave:1> simplex(A,b,c,m,n)

******************** Fase1 ********************


Iteracao 0:
|  Var Basicas       Valor
|  ===========  ==========
|           x5    1.000000
|
|  Custo em x
|  ==========
|       1.000
|

Iteracao 1:
|  Var Basicas       Valor
|  ===========  ==========
|           x1    1.000000
|
|  Indice var basicas   Componente da direcao
|  ==================   ======================
|                    1               -1.000000
|
|  Custo em x      Teta         Entra na Base   Sai da Base
|  ==========   ==========      =============   ===========
```

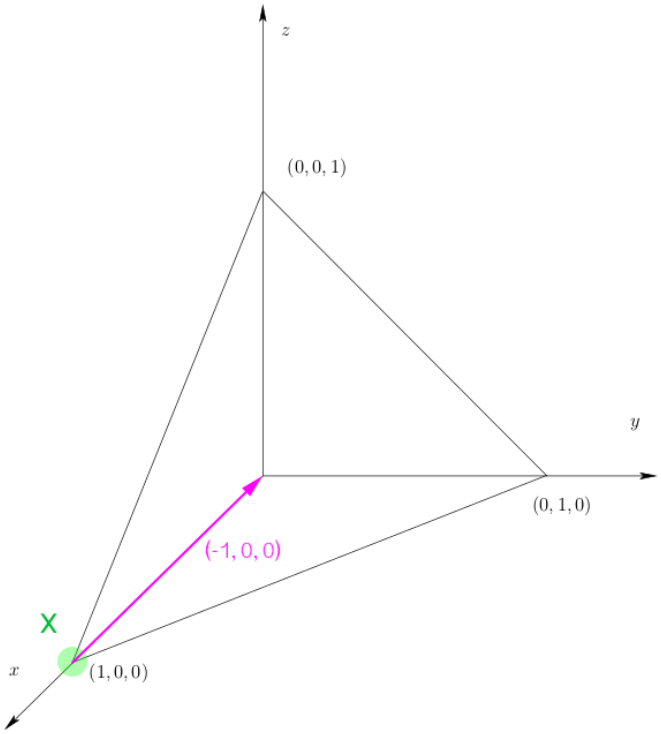
```
|       0.000         1.000                   x1              x5
|
```

A fase 1 consiste em achar uma solução para o problema principal através de um problema auxiliar. Nesse caso o problema auxiliar tem somente uma variável a mais (x_5), pois $m = 1$. Por esse motivo, na Iteração 0, x_5 está na base e vale $b = 1$. Nessa situação, $c^T x = x_5 = 1$. Como essa não é uma solução ótima, foi encontrada uma direção em que o custo reduzido é menor que 0. Vemos na Iteração 1: que essa direção é onde $j = 1$. Verificamos que $\overline{c}_j = c_j - c_B^T B^{-1} A_j = 0 - 1 * 1 * 1 = -1$. $x + \theta d$ é calculado resultando em $x_5 = 0$ e $x_1 = 1$. Nesse novo ponto, o custo vale 0. Nesse ponto, todos os custos reduzidos são maiores ou iguais a 0, e essa é a solução ótima. Como x_1 não é uma variável artificial, essa já é a solução inicial para o problema inicial.

```

********************* Fase2 *********************


Iteracao 0:
|  Var Basicas        Valor
|  ============  ==========
|            x1    1.000000
|
|  Custo em x
|  ==========
|       1.000
|

Iteracao 1:
|  Var Basicas        Valor
|  ============  ==========
|            x4    1.000000
|
|  Indice var basicas   Componente da direcao
|  ==================   ======================
|                    4               -1.000000
|
|  Custo em x      Teta         Entra na Base   Sai da Base
|  ==========   ==========      =============   ===========
|       0.000         1.000                   x4              x1
|
```

```
56
57
58 Solucao encontrada:
59 x =
60
61     0
62     0
63     0
64     1
```

A Iteração 0 da fase 2 começa com $x_1 = 1$, pois foi a solução inicial encontrada na Fase 1. A figura a seguir ilustra a situação:



Na Iteração 1, a direção $j = 4$ é escolhida por ter custo reduzido menor que 0. Assim, x_4 entra na base e x_1 a deixa. No novo ponto onde $x_4 = 1$ e as demais componentes de x são 0, não há custo reduzido menor que 0, portanto essa é a solução ótima para nosso problema.

5.2 Custo ótimo $-\infty$

Suponha o problema:

$$\begin{array}{rl} \text{minimizar} & -x_1 - x_3 \\ \text{sujeito a} & x_1 + x_2 = 1 \\ & x \geq 0 \end{array}$$

Para esse problema, temos:

- $A = \begin{pmatrix} 1 & 1 & 0 \end{pmatrix}$
- $b = \begin{pmatrix} 1 \end{pmatrix}$
- $c = \begin{pmatrix} -1 & 0 & -1 \end{pmatrix}$
- $m = 1$
- $n = 3$

É fácil ver que como a unica restrição em x_3 é $x_3 > 0$, e temos que minimizar $-x_1 - x_3$, x_3 pode ser tão grande quanto queiramos e portanto o custo ótimo será $-\infty$.

```
octave:1> simplex(A,b,c,m,n)

******************** Fase1 ********************


Iteracao 0:
|  Var Basicas       Valor
|  ===========  ==========
|           x4    1.000000
|
|  Custo em x
|  ==========
|       1.000
|

Iteracao 1:
|  Var Basicas       Valor
|  ===========  ==========
|           x1    1.000000
|
|  Indice var basicas   Componente da direcao
|  ==================   ======================
|                   1                -1.000000
```

```
|
|  Custo em x      Teta         Entra na Base   Sai da Base
|  ==========   ==========      =============   ===========
|       0.000         1.000                x1            x4
|
```

Como $m = 1$, a unica variável artificial será x_4, por isso na Iteração 0 da Fase 1, ela é a unica variável na base. No entanto na direção $j = 1$, o custo reduzido é menor que 0, portanto a variável x_4 sai da base e a x_1 entra. Nesse novo ponto, os custos reduzidos são maiores ou iguais a zero e não há variáveis artificiais na base. Assim, $x = (1, 0, 0, 0)^T$ é solução inicial do problema.

```

********************* Fase2 *********************


Iteracao 0:
|  Var Basicas       Valor
|  ===========  ==========
|           x1    1.000000
|
|  Custo em x
|  ==========
|      -1.000
|


O custo otimo e -infinito, com direcao:
d =

   -0
    0
    1
A partir de:
x =

    1
    0
    0
```

O problema começa com o x que calculamos na fase anterior. O j escolhido é 3, pois $\bar{c}_3 = -1 + 0 = -1$. No entanto, $u = B^-1 * A_j = A_1^-1 * A_3 =$

$1 * 0 = 0$. Portanto não existe $u > 0$, logo podemos escolher θ tão grande quanto queiramos e nosso custo será cada vez menor. A direção que leva a $-\infty$ é tal que d_3 = 1, e como já vimos, $u = 0$. Portanto $d = (0, 0, 1)^T$.

5.3 Inviável

Suponha o problema:

$$\begin{array}{rl} \text{minimizar} & -x_1 - x_2 - x_3 \\ \text{sujeito a} & x_1 = 1 \\ & 2x_1 = 1 \\ & x \geq 0 \end{array}$$

Para esse problema, temos:

- $A = \begin{pmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \end{pmatrix}$
- $b = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$
- $c = \begin{pmatrix} -1 & -1 & -1 \end{pmatrix}$
- $m = 2$
- $n = 3$

Esse problema é obviamente inviável, pois x_1 não pode ser ao mesmo tempo 1 e 0.5.

```
octave:1> simplex(A,b,c,m,n)

******************** Fase1 ********************


Iteracao 0:
|  Var Basicas       Valor
|  ===========  ==========
|           x4    1.000000
|           x5    1.000000
|
|  Custo em x
```

```
|   ==========
|        2.000
|

Iteracao 1:
|   Var Basicas        Valor
|   ============  ==========
|             x4    0.500000
|             x1    0.500000
|
|   Indice var basicas   Componente da direcao
|   ==================   ======================
|                    4               -1.000000
|                    1               -2.000000
|
|   Custo em x       Teta     Entra na Base    Sai da Base
|   ==========   ==========  ==============   ============
|        0.500         0.500              x1             x5
|


O problema e inviavel
```

Vemos que o problema começa com as duas variáveis artificiais x_4 e x_5 na base, pois $m = 2$. Em 1 iteração achamos a solução ótima para o problema auxiliar. No entanto como pode-se ver, x_4 está entre as variáveis básicas, portanto o problema principal é inviável.

Referências

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