UNIVERSIDADE DE SÃO PAULO

ROBERTO OLIVEIRA BOLGHERONI - 11796430

EXERCÍCIO PROGRAMA 1

Otimização Linear - Método Simplex

São Paulo

2021

1. INTRODUÇÃO

O objetivo deste trabalho é relatar a implementação da segunda fase do Método Simplex de resolução de Problemas de Programação Linear no formato padrão, para problemas sem soluções degeneradas. Foram feitas três diferentes implementações seguindo padrões diferentes, a saber: Simplex ingênuo, revisado e full-tableau. Embora sigam a mesma sequência de passos (descrita a seguir), utilizam recursos de forma diferente para obter seus resultados.

O Método simplex:

Seja um problema PL no formato padrão, representado por m restrições em n variáveis, dada pelo sistema $Ax = b,$ - sendo A uma matriz de tamanho m*n, b um vetor de m posições, x uma solução viável básica de n posições para o problema - e uma função de custos objetiva c, iniciamos a fase 2 com uma solução viável básica inicial x, dada pela base B, esta última formada por uma sequência de m índices de colunas L.I. de A. Então podemos iniciar a primeira iteração:

1. Colete x, a solução viável básica, e a base B
2. Compute os custos reduzidos para cada variável não-básica (não presente em B). Caso todos sejam não-negativos, então a solução viável básica atual é ótima e o algoritmo termina. Se não, escolha alguma variável j cujo custo seja negativo.
3. Calcule $u = B^{-1}A_j$. Caso nenhuma componente de u seja positiva, definimos $\theta^* = -\infty$. Então o custo ótimo é $-\infty$ e o algoritmo termina.
4. Caso alguma componente de u seja positiva, então compute
$$\theta^* = min_{\{i=1,\dots,m \mid u_i>0\}}\{\frac{x_{B(i)}}{u_i}\}$$
5. Seja l tal que $\theta^* = \frac{x_{B(l)}}{u_l}$. Forme uma nova base substituindo $A_{B(l)}$ por A_j. Seja y a nova solução viável básica. Então $y_j = \theta^*$ e $y_{B(i)} = x_{B(i)} - \theta^* u_i$, para todo $i \neq l$. Inicie a próxima iteração com a nova base e y como solução viável básica.

Cada implementação realiza o método simplex a sua maneira.

1.1 Ingênuo:

O método ingênuo é o mais simples dos três. Ele segue estritamente os passos acima. Para tanto, em cada iteração, temos apenas os índices da base e a solução viável básica, então B e B^{-1}, bem como os custos reduzidos das variáveis não básicas, são sempre recalculados, exigindo custo assintótico $O(m^3 + mn)$ de operações aritméticas.

1.2 Revisado:

O revisado busca evitar o recálculo de B^{-1}, utilizando a matriz de uma iteração para calcular a próxima através de uma sequência de operações de multiplicação matriz-vetor. O princípio por trás desse método é que de uma iteração para outra, apenas uma coluna de B é alterada, portanto B^{-1} pode ser calculada sem muito esforço. Sendo u o vetor descrito no método simplex original, $\overline{B}$ a base para a próxima iteração, temos que $B^{-1}\overline{B} = I$, a matriz identidade. Como B e $\overline{B}$ diferem apenas na l-ésima coluna, temos:

$$\mathbf{B}^{-1}\overline{\mathbf{B}} = \begin{bmatrix} | & & | & | & | & & | \\ \mathbf{e}_1 & \cdots & \mathbf{e}_{\ell-1} & \mathbf{u} & \mathbf{e}_{\ell+1} & \cdots & \mathbf{e}_m \\ | & & | & | & | & & | \end{bmatrix}$$

$$= \begin{bmatrix} 1 & & u_1 & & \\ & \ddots & \vdots & & \\ & & u_\ell & & \\ & & \vdots & \ddots & \\ & & u_m & & 1 \end{bmatrix},$$

Figura 1: *Representação da matriz resultado da multiplicação da inversa da base atual pela próxima base.*

É possível concluir que podemos obter uma sequência de operações elementares nas linhas da matriz acima que a transforme na matriz identidade, que pode ser associada a um vetor Q. Ou seja, de forma que $QB^{-1}\overline{B} = I$. Mas seja $V = QB^{-1}$. Note que $V\overline{B} = I, \Rightarrow V = \overline{B}^{-1} \Rightarrow QB^{-1} = \overline{B}^{-1}$. Ou seja, conseguimos obter uma sequência de operações elementares que aplicadas a matriz inversa de uma iteração nos fornece a da próxima, sequência esta dada pela transformação do vetor u no vetor e_l, o l-ésimo vetor canônico.

O método simplex revisado se aproveita dessa propriedade. Em cada iteração, além dos parâmetros que uma iteração ingênua pede, recebe também a matriz B^{-1} e segue normalmente o método simplex - calcula os custos reduzidos, o vetor u, quais variáveis entram/saem da base, a próxima solução viável básica, etc. No entanto, ao passar para uma próxima iteração, calcula a próxima base invertida de acordo com o descrito acima e a passa como parâmetro.

Apesar de não recalcular B^{-1} em cada iteração, a fim de evitar o acúmulo de erros de arredondamento entre uma iteração e outra, pode ser interessante recalcular B^{-1} a partir de B periodicamente.

Em cada iteração, calcular B^{-1} e os custos reduzidos exige até O(m² + mn) operações aritméticas, o que faz desse método mais eficiente que o ingênuo.

1.3 Full-Tableau:

O método full-tableau consiste em manter uma matriz de tamanho (m+1)(n+1) que contenha todas as informações necessárias para uma iteração do método simplex. Na transição de uma iteração para a outra, atualizamos o tableau. Ele é organizado da seguinte forma:

$-\mathbf{c}'_B\mathbf{x}_B$	$\bar{c}_1$	$\dots$	$\bar{c}_n$
$x_{B(1)}$	$\vert$		$\vert$
$\vdots$	$\mathbf{B}^{-1}\mathbf{A}_1$	$\dots$	$\mathbf{B}^{-1}\mathbf{A}_n$
$x_{B(m)}$	$\vert$		$\vert$

***Figura 2:** Representação de um tableau do método simplex.*

- Chamamos a primeira coluna de 0-ésima coluna, que armazena o valor atual da função objetiva e os valores das variáveis básicas da solução viável básica atual;
- Chamamos a primeira linha de 0-ésima linha, que armazena o valor atual da função objetiva e os custos reduzidos atuais de cada variável - as básicas tem sempre custo reduzido valendo 0;
- o resto do tableau corresponde a uma matriz obtida ao multiplicar A pela inversa da base atual

Como funciona uma iteração do método full-tableau:

1. Recebe-se um tableau associado a um conjunto de índices da base B e uma solução viável básica x
2. Examinam-se os custos reduzidos na 0-ésima linha do tableau. Caso todos sejam não-negativos, então a solução atual é ótima e o método termina.
3. Caso haja algum custo reduzido negativo, escolha uma coluna j que contenha um custo reduzido negativo. j corresponde ao índice da variável que vai entrar na base; o vetor $u = B^{-1}A_j$ pode ser obtido pela j-ésima coluna do tableau, pulando a 0-ésima linha. Se nenhuma componente de u for positiva, então o algoritmo termina e retorna que o custo ótimo é $-\infty$.
4. Caso contrário, determine a linha l para qual $\theta^* = min_{\{i=1,...,m \mid u_i>0\}}\{\frac{x_{B(i)}}{u_i}\} = \frac{x_{B(l)}}{u_l}$. Chamamos l de linha pivotal. B(l) é o índice da variável que vai sair da base.
5. Adicione a cada linha do tableau um múltiplo da linha pivotal de forma que a j-ésima coluna do tableau (o vetor u) seja transformado no l-ésimo vetor canônico - queremos que valha 1 na l-ésima entrada e 0 nas demais. Fazemos o mesmo processo para o 0-ésima linha, ou seja, adicionamos um múltiplo da linha pivotal de forma que o custo reduzido da j-ésima variável valha 0 ao final do processo. Com essa sequência de operações, obtemos o tableau para a próxima iteração, que pode ser iniciada.

2. IMPLEMENTAÇÕES

Nesta seção serão apresentadas as seções do código implementado de forma a justificar seu funcionamento e sua fiel implementação aos métodos descritos na seção 1.

2.1 Ingênuo

Como descrito na introdução, o método ingênuo implementa estritamente a descrição do método simplex:

1. Coletando x e a base B. Recebemos x e o vetor de índices da base indB por parâmetros. Para montar B, concatenamos as m colunas de A cujos índices pertencem a indB, na ordem recebida em indB.

```
    B = [];
    disp("Variaveis basicas:");
    for b_i = indB
        disp ([b_i, x(b_i)])
        B = [B A(:, b_i)];
    endfor
```

2. Calculando custos reduzidos. Calculamos a inversa de B e c_B, que corresponde ao vetor de custos nos índices das variáveis da base. Para evitar repetir essa operação, armazenamos c_B*B_inv na variável p.

```
    B_inv = inv(B)
    c_B = c(indB);
    disp(["Vetor de custos da base: ", mat2str(c_B)]);
    p = c_B*B_inv; # deveria ser c_B' mas os vetores aqui ja sao
deitados por padrao
    disp(["p: ", mat2str(p)]);
```

   Agora passamos ao cálculo dos custos reduzidos. Apesar de não ser exigido no enunciado, foi implementada a regra de Bland; calculamos os custos reduzidos das variáveis em ordem crescente do índice até acharmos o primeiro que seja negativo. Caso seja encontrado, não calculamos para o resto das variáveis. Armazenamos na variável hasNegativeCost a informação se algum dos custos reduzidos é negativo, inicialmente sendo falso. Para cada variável j de 1 até n, caso j seja não básica (não esteja presente no vetor indB), então calculamos seu custo reduzido. Caso seja negativo, quebramos o laço.

```
    % calcular custos reduzidos para as variaveis nao basicas
    disp(["Custos reduzidos das nao basicas (ate primeiro 
negativo): "]);
    hasNegativeCost = false;
    for j = 1:n
        if !any( indB == j)
            c_red = c(j) - p * A(:, j);
            disp([j, c_red]);
            if (c_red < 0)
                hasNegativeCost = true;
                break
            endif
        endif
    endfor
```

Ao final do processo, caso hasNegativeCost seja falso, então a solução atual é ótima: encerramos o algoritmo e exibimos o resultado. Caso contrário, j armazena a variável de menor índice com custo reduzido negativo, armazenado em c_red. Ou seja, j será a variável a entrar na base.

```
    if(!hasNegativeCost)
        disp(strcat("Solucao otima encontrada com custo= ", 
mat2str(c*x'), ":"));
        for k = 1:n
            disp([k, x(k)]);
        endfor
        ind = 1;
        v = x;
        return
    endif
    display("Ha custo reduzido negativo");
    disp(["Entra na base: ", int2str(j)]);
```

3. Calculamos $u = B^{-1}A_j$.

```
    u = B_inv * A(:, j);
    disp(["Direcao a ser percorrida para proxima instancia de x: 
u=", mat2str(u)]);
```

Checamos uma a uma as componentes de u para ver se há alguma positiva, informação armazenada em hasPositive, inicialmente falso.

Concomitantemente, para toda componente $u_i > 0$, calculamos $\frac{x_{B(i)}}{u_i}$, armazenando o valor mínimo dessa razão na variável teta e o i correspondente na variável t.

```
    t = -1;
    teta = 0;
    hasPositive = false;
    for i = 1:m
        if (u(i) > 0)
            val = x(indB(i))/u(i);
            if (!hasPositive)
                t = i;
                teta = val;
                hasPositive = true;
            endif
            if (val < teta)
                t = i;
                teta = val;
            endif
        endif
    endfor
```

Caso hasPositive permaneça com valor falso ao final do laço, então conclui-se que o problema tem custo ótimo $-\infty$. Então calculamos a direção d em que a função de custo decresce infinitamente, armazenada na variável d, exibimos os resultados e encerramos o algoritmo.

```
    else
        disp("Problema de solucao ilimitada");
        ind =-1 ;
        d = zeros(1, n);
        d_B = u;
        for i = 1:m
            d(indB(i)) = d_B(i);
        endfor
        d(j) = 1;
        v=d;
        disp(["direcao em que decresce infinitamente: ",
mat2str(d)]);
```

```
        return
    endif
```

4. Caso contrário, indB(t) indica qual variável vai sair da base; teta armazena o valor θ^*.

```
    if (hasPositive)
        display("Valor de u nao negativo encontrado.");
        disp(["Sai da base: ", int2str(indB(t))]);
        disp(["Teta*: ", num2str(teta)]);
```

5. Realizamos a atualização de indB, calculamos a nova solução viável básica, ou seja, deslocamos x por $\theta^* d$ e retornamos o resultado da próxima iteração recursiva do método simplex.

```
        x(j) = teta;
        for i = 1:m
            if (i != t)
                x(indB(i)) -= teta*u(i);
            else
                x(indB(i)) = 0;
            endif
        endfor
        indB(t) = j;
        disp("\n");
        [ind v] = simplex_ing_intern(A,b,c,m,n,x,indB,
iteracao+1);
        return
```

2.2 Revisado

O método revisado só se diferencia do método ingênuo nos seguintes trechos:

- O cálculo manual (inversão direta) de Binv só é feito a cada 10 iterações. Nas outras vezes, utiliza-se a matriz fornecida nos parâmetros da função.

```
    if (mod(iteracao, 10) == 0)
        display("Binv recalculado");
```

```
        Binv = inv(B);
    endif
```

- O Cálculo da matriz inversa da próxima iteração. A ser armazenado na variável nextBinv, inicialmente recebe o valor de Binv. Sobre cada linha $i \neq t$ da base inversa atual, iremos adicionar um múltiplo da linha t (note que indB(t) = l), que corresponde à variável que vai sair da base, de forma que u(i) corresponda à i-ésima posição do l-ésimo vetor canônico. Ou seja, multiplicamos a linha t por -u(i)/u(t). Já a t-ésima linha apenas é dividida por u(t), ou seja, é alterada de forma que u(t) seja transformado em 1. Exibimos o resultado e chamamos a próxima iteração do método simplex.

```
% calculando a proxima base inversa
        nextBinv = Binv;
        for i = 1:m
            if (i != t)
                nextBinv(i, :) += nextBinv(t, :)*(-u(i)/u(t));
            endif
        endfor
        nextBinv(t, :) = nextBinv(t, :)/(u(t));

        display("Proxima Base de indices (invertida)
encontrada:");
        nextBinv
```

2.3 Full-tableau

Para o método tableau, as mudanças são maiores. Como apenas recebemos o tableau e os índices das variáveis básicas em uma dada iteração, precisamos recuperar todas as informações contidas nele: n, m, x_b = valor das variáveis básicas, c_t = custos reduzidos.

```
    n = size(tableau)(2)-1;
    m = size(tableau)(1)-1;

    disp("Variaveis basicas:");
    for i = 1:m
        disp ([indB(i), tableau(1+i, 1)]);
    endfor
    #display("Instancia de x_b:");
    x_b = tableau(2:end, 1)
```

```
    c_t = tableau(1, 2:end);
    disp(["Vetor de custos reduzidos: ", mat2str(c_t)]);
```

A checagem dos custos reduzidos pode ser apenas lida de c_t em vez de calculada, como nos outros métodos. No entanto, segue o mesmo princípio: procuramos a primeiro custo cujo valor seja negativo (e que não corresponda a uma variável básica, checagem conservadora feita para evitar erros, pois se espera que o custo reduzido de qualquer variável básica seja nulo).

```
    hasNegativeCost = false;
    for j = 1:n
        c_red = c_t(j);
        if (c_red < 0 && !any( indB == j) )
            hasNegativeCost = true;
            break
        endif
    endfor
```

Caso não haja custo negativo, então o método termina, pois já encontramos a solução ótima. Outra vantagem do método tableau é que não precisamos calcular o custo da função objetiva atual, pois já está armazenado na primeira posição do tableau, com sinal invertido. Precisamos, no entanto, construir x = solução ótima, que é da forma: x(i) = 0, para todo i não básico; x(i) = valor correspondente a i na 0-ésima coluna do tableau, para todo i básico. Armazenamos x na variável v e retornamos, finalizando o algoritmo.

```
    if(!hasNegativeCost)
        custo = -tableau(1, 1);
        ind = 1;
        v = zeros(1, n);
        for i = 1:m
            v( indB(i) ) = tableau(i+1, 1);
        endfor

        disp(strcat("Solucao otima encontrada com custo= ",
num2str(custo), ":"));
        for i = 1:n
            disp([i, v(i)]);
        endfor
        return
    endif
```

Caso contrário, o custo atual não é ótimo; podemos reduzir inserindo a variável j. Coletamos u = j-ésima coluna do tableau.

```
    display("Ha custo reduzido negativo");
    disp(["Entra na base: ", int2str(j)]);

    u = tableau(2:end, j+1);
    disp(["Direcao a ser percorrida para proxima instancia de x: u=",
mat2str(u)]);
```

Caso não haja valor positivo em u e θ^* seja avaliado como $-\infty$, construímos a direção d da mesma forma que nos outros métodos.

Caso contrário, a decisão da linha pivotal armazenada em t e o valor de θ^* armazenado em teta também são feitas de forma igual à dos outros métodos.
A diferença será no cálculo do tableau da próxima iteração do método. De forma análoga ao método revisado, queremos adicionar à cada linha do tableau - inclusive a 0-ésima linha - um múltiplo da linha pivotal de forma que a j-ésima coluna do tableau corresponda ao l-ésimo vetor canônico. Precisamos lembrar que apesar de nos referirmos à 0-ésima linha e à 0-ésima coluna por índices 0, na programação elas têm índice 1. Portanto, é necessário corrigir alguns índices para acessá-los no tableau. Após calculado o tableau da próxima iteração e atualizado o vetor de índices da base, trocando l por j, podemos chamar a próxima iteração do método simplex.

```
        indB(t) = j;
        % calculando o proximo tableau
        for i = 1:m+1
            if (i != t+1)
                tableau(i, :) += tableau(t+1, :)*(-tableau(i, j +
1)/tableau(t +1, j+ 1));
            endif
        endfor
        tableau(t+1, :) = tableau(t+1, :)/(u(t));

        #display("Proximo tableau encontrado:");
        #tableau
        [ind v] = simplex_tab_intern(indB, tableau, iteracao + 1);
        return
```

3. EXEMPLOS

Nesta seção a execução do método simplex será acompanhada, em cada implementação, para dois problemas diferentes: um com solução ótima finita e outro ilimitado. Acompanhando este relatório estarão os arquivos que permitem realizar a execução destes mesmos exemplos, contendo as implementações de cada método e instruções para realização dos testes.

3.1 Solução ótima finita

O problema é dado pelas seguintes características iniciais:

Restrições

```
A =

   1   2   0   1
   0   1   1   1
```

```
b =

   10    3
```

```
m = 2;
n = 4;
```

Valor da base inicial

```
B = [ [1; 0] [2; 1] ];
```

```
B =

   1   2
   0   1
```

```
Binv =

   1  -2
   0   1
```

Função de custos

```
c = [4  5  1 -1];
```

Valor da solução viável básica inicial

```
x = [4 3 0 0];
```

Tableau inicial

```
tableau =

  -31     0     0     4    -2
    4     1     0    -2    -1
    3     0     1     1     1
```

Iterações:

1. Os métodos ingênuo e revisado coletam os dados B, Binv, vetor de custos na base, o valor da função objetivo e p e exibem.

```
octave:2> naive
iteracao = 1
Iterando: 1
Variaveis basicas:
   1   4
   2   3
Valor da funcao objetivo: 31
B =

   1   2
   0   1

B_inv =

   1  -2
   0   1

Vetor de custos da base: [4 5]
p: [4 -3]
```

```
octave:1> revised
Iterando: 1
Variaveis basicas:
   1   4
   2   3
Valor da funcao objetivo: 31
B =

   1   2
   0   1

Binv =

   1  -2
   0   1

Vetor de custos da base: [4 5]
p: [4 -3]
```

O tableau coleta o tableau, os índices, os valores das variáveis básicas e o valor da função objetivo e exibe-os.

```
octave:1> tableauV
Iterando: 1
tableau =

  -31    0    0    4   -2
    4    1    0   -2   -1
    3    0    1    1    1

Variaveis basicas:
   1   4
   2   3
x_b =

   4
   3

Valor da funcao objetivo: 31
```

O ingênuo e revisado calculam os custos reduzidos até o primeiro negativo, imprimindo o índice da variável ao lado do custo. j é escolhido para o valor 4 e u é calculado pela multiplicação Binv * A_j.

```
Custos reduzidos das nao basicas (ate primeiro negativo):
   3   4
   4  -2
Ha custo reduzido negativo
Entra na base: 4
Direcao a ser percorrida para proxima instancia de x: u=[-1;1]
```

O tableau apenas imprime a 0-ésima linha e a percorre até encontrar o primeiro valor negativo e seu índice, que é 4, depois coleta u = j-ésima = 4ª coluna.

```
Vetor de custos reduzidos: [0 0 4 -2]
Ha custo reduzido negativo
Entra na base: 4
Direcao a ser percorrida para proxima instancia de x: u=[-1;1]
```

Os três algoritmos vasculham u e determinam o índice que minimiza teta*, o índice 2, com valor teta* = 3.

```
Valor de u positivo encontrado.
Sai da base: 2
Teta*: 3
```

Enquanto o ingênuo passa para a próxima iteração, o revisado calcula a próxima base inversa e a imprime.

```
Proxima Base de indices (invertida) encontrada:
nextBinv =

   1  -1
   0   1
```

Já o tableau calcula o próximo tableau.

```
Proximo tableau encontrado:
tableau =

  -25    0    2    6    0
    7    1    1   -1    0
    3    0    1    1    1
```

2. Na segunda iteração, temos para o ingênuo e o revisado a exibição dos itens atuais. Enquanto o ingênuo calcula a base inversa a partir da inversão direta de B, o revisado carrega o resultado da última iteração.

```
Iterando: 2
Variaveis basicas:
   1   7
   4   3
Valor da funcao objetivo: 25
B =

   1   1
   0   1

B_inv =

   1  -1
   0   1

Vetor de custos da base: [4 -1]
```

Para o tableau, é exibido o tableau atual (igual ao calculado na última iteração), as variáveis básicas com seus valores e o valor da função objetivo.

```
Iterando: 2
tableau =

  -25    0    2    6    0
    7    1    1   -1    0
    3    0    1    1    1

Variaveis basicas:
   1   7
   4   3
x_b =

   7
   3

Valor da funcao objetivo: 25
```

Os métodos ingênuo e revisado calculam p e os custos reduzidos das variáveis não-básicas (2 e 3).

```
p: [4 -5]
Custos reduzidos das nao basicas (ate primeiro negativo):
   2   2
   3   6
```

O tableau imprime a 0-ésima linha contendo o vetor de custos reduzidos de todas as variáveis. Note que como as variáveis básicas são 1 e 4, seus custos reduzidos valem 0. Já para as não-básicas 2 e 3, seus custos reduzidos são iguais àqueles encontrados nos métodos ingênuo e revisado.

```
Vetor de custos reduzidos: [0 2 6 0]
```

Todos os métodos constatam que não há custos negativos e que a solução atual é ótima. Os métodos ingênuo e revisado calculam o custo ótimo, já tendo o valor da solução, enquanto o tableau calcula a solução, já tendo o valor ótimo. Todos exibem os resultados, os retornam e finalizam a execução nesta segunda iteração.

```
Solucao otima encontrada com custo=25:
   1   7
   2   0
   3   0
   4   3
ind = 1
v =

   7   0   0   3

ans = 1
```

3.2 Solução ótima ilimitada

Adicionando ao problema da seção 3.1 uma variável sem restrições cujo custo na função objetiva é negativo, obtemos um problema com solução ilimitada - seu crescimento não é restringido e quanto maior seu valor, menor a função objetiva.
O problema é dado pelas seguintes características iniciais:

Restrições

```
A =

   1   2   0   1   0
   0   1   1   1   0
```

```
b =

   10    3
```

```
m = 2;
n = 5;
```

Valor da base inicial

```
B = [ [1; 0] [2; 1] ];
```

```
B =

   1   2
   0   1
```

```
Binv =

   1  -2
   0   1
```

Função de custos

```
c = [4  5  1 -1 -1];
```

Valor da solução viável básica inicial

```
x = [4 3 0 0 0];
```

Tableau inicial

```
tableau =

  -31     0     0     4    -2    -1
    4     1     0    -2    -1     0
    3     0     1     1     1     0
```

Iterações:

1) Os métodos ingênuo e revisado coletam os dados B, Binv, vetor de custos na base, valor da função objetiva e p e exibem.

```
octave:1> naive
iteracao = 1
Iterando: 1
Variaveis basicas:
   1   4
   2   3
Valor da funcao objetivo: 31
B =

   1   2
   0   1

B_inv =

   1  -2
   0   1

Vetor de custos da base: [4 5]
```

```
octave:1> revised
Iterando: 1
Variaveis basicas:
   1   4
   2   3
B =

   1   2
   0   1

Valor da funcao objetivo: 31
Binv =

   1  -2
   0   1

Vetor de custos da base: [4 5]
```

O tableau coleta o tableau, os índices e os valores das variáveis básicas e o valor da função objetivo.

```
octave:1> tableauV
Iterando: 1
tableau =

  -31    0    0    4   -2   -1
    4    1    0   -2   -1    0
    3    0    1    1    1    0

Variaveis basicas:
   1   4
   2   3
x_b =

   4
   3

Valor da funcao objetivo: 31
```

O ingênuo e revisado calculam os custos reduzidos até o primeiro negativo, imprimindo o índice da variável ao lado do custo. j é escolhido para o valor 4 e u é calculado pela multiplicação Binv * A_j.

```
Custos reduzidos das nao basicas (ate primeiro negativo):
   3   4
   4  -2
Ha custo reduzido negativo
Entra na base: 4
Direcao a ser percorrida para proxima instancia de x: u=[-1;1]
```

O tableau apenas imprime a 0-ésima linha e a percorre até encontrar o primeiro valor negativo e seu índice, que é 4, depois coleta u = j-ésima = 4ª coluna.

```
Vetor de custos reduzidos: [0 0 4 -2 -1]
Ha custo reduzido negativo
Entra na base: 4
Direcao a ser percorrida para proxima instancia de x: u=[-1;1]
```

Os três algoritmos vasculham u e determinam o índice que minimiza teta*, o índice 2, com valor teta* = 3.

```
Valor de u positivo encontrado.
Sai da base: 2
Teta*: 3
```

Enquanto o ingênuo passa para a próxima iteração, o revisado calcula a próxima base inversa e a imprime.

```
Proxima Base de indices (invertida) encontrada:
nextBinv =

   1  -1
   0   1
```

Já o tableau calcula o próximo tableau.

```
Proximo tableau encontrado:
tableau =

  -25    0    2    6    0   -1
    7    1    1   -1    0    0
    3    0    1    1    1    0
```

2) Na segunda iteração, temos para o ingênuo e o revisado a exibição dos itens atuais. Enquanto o ingênuo calcula a base inversa a partir da inversão direta de B, o revisado carrega o resultado da última iteração.

```
Iterando: 2
Variaveis basicas:
   1   7
   4   3
Valor da funcao objetivo: 25
B =

   1   1
   0   1

B_inv =

   1  -1
   0   1

Vetor de custos da base: [4 -1]
```

Para o tableau, é exibido o tableau atual (igual ao calculado na última iteração), as variáveis básicas com seus valores e o valor da função objetivo.

```
Iterando: 2
tableau =

  -25    0    2    6    0   -1
    7    1    1   -1    0    0
    3    0    1    1    1    0

Variaveis basicas:
   1   7
   4   3
x_b =

   7
   3

Valor da funcao objetivo: 25
```

Os métodos ingênuo e revisado calculam p e os custos reduzidos das variáveis não-básicas (2, 3 e 5) até encontrarem o primeiro que seja negativo, que é 5. Calculam u.

```
p: [4 -5]
Custos reduzidos das nao basicas (ate primeiro negativo):
   2   2
   3   6
   5  -1
Ha custo reduzido negativo
Entra na base: 5
Direcao a ser percorrida para proxima instancia de x: u=[0;0]
```

O tableau imprime a 0-ésima linha contendo o vetor de custos reduzidos de todas as variáveis. Note que como as variáveis básicas são 1 e 4, seus custos reduzidos valem 0. Já para as não-básicas 2, 3 e 5, seus custos reduzidos são iguais àqueles encontrados nos métodos ingênuo e revisado. Ele encontra o primeiro custo negativo na variável de índice 5, que será introduzida na base, e obtém u.

```
Vetor de custos reduzidos: [0 2 6 0 -1]
Ha custo reduzido negativo
Entra na base: 5
Direcao a ser percorrida para proxima instancia de x: u=[0;0]
```

Todos os métodos vasculham u mas não encontram nenhuma componente positiva. É determinado, portanto, que o problema tem solução ilimitada. Calculam a direção d, cujos valores nas variáveis básicas correspondem ao vetor u, em que a solução decresce infinitamente, e encontram o vetor d = [0, 0, 0, 0 ,1]: ou seja, crescendo a 5ª variável irrestritamente, sempre acharemos uma solução de valor menor na função objetiva. Os métodos exibem e retornam seus resultados, finalizando a execução.

```
Problema de solucao ilimitada
direcao em que decresce infinitamente: [0 0 0 0 1]
ind = -1
v =

   0   0   0   0   1

ans = -1
```