Projeto Computacional

Gabriel Belém Barbosa
RA: 234672
Vanessa Vitória de Arruda Pachalki
RA: 244956

01 de Outubro de 2021

Conteúdo

1 Exercício **3**

2 Algoritmo por partes **3**
- 2.1 Inicialização e entrada . . . 3
- 2.2 Forma Big-M e base inicial . . . 3
- 2.3 Método simplex . . . 4
- 2.4 Análise da factibilidade do PL original . . . 7

3 Algoritmo completo **7**

4 Exemplos **9**
- 4.1 PL infactível . . . 9
- 4.2 PL factível . . . 10
- 4.3 PL sem solução ótima limitada . . . 12

5 Referências Bibliográficas **13**

1 Exercício

Neste trabalho foi desenvolvido um projeto computacional com o objetivo de implementar o método simplex estudado na matéria de MS428. Este método é uma ferramenta muito útil para a resolução de problemas de programação linear (PL). Para um melhor entendimento da implementação, no decorrer do trabalho são apresentados os raciocícios utilizados no algoritmo e, além disso, alguns exemplos para auxiliar o usuário na utilização e demonstrar o funcionamento do algoritmo em alguns PL's com comportamentos distintos. Esse projeto se pautará na nomenclatura e teoria apresentadas em aula.

2 Algoritmo por partes

2.1 Inicialização e entrada

Para iniciar o algoritmo é criado o array (matriz) `A`, matriz A do PL na forma padrão, o vetor `b`, definido analogamente, e, por fim, o vetor `custos`, vetor denominado c originalmente. É importante ressaltar que o problema deve ser informado na forma padrão. A ordem utilizada para adquirir esses dados do usuário foi definida como:

1) Solicitar o tamanho da matriz A;

2) Solicitar os valores por posição na matriz A sequencialmente (percorrendo por colunas e depois por linhas);

3) Solicitar os valores de b sequencialmente;

4) Solicitar os valores dos custos sequencialmente.

Algoritmo:

```
A=[];
b=[];
custos=[];
m=input("Número de linhas: ");
n=input("Número de colunas: ");
for i=1:m
    aux=[];
    for j=1:n
        v=input(strcat("Elemento a_",num2str(i),num2str(j),":"));
        aux=[aux, v];
    endfor
    A=[A; aux];
endfor
for i=1:m
    v=input(strcat("Elemento b_",num2str(i),":"));
    b=[b; v];
endfor
for i=1:n
    v=input(strcat("Elemento c_",num2str(i),":"));
    custos=[custos; v];
endfor
```

2.2 Forma Big-M e base inicial

O método utilizado para solucionar o PL foi o Big-M. Esse método tem como característica a inserção de uma variável auxiliar grande o suficiente na função de custo e a utilização de variáveis de folga nas restrições. A seguir, algumas variáveis são criadas. Primeiro `iB` e `iN`, que são os índices das colunas de A que pertencem à base ou não, respectivamente. Então `M` foi definido como 100 vezes o maior custo fornecido. Em seguida, a variável `where_y`, que conterá quais linhas de A precisarão receber uma variável auxiliar, foi inicializada com todas as linhas de A. Algoritmo:

```
iB=[];
iN=[];
m=rows(A);
n=columns(A);
where_y=linspace(1, m, m);
M=100*max(abs(custos));
i=0;
for c=transpose(custos)
    i++;
    if (c==0)
```

```
        j=find(A(:, i)!=0);
        if ((rows(j)==1) & (sign(A(j, i))*sign(b(j))>=0))
            j=find(where_y==j);
            if (j)
                where_y(j)=[];
                iB=[iB, i];
            else
                iN=[iN,i];
            endif
        else
            iN=[iN, i];
        endif
    else
        iN=[iN, i];
    endif
endfor
j=n;
for i=where_y
    j++;
    aux=zeros(m, 1);
    if (b(i)>=0)
        aux(i)=1;
    else
        aux(i)=-1;
    endif
    A=[A, aux];
    custos=[custos; M] ;
    iB=[iB, j];
endfor
A
iB
sol=[];
B=[];
Cb=[];
limitada=1;
for i=iB
    B=[B, A(:, i)];
    Cb=[Cb; custos(i)];
endfor
B
while (1)
    .
    .
    .
endwhile
```

Um `for` que percorre as variáveis da forma padrão. Se uma variável possuir custo associado nulo, se a coluna associada a ela em A tiver um único elemento não nulo e o sinal de tal elemento for igual ao sinal do elemento de b daquela linha, esse elemento entrará para a base inicial (e seu índice será adicionado a `iB`), e o elemento de `where_y` que representa a linha de b que será satisfeita por essa variável é excluído, pois não há necessidade de criar uma variável auxiliar nessa linha. Caso contrário, a variável não estará na base inicial (e seu índice será adicionado a `iN`).

Após isso, os elementos restantes de `where_y` são usados parar criar as variáveis auxiliares (cujos pesos `M` são adicionados aos custos) nas linhas que não foram contempladas, com sinais adequados para os elementos de b de suas respectivas linhas (elementos de `where_y`).

Finalmente, a matriz básica inicial B (`B`) e o vetor de custos básicos c_B (`Cb`) são criados por um `for` que percorre os elementos de `iB` e acopla as colunas de A e custos correspondentes em B e c_B, respectivamente.

2.3 Método simplex

A partir da base adquirida anteriormente, o algoritmo entra em `while(1)` (fase simplex) que só será quebrado quando uma solução for obtida ou a infactibildiade ou não existência de solução ótima limitada seja detectada pelo algoritmo. Os seguintes passos são então efetuados:

- Passo 1: Cálculo da solução básica

$$B\hat{\mathbf{x}}_\mathbf{B} = b$$

$$\hat{\mathbf{x}}_\mathbf{B} = B \setminus b$$

Usando a função `linsolve` do `Octave` (através do atalho \), a solução básica ($\hat{x}_B$) acima é encontrada. Sabe-se que quando $\hat{x}_B \ngeq 0$ a solução é infactível. O programa então busca qualquer elemento menor que zero em $\hat{x}_B$, e, se este existe, o `while` é quebrado, e a infactibilidade será detectada mais à frente na seção 2.4. Caso contrário, é calculado (e printado) o valor da função na solução básica, que é $c_B^T \hat{x}_B$ e o método avança para o próximo passo. Algoritmo:

```
Xb=B\b
if (any(Xb<0))
    break;
endif
transpose(Cb)*Xb
```

- Passo 2: Cálculo dos custos relativos
  - 2.1: Cálculo do vetor multiplicador simplex

$$\lambda^{\mathbf{T}} = \mathbf{c_B^T B^{-1}}$$

$$B^T \lambda = \mathbf{c_B}$$

$$\lambda^T = (B^T \setminus \mathbf{c_B})^T$$

Novamente usando a função `linsolve`, o sistema linear acima é resolvido. Algoritmo:

```
lambda_T=transpose(transpose(B)\Cb);
```

  - 2.2 e 2.3: Custos relativos e escolha da variável a entrar na base
Existem diversos métodos para escolher a variável que entrará na base. Neste caso, optou-se por escolher a variável não básica cujo custo relativo é o menor. Assim, temos:

$$\hat{c}_{N_i} = c_{N_i} - \lambda^T \mathbf{a_{N_i}}$$

$$\hat{c}^*_{N_i} = min\{\hat{c}_{N_i}, i = 1, 2, ..., m-n\}$$

A variável `minimo_redux` é criada para armazenar o menor custo relativo ($\hat{c}^*_{N_i}$). Ela é inicializada como como o maior float possível (o que implica que o primeiro custo relativo, independente de seu valor, atualizará esse valor inicial). Durante a primeira iteração garantidamente será criada a variável `N_index`, o índice (em N) da variável com o menor custo relativo. O algoritmo então percorre os elementos de `iN` e calcula seu custo relativo e, caso necessário, atualiza o novo mínimo e `N_index`. Algoritmo:

```
minimo_redux=realmax;
j=0;
for i=iN
    j++;
    custo_redux=custos(i)-lambda_T*A(:, i);
    if (custo_redux<minimo_redux)
        minimo_redux=custo_redux;
        N_index=j;
    endif
endfor
in_index=iN(N_index)
```

Por fim, uma variável `in_index` é criada e definida como o elemento na posição `N_index` de `iN`, isto é, essa nova variável é o índice da variável/coluna de A que possivelmente entrará na base.

- Passo 3: Teste de otimalidade

Neste passo, se for encontrado algum valor $\hat{c}_{N_i} < 0$, significa que a solução não é ótima e por isso o método deve prosseguir para o próximo passo. Basta portanto checar pela não negatividade do mínimo custo encontrado, e caso esta se confirme, o algoritmo quebra o `while`, e a otimalidade será detectada mais à frente na seção 2.4. Algoritmo:

```
if (minimo_redux>=0)
    break;
endif
```

- Passo 4: Cálculo da direção simplex

$$\mathbf{y} = B^{-1}\mathbf{a_{N_i}}$$

$$B\mathbf{y} = \mathbf{a_{N_i}}$$

$$\mathbf{y} = B \setminus \mathbf{a_{N_i}}$$

Novamente usando a função `linsolve`, o sistema linear acima é resolvido. Algoritmo:

```
y=B\A(:, in_index);
```

- Passo 5: Determinar passo e variável a sair da base

Nesta etapa, são calculados, para todos os $y_j > 0$, o passo máximo associado às variáveis de B. A variável `min_epsilon` é criada como o maior valor de float (de forma semelhante ao passo 2.2 e 2.3) e conterá $\hat{\varepsilon}$ definido como

$$\hat{\varepsilon} = min\left\{\frac{\hat{x}_{B_j}}{y_j},\ \text{t.q. } y_j > 0, j = 1, 2..., n\right\}$$

Também é criada `B_index`, índice (em B) da coluna que sairá da base, que começa como 0, fato que será importante a seguir. Um `for` percorre os elementos de y e, se o elemento for positivo, calcula seu passo máximo, e atualiza tanto `min_epsilon` quanto `B_index`, caso necessário. Note que, se não existir nenhum elemento de y maior que zero (isto é, $y \leq 0$), o algoritmo não entra na primeira condicional e ao final do `for`, `iB_index` nunca é atualizado e continua nulo. Algoritmo:

```
min_epsilon=realmax;
B_index=0;
for j=1:m
    if (y(j)>0)
        epsilon=Xb(j)/y(j);
        if (epsilon<=min_epsilon)
            B_index=j;
            min_epsilon=epsilon;
        endif
    endif
endfor

if (B_index)
    .
    .
    .
else
    display("Sem solucao limitada.");
    limitada=0;
    break;
endif
```

Em seguida, como $y \leq 0$ ocorre quando o PL não possui solução ótima finita, é checado se `iB_index==0` (que é equivalente a essa condição de y, como explciado anteriormente). Em caso afirmativo, o `while` é quebrado, e esse fato será printado no `else` da condicional, e a variável `limitada` é definida como nula para que seja distinguido esse caso na hora do print da solução final. Caso contrário, a variável da base que representar o menor valor obtido será aquela que deixará a base e o método prossegue.

- Passo 6: Atualizar

Nesse passo basta fazer a atualização da base e recomeçar o método a partir do passo 1. A variável `out_index` é criada e definida como o elemento na posição `B_index` de `iB`, isto é, essa nova variável é o índice da variável/coluna de A que sairá da base. As substituições são então feitas de acordo com a definição das variáveis envolvidas. Algoritmo:

```
out_index=iB(B_index)
iN(N_index)=out_index;
iB(B_index)=in_index
B(:, B_index)=A(:, in_index)
Cb(B_index)=custos(in_index);
```

2.4 Análise da factibilidade do PL original

Após sair do `while`, ou seja, ao final do método, é preciso identificar e reportar o motivo do fim do algoritmo. Se alguma variável auxiliar do método Big-M continuar na base nesse ponto, o que é checado buscando-se qualquer elemento em `iB` maior que o número de variáveis do PL na forma padrão, n, tem-se que o PL na forma padrão é infactível, e o algoritmo printa tal conclusão.

Algoritmo:

```
while (1)
    .
    .
    .
endwhile
if any(iB>n)
        display("PL infactivel.");
elseif(limitada)
        display("Solucao otima (com indices correspondentes) e valor otimo.")
        Xb
        iB
        transpose(Cb)*Xb
endif
```

3 Algoritmo completo

```
#Inicializacao e entrada
A=[];
b=[];
custos=[];
m=input("Número de linhas: ");
n=input("Número de colunas: ");
for i=1:m
    aux=[];
    for j=1:n
        v=input(strcat("Elemento a_",num2str(i),num2str(j),":"));
        aux=[aux, v];
    endfor
    A=[A; aux];
endfor
for i=1:m
    v=input(strcat("Elemento b_",num2str(i),":"));
    b=[b; v];
endfor
for i=1:n
    v=input(strcat("Elemento c_",num2str(i),":"));
    custos=[custos; v];
endfor
function sol=Big_M(A, custos, b)

    #Forma Big-M e base inicial
    iB=[];
    iN=[];
    m=rows(A);
    n=columns(A);
    where_y=linspace(1, m, m);
    M=100*max(abs(custos));
    i=0;
    for c=transpose(custos)
        i++;
        if (c==0)
            j=find(A(:, i)!=0);
            if ((rows(j)==1) & (sign(A(j, i))*sign(b(j))>=0))
              j=find(where_y==j);
              if (j)
                where_y(j)=[];
                iB=[iB, i];
              else
                iN=[iN,i];
              endif
          else
              iN=[iN, i];
```

```
            endif
        else
            iN=[iN, i];
        endif
    endfor
    j=n;
    for i=where_y
        j++;
        aux=zeros(m, 1);
        if (b(i)>=0)
            aux(i)=1;
        else
            aux(i)=-1;
        endif
        A=[A, aux];
        custos=[custos; M] ;
        iB=[iB, j];
    endfor

    #Matriz A na forma Big-M
    A
    iB
    sol=[];
    B=[];
    Cb=[];
    limitada=1;
    for i=iB
        B=[B, A(:, i)];
        Cb=[Cb; custos(i)];
    endfor

    #Base
    B

    #Metodo simplex
    while (1)

        #Passo 1
        Xb=B\b
        if (any(Xb<0))
            break;
        endif

        #f da soluçao basica
        transpose(Cb)*Xb
        sol=[sol, [Xb; transpose(iB)]];

        #Passo 2.1
        lambda_T=transpose(transpose(B)\Cb);

        #Passo 2.2 e 2.3
        minimo_redux=realmax;
        j=0;
        for i=iN
            j++;
            custo_redux=custos(i)-lambda_T*A(:, i);
            if (custo_redux<minimo_redux)
                minimo_redux=custo_redux;
                N_index=j;
            endif
        endfor
        in_index=iN(N_index)

        #Passo 3
        if (minimo_redux>=0)
            break;
        endif

        #Passo 4
        y=B\A(:, in_index);

        #Passo 5
        min_epsilon=realmax;
```

```
        B_index=0;
        for j=1:m
            if (y(j)>0)
                epsilon=Xb(j)/y(j);
                if (epsilon<=min_epsilon)
                    B_index=j;
                    min_epsilon=epsilon;
                endif
            endif
        endfor

        if (B_index)
            out_index=iB(B_index)
            iN(N_index)=out_index;
            iB(B_index)=in_index
            B(:, B_index)=A(:, in_index)
            Cb(B_index)=custos(in_index);
        else
            display("Sem solucao limitada.");
            limitada=0;
            break;
        endif
    endwhile

    #Analise da factibilidade do PL original
    if any(iB>n)
        display("PL infactivel.");
    elseif(limitada)
        display("Solucao otima (com indices correspondentes) e valor otimo.")
        Xb
        iB
        transpose(Cb)*Xb
    endif
endfunction

sol=Big_M(A, custos, b);
```

4 Exemplos

A seguir, com o intuito de testar o algoritmo, encontram-se alguns exemplos de PL que requerem diferentes saídas. Tais PL's foram obtidos das atividades da disciplina, e possuem comportamento conhecido. No algoritmo foram limitado os prints a alguns elementos essenciais; para observar com mais detalhes seu funcionamento, recomenda-se chamar mais prints (retirando-se `;` do final das linhas de código, quando desejado) na atualização de outras variáveis, como `lambda_T`, `y`, `minimo_redux` e `min_epsilon`.

4.1 PL infactível

$min\ f(x) = -6x_1 + 4x_2$

$$\text{Sujeito a: } \begin{cases} x_1 + x_2 + x_3 = 4 \\ 2x_1 + 3x_2 - x_4 = 18 \\ x_1 \geq 0, x_2 \geq 0, x_3 \geq 0, x_4 \geq 0 \end{cases}$$

Cópia do terminal:

```
Número de linhas: 2
Número de colunas: 4
Elemento a_11:1
Elemento a_12:1
Elemento a_13:1
Elemento a_14:0
Elemento a_21:2
Elemento a_22:3
Elemento a_23:0
Elemento a_24:-1
```

```
Elemento b_1:4
Elemento b_2:18
Elemento c_1:-6
Elemento c_2:4
Elemento c_3:0
Elemento c_4:0
A =

   1   1   1   0   0
   2   3   0  -1   1

iB =

   3   5

B =

   1   0
   0   1

Xb =

    4
   18

ans = 10800
in_index = 2
out_index = 3
iB =

   2   5

B =

   1   0
   3   1

Xb =

   4
   6

ans = 3616
in_index = 1
PL infactivel.
```

4.2 PL factível

$min\ f(x) = 2x_1 + x_2$

$$\text{Sujeito a:} \begin{cases} x_1 + 6.5x_2 - x_3 = 5 \\ 2x_1 + x_2 - x_4 = 4 \\ 5x_1 + 4x_2 + x_5 = 20 \\ x_1 \geq 0, x_2 \geq 0, x_3 \geq 0, x_4 \geq 0, x_5 \geq 0 \end{cases}$$

Cópia do terminal:

```
Número de linhas: 3
Número de colunas: 5
```

```
Elemento a_11:1
Elemento a_12:6.5
Elemento a_13:-1
Elemento a_14:0
Elemento a_15:0
Elemento a_21:2
Elemento a_22:1
Elemento a_23:0
Elemento a_24:-1
Elemento a_25:0
Elemento a_31:5
Elemento a_32:4
Elemento a_33:0
Elemento a_34:0
Elemento a_35:1
Elemento b_1:5
Elemento b_2:4
Elemento b_3:20
Elemento c_1:2
Elemento c_2:1
Elemento c_3:0
Elemento c_4:0
Elemento c_5:0
A =

   1.0000   6.5000  -1.0000        0        0   1.0000        0
   2.0000   1.0000        0  -1.0000        0        0   1.0000
   5.0000   4.0000        0        0   1.0000        0        0

iB =

   5   6   7

B =

   0   1   0
   0   0   1
   1   0   0

Xb =

   20
    5
    4

ans = 1800
in_index = 2
out_index = 6
iB =

   5   2   7

B =

        0   6.5000        0
        0   1.0000   1.0000
   1.0000   4.0000        0
```

```
Xb =

   16.9231
    0.7692
    3.2308

ans = 646.92
in_index = 1
out_index = 7
iB =

   5   2   1

B =

        0   6.5000   1.0000
        0   1.0000   2.0000
   1.0000   4.0000   5.0000

Xb =

   9.2500
   0.5000
   1.7500

ans = 4
in_index = 3
Solucao otima (com indices correspondentes) e valor otimo.
Xb =

   9.2500
   0.5000
   1.7500

iB =

   5   2   1

ans = 4
```

4.3 PL sem solução ótima limitada

$min\ f(x) = -2x_1 - 1x_2$

$$\text{Sujeito a: } \begin{cases} -2x_1 + x_2 + x_3 = 2 \\ -2x_1 + 3x_2 - x_4 = -6 \\ x_1 \geq 0, x_2 \geq 0, x_3 \geq 0, x_4 \geq 0 \end{cases}$$

Cópia do terminal:

```
Número de linhas: 2
Número de colunas: 4
Elemento a_11:-2
Elemento a_12:1
Elemento a_13:1
Elemento a_14:0
Elemento a_21:-2
Elemento a_22:3
Elemento a_23:0
```

```
Elemento a_24:-1
Elemento b_1:2
Elemento b_2:-6
Elemento c_1:-2
Elemento c_2:-1
Elemento c_3:0
Elemento c_4:0
A =

  -2   1   1   0
  -2   3   0  -1

iB =

   3   4

B =

   1   0
   0  -1

Xb =

   2
   6

ans = 0
in_index = 1
out_index = 4
iB =

   3   1

B =

   1  -2
   0  -2

Xb =

   8
   3

ans = -6
in_index = 2
Sem solucao limitada.
```

5 Referências Bibliográficas

- Arenales, M.; Armentano, V. A.; Morabito, R.; Yanasse, H. H. **Pesquisa operacional**. Rio de Janeiro: Campus, 2007.
- Bazaraa, M. Jarvis, J., Sherali, H. **Linear Programming and Network Flows**. John Wiley & Sons, 1990.