Universidade Federal do Piauí – UFPI
Centro de Ciências da Natureza – CCN
Departamento de Ciências da Computação – DC
Bacharelado em Ciência da Computação
Disciplina: Programação Linear
Professor: Antônio Costa de Oliveira

Relatório de Programação Linear

(Modelagem, resolução de problemas, código-fonte e explicação do código simplex)

Hugo Santos Piauilino Neto
Luís Guilherme Teixeira dos Santos
Natasha Rebelo Oliveira

Fevereiro de 2016

MODELAGENS E RESOLUÇÕES DOS PROBLEMAS

1. Planejamento Urbano

Variáveis de decisão:

X_1 = Quantidade de apartamentos funcionais;
X_2 = Quantidade de apartamentos duplex;
X_3 = Quantidade de apartamentos residenciais simples;
X_4 = Quantidade de área de comercio varejista.

Função objetivo:

Max Z = 600 X_1 + 750 X_2 + 1200 X_3 + 100 X_4

*Visa maximizar os lucros gerados pela construção dos três tipos de apartamentos e com o aluguel das áreas para comércio varejista.

Sujeito a:

$$\begin{cases} X1 \leq 500\ (Apartamento\ 1) \\ X2 \leq 300\ (Apartamento\ 2) \\ X3 \leq 250\ (Apartamento\ 3) \\ X4 \leq 10000\ (Área\ Comercial) \\ 2\,X2 - X1 - X3 \geq 0\ (Demanda\ Apartamento\ 2) \\ 10\,X1 + 15\,X2 + 18\,X3 - X4 \geq 0\big(Demanda\ Área\ Comércial\big) \\ X1, X2, X3, X4 \geq 0 \end{cases}$$

Existem dois grupos com restrições relacionadas entre si:

1° grupo – Restrições relacionadas a demanda de inquilinos de cada apartamento:

- Demanda máxima estimada do apartamento 1 é de 500 apartamentos;
- Demanda máxima estimada do apartamento 2 é de 300 apartamentos;
- Demanda máxima estimada do apartamento 3 é de 250 apartamentos;
- Demanda máxima estimada do apartamento 2 é de no mínimo 50% do número de apartamentos 1 e 3.

2° grupo – Restrições relacionadas ao espaço para o comércio varejista:

- O comércio varejista é proporcional ao número de apartamentos à razão de 10 pés^2, 15 pés^2 e 18 pés^2 para os apartamentos 1, 2 e 3.

Resolução:

```
MAX 600 X1 + 750 X2 + 1200 X3 + 100 X4
ST
X1 <= 500
X2 <= 300
X3 <= 250
X4 <= 10000
2 X2 - X1 - X3 >= 0
10 X1 + 15 X2 + 18 X3 - X4 >= 0
END
```

Figura 1. Entrada de dados do software LINDO para o problema 1.

```
LP OPTIMUM FOUND AT STEP      4

       OBJECTIVE FUNCTION VALUE

       1)      1735000.

 VARIABLE        VALUE          REDUCED COST
       X1       350.000000          0.000000
       X2       300.000000          0.000000
       X3       250.000000          0.000000
       X4     10000.000000          0.000000

      ROW   SLACK OR SURPLUS     DUAL PRICES
       2)       150.000000          0.000000
       3)         0.000000       1950.000000
       4)         0.000000        600.000000
       5)         0.000000        100.000000
       6)         0.000000       -600.000000
       7)      2500.000000          0.000000

NO. ITERATIONS=       4

RANGES IN WHICH THE BASIS IS UNCHANGED:

                          OBJ COEFFICIENT RANGES
VARIABLE         CURRENT        ALLOWABLE        ALLOWABLE
                   COEF          INCREASE         DECREASE
      X1       600.000000       600.000000       600.000000
      X2       750.000000        INFINITY       1950.000000
      X3      1200.000000        INFINITY        600.000000
      X4       100.000000        INFINITY        100.000000

                          RIGHTHAND SIDE RANGES
     ROW         CURRENT        ALLOWABLE        ALLOWABLE
                   RHS           INCREASE         DECREASE
       2       500.000000        INFINITY        150.000000
       3       300.000000        75.000000        71.428574
       4       250.000000       350.000000       150.000000
       5     10000.000000      2500.000000     10000.000000
       6         0.000000       250.000000       150.000000
       7         0.000000      2500.000000        INFINITY
```

Figura 2. Valores de cada uma das variáveis do problema 1.

```
THE TABLEAU

     ROW  (BASIS)         X1        X2        X3        X4     SLK   2    SLK   3
       1 ART            0.000     0.000     0.000     0.000      0.000  1950.000
       2 SLK      2     0.000     0.000     0.000     0.000      1.000    -2.000
       3        X1      1.000     0.000     0.000     0.000      0.000     2.000
       4        X2      0.000     1.000     0.000     0.000      0.000     1.000
       5        X4      0.000     0.000     0.000     1.000      0.000     0.000
       6        X3      0.000     0.000     1.000     0.000      0.000     0.000
       7 SLK      7     0.000     0.000     0.000     0.000      0.000    35.000

     ROW   SLK    4   SLK    5   SLK    6   SLK    7
       1  0.60E+03  0.10E+03  0.60E+03  0.00E+00  0.17E+07
       2     1.000     0.000    -1.000     0.000   150.000
       3    -1.000     0.000     1.000     0.000   350.000
       4     0.000     0.000     0.000     0.000   300.000
       5     0.000     1.000     0.000     0.000 10000.000
       6     1.000     0.000     0.000     0.000   250.000
       7     8.000    -1.000    10.000     1.000  2500.000
```

Figura 3. Tableau gerado pelo LINDO para o problema 1.

Na Figura 3 analisamos a solução ótima, a base está marcada de verde, com a linha azul estão os coeficientes relativos na função objetivo, a interseção entre um retângulo azul e um retângulo vermelho representa os coeficientes relativos para cada uma das variáveis básicas. A solução ótima para este PPL é (350, 300, 250, 10000), resultando em 1735000 de receita mensal em dólares.

2. Programação e Distribuição da Produção

De Fábrica:	Para Depósito:	
	1. Denver	2. Cincinnati
1. Detroit	\$1253	\$637
2. Atlanta	\$1398	\$841

De Depósito:	Para Cidade Cliente:		
	1. Los Angeles	2. Chicago	3. Philadelphia
1. Denver	\$1059	\$996	\$1691
2. Cincinnati	\$2786	\$802	\$700

Variáveis de decisão:

$X_{i\,j\,k}$ = Quantidade de carros que serão produzidos onde:

- “i” representa a fábrica e varia entre 1 e 2;
- “j” representa o depósito e varia entre 1 e 2;
- “k” representa as cidades clientes e varia entre 1 e 3.

$\mathbf{X_{111}}$ **=** Quantidade de carros produzidos na fábrica 1, armazenados no depósito 1 e vendidos para a cidade cliente 1.

$\mathbf{X_{112}}$ **=** Quantidade de carros produzidos na fábrica 1, armazenados no depósito 1 e vendidos para a cidade cliente 2.

$\mathbf{X_{113}}$ **=** Quantidade de carros produzidos na fábrica 1, armazenados no depósito 1 e vendidos para a cidade cliente 3.

$\mathbf{X_{121}}$ **=** Quantidade de carros produzidos na fábrica 1, armazenados no depósito 2 e vendidos para a cidade cliente 1.

$\mathbf{X_{122}}$ **=** Quantidade de carros produzidos na fábrica 1, armazenados no depósito 2 e vendidos para a cidade cliente 2.

$\mathbf{X_{123}}$ **=** Quantidade de carros produzidos na fábrica 1, armazenados no depósito 2 e vendidos para a cidade cliente 3.

$\mathbf{X_{211}}$ **=** Quantidade de carros produzidos na fábrica 2, armazenados no depósito 1 e vendidos para a cidade cliente 1.

$\mathbf{X_{212}}$ **=** Quantidade de carros produzidos na fábrica 2, armazenados no depósito 1 e vendidos para a cidade cliente 2.

$\mathbf{X_{213}}$ **=** Quantidade de carros produzidos na fábrica 2, armazenados no depósito 1 e vendidos para a cidade cliente 3.

$\mathbf{X_{221}}$ **=** Quantidade de carros produzidos na fábrica 2, armazenados no depósito 2 e vendidos para a cidade o cliente 1.

X_{222} = Quantidade de carros produzidos na fábrica 2, armazenados no depósito 2 e vendidos para a cidade cliente 2.

X_{223} = Quantidade de carros produzidos na fábrica 2, armazenados no depósito 2 e vendidos para a cidade cliente 3.

Função objetivo:

Min Z = 12312 $\mathbf{X_{111}}$ + 12249 $\mathbf{X_{112}}$ + 12944 $\mathbf{X_{113}}$ + 13423 $\mathbf{X_{121}}$ + 11439 $\mathbf{X_{122}}$ + 11337 $\mathbf{X_{123}}$ + 12457 $\mathbf{X_{211}}$ + 12394 $\mathbf{X_{212}}$ + 12989 $\mathbf{X_{213}}$ + 13627 $\mathbf{X_{221}}$ + 11643 $\mathbf{X_{222}}$ + 11541 $\mathbf{X_{223}}$

*Visa minimizar os custos gerados pelo transporte dos carros produzidos, onde o coeficiente de cada variável representa a quantidade de carros que serão transportados por cada rota.

Soma simplificada de:

10000 + (1253+1059)	$\mathbf{X_{111}}$	= 12312 $\mathbf{X_{111}}$
10000 + (1253+996)	$\mathbf{X_{112}}$	= 12249 $\mathbf{X_{112}}$
10000 + (1253+1691)	$\mathbf{X_{113}}$	= 12944 $\mathbf{X_{113}}$
10000 + (637+2786)	$\mathbf{X_{121}}$	= 13423 $\mathbf{X_{121}}$
10000 + (637+802)	$\mathbf{X_{122}}$	= 11439 $\mathbf{X_{122}}$
10000 + (637+700)	$\mathbf{X_{123}}$	= 11337 $\mathbf{X_{123}}$
10000 + (1398+1059)	$\mathbf{X_{211}}$	= 12457 $\mathbf{X_{211}}$
10000 + (1398+996)	$\mathbf{X_{212}}$	= 12394 $\mathbf{X_{212}}$
10000 + (1298+1691)	$\mathbf{X_{213}}$	= 12989 $\mathbf{X_{213}}$
10000 + (841+2786)	$\mathbf{X_{221}}$	= 13627 $\mathbf{X_{221}}$
10000 + (841+802)	$\mathbf{X_{222}}$	= 11643 $\mathbf{X_{222}}$
10000 + (841+700)	$\mathbf{X_{223}}$	= 11541 $\mathbf{X_{223}}$

Onde o valor de produção de cada carro é somado aos valores de transporte da fábrica para o depósito e do depósito para o cliente final, respectivamente.

Sujeito a:

$$\begin{cases} X111 + X112 + X113 + X121 + X122 + X123 \leq 110\ (Fábrica\ 1) \\ X211 + X212 + X213 + X221 + X222 + X223 \leq 100\ (Fábrica\ 2) \\ X111 + X211 + X121 + X221 \geq 80\ (Cliente\ 1) \\ X112 + X212 + X122 + X222 \geq 70\ (Cliente\ 2) \\ X113 + X213 + X123 + X223 \geq 60\ (Cliente\ 3) \\ X111, X112, X113, X121, X122, X123, X211, X212, X213, X221, X222, X223 \geq 0 \end{cases}$$

Existem dois grupos com restrições relacionadas entre si:

1° grupo – Restrições relacionadas a capacidade de produção de cada fábrica:

- Produção máxima da fábrica 1 é de 110 carros/semana;
- Produção máxima da fábrica 2 é de 100 carros/semana.

2° grupo – Restrições relacionadas com o compromisso de venda de cada cidade:

- Compromisso de venda mínimo do cliente 1 é de 80 carros/semana;

- Compromisso de venda mínimo do cliente 2 é de 70 carros/semana;
- Compromisso de venda mínimo do cliente 3 é de 60 carros/semana.

Resolução:

```
MIN 12312X111 + 12249X112 + 12944X113 + 13423X121 + 11439X122 + 11337X123
+ 12457X211 + 12394X212 + 12989X213 + 13627X221 + 11643X222 + 11541X223
ST
X111 + X112 + X113 + X121 + X122 + X123 <= 110
X211 + X212 + X213 + X221 + X222 + X223 <= 100
X111 + X211 + X121 + X221 >= 80
X112 + X212 + X122 + X222 >= 70
X113 + X213 + X123 + X223 >= 60
END
```

Figura 4. Entrada de dados do software LINDO para o problema 2.

```
LP OPTIMUM FOUND AT STEP      4

       OBJECTIVE FUNCTION VALUE

       1)      2481590.

 VARIABLE        VALUE          REDUCED COST
     X111         0.000000         59.000000
     X112         0.000000        810.000000
     X113         0.000000       1607.000000
     X121         0.000000       1170.000000
     X122        50.000000          0.000000
     X123        60.000000          0.000000
     X211        80.000000          0.000000
     X212         0.000000        751.000000
     X213         0.000000       1448.000000
     X221         0.000000       1170.000000
     X222        20.000000          0.000000
     X223         0.000000          0.000000

      ROW   SLACK OR SURPLUS     DUAL PRICES
       2)         0.000000        204.000000
       3)         0.000000          0.000000
       4)         0.000000     -12457.000000
       5)         0.000000     -11643.000000
       6)         0.000000     -11541.000000

NO. ITERATIONS=       4

RANGES IN WHICH THE BASIS IS UNCHANGED:

                           OBJ COEFFICIENT RANGES
VARIABLE         CURRENT        ALLOWABLE        ALLOWABLE
                  COEF          INCREASE         DECREASE
    X111     12312.000000       INFINITY         59.000000
    X112     12249.000000       INFINITY        810.000000
    X113     12944.000000       INFINITY       1607.000000
    X121     13423.000000       INFINITY       1170.000000
    X122     11439.000000      59.000000          0.000000
    X123     11337.000000       0.000000      11541.000000
    X211     12457.000000      59.000000      12457.000000
    X212     12394.000000       INFINITY        751.000000
    X213     12989.000000       INFINITY       1448.000000
    X221     13627.000000       INFINITY       1170.000000
    X222     11643.000000       0.000000         59.000000
    X223     11541.000000       INFINITY          0.000000

                           RIGHTHAND SIDE RANGES
     ROW         CURRENT        ALLOWABLE        ALLOWABLE
                   RHS          INCREASE         DECREASE
       2       110.000000      20.000000          0.000000
       3       100.000000       INFINITY          0.000000
       4        80.000000       0.000000         80.000000
       5        70.000000       0.000000         20.000000
       6        60.000000       0.000000         20.000000
```

Figura 5. Valores de cada uma das variáveis do problema 2.

```
THE TABLEAU

    ROW  (BASIS)        X111       X112       X113       X121       X122       X123
      1 ART           59.000    810.000   1607.000   1170.000      0.000      0.000
      2      X122      1.000      1.000      0.000      1.000      1.000      0.000
      3 SLK     3      0.000      0.000      0.000      0.000      0.000      0.000
      4      X211      1.000      0.000      0.000      1.000      0.000      0.000
      5      X222     -1.000      0.000      0.000     -1.000      0.000      0.000
      6      X123      0.000      0.000      1.000      0.000      0.000      1.000

    ROW         X211       X212       X213       X221       X222       X223   SLK     2
      1        0.000    751.000   1448.000   1170.000      0.000      0.000    204.000
      2        0.000      0.000     -1.000      0.000      0.000     -1.000      1.000
      3        0.000      0.000      0.000      0.000      0.000      0.000      1.000
      4        1.000      0.000      0.000      1.000      0.000      0.000      0.000
      5        0.000      1.000      1.000      0.000      1.000      1.000     -1.000
      6        0.000      0.000      1.000      0.000      0.000      1.000      0.000

    ROW     SLK     3  SLK     4  SLK     5  SLK     6
      1    0.00E+00   0.12E+05   0.12E+05   0.12E+05  -0.25E+07
      2       0.000      0.000      0.000      1.000     50.000
      3       1.000      1.000      1.000      1.000      0.000
      4       0.000     -1.000      0.000      0.000     80.000
      5       0.000      0.000     -1.000     -1.000     20.000
      6       0.000      0.000      0.000     -1.000     60.000
```

Figura 6. Tableau gerado pelo LINDO para o problema 2.

Na Figura 5 podemos observa que as variáveis X_{122} e X_{222} são respectivamente 50 e 20, atendendo a demanda do cliente 2 que é de 70 carros, X_{123} atende a demanda do cliente 3 de 60 carros e X_{211} atende a demanda do cliente 1 com a quantidade de 80 carros.

Na Figura 6 analisamos a solução ótima, a base está marcada de verde, com a linha azul estão os coeficientes relativos na função objetivo, a interseção entre um retângulo azul e um retângulo vermelho representa os coeficientes relativos para cada uma das variáveis básicas.

Analisando a Figuras 6, percebemos que não existe variável básica com coeficiente relativo na função objetivo igual à zero, portanto a solução é única.

3. Investimento Financeiro

SUBSIDIÁRIA	PROJETO	TAXA DE RETORNO	LIMITE DO INVESTIMENTO
1	1	8%	US$ 6 milhões
	2	6%	US$ 5 milhões
	3	7%	US$ 9 milhões
2	4	5%	US$ 7 milhões
	5	8%	US$ 10 milhões
	6	9%	US$ 4 milhões
3	7	10%	US$ 6 milhões
	8	6%	US$ 3 milhões

Variáveis de decisão:

$X_{i\,j}$ = Dólares que serão investidos onde:

- "i" representa a subsidiária e varia entre 1 e 3;
- "j" representa o projeto e varia entre 1 e 9.

X_{11} = Dólares que serão investidos no projeto 1 que pertence a subsidiária 1.

X_{12} = Dólares que serão investidos no projeto 2 que pertence a subsidiária 1.

X_{13} = Dólares que serão investidos no projeto 3 que pertence a subsidiária 1.

X_{24} = Dólares que serão investidos no projeto 4 que pertence a subsidiária 2.

X_{25} = Dólares que serão investidos no projeto 5 que pertence a subsidiária 2.

X_{26} = Dólares que serão investidos no projeto 6 que pertence a subsidiária 2.

X_{37} = Dólares que serão investidos no projeto 7 que pertence a subsidiária 3.

X_{38} = Dólares que serão investidos no projeto 8 que pertence a subsidiária 3.

Função objetivo:

Max Z = 0.08 $\mathbf{X_{11}}$ + 0.06 $\mathbf{X_{12}}$ + 0.07 $\mathbf{X_{13}}$ + 0.05 $\mathbf{X_{24}}$ + 0.08 $\mathbf{X_{25}}$ + 0.09 $\mathbf{X_{26}}$ + 0.1 $\mathbf{X_{37}}$ + 0.06 $\mathbf{X_{38}}$

*Visa maximizar a taxa de retorno obtida com os investimentos em cada projeto, onde o coeficiente de cada variável representa a taxa de retorno de cada investimento por subsidiária e projeto.

Sujeito a:

$$
\begin{cases}
X11 + X12 + X13 + X24 + X25 + X26 + X37 + X38 \leq 30000000 \\
X11 + X12 + X13 \geq 3000000 \\
X24 + X25 + X26 \geq 5000000 \\
X37 + X38 \geq 8000000 \\
X24 + X25 + X26 \leq 17000000 \\
X11 \leq 6000000 \\
X12 \leq 5000000 \\
X13 \leq 9000000 \\
X24 \leq 7000000 \\
X25 \leq 10000000 \\
X26 \leq 4000000 \\
X37 \leq 6000000 \\
X38 \leq 3000000 \\
X11, X12, X13, X24, X25, X26, X37, X38 \geq 0
\end{cases}
$$

**Restrições do Projeto:

- O investimento máximo é de US$ 30 milhões;
- O investimento da subsidiária 1 será de no mínimo de US$ 3 milhões;
- O investimento da subsidiária 2 será de no mínimo de US$ 5 milhões;
- O investimento da subsidiária 3 será de no mínimo de US$ 8 milhões;
- O investimento da subsidiária 2 será de no máximo de US$ 17 milhões;
- O projeto 1 terá um máximo de 6 milhões;
- O projeto 2 terá um máximo de 5 milhões;
- O projeto 3 terá um máximo de 9 milhões;
- O projeto 4 terá um máximo de 7 milhões;
- O projeto 5 terá um máximo de 10 milhões;
- O projeto 6 terá um máximo de 4 milhões;
- O projeto 7 terá um máximo de 6 milhões;
- O projeto 8 terá um máximo de 3 milhões;

Resolução:

```
Max 0.08 X11 + 0.06 X12 + 0.07 X13 + 0.05 X24 + 0.08 X25 + 0.09 X26 + 0.1 X37 + 0.06 X38
ST
X11 + X12 + X13 + X24 + X25 + X26 + X37 + X38 <= 30000000
X11 + X12 + X13 >= 3000000
X24 + X25 + X26 >= 5000000
X37 + X38 >= 8000000
X24 + X25 + X26 <= 17000000
X11 <= 6000000
X12 <= 5000000
X13 <= 9000000
X24 <= 7000000
X25 <= 10000000
X26 <= 4000000
X37 <= 6000000
X38 <= 3000000
END
```

Figura 7. Entrada de dados do software LINDO para o problema 3.

```
LP OPTIMUM FOUND AT STEP      7

      OBJECTIVE FUNCTION VALUE

      1)      2500000.

 VARIABLE        VALUE          REDUCED COST
      X11    6000000.000000         0.000000
      X12          0.000000         0.010000
      X13    2000000.000000         0.000000
      X24          0.000000         0.020000
      X25   10000000.000000         0.000000
      X26    4000000.000000         0.000000
      X37    6000000.000000         0.000000
      X38    2000000.000000         0.000000

      ROW   SLACK OR SURPLUS     DUAL PRICES
       2)          0.000000         0.070000
       3)    5000000.000000         0.000000
       4)    9000000.000000         0.000000
       5)          0.000000        -0.010000
       6)    3000000.000000         0.000000
       7)          0.000000         0.010000
       8)    5000000.000000         0.000000
       9)    7000000.000000         0.000000
      10)    7000000.000000         0.000000
      11)          0.000000         0.010000
      12)          0.000000         0.020000
      13)          0.000000         0.040000
      14)    1000000.000000         0.000000

NO. ITERATIONS=       7

RANGES IN WHICH THE BASIS IS UNCHANGED:

                          OBJ COEFFICIENT RANGES
VARIABLE          CURRENT        ALLOWABLE        ALLOWABLE
                   COEF          INCREASE         DECREASE
     X11         0.080000        INFINITY         0.010000
     X12         0.060000        0.010000         INFINITY
     X13         0.070000        0.010000         0.010000
     X24         0.050000        0.020000         INFINITY
     X25         0.080000        INFINITY         0.010000
     X26         0.090000        INFINITY         0.020000
     X37         0.100000        INFINITY         0.040000
     X38         0.060000        0.010000         INFINITY

                          RIGHTHAND SIDE RANGES
     ROW          CURRENT        ALLOWABLE        ALLOWABLE
                    RHS          INCREASE         DECREASE
       2  30000000.000000   7000000.000000   2000000.000000
       3   3000000.000000   5000000.000000         INFINITY
       4   5000000.000000   9000000.000000         INFINITY
       5   8000000.000000   1000000.000000   2000000.000000
       6  17000000.000000         INFINITY   3000000.000000
       7   6000000.000000   2000000.000000   6000000.000000
       8   5000000.000000         INFINITY   5000000.000000
       9   9000000.000000         INFINITY   7000000.000000
      10   7000000.000000         INFINITY   7000000.000000
      11  10000000.000000   2000000.000000   7000000.000000
      12   4000000.000000   2000000.000000   4000000.000000
      13   6000000.000000   2000000.000000   1000000.000000
      14   3000000.000000         INFINITY   1000000.000000
```

Figura 8. Valores de cada uma das variáveis do problema 3.

```
THE TABLEAU

ROW   (BASIS)        X11        X12        X13        X24        X25        X26
  1 ART            0.000      0.010      0.000      0.020      0.000      0.000
  2          X13   0.000      1.000      1.000      1.000      0.000      0.000
  3 SLK        3   0.000      0.000      0.000      1.000      0.000      0.000
  4 SLK        4   0.000      0.000      0.000     -1.000      0.000      0.000
  5 SLK       14   0.000      0.000      0.000      0.000      0.000      0.000
  6 SLK        6   0.000      0.000      0.000      1.000      0.000      0.000
  7          X11   1.000      0.000      0.000      0.000      0.000      0.000
  8 SLK        8   0.000      1.000      0.000      0.000      0.000      0.000
  9 SLK        9   0.000     -1.000      0.000     -1.000      0.000      0.000
 10 SLK       10   0.000      0.000      0.000      1.000      0.000      0.000
 11          X25   0.000      0.000      0.000      0.000      1.000      0.000
 12          X26   0.000      0.000      0.000      0.000      0.000      1.000
 13          X37   0.000      0.000      0.000      0.000      0.000      0.000
 14          X38   0.000      0.000      0.000      0.000      0.000      0.000

ROW          X37        X38  SLK     2  SLK     3  SLK     4  SLK     5  SLK     6
  1        0.000      0.000      0.070      0.000      0.000      0.010      0.000
  2        0.000      0.000      1.000      0.000      0.000      1.000      0.000
  3        0.000      0.000      1.000      1.000      0.000      1.000      0.000
  4        0.000      0.000      0.000      0.000      1.000      0.000      0.000
  5        0.000      0.000      0.000      0.000      0.000      1.000      0.000
  6        0.000      0.000      0.000      0.000      0.000      0.000      1.000
  7        0.000      0.000      0.000      0.000      0.000      0.000      0.000
  8        0.000      0.000      0.000      0.000      0.000      0.000      0.000
  9        0.000      0.000     -1.000      0.000      0.000     -1.000      0.000
 10        0.000      0.000      0.000      0.000      0.000      0.000      0.000
 11        0.000      0.000      0.000      0.000      0.000      0.000      0.000
 12        0.000      0.000      0.000      0.000      0.000      0.000      0.000
 13        1.000      0.000      0.000      0.000      0.000      0.000      0.000
 14        0.000      1.000      0.000      0.000      0.000     -1.000      0.000

ROW   SLK     7  SLK     8  SLK     9  SLK    10  SLK    11  SLK    12  SLK    13
  1       0.010      0.000      0.000      0.000      0.010      0.020      0.040
  2      -1.000      0.000      0.000      0.000     -1.000     -1.000      0.000
  3       0.000      0.000      0.000      0.000     -1.000     -1.000      0.000
  4       0.000      0.000      0.000      0.000      1.000      1.000      0.000
  5       0.000      0.000      0.000      0.000      0.000      0.000      1.000
  6       0.000      0.000      0.000      0.000     -1.000     -1.000      0.000
  7       1.000      0.000      0.000      0.000      0.000      0.000      0.000
  8       0.000      1.000      0.000      0.000      0.000      0.000      0.000
  9       1.000      0.000      1.000      0.000      1.000      1.000      0.000
 10       0.000      0.000      0.000      1.000      0.000      0.000      0.000
 11       0.000      0.000      0.000      0.000      1.000      0.000      0.000
 12       0.000      0.000      0.000      0.000      0.000      1.000      0.000
 13       0.000      0.000      0.000      0.000      0.000      0.000      1.000
 14       0.000      0.000      0.000      0.000      0.000      0.000     -1.000

ROW   SLK    14
  1 0.00E+00    0.25E+07
  2 0.00E+00    0.20E+07
  3 0.00E+00    0.50E+07
  4 0.00E+00    0.90E+07
  5   1.0       0.10E+07
  6 0.00E+00    0.30E+07
  7 0.00E+00    0.60E+07
  8 0.00E+00    0.50E+07
  9 0.00E+00    0.70E+07
 10 0.00E+00    0.70E+07
 11 0.00E+00    0.10E+08
 12 0.00E+00    0.40E+07
 13 0.00E+00    0.60E+07
 14 0.00E+00    0.20E+07
```

Figura 9. Tableau gerado pelo LINDO para o problema 3.

Na Figura 9 analisamos a solução ótima, a base está marcada de verde, com a linha azul estão os coeficientes relativos na função objetivo, a interseção entre um retângulo azul e um retângulo vermelho representa os coeficientes relativos para cada uma das variáveis básicas. Analisando a Figuras 9, percebemos que não existe variável básica com coeficiente relativo na função objetivo igual à zero, portanto a solução é única.

EXPLICAÇÕES MÉTODO SIMPLEX

O método *Simplex* é um algoritmo utilizado para calcular algebricamente Problemas de Programação Linear (PPL) e tenta encontrar uma ou mais soluções ótimas para cada problema apresentado. A solução ótima de um modelo é uma solução viável do sistema, ou seja, um ponto extremo do hiperplano gerado pelas restrições. Para resolver um PPL é necessário conhecer alguma solução inicial do sistema, ou seja, um dos pontos do hiperplano gerado.

Se a solução inicial for ótima o processo é encerrado, caso contrário um dos pontos adjacentes fornece um valor melhor que o inicial. Neste caso, o método faz a mudança de um ponto que otimize o PPL. Esse procedimento é repetido até que seja obtido um ponto extremo que seja a solução ótima. A seguir, será apresentada a explicação do código do método *simplex*, com as variáveis e funções que ajudarão na compreensão código.

O PPL não precisa estar na forma padrão para o programa funcionar. Para inserir o Problema de Programação Linear (PPL), deve-se fornecer o número de restrições do problema e o número de variáveis. Para colocar a matriz na forma padrão, na função **main**, a variável “aux”, guarda a posição da matriz a partir da qual são colocadas as variáveis de folga, o que leva o aumento do número de colunas da matriz. No código, o tamanho da matriz “restricoes” apenas copia os coeficientes de uma para outra, pois *padrao[i][j] = restricoes[i][j].*

Após algumas condições e comandos de repetição no código, a função “Z” objetivo é colocada na forma padrão. O sinal é alterado caso a função objetivo seja de maximização (equivale a multiplicar todo o Z por -1). Como a matriz “restricoes” não é utilizada na forma padrão, foi usada uma outra variável para desalocar o espaço, que é executada pela seguinte atribuição presente no código: *restricoes = desaloca_matriz(l, restrições).*

Em seguida, a matriz “tableaux” guarda o *tableau* (inicial e os outros até chegar na solução ótima). Para que seja possível encontrar o *tableau*, é preciso adicionar mais uma coluna (c) para "Z" e uma linha (l) para a “base”. Feito isso, o programa define a primeira linha do *tableau* e caso seja um problema de minimização, multiplica-se a linha por “-1”, inclusive o “Z”.

Ao atribuir a coluna de “Z”, definimos “1” na primeira linha e “0” nas outras. O “Z” é atribuído como a linha base do *tableau* e as restrições também são acrescentadas.

Na função **primeira_fase**, o parâmetro “W” (que é inicialmente declarado na função **main**), é verificado mediante à variável “isOtimo”, a qual verifica o ótimo do *tableau*, ou seja, caso a variável “isOtimo” seja igual à “1”, o parâmetro “W” não é igual à zero, podendo assim escolher as variáveis que vão entrar (in) e sair (aux), gerando assim uma nova solução.

Se o ótimo (variável “isOtimo”) for igual a 2, a função não tem solução, caso contrário, o “W” é igual a zero e finaliza a primeira fase. No entanto, se “isOtimo” for 0 (zero), inicia-se a segunda fase e o *tableau* é mostrado na tela, assim como o resultado.

Na função **imprime_tableaux**, o vetor guarda os índices dos “x” para serem impressos embaixo do *tableau*. A função **calloc** cria um vetor de tamanho dinâmico. Após imprimir o *tableau*, no código existe um comando de repetição *for* que percorre o vetor e busca se o índice do “x” em questão que está na base e dependendo das condições apresentadas no código, os valores calculados da matriz “tableaux” são impressos, ou valor igual a 0.

Caso o elemento da linha “0” e coluna “0” da matriz “tableaux” seja igual a “-1” e o elemento da linha “0” e coluna “c -1” (numero de colunas menos 1) da mesma matriz seja diferente de “0”, o valor de “Z” maximizado ou “-Z” minimizado será impresso.

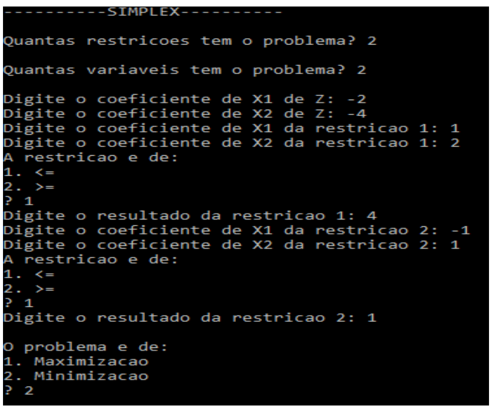
As funções **variable_in** e **variable_out** encontram quem deve entrar e sair da base respectivamente, portanto, depois de impresso o *tableau* inicial, a partir da primeira iteração (as interações são feitas na função **iteração_tableaux**), caso não seja encontrada a solução ótima, uma variável deverá entrar e outra deverá sair, dando continuidade ao procedimento até encontrar o ótimo do problema.

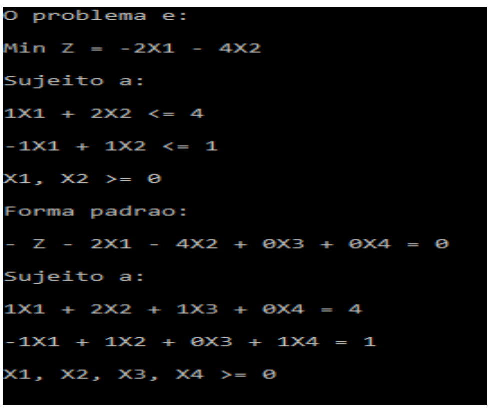
A função responsável por testar as duas funções descritas anteriormente (**variable_in** e **variable_out**) é a **resultado_tableaux**, a qual determina se a solução é degenerada, se o problema possui múltiplas soluções, ou se a função tem solução infinita, ou mesmo se é solução ótima, ou não, pois essa função aplica os cálculos do *tableau*.

Ainda sobre a função **resultado_tableaux**, um comando de repetição *for* é responsável por realizar o cálculo da nova linha pivô e essa nova linha será dividida pela intersecção entre a linha e a coluna pivô. Um outro *for* inicia o percurso pela primeira linha do *tableau* e existe uma condição que determina se prossegue com os cálculos, caso a linha atual não seja a linha pivô, pois ela já foi calculada antes. Dentro desse *for*, existe um que caso a condição não seja a linha pivô, ele percorrerá a linha toda. Uma variável auxiliar presente no for interno guarda a intersecção entre a linha atual com a coluna pivô. Os novos valores de cada linha, são cada valor subtraído da multiplicação de “aux” (variável auxiliar) com o valor correspondente na mesma coluna, na linha pivô.

Nos exemplos que sucedem, foi-se detalhado cada um deles e o programa mostra quando a solução ótima é encontrada, além de dizer qual o tipo de solução, assim como foi apresentado no código.

- Primeiro exemplo





```
Tableaux:

        Z       X1      X2      X3      X4      b

base    -1.0    -2.0    -4.0    0.0     0.0     0.0

X3      0.0     1.0     2.0     1.0     0.0     4.0

X4      0.0     -1.0    1.0     0.0     1.0     1.0

X1 = 0; X2 = 0; X3 = 4.0; X4 = 1.0;

Z = 0.0


A SOLUCAO NAO E OTIMA!!!

ENTRA A VARIAVEL X2
SAI A VARIAVEL X4

NOVA SOLUCAO:


        Z       X1      X2      X3      X4      b

base    -1.0    0.0     0.0     2.0     0.0     8.0

X1      0.0     1.0     0.0     0.3     -0.7    0.7

X2      0.0     0.0     1.0     0.3     0.3     1.7

X1 = 0.7; X2 = 1.7; X3 = 0; X4 = 0;

Z = -8.0


A SOLUCAO E OTIMA!!!
```

```
NOVA SOLUCAO:


        Z       X1      X2      X3      X4      b

base    -1.0    -6.0    0.0     0.0     4.0     4.0

X3      0.0     3.0     0.0     1.0     -2.0    2.0

X2      0.0     -1.0    1.0     0.0     1.0     1.0

X1 = 0; X2 = 1.0; X3 = 2.0; X4 = 0;

Z = -4.0


A SOLUCAO NAO E OTIMA!!!

ENTRA A VARIAVEL X1
SAI A VARIAVEL X3
```

- Segundo exemplo

```
Quantas variaveis tem o problema? 2

Digite o coeficiente de X1 de Z: -5
Digite o coeficiente de X2 de Z: -2
Digite o coeficiente de X1 da restricao 1: 4
Digite o coeficiente de X2 da restricao 1: 3
A restricao e de:
1. <=
2. >=
? 1
Digite o resultado da restricao 1: 12
Digite o coeficiente de X1 da restricao 2: 1
Digite o coeficiente de X2 da restricao 2: 0
A restricao e de:
1. <=
2. >=
? 1
Digite o resultado da restricao 2: 3
Digite o coeficiente de X1 da restricao 3: 0
Digite o coeficiente de X2 da restricao 3: 1
A restricao e de:
1. <=
2. >=
? 1
Digite o resultado da restricao 3: 4

O problema e de:
1. Maximizacao
2. Minimizacao
? 1
```

```
O problema e:

Max Z = -5X1 - 2X2

Sujeito a:

4X1 + 3X2 <= 12

1X1 + 0X2 <= 3

0X1 + 1X2 <= 4

X1, X2 >= 0

Forma padrao:

Z + 5X1 + 2X2 + 0X3 + 0X4 + 0X5 = 0

Sujeito a:

4X1 + 3X2 + 1X3 + 0X4 + 0X5 = 12

1X1 + 0X2 + 0X3 + 1X4 + 0X5 = 3

0X1 + 1X2 + 0X3 + 0X4 + 1X5 = 4

X1, X2, X3, X4, X5 >= 0
```

```
Tableaux:

        Z       X1      X2      X3      X4      X5      b

base    -1.0    -5.0    -2.0    0.0     0.0     0.0     0.0

X3      0.0     4.0     3.0     1.0     0.0     0.0     12.0

X4      0.0     1.0     0.0     0.0     1.0     0.0     3.0

X5      0.0     0.0     1.0     0.0     0.0     1.0     4.0

X1 = 0; X2 = 0; X3 = 12.0; X4 = 3.0; X5 = 4.0;

Z = 0.0


A SOLUCAO NAO E OTIMA!!!

ENTRA A VARIAVEL X1
SAI A VARIAVEL X3
```

```
NOVA SOLUCAO:


        Z       X1      X2      X3      X4      X5      b

base    -1.0    0.0     1.8     1.3     0.0     0.0     15.0

X1      0.0     1.0     0.8     0.3     0.0     0.0     3.0

X4      0.0     0.0     -0.8    -0.3    1.0     0.0     0.0

X5      0.0     0.0     1.0     0.0     0.0     1.0     4.0

X1 = 3.0; X2 = 0; X3 = 0; X4 = 0.0; X5 = 4.0;

Z = -15.0


A SOLUCAO E DEGENERADA!

A SOLUCAO E OTIMA!!!
```

- Terceiro Exemplo

```
----------SIMPLEX----------

Quantas restricoes tem o problema? 2

Quantas variaveis tem o problema? 2

Digite o coeficiente de X1 de Z: -1
Digite o coeficiente de X2 de Z: -3
Digite o coeficiente de X1 da restricao 1: 1
Digite o coeficiente de X2 da restricao 1: -2
A restricao e de:
1. <=
2. >=
? 1
Digite o resultado da restricao 1: 4
Digite o coeficiente de X1 da restricao 2: -1
Digite o coeficiente de X2 da restricao 2: 1
A restricao e de:
1. <=
2. >=
? 1
Digite o resultado da restricao 2: 3

O problema e de:
1. Maximizacao
2. Minimizacao
? 2
```

```
O problema e:
Min Z = -1X1 - 3X2
Sujeito a:
1X1 - 2X2 <= 4
-1X1 + 1X2 <= 3
X1, X2 >= 0
Forma padrao:
- Z - 1X1 - 3X2 + 0X3 + 0X4 = 0
Sujeito a:
1X1 - 2X2 + 1X3 + 0X4 = 4
-1X1 + 1X2 + 0X3 + 1X4 = 3
X1, X2, X3, X4 >= 0
```

```
Tableaux:

        Z       X1      X2      X3      X4      b

base    -1.0    -1.0    -3.0    0.0     0.0     0.0

X3      0.0     1.0     -2.0    1.0     0.0     4.0

X4      0.0     -1.0    1.0     0.0     1.0     3.0

X1 = 0; X2 = 0; X3 = 4.0; X4 = 3.0;

Z = 0.0


A SOLUCAO NAO E OTIMA!!!

ENTRA A VARIAVEL X2
SAI A VARIAVEL X4
```

```
NOVA SOLUCAO:


        Z       X1      X2      X3      X4      b

base    -1.0    -4.0    0.0     0.0     3.0     9.0

X3      0.0     -1.0    0.0     1.0     2.0     10.0

X2      0.0     -1.0    1.0     0.0     1.0     3.0

X1 = 0; X2 = 3.0; X3 = 10.0; X4 = 0;

Z = -9.0


A FUNCAO TEM SOLUCAO INFINITA!!!
```

CÓDIGO FONTE

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#include<math.h>
#include<windows.h>

double modulo(double n);
void imprime_tableaux(int l, int c, double tableaux[][100], int * tipo_restricao);
void imprime_problema(int op, int l, int c, double * Z, double restricoes[][100], int *
tipo_restricao);
void imprime_na_forma_padrao(int l, int c, double * Z, double padrao[][100], int * tipo_restricao,
int q_var, int op);
int menu();
void salvar_valores(double * Z, double restricoes[][100], int l, int c, int * tipo_restricao);
int verificar_otimo(double tableaux[][100], int c, int l, int cPivo);
int variable_in(double tableaux[][100], int c);
int variable_out(double tableaux[][100], int ind, int l, int c);
void resultado_tableaux(double tableaux[][100], int l, int c, int * tipo_restricao);
void iteracao_tableaux(double tableaux[][100], int l, int c, int lPivo, int cPivo);
void primeira_fase(double tableaux[][100], double * W, int l, int c, int * tipo_restricao);
int win(double * W, int c);
int verificar_w(double * W, int c);
void iteracao_w(double tableaux[][100], double * W, int l, int c, int lPivo, int cPivo);
void imprime_w(int l, int c, double tableaux[][100], double * W);

double modulo(double n) { //Funcao que retorna o modulo de um double
    if(n >= 0){ return n; }else{ return (n * (-1)); }
}

void imprime_w(int l, int c, double tableaux[][100], double * W) {
    int i, j, isBasic, k, * x, cont = 0;
    x = (int * )calloc(l - 1, sizeof(int));
    printf("\n\n\tW\t");
    for(j = 0; j < c - 2; j++){
        printf("X%d\t", j + 1);
            }
    printf("b\n\nbase\t");
    for(j = 0; j < c; j++){
        printf("%.1lf\t", W[j]);
            }
    printf("\n\n");
    for(i = 1; i < l; i++) { //1.1 - imprime as outras linhas
        for(j = 1; j < c; j++){
            if(tableaux[i][j] == 1) {
                isBasic = 0;
                for(k = 1; k < l; k++){
                    if(tableaux[k][j] != 0){ isBasic = isBasic + 1; }
                                    }
                if(isBasic == 1) {
                    x[cont++] = j;
                    printf("X%d\t", j);
                    break;
                }
            }
                    }
        for(j = 0; j < c; j++){ printf("%.1lf\t", tableaux[i][j]); }
        printf("\n\n");
    }
    getch();
```

```c
}

void iteracao_w(double tableaux[][100], double * W, int l, int c, int lPivo, int cPivo) {
    int i, j;
    double aux;
    aux = tableaux[lPivo][cPivo];
    for(i = 0; i < c; i++){ tableaux[lPivo][i] = (tableaux[lPivo][i] / aux); }
    for(i = 0; i < l; i++){
        if(i != lPivo){
            for(j = 0; j < c; j++) {
                if(j == 0){ aux = tableaux[i][cPivo]; }
                tableaux[i][j] = tableaux[i][j] - (aux * tableaux[lPivo][j]);
            }
                        }
            }
    for(j = 0; j < c; j++) {
        if(j == 0){ aux = W[cPivo]; }
        W[j] = W[j] - (aux * tableaux[lPivo][j]);
    }
}

int verificar_w(double * W, int c) {
    int i, aux = 0;
    for(i = 1; i < c - 1; i++) {
        if(W[i] < 0) {
            aux = 1;
            break;
        }
    }
    if(aux == 0 && W[c - 1] != 0){ aux = 2; }
    return aux;
}

int win(double * W, int c) {
    int i, indice = 0;
    double menor = 0;
    for(i = 1; i < c - 1; i++) {
        if(W[i] < menor) {
            menor = W[i];
            indice = i;
        }
    }
    return indice;
}

void primeira_fase(double tableaux[][100], double * W, int l, int c, int * tipo_restricao) {
    int isOtimo, in, out, i, aux, j, cPivo;
    cPivo = win(W, c);
    isOtimo = verificar_w(W, c);
    if(isOtimo == 1) {
        printf("\nW NAO E IGUAL A ZERO!!!\n");
        in = win(W, c);
        printf("\nENTRA A VARIAVEL X%d\n", in);
        out = variable_out(tableaux, in, l, c);
        aux = 0;
        for(j = 1; j < c - 1; j++) {
            if(tableaux[out][j] == 1) {
                for(i = 1; i < l; i++) { if(tableaux[i][j] != 0) { aux++; } }
            }
            if(aux == 1) {
```

```
            aux = j;
            break;
        } else{ aux = 0; }
    }
    printf("SAI A VARIAVEL X%d\n\n\nNOVA SOLUCAO:\n", aux);
    getch();
    iteracao_w(tableaux, W, l, c, out, in);
    imprime_w(l, c, tableaux, W);
    primeira_fase(tableaux, W, l, c, tipo_restricao);
  } else {
    if(isOtimo == 2){ printf("\nA FUNCAO NAO TEM SOLUCAO!!!\n");
              }else{ printf("\nW E IGUAL A ZER0!!!\n\nFIM DA PRIMEIRA FASE\n"); }
    getch();
    if(isOtimo == 0) {
      printf("\n\nINICIO DA SEGUNDA FASE:\n");
      imprime_tableaux(l, c, tableaux, tipo_restricao);
      resultado_tableaux(tableaux, l, c, tipo_restricao);
    }
  }
}

void imprime_tableaux(int l, int c, double tableaux[][100], int * tipo_restricao) { //1.1 - função que
imprime o tableaux
  int i, j, isBasic, k, * x, cont = 0, tam, k2;
  x = (int * )calloc(l - 1, sizeof(int)); //2.0 - vetor que guarda os índices dos x pra serem
impressos embaixo do tableaux
  printf("\n\n\tZ\t");
  tam = c - 2;
  for(i = 0; i < l - 1; i++) { tam = tam - tipo_restricao[i];}
  for(j = 0; j < tam; j++){ printf("X%d\t", j + 1); }
  k = 0;
  for(j = tam; j < c - 2; j++) {
    printf("X%d\t", j + 1);
    if(tipo_restricao[k++] == 2) { j++; }
  }
  printf("b\n\nbase\t");
  k = 0;
  for(j = 0; j < c; j++) {
    printf("%.1lf\t", tableaux[0][j]);   //1.1 - imprime a primeira linha do tableaux
    if(j >= tam + 1 && j < c - 1){
      if(tipo_restricao[k++] == 2) { j++; }
              }
  }
  printf("\n\n");
  for(i = 1; i < l; i++) {//1.1 - imprime as outras linhas
    for(j = 1; j < c; j++) {
      if(tableaux[i][j] == 1) { //1.1 - procura por 1 na linha
        isBasic = 0;
        for(k = 1; k < l; k++) //1.1 - se encontrar percorre toda a coluna
          if(tableaux[k][j] != 0) { isBasic = isBasic + 1; }
        if(isBasic == 1) { //1.1 - se o 1 for o único elemento diferente de 0 na coluna, a
variável é básica
          x[cont++] = j;  //2.0 - Nesse ponto x guarda os indices dos X que estao na base
          printf("X%d\t", j); //1.1 - imprima ela
          break;
        }
      }
    }
    k2 = 0;
    for(j = 0; j < c; j++) {
```

```
            printf("%.1lf\t", tableaux[i][j]);   //1.1 - imprime os elementos de cada linha do tableaux
            if(j >= tam + 1 && j < c - 1){
                if(tipo_restricao[k2++] == 2) { j++; }
                                }
        }
        printf("\n\n");
    }
    k2 = 0;
    for(i = 0; i < c - 2; i++) { //2.0 - c-2 pois a quantidade de X é a quantidade de colunas menos a
linha Z e a linha b
        k = 0;  //2.0 - k indica se o x esta ou não na base
        printf("X%d = ", i + 1);
        for(j = 0; j < cont; j++)//2.0 - percorre o vetor e busca se o índice do x em questao esta na
base
            if(i + 1 == x[j]) { //2.0 - se sim imprime o valor dele
                printf("%.1lf; ", tableaux[j + 1][c - 1]);
                k = 1;
                break;
            }
        if(k == 0){ printf("0; "); }
        if(i >= tam) { if(tipo_restricao[k2++] == 2) { i++; } }
    }
    if(tableaux[0][0] == -1 && tableaux[0][c - 1] != 0) //2.0 - Imprime o valor de Z (Max) ou -Z (Min)
        printf("\n\nZ = %.1lf\n\n", tableaux[0][c - 1] * (-1));
    else { printf("\n\nZ = %.1lf\n\n", tableaux[0][c - 1]); }
    getch();
}

//1.1 - função que imprime o problema
void imprime_problema(int op, int l, int c, double * Z, double restricoes[][100], int *
tipo_restricao) {
    int i, j, cont = 0;
    printf("O problema e:\n"); //Imprime o problema
    //Dependendo da opção op imprime Max ou Min
    if(op == 1) { printf("\nMax Z = "); }
    if(op == 2) { printf("\nMin Z = "); }
    for(j = 0; j < c - 1; j++) { //Imprime Z
        if(j == 0)  //Se for o primeiro elemento apenas imprime normalmente
            printf("%.0lfX%d ", Z[j], j + 1);
        else
            printf("%.0lfX%d ", modulo(Z[j]), j + 1); //Se for outro elemento imprime apenas o módulo
        if(j != c - 2) { //Aqui imprime o sinal para que fique alinhado
            if(Z[j + 1] >= 0) { printf("+ "); }
            else { printf("- "); }
        }
    }
    //Imprime as restrições
    printf("\n\nSujeito a:\n\n");
    for(i = 0; i < l; i++) {
        for(j = 0; j < c - 1; j++) { //Vai até o utimo X
            if(j == 0) { printf("%.0lfX%d ", restricoes[i][j], j + 1); }
            else { printf("%.0lfX%d ", modulo(restricoes[i][j]), j + 1); }
            if(j != c - 2) {
                if(restricoes[i][j + 1] >= 0) { printf("+ "); }
                                else {  printf("- "); }
            } else { //Se for o ultimo X, imprime a igualdade e o numero depois dela
                if(tipo_restricao[cont] == 1) { printf("<= "); }
                else { printf(">= "); }
                cont++;
                printf("%.0lf ", restricoes[i][c - 1]);
```

```
            }
        }
        printf("\n\n");
    }
    //Imprime as restrições de >=0
    for(j = 0; j < c - 1; j++) {
        printf("X%d", j + 1);
        if(j != c - 2) { printf(", "); }
    }
    printf(" >= 0");
    getch();
}

//1.1 - função que imprime o problema na forma padrão
void imprime_na_forma_padrao(int l, int c, double * Z, double padrao[][100], int * tipo_restricao,
int q_var, int op) {
    int i, j, cont = 0;
    printf("\n\nForma padrao:\n"); //Imprimindo na forma padrão
    if(op == 1){ printf("\nZ "); }
    else { printf("\n- Z "); }
    for(j = 0; j < q_var; j++) {
        if(Z[j] >= 0){ printf("+ "); }
        else { printf("- "); }
        printf("%.0lfX%d ", modulo(Z[j]), j + 1);
    }
    for(i = q_var + 1; cont < l; i++) {
        if(tipo_restricao[cont] == 1) { printf("+ 0X%d ", i); }
        else {
            printf("+ 0X%d ", i);
            i++;
            printf("+ 0X%d ", i);
        }
        cont++;
    }
    printf("= 0\n\nSujeito a:\n\n");
    for(i = 0; i < l; i++) {
        for(j = 0; j < c - 1; j++) {
            if(j == 0) { printf("%.0lfX%d ", padrao[i][j], j + 1); }
            else { printf("%.0lfX%d ", modulo(padrao[i][j]), j + 1); }

            if(j != c - 2) {
                if(padrao[i][j + 1] >= 0) { printf("+ "); }
                else { printf("- "); }
            } else { printf("= %.0lf ", padrao[i][c - 1]); }
        }
        printf("\n\n");
    }
    for(j = 0; j < c - 1; j++) {
        printf("X%d", j + 1);
        if(j != c - 2){ printf(", "); }
    }
    printf(" >= 0");
    getch();
}

int menu() {//1.2 - função p/ selecionar a opção
    int op;
    do {
        printf("\nO problema e de:\n1. Maximizacao\n2. Minimizacao\n? ");
        scanf("%d", & op);
```

```
        if(op != 1 && op != 2) { printf("Opcao invalida!\n"); }
    } while(op != 1 && op != 2);
    system("cls");
    return op;
}

//1.2 - função p/ ler valores de Z e da matriz de restrições - A
void salvar_valores(double * Z, double restricoes[][100], int l, int c, int * tipo_restricao) {
    int i, j, cont = 0;
    for(i = 0; i < c - 1; i++) {
        printf("Digite o coeficiente de X%d de Z: ", i + 1);
        scanf("%lf", & Z[i]);
    }
    for(i = 0; i < l; i++)
        for(j = 0; j < c; j++) {
            if(j == c - 1) { //Verifica se vai ler um coeficiente de X ou a igualdade da restrição
                do {
                    printf("A restricao e de:\n1. <=\n2. >=\n? ");
                    scanf("%d", & tipo_restricao[cont]);
                    if(tipo_restricao[cont] != 1 && tipo_restricao[cont] != 2) { printf("Opcao invalida!\n");
}
                } while(tipo_restricao[cont] != 1 && tipo_restricao[cont] != 2);
                cont++;
                printf("Digite o resultado da restricao %d: ", i + 1);
            } else { printf("Digite o coeficiente de X%d da restricao %d: ", j + 1, i + 1); }
            scanf("%lf", & restricoes[i][j]);
        }
}

//1.2 Verifica se a função é ótima
int verificar_otimo(double tableaux[][100], int c, int l, int cPivo) { //1.6 - O cabeçalho da função
foi alterado, pois há a necessidade de receber o numero de linhas do tableaux e a coluna pivô.
    int i, aux = 0;
    if(cPivo != 0) { //1.8.1 - Precisei fazer isso pra corrigir um bug, atribui 0 como valor inicial do
indice de variable_in, pois no ultimo tableaux, como não tem variaveis pra entrar a função vai
jogar um lixo em cPivo, consequentemente quando tu tentar ler ela aqui o programa vai parar
de funcionar, pois não existe a coluna "lixo"
        for(i = 0; i < l; i++) //1.6 - para percorrer toda a coluna pivô
            if(tableaux[i][cPivo] > 0) { aux++; } //1.6 - Se pelo algum elemento da coluna pivô for
positivo, aux é incrementado, indicando que a solução não é infinita
        if(aux == 0) //1.6 - Se nesse ponto do programa aux for igual 0, indica que na coluna pivô
só há valores negativos
            return 2; //1.6 - Retorna 2 indicando que a solução é infinita
        aux = 0;
    }
    for(i = 1; i < c - 1; i++) //1.2 - percorre somente as culunas das variáveis Xi.
        if(tableaux[0][i] < 0) {
            aux = 1; //1.2 - aux recebe 1, indicando que ainda há variáveis negativas, portanto a
solução não é otima.
            break;
        }
    return aux; //1.2 - se aux aqui for 0, indica que a solução é ótima
}

//1.2 - função que encontra quem deve entrar na base.
int variable_in(double tableaux[][100], int c) {
    int i, indice = 0; //1.2 - indice vai indicar a coluna do tlabeux que esta a variável que deve
entrar na base.
    double menor = 0.0; //1.2 - variavel auxiliar pra ir comparando os valores das variáveis Xi
```

```
    for(i = 1; i < c - 1; i++) //1.5 - i inicia com 1, pq não precisa comparar a coluna do Z e vai até
c-1, pq não compara a umtima coluna de resultados.
        if(tableaux[0][i] < menor) {
            menor = tableaux[0][i];
            indice = i; //1.2 - Se por exemplo, variável que for entrar na base for X1, o indice vai
guardar 1
        }
    return indice;
}

//1.2 - função que encontra quem deve sair da base.          sai_da_base
int variable_out(double tableaux[][100], int ind, int l, int c) {
    int i, indice; // 1.2- indice guarda a linha que esta a variável que vai sair.
    double menor = 100000.0, result; //1.2 - result guarda o resultado da divisão dos elementos
de B pelos valores da coluna pivô
    for(i = 1; i < l; i++) //1.2 - começa a percorrer depois da linha do Z
        if(tableaux[i][ind] > 0) { //1.2 - só irá dividir se o denominador for maior que 0
            result = tableaux[i][c - 1] / tableaux[i][ind]; //1.2 - divisão da última coluna do tableaux
pelos elementos da linha pivô.
            if(result < menor) { // 1.2 - vai comparando se o resultado da divisão é menor que o
valor da divisão anterior
                menor = result; //1.2 - se for menor, armazena este resultado como o novo menor
                indice = i;
            }
        }
    return indice;
}

//1.5 - função para testar se as funções variable_in e variable_out estao funcionando
void resultado_tableaux(double tableaux[][100], int l, int c, int * tipo_restricao) {
    int isOtimo, in, out, i, aux, j, cPivo;
    cPivo = variable_in(tableaux, c); //1.6 - A função variable_in só é chamada pq a função
abaixo(verificar_otimo) precisa saber quem é a coluna pivô
    isOtimo = verificar_otimo(tableaux, c, l, cPivo);
    aux = 0; //1.8 - Verifica se a solução é multipla
    for(i = 1; i < c - 1; i++){ //1.8 - Percorre a linha da base guardando a quantidade de zeros em
aux
        if(tableaux[0][i] == 0){ aux++; }
            }
    for(i = 0; i < l - 1; i++) { if(tipo_restricao[i] == 2) {aux--;}}
    if(aux > l - 1) //1.8 - Se houverem mais zeros do que variaveis na base a solução é multipla
        printf("\nO PROBLEMA POSSUI MULTIPLAS SOLUCOES!\n");
    //1.7 - Verifica se a solução é degenerada
    aux = 0;
    for(i = 1; i < l; i++)//1.7 - A linha da base não conta, por isso começa do 1
        if(tableaux[i][c - 1] <= 1e-10 && tableaux[i][c - 1] >= -1e-10) //1.7 - Se houver algum 0 na
ultima coluna (b) incrementa aux
            aux++;
    if(aux > 0) //1.7 - Se aux for maior que 0 significa que alguma variavel básica é 0 (solução
degenerada)
        printf("\nA SOLUCA0 E DEGENERADA!\n");
    if(isOtimo == 1) {
        printf("\nA SOLUCAO NAO E OTIMA!!!\n");
        in = variable_in(tableaux, c);
        printf("\nENTRA A VARIAVEL X%d\n", in);
        out = variable_out(tableaux, in, l, c);
        aux = 0; //2.0 - baseado no índice de out, encontra qual X vai sair da base (quase o msm
algoritmo que esta em imprimir_tableaux)
        for(j = 1; j < c - 1; j++) {
            if(tableaux[out][j] == 1){ for(i = 1; i < l; i++) { if(tableaux[i][j] != 0){ aux++; } } }
```

```
            if(aux == 1) {
                aux = j;
                break;
            } else { aux = 0; }
        }
        printf("SAI A VARIAVEL X%d\n\n\nNOVA SOLUCAO:\n", aux);
        getch();
        iteracao_tableaux(tableaux, l, c, out, in);
        imprime_tableaux(l, c, tableaux, tipo_restricao);
        resultado_tableaux(tableaux, l, c, tipo_restricao); //1.5 - como não é otimo chama a função
recursivamente.
    } else {
        if(isOtimo == 2){ printf("\nA FUNCAO TEM SOLUCAO INFINITA!!!\n"); }
        else{ printf("\nA SOLUCAO E OTIMA!!!\n"); }
        getch();
    }
}

//1.5 - função que aplica os cálculos no tableuax
void iteracao_tableaux(double tableaux[][100], int l, int c, int lPivo, int cPivo) {
    int i, j;
    double aux;
    aux = tableaux[lPivo][cPivo];
    for(i = 0; i < c; i++) { tableaux[lPivo][i] = tableaux[lPivo][i] / aux; }//A nova linha será ela
mesma dividida pela intersecção entre a linha e a coluna pivôs
    for(i = 0; i < l; i++) { //1.5 - inicia o percurso pela primeira linha do tableaux
        if(i != lPivo){ //1.5 - só prossigo com os cálculos se a linha atual não for a linha pivô, pois
ela já foi calculada antes
            for(j = 0; j < c; j++) { //1.5 - caso não seja a LP, percorre toda a linha
                if(j == 0) { aux = tableaux[i][cPivo]; }//1.5 - aux guarda a intersecção entre a linha
atual com a coluna pivô
                tableaux[i][j] = tableaux[i][j] - (aux * tableaux[lPivo][j]); // 1.5 - Os novos valores de
cada linha, é cada valor subtraido da multiplicação de aux com o valor correspondente na
mesma coluna, só que na linha pivô
            }
                        }
    }
}

// Main
int main() {
    int op, i, j, l, c, aux, tipo_restricao[100], q_var, cont = 0, tamZ;
    double restricoes[100][100], padrao[100][100], Z[100], W[100];
            system("cls");
    printf("----------SIMPLEX----------\n");
    do {
        printf("\nQuantas restricoes tem o problema? ");  //A quantidade de restrições representa a
quantidade de linhas da matriz
        scanf("%d", & l);
        if(l <= 0) printf("Quantidade invalida!\n");
    } while(l <= 0);
    do {
        printf("\nQuantas variaveis tem o problema? "); //A quantidade de variáveis + 1 representa
a quantidade de colunas da matriz
        scanf("%d", & c);
        if(c <= 0) printf("Quantidade invalida!\n");
    } while(c <= 0);
    tamZ = c;
    c = c + 1;
    printf("\n");
```

```
    salvar_valores(Z, restricoes, l, c, tipo_restricao); // 1.2 - função criada separadamente pra ler as entradas de dados
    op = menu(); //1.2 - função menu criada só pra diminuir um pouco o código na main
    imprime_problema(op, l, c, Z, restricoes, tipo_restricao);
    aux = 0;
    q_var = c - 1;  //Colocando a matriz na forma padrão
    for(i = 0; i < l; i++){ c = c + tipo_restricao[i]; }
    for(i = 0; i < l; i++){
        for(j = 0; j < c - 1; j++){
            if(j < q_var){ //Ate o tamanho da matriz restricoes apenas copia os coeficientes de uma pra outra
                padrao[i][j] = restricoes[i][j];
                            } else {
                if(j == q_var){  //Copia a igualdade da matriz restrições para o fim da matriz padrao
                    padrao[i][c - 1] = restricoes[i][j];
                                    }
                padrao[i][j] = 0;
            }
                    }
          }
    for(j = 0; j < c - 1; j++){
        if(j >= q_var) {
            if(tipo_restricao[cont] == 1){  padrao[aux][j] = 1;
                            } else {
                if(padrao[aux][j - 1] == -1){  padrao[aux][j] = 1;
                                    } else {
                    padrao[aux][j] = -1;
                    cont--;
                    aux--;
                }
            }
            cont++;
            aux = aux + 1;
        }
          }
    //1.1 - Colocando Z na forma padrão
    if(op == 1){ for(j = 0; j < q_var; j++){ Z[j] = Z[j] * (-1); } }//1.3 - só muda o sinal se for de maximização (equivale a multiplicar todo o Z por -1)
    Z[j] = 0;
    imprime_na_forma_padrao(l, c, Z, padrao, tipo_restricao, q_var, op);
    double tableaux[100][100]; //1.1 - tableaux é a matriz que vai guardar o tableaux  --
    c = c + 1; //1.1 - precisa de mais uma coluna para o Z
    l = l + 1; //1.1 - precisa de mais uma linha para a base
    if(op == 1){ tableaux[0][0] = 1; //1.1. - Definindo a primeira coluna do tableaux
          }else{ tableaux[0][0] = -1; } //1.3 - caso seja de minimização multiplica-se a linha base por -1, inclusive o Z
    for(i = 1; i < l; i++){ tableaux[i][0] = 0; } //Atribuição da coluna do Z que tem 1 na primeira linha e 0 nas outras
    for(j = 1; j < tamZ + 1; j++){ tableaux[0][j] = Z[j - 1]; } //Atribuindo o vetor Z como a linha base no tableaux
    for(j = tamZ + 1; j < c; j++){ tableaux[0][j] = 0; }
    for(i = 1; i < l; i++){ //1.1 - Colocando as restrições no tableaux
        for(j = 1; j < c; j++){ tableaux[i][j] = padrao[i - 1][j - 1]; }
          }
    aux = 0;
    for(i = 0; i < l - 1; i++){
        if(tipo_restricao[i] == 2) {
            aux = 1;
            break;
        }
```

```
        }
printf("\n\nTableaux:");
if(aux == 1) {
    W[0] = -1;
    for(i = 0; i < l - 1; i++){
        if(tipo_restricao[i] == 2) {
            for(j = 0; j < c - 1; j++){ W[j + 1] = W[j + 1] + (padrao[i][j] * (-1)); }
            for(j = c - 1; j != 0; j--) {
                if(W[j] == -1) {
                    W[j] = 0;
                    break;
                }
            }
        }
                    }
    imprime_w(l, c, tableaux, W);
    primeira_fase(tableaux, W, l, c, tipo_restricao);
} else {
    imprime_tableaux(l, c, tableaux, tipo_restricao);
    resultado_tableaux(tableaux, l, c, tipo_restricao);
}
return 0; }
```