

Maximizando com Simpléx

Mário Leite

...

Em muitas situações em que envolva tomar decisões baseadas em otimização de processos, é fundamental empregar métodos que nos dão alguma sugestão que nos leve à resultados mais verdadeiros e condizentes com a situação exposta. O “Método Simpléx” é o mais utilizados nesses casos, baseado em cálculos complexos, que nos poupa de cometer erros críticos ao tentar resolver o problema manualmente. Esse método (conhecido apenas como “Simpléx”) é um dos algoritmos mais eficazes para resolver problemas de programação linear e é amplamente utilizado em diversas áreas como: economia, engenharia e logística. O seu emprego e funcionamento se baseiam em seis etapas, fundamentais, para que esse algoritmo funcione perfeitamente:

1) **Formulação do problema:** É a base de todo o processo, pois é aqui que o usuário descreve o que deve ser feito, onde é descrito o problema da otimização (*maximização* ou *minimização*). O problema da otimização é formulado na forma padrão que, envolvendo a definição da função objetivo sujeita a restrições lineares que devem ser expressas em termos de variáveis de decisão.

2) **Tabela Simplex:** O “Método Simplex” opera utilizando uma tabela, também conhecida como “tabela simplex”, que contém as informações sobre as variáveis de decisão, as restrições e a função objetivo.

3) **Escolha da variável básica e não básica:** Inicialmente, o método seleciona uma *variável básica* e uma *variável não básica* que assume um valor positivo na solução ótima; enquanto a variável não básica é uma variável que assume um valor zero.

4) **Iteração:** O algoritmo itera, através de diferentes soluções candidatas (soluções possíveis do problema), movendo-se de uma solução viável para outra com um valor de função objetivo maior até que a solução ótima seja alcançada. Isto é feito através de operações chamadas de *pivoteamento*, onde as *variáveis básicas* e *não básicas* são trocadas para melhorar a solução, através de muitas iterações em *loop*.

5) **Critério de parada:** Como o processo ocorre num *loop*, o algoritmo continua iterando até que uma solução ótima seja encontrada ou até que seja determinado que não há solução ótima possível; por isto tem que haver uma parada caso a função objetivo seja ilimitada.

6) **Solução ótima:** Assim que o algoritmo convergir para uma solução ótima as variáveis de decisão são fixadas em seus valores correspondentes na solução ótima, fornecendo assim a solução final para o problema.

O programa “**ProgMaximizacao**”, codificado em **Visualg** (que pode ser convertido para qual outra linguagem real) é uma solução simples para um problema de maximização, assim descrito:

Contexto:

Considere uma empresa que produz dois tipos de ração para cães: Tipo A e Tipo B. Para produzir essas rações são utilizados cereais e carne sabe-se que:

- Ração Tipo A: 5kg de cereais e 1 Kg de carne.
- Ração Tipo B: 2Kg de cereais e 4Kg de carne.

Suponha que o Kg de carne custa \$4.00 e o de cereais \$1.00; o pacote de ração Tipo A custa \$20.00 e o de Tipo B \$30.00. A disponibilidade mensal de suprimentos é de 10000 kg de carne e 30000 kg de cereais.

Análise do problema:

Custo da carne:

- $1 \times 4 = \$4.00$ (para a ração Tipo A)
- $4 \times 4 = \$16.00$ (para a ração Tipo B)

Custo do cereal:

- $5 \times 1 = \$5.00$ (para a ração Tipo A)
- $2 \times 1 = \$2.00$ (para a ração Tipo B)

Custo total:

- Carne = \$9.00
- Cereal = \$18.00

Preços:

Carne = \$4.00

Cereal = \$1.00

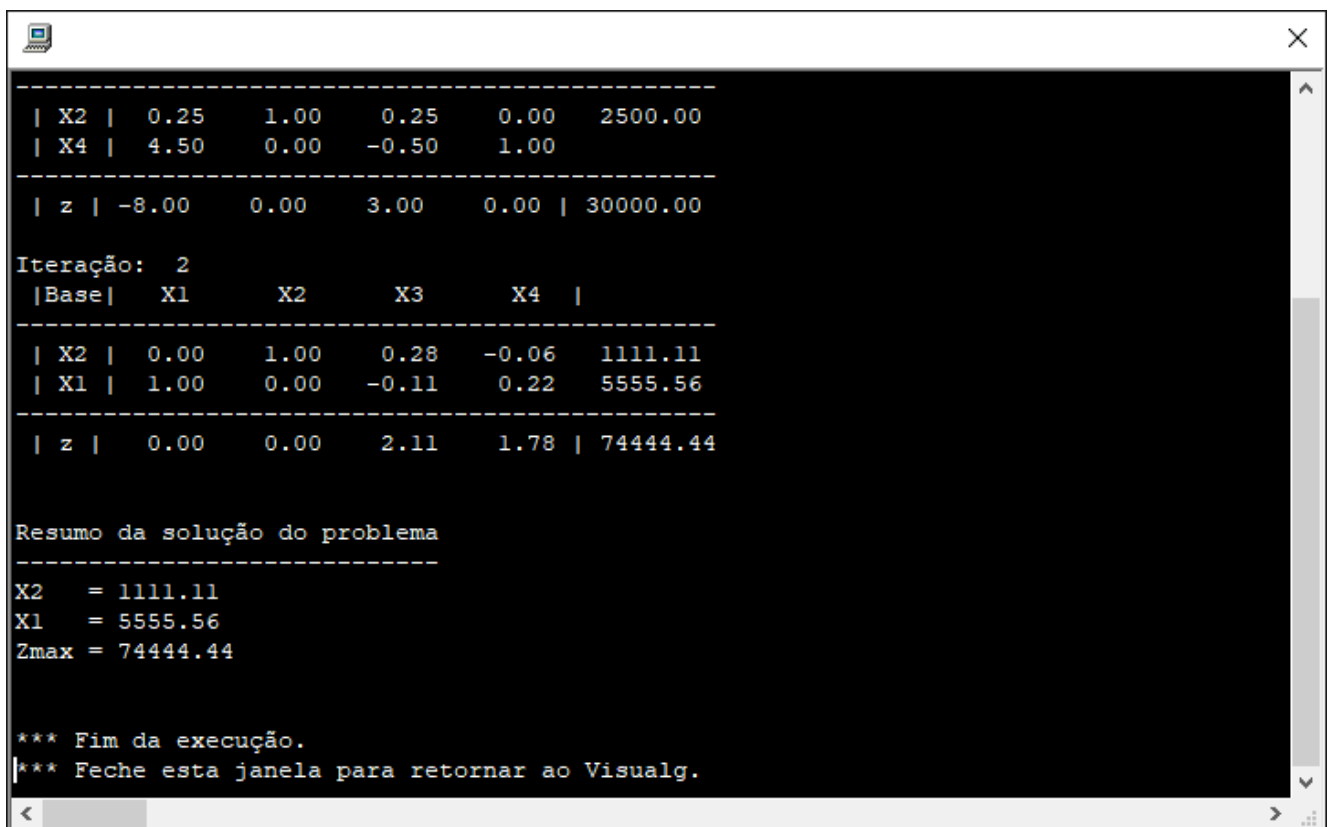
Lucro:

Carne = $(\$20.00 - \$9.00) = \$11.00$

Cereal = $(\$30.00 - \$18.00) = \$12.00$

Questão a ser resolvida:

Quais deverão ser as quantidades de cada ração devem ser produzidas para maximizar o lucro?



```
-----
| X2 | 0.25  1.00  0.25  0.00  2500.00
| X4 | 4.50  0.00 -0.50  1.00
-----
| z | -8.00  0.00  3.00  0.00 | 30000.00

Iteração: 2
|Base|  X1    X2    X3    X4 |
-----
| X2 | 0.00  1.00  0.28 -0.06 | 1111.11
| X1 | 1.00  0.00 -0.11  0.22 | 5555.56
-----
| z | 0.00  0.00  2.11  1.78 | 74444.44

Resumo da solução do problema
-----
X2  = 1111.11
X1  = 5555.56
Zmax = 74444.44

*** Fim da execução.
*** Feche esta janela para retornar ao Visualg.
```

Saída do programa “ProgMaximizacao”

Algoritmo "ProgMaximizacao"

//Faz a maximização de um Problema de Programação Linear com o Método Simpléx,
//para dados já preestabelecidos num exemplo prático.

//Autor : Mário Leite

//E-mail : marleite@gmail.com

//-----

//Variáveis globais

Var MatA, MatB, MatZ: vetor[1..5,1..5] de real *//define dimensões das matrizes*

MatQuadro, MatQuadro1: vetor[1..5,1..5] de real

VetDiv: vetor[1..5] de real

m, n, nl, nc: inteiro

VetBase, VetV, VetX, VetVarBase: vetor[1..5] de caractere

TemSolucao: logico

//-----

Procedimento ProCriaMatrizes (m,n:inteiro)

//Cria as matizes do sistema

Var i, j, k, op: inteiro

Inicio

{Matriz dos coeficientes: custos das rações e cereais (incluindo as variáveis de folga)}

MatA[1,1] <- 1

MatA[1,2] <- 4

MatA[1,3] <- 1

MatA[1,4] <- 0

MatA[2,1] <- 5

MatA[2,2] <- 2

MatA[2,3] <- 0

MatA[2,4] <- 1

{Matriz dos termos independentes: disponibilidades}

{Vetor-coluna dos termos independentes: m+1 elementos}

MatB[1,1] <- 10000

MatB[2,1] <- 30000

MatB[3,1] <- 0

{Matriz da função objetivo z: sujeita à maximização}

MatZ[1,1] <- 11

MatZ[1,2] <- 12

MatZ[1,3] <- 0

MatZ[1,4] <- 0

op <- -1 *//define o tipo de otimização [Max: -1]*

Para j **De** 1 **Ate** 4 **Faca**

MatZ[1,j] <- op*MatZ[1,j]

FimPara

{Define a Base: inicialmente com as variáveis de folga}

Para k **De** 1 **Ate** m **Faca**

VetBase[k] <- "X" + NumpCarac(n+k) *//títulos das variáveis de folga*

FimPara

VetBase[m+1] <- " z " *//título da função objetivo*

{Vetor dos títulos das variáveis (reais e de folga: linha 0)}

Para j **De** 1 **Ate** m **Faca**

VetV[j] <- "X" + NumpCarac(j)

FimPara

```

{Copia a Matriz A no Quadrol}
  Para i De 1 Ate m Faca
    Para j De 1 Ate (m+n) Faca
      MatQuadrol[i,j] <- MatA[i,j]
    FimPara
  FimPara
FimProcedimento //fim do procecimento "ProCriaMatrizes"

//-----
Procedimento ProInsereLinhasColunas (m,n:inteiro)
//Insere linhas e colunas para compor o Quadro Simpl x.
  Var i, j: inteiro
Inicio
{Insere a linha da Fun  o Objetivo (z) no Quadrol}
  Para i De (m+1) Ate (m+1) Faca
    Para j De 1 Ate (m+n) Faca
      MatQuadrol[i,j] <- MatZ[1,j]
    FimPara
  FimPara

  {Insere a coluna das Disponibilidades (vetor b) no Quadrol}
  Para i De 1 Ate (m+1) Faca
    Para j De 1 Ate (m+n+1) Faca
      Se(j=(n+m+1)) Entao
        MatQuadrol[i,5] <- MatB[i,1]
      FimSe
    FimPara
  FimPara
FimProcedimento //fim do procedimento "ProInsereLinhasColunas"

//-----
Procedimento ProResolveSistema (m,n:inteiro)
//Rotina para tentar resolver o problema de maximiza  o
  Var i, j, k, Tracos: inteiro
      cp, lp, loop, Nelcp: inteiro
      pivot, ZNeg, MaisNeg, Fator: real
      Verdade: logico
Inicio
  nl <- (m+1) //n  mero total de linhas dos quadros
  nc <- (n+m+1) //n  mero total de colunas dos quadros (excluindo a Base)
  Tracos <- Int((nc*60)/6.2) //n  mero de tra  os horizontais do quadro
  {Exibe o Quadrol do Simplex (sem a Base)}
  Escreval("Quadro inicial do Simpl x")
  Escreval("-----")
  Para i De 1 Ate nl Faca
    Para j De 1 Ate nc Faca
      Se(j=nc) Entao //valor MatB
        Escreva(" ", MatQuadrol[i,j]) //formata mais   direita
      Senao
        Escreva(MatQuadrol[i,j], " ")
      FimSe
    FimPara
    Escreval("")
  FimPara
  Escreval("-----")
  Escreval("")
  Escreval("")
  TemSolucao <- Verdadeiro
  loop <- 1
  Verdade <- Verdadeiro
  {Copia MatQuadrol[] em MatQuadro []}
  Para i De 1 Ate nl Faca

```

```

Para j De 1 Ate nc Faca
    MatQuadro[i,j] <- MatQuadro1[i,j]
FimPara
FimPara
{Loop para determinar a solução do problema de maximização}
Enquanto (Verdade) Faca
    {Ainda existe algum z(j) negativo!?!}
    ZNeg <- 0
    Para j De 1 Ate (nc-1) Faca
        Se(MatQuadro[nl,j]<0) Entao
            ZNeg <- ZNeg + 1
        FimSe
    FimPara
    Se(ZNeg=0) Entao
        Verdade <- Falso
        Interrompa //abandona o loop; não precisa mais nenhuma iteração
    FimSe
    Escreval("Iteração: ", loop)
    {Determina o elemento mais negativo em z}
    MaisNeg <- 0
    Para j De 1 Ate (nc-1) Faca
        Se(MatQuadro[nl,j] < MaisNeg) Entao
            MaisNeg <- MatQuadro[nl,j]
            cp <- j //coluna do pivot
        FimSe
    FimPara
    {Determina o divisor (b/Quadro[i,cp] da coluna do pivot}
    k <- 0
    Para i De 1 Ate m Faca
        Se(MatQuadro[i,cp] > 0) Entao
            k <- k + 1
            Se(k=1) Entao
                VetDiv[1] <- MatQuadro[i,nc]/MatQuadro[1,cp]
                pivot <- MatQuadro[i,cp]
                lp <- i //linha do pivot
            Senao
                VetDiv[i] <- MatQuadro[i,nc]/MatQuadro[i,cp]
                Se(VetDiv[i] < VetDiv[i-1]) Entao
                    pivot <- MatQuadro[i,cp]
                    lp <- i
                FimSe
            FimSe
        FimSe
    FimPara
    {Verifica se o sistema admite alguma solução}
    Nelcp <- 0 //número de elementos da coluna do pivot
    Para i De 1 Ate m Faca
        Se(MatQuadro[i,cp] < 0) Entao
            Nelcp <- Nelcp + 1
        FimSe
    FimPara
    Se(Nelcp=m) Entao
        Escreval("O sistema não tem solução; será encerrado...!")
        Timer(5000) //faz uma pausa de 5 segundos
        TemSolucao <- Falso
        Interrompa //sistema não tem solução: sai do loop de solução
    FimSe
    {Reduz o pivot a 1 fazendo lp/pivot}
    Para j De 1 Ate nc Faca
        MatQuadro[lp,j] <- MatQuadro[lp,j]/pivot
    FimPara

```

```

Para i De 1 Ate m Faca
  Se((i<>lp) e (MatQuadro[i,cp]<>0)) Entao
    Fator <- -MatQuadro[i,cp]
    Para j De 1 Ate nc Faca
      MatQuadro[i,j] <- Fator*MatQuadro[lp,j] + MatQuadro[i,j]
    FimPara
  FimSe
FimPara
{Reduz a zero o elemento mais negativo em z (MaisNeg)}
Para j De 1 Ate nc Faca
  MatQuadro[nl,j] <- -MaisNeg*MatQuadro[lp,j] + MatQuadro[nl,j]
FimPara
{Insere nova variável na Base}
Para i De 1 Ate m Faca
  Se(MatQuadro[i,cp]=1.00) Entao
    VetBase[i] <- VetV[cp]
  FimSe
FimPara
{Monta o dispositivo do Simpléx (a Base) para efeito de confirmação}
Escreva(" ")
Escreva("|Base|")
Para j De 1 Ate (nc-1) Faca
  VetX[j] <- "X" + NumpCarac(j)
  Se(j=1) Entao
    Escreva(" ", VetX[1])
  Senao
    Escreva(" ", VetX[j])
  FimSe
  Timer(100) //interrompe temporariamente o processamento por 0.1 segundos
FimPara
Escreva(" |")
Escreval("")
Para j De 1 Ate Tracos Faca
  Escreva("-")
  Timer(10)
FimPara
Escreval("")
{Imprime os elementos nos quadros do Simpléx}
Para i De 1 Ate m Faca
  Escreva(" | ", VetBase[i], " | ")
  Para j De 1 Ate nc Faca
    Se(j=1) Entao
      Se(MatQuadro[i,j]>=0) Entao
        Escreva(" ", MatQuadro[i,j]:4:2)
      Senao
        Escreva(MatQuadro[i,j])
    FimSe
  Senao
    Se(MatQuadro[i,j]>=0) Entao //analisa tamanho do elemento
      Se((MatQuadro[i,j]>=0) e (MatQuadro[i,j]<10)) Entao
        Escreva(" ",MatQuadro[i,j]:4:2)
      FimSe
      Se((MatQuadro[i,j]>=10) e (MatQuadro[i,j]<100)) Entao
        Escreva(" ",MatQuadro[i,j]:4:2)
      FimSe
      Se((MatQuadro[i,j]>=100) e (MatQuadro[i,j]<1000)) Entao
        Escreva(" ",MatQuadro[i,j]:4:2)
      FimSe
      Se((MatQuadro[i,j]>=1000) e (MatQuadro[i,j]<10000)) Entao
        Escreva(" ",MatQuadro[i,j]:4:2)
    FimSe
  FimSe

```

```

        FimSe
    Senao
        Se(Abs(MatQuadro[i,j])>=0) e (Abs(MatQuadro[i,j])<10) Entao
            Escreva(" ",MatQuadro[i,j]:4:2)
        FimSe
        Se(Abs(MatQuadro[i,j])>=10) e (Abs(MatQuadro[i,j])<100) Entao
            Escreva(" ",MatQuadro[i,j]:4:2)
        FimSe
        Se(Abs(MatQuadro[i,j])>=100) e (Abs(MatQuadro[i,j])<1000) Entao
            Escreva(" ",MatQuadro[i,j]:4:2)
        FimSe
        Se((Abs(MatQuadro[i,j])>=1000) e (Abs(MatQuadro[i,j])<10000)) Entao
            Escreva(" ",MatQuadro[i,j]:4:2)
        FimSe
    Fimse
Fimse
    Timer(100)
FimPara
    Escreval("")
FimPara //fim do loop de impressão da montagem do Simpléx
{Traça a linha da função objetivo}
Para j De 1 Ate Tracos Faca
    Escreva("-")
    Timer(10)
FimPara
    Escreval("")
    {Traça dos elementos do Simpléx com formatação}
    Para i De nl Ate nl Faca
        Escreva(" |", VetBase[i], "| ")
        Para j De 1 Ate nc Faca
            Se(j=1) Entao
                Se(MatQuadro[i,j]>=0) Entao
                    Escreva(" ", MatQuadro[i,j]:4:2)
                Senao
                    Escreva(MatQuadro[i,j]:4:2)
            FimSe
        Senao
            Se(j=nc) Entao
                Escreva(" | ",MatQuadro[i,j]:4:2)
            Senao
                Se(MatQuadro[i,j]>=0) Entao //analisa tamanho do negativo
                    Se((MatQuadro[i,j]>=0) e (MatQuadro[i,j]<10)) Entao
                        Escreva(" ",MatQuadro[i,j]:4:2)
                    FimSe
                    Se((MatQuadro[i,j]>=10) e (MatQuadro[i,j]<100)) Entao
                        Escreva(" ",MatQuadro[i,j]:4:2)
                    FimSe
                    Se((MatQuadro[i,j]>=100) e (MatQuadro[i,j]<1000)) Entao
                        Escreva(" ",MatQuadro[i,j]:4:2)
                    FimSe
                    Se((MatQuadro[i,j]>=1000) e (MatQuadro[i,j]<10000)) Entao
                        Escreva(" ",MatQuadro[i,j]:4:2)
                    FimSe
                    Se(Abs(MatQuadro[i,j])>=10000) Entao
                        Escreva(" ",MatQuadro[i,j]:4:2)
                    FimSe
                Senao //analisa tamanho do elemento negativo
                    Se(Abs(MatQuadro[i,j])>=0) e (Abs(MatQuadro[i,j])<10) Entao
                        Escreva(" ",MatQuadro[i,j]:4:2)
                    FimSe
                    Se(Abs(MatQuadro[i,j])>=10) e (Abs(MatQuadro[i,j])<100) Entao

```

```

        Escreva(" ",MatQuadro[i,j]:4:2)
    FimSe
    Se(Abs(MatQuadro[i,j])>=100) e (Abs(MatQuadro[i,j])<1000) Entao
        Escreva(" ",MatQuadro[i,j]:4:2)
    FimSe
    Se(Abs(MatQuadro[i,j])>=1000) e (Abs(MatQuadro[i,j])<10000) Entao
        Escreva(" ",MatQuadro[i,j]:4:2)
    FimSe
    Se(Abs(MatQuadro[i,j])>=10000) Entao
        Escreva(" ",MatQuadro[i,j]:4:2)
    FimSe
FimSe
FimSe
FimSe
FimPara
Escreval("")
Escreval("")
FimPara
loop <- loop + 1 //nova iteração
FimEnquanto //fim do loop da solução do problema
Escreval("")
FimProcedimento //fim do procedimento "ProResolveSistema"
//-----

Procedimento ProExibeSolucao(m,n:inteiro)
//Mostra a solução do problema de maximização.
Var i: inteiro
    zMax: real
    Resumol, Resumo2, StrBase: caractere
Inicio
Se(TemSolucao) Entao //o sistema tem solução
    Para i De 1 Ate (n+m) Faca
        VetVarBase[i] <- "X"
    FimPara
    Resumol <- "Resumo da solução do problema"
    Resumo2 <- "-----"
    Para i De 1 Ate Compr(Resumol) Faca
        Escreva(Copia(Resumol,i,1))
        Timer(10)
    FimPara
    Escreval("")
    Timer(100)
    Para i De 1 Ate Compr(Resumo2) Faca
        Escreva(Copia(Resumo2,i,1))
        Timer(10)
    FimPara
    Escreval("")
    {Imprime as variáveis da Base}
    Para i De 1 Ate (nl-1) Faca
        MatQuadro[i,nc] <- Int(MatQuadro[i,nc]*100+0.50)/100 //com duas decimais
        Escreval(VetBase[i], " =", MatQuadro[i,nc])
    FimPara
    {Imprime com valor zero as variáveis que não estão na Base}
    StrBase <- ""
    Para i De 1 Ate m Faca
        StrBase <- StrBase + VetBase[i]
    FimPara
    {Imprime o valor otimizado da função objetivo z}
    zMax <- Int(MatQuadro[nl,nc]*100+0.50)/100 //zMax com duas decimais
    Escreval("Zmax =", zMax)

```



```

    Senao
        Escreval(" O sistema assim colocado, não tem solução!!")
    FimSe
    Escreval("")
FimProcedimento //fim do procedimento "ProExibeSolucao"

//=====
//Programa principal
Inicio
    LimpaTela
    m <- 2 //número de equações
    n <- 2 //número de incógnitas
    ProCriaMatrizes(m,n) //monta as matrizes do sistemas
    ProInsereLinhasColunas(m,n) //insere linhas/colunas no Quadro Simpléx
    ProResolveSistema(m,n) //resolve o sistema
    ProExibeSolucao(m,n) //mostra os resultados
FimAlgoritmo //Fim do programa "ProgMaximizacaoV

```