

MÉTODO SIMPLEX

Leonardo Reis Dias; Vanessa da Silva Machado

Qual a finalidade do método?

Permite que se encontre valores ideais em situações em que diversos aspectos precisam ser respeitados. Diante de um problema, são estabelecidas inequações que representam restrições para as variáveis. A partir daí, testa-se possibilidades de maneira a otimizar o resultado da forma mais rápida possível.

O uso mais comum do Simplex é para se maximizar um resultado, ou seja, encontrar o maior valor possível para um total. Problemas típicos para se resolver com o Simplex são os que buscam quantidades ideais de produtos a serem comercializados, com restrições referentes ao armazenamento e à fabricação dos mesmos

Como foi idealizado?

A ideia é isolar uma função como sendo o objetivo. As quantidades que se deseja otimizar são representadas por variáveis aqui chamadas de

x_1, x_2, etc , e a função objetivo apresenta-se como $a_1x_1 + a_2x_2 + etc$, sendo

a_1, a_2, etc os coeficientes das variáveis. Estes demonstram a proporcionalidade entre elas. Geralmente são números racionais obtidos no problema que se deseja resolver.

As restrições são apresentadas como inequações. Indicam peculiaridades como o facto de uma empresa só conseguir armazenar um determinado peso ou quantidade dos produtos, por exemplo. De entre as possibilidades de valores para as variáveis que atendam às restrições, o algoritmo deve encontrar aqueles que dão à função objetivo o maior total possível.

A equipe pensou em implementar o método Simplex utilizando a linguagem C++, utilizando de funções e cálculos matemáticos.

O código:

Primeiramente, iremos passar por cada parte do código para explicações de suas partes, e logo em seguida um exercício como exemplo será utilizado.

Com a função principal e as funções limitantes monta-se uma tabela onde cada coluna representa os elementos de **Z (lucro máximo ou custo mínimo)**, seguido por cada um dos **X's**, e logo após as **variáveis de folga (uma para cada função limitante)** e ao final a variável **B (constantes)**.

O conteúdo da tabela são os valores numéricos que acompanham os **X's** e **xF's (folga)**, como também o **Z**. Menos a coluna de **B** que são os valores numéricos constantes das funções.

Primeiro:

Com a tabela formada, temos de buscar a **Variável que Entra**, cujo é o menor valor negativo da tabela. A coluna em que esta variável está fica como **Coluna Pivo**.

Para obter essa variável, percorremos a matriz (tabela) e filtramos o menor valor e consigo, sua coluna, como visto abaixo:

```
void buscarVariavelEntrada(){
    variavel_entrada=1000;
    linhaMenorElemento=0;
    colunaPivo=0;

    printf("\n\nProcurando Variavel Que Entra: \n");
    Sleep(4000);

    for(int i=0;i<totalLinhas;i++){
        for(int j=0;j<totalColunas;j++){
            if(tabela[i][j] < variavel_entrada){
                variavel_entrada = tabela[i][j];
                linhaMenorElemento = i;
                colunaPivo = j;
            }
        }
    }

    printf("Assim temos que a Variavel que Entra eh |%.2lf|\n",variavel_entrada);
}
```

Segundo:

Agora com a **Coluna Pivô** iremos buscar a **Linha Pivô** que consiste em dividir os valores da coluna **B** pelo seu respectivo da **Coluna Pivô**, o menor valor **positivo** obtido será o que definirá a **Linha Pivô**.

E o **Elemento Pivô** é o valor que se encontra **tanto na Coluna como na Linha Pivô**.

```
void buscarLinhaPivo(){
    menorPivo=1000;
    linhaPivo=0;
    elementoPivo=0;

    printf("\nObtendo Linha Pivo: \n\n");
    Sleep(5000);

    for(int i=0;i<totalLinhas;i++){
        if(i == linhaMenorElemento) continue;
        coluna_variavel_entrada = tabela[i][totalColunas-1] / tabela[i][colunaPivo];
        if(coluna_variavel_entrada < menorPivo && coluna_variavel_entrada>0){
            menorPivo = coluna_variavel_entrada;
            linhaPivo = i;
        }
    }
    elementoPivo=tabela[linhaPivo][colunaPivo];
    printf("Linha Pivo: %d\n",linhaPivo);
    printf("\nElemento Pivo: %.2lf\n",elementoPivo);
}
```

Terceiro:

Agora com o **Elemento Pivô** obtido, iremos calcular uma **Nova Linha Pivô**. Isso será feito dividindo cada elemento da **Linha Pivô** pelo **Elemento Pivô**, e assim a nova linha formará.

```
void calcularNovaLinhaPivo(){
    printf("\nCalculando Nova Linha Pivo: \n\n");
    Sleep(5000);

    printf("Nova Linha Pivo: \n");
    for(int i=0;i<totalColunas;i++){
        tabela[linhaPivo][i] = tabela[linhaPivo][i] / elementoPivo;
        printf("%.2lf ",tabela[linhaPivo][i]) ;
    }
    printf("\n");

    calcularNovasLinhas();
}
```

Quarto:

Com a **Nova Linha Pivô**, iremos calcular as **Novas Linhas** da tabela, substituindo assim a tabela original.

Essas **Novas Linhas** serão calculadas da seguinte maneira:

- Multiplicaremos a **Linha Pivô** pelo representante da linha que está sendo trocada.
 - Esse representante é o elemento presente na **Coluna Pivô** da linha em questão.
 - Deve-se multiplicar pelo inverso do representante. (Ex.: Se for 1, multiplica-se por -1)
- Adicionaremos essa linha obtida da multiplicação à linha que estamos trocando.
- Nova Linha X obtida.

```
void calcularNovasLinha() {
    printf("\nCalculando novas Linhas: \n");
    Sleep(5000);

    int aux=0;
    while(aux<totalLinha){
        if(aux!=linhaPivo){
            printf("\nNova %d º Linha: \n",aux+1);

            double auxLinha[totalColunas];
            double representanteInverso = tabela[aux][colunaPivo] * -1;
            for(int i=0;i<totalColunas;i++){
                auxLinha[i] = tabela[linhaPivo][i] * representanteInverso;
                tabela[aux][i] += auxLinha[i];

                printf("%.2lf ", tabela[aux][i]);
            }
            printf("\n");
        }
        aux++;
        Sleep(2000);
    }
}
```

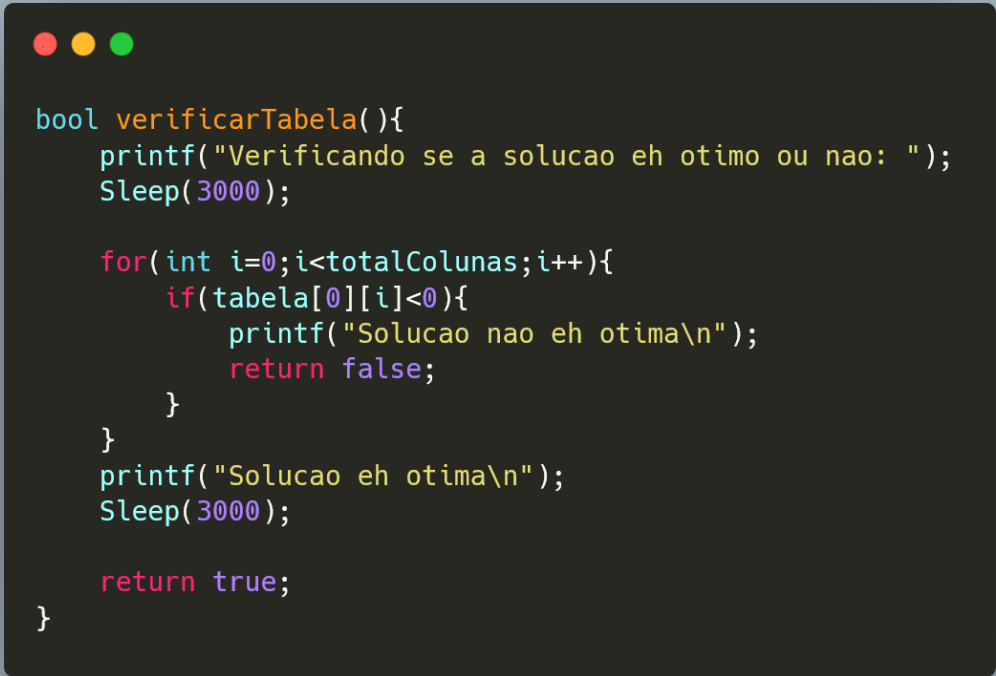
Quinto:

Agora com a nova tabela formada, iremos verificar se a solução é ótima ou não. Uma solução é ótima se, e somente se, a primeira linha da tabela não tiver nenhum valor negativo.

Caso tenha, teremos de começar novamente a partir dessa nova tabela até obter uma solução ótima.

Caso não tenha, a solução ótima foi encontrada.

Para verificar, apenas percorremos um FOR na primeira linha da tabela verificando se há números negativos. Caso haja, a função retornará FALSE, se não, TRUE simbolizando que é solução ótima.



```
bool verificarTabela(){
    printf("Verificando se a solucao eh otimo ou nao: ");
    Sleep(3000);

    for(int i=0;i<totalColunas;i++){
        if(tabela[0][i]<0){
            printf("Solucao nao eh otima\n");
            return false;
        }
    }
    printf("Solucao eh otima\n");
    Sleep(3000);

    return true;
}
```

Sexto:

Tomando-se que a solução ótima foi encontrada, iremos buscar as **Variáveis Básicas e Não Básicas**:

- Variáveis Básicas: São as variáveis cujo a coluna possua apenas um valor 1 e o resto 0.
- Variáveis Não Básicas: São as outras que não atendem ao requisito acima.

A busca foi feita percorrendo linha a linha de cada coluna, verificando se a soma daquela coluna é igual a 1, se for, aquela coluna representa uma **Variável Básica**, se não, representa uma **Variável Não Básica**.

```
void buscarVariaveisBasicas_aoBasicas(){
    double aux=0;

    for(int i=1;i<totalColunas-1;i++){
        for(int j=0;j<totalLinhas;j++){
            aux += tabela[j][i];
        }
        if(aux==1){
            variaveisBasicas[contBasicas] = i;
            contBasicas++;
        }
        else{
            variaveisNaoBasicas[contNaoBasicas] = i;
            contNaoBasicas++;
        }
        aux=0;
    }
}
```

Sétimo:

Agora sabendo das **Variáveis Básicas** e **Não Básicas**, iremos calcular as mesmas.

- Variáveis Básicas: São calculadas a partir do valor 1 da coluna, pegando na mesma linha da Coluna B.
- Variáveis Não Básicas: São zeradas.

O cálculo foi feito para que ache o valor de todas variáveis, independente do exercício e sua quantidade de variáveis. Para então serem exibidas em ordem primeiro as Básicas e depois as Não Básicas.

```
void calcularVariaveisBasicas_NaoBasicas(){
    printf("\nCalculando Resposta Final e Variaveis Basicas e Nao Basicas: \n");
    Sleep(3000);

    printf("Variaveis Basicas: \n");

    for(int i=0;i<contBasicas;i++){
        if(variaveisBasicas[i] <= qtdX){
            double valorResp=0;
            for(int j=0;j<totalLinhas;j++){
                if(tabela[j][variaveisBasicas[i]] == 1){
                    valorResp = tabela[j][totalColunas-1];
                    break;
                }
            }
            printf("X%d = %.2lf\n",variaveisBasicas[i],valorResp);
        }
        else if(variaveisBasicas[i] > qtdX){
            double valorResp=0;
            for(int j=0;j<totalLinhas;j++){
                if(tabela[j][variaveisBasicas[i]] == 1){
                    valorResp = tabela[j][totalColunas-1];
                    break;
                }
            }
            printf("xF%d = %.2lf\n",variaveisBasicas[i]-qtdX,valorResp);
        }
    }

    printf("\n\nVariaveis Nao Basicas: \n");
    double zero=0;
    for(int i=0;i<contNaoBasicas;i++){
        if(variaveisNaoBasicas[i] <= qtdX){
            printf("X%d = %.2lf\n",variaveisNaoBasicas[i],zero);
        }
        else if(variaveisNaoBasicas[i] > qtdX){
            printf("xF%d = %.2lf\n",variaveisNaoBasicas[i]-qtdX,zero);
        }
    }
}
```

Oitavo:

A resposta final **Z (Lucro máximo ou Custo Mínimo)** é feita da mesma forma que as **Variáveis Básicas**.

Pegando a linha da coluna Z, cujo elemento é diferente de 0 e fazer a divisão do elemento da coluna B, que está na mesma linha, por ele.

```
printf("\n\nCalculando Z: \n");
Sleep(2000);

double respostaFinal=0;

for(int i=0;i<totalLinhas;i++){
    if(tabela[i][0] != 0){
        respostaFinal = tabela[i][totalColunas-1] / tabela[i][0];
        break;
    }
}

printf("Valor de Z (LUCRO MAXIMO OU GASTO MINIMO): %.2lf\n",respostaFinal);
```

Nono:

```
int main(){
    system("color 0A");

    printf("Quantas funcoes limitantes? "); cin >> qtdFuncoesLimitantes;
    printf("Quantos X existem? "); cin >> qtdX;
    totalColunas = qtdX + qtdFuncoesLimitantes+2;
    totalLinhas = qtdFuncoesLimitantes+1;

    printf("Informe os valores da tabela inicial formada linha a linha: \n");
    for(int i=0;i<totalLinhas;i++){
        for(int j=0;j<totalColunas;j++){
            cin >> tabela[i][j];
        }
    }

    printf("\n-----\n");
    printf("Formato da primeira tabela: \n");
    for(int i=0;i<totalLinhas;i++){
        printf("\n");
        for(int j=0;j<totalColunas;j++){
            cout << tabela[i][j] << " ";
        }
    }

    do{
        printf("\n#####\n");

        iniciarMetodoSimplex();

    }while(!verificarTabela());

    buscarVariaveisBasicas_naoBasicas();
    calcularVariaveisBasicas_NaoBasicas();
}
```


Como visto acima, o programa inicia-se com o usuário digitando o número de Funções Limitantes na operação, como também a quantidade de X's do exercício.

Logo em seguida é requisitado que o usuário entre com a primeira tabela, sendo essa tendo sido montada por ele.

E então o programa iniciará a função **iniciarMetódoSimplex** que nada mais é que uma função que chamará as outras funções explicadas acima. Sendo que toda vez que a solução não for ótima, essa função será chamada e todas as outras também para que o cálculo continue.

Exemplo:

Tomando o seguinte exercício:

$$\text{Maximizar } Z = 2x_1 + 3x_2 + x_3$$

$$\begin{aligned} \text{Limitantes: } & x_1 + x_2 + x_3 \leq 40 \\ & 2x_1 + x_2 - x_3 \leq 20 \\ & 3x_1 + 2x_2 - x_3 \leq 30 \\ & x_1, x_2, x_3 \geq 0 \end{aligned}$$

Agora preparando as funções para a tabela:

$$Z - 2x_1 - 3x_2 - x_3 = 0$$

$$x_1 + x_2 + x_3 + xF_1 = 40$$

$$2x_1 + x_2 - x_3 + xF_2 = 20$$

$$3x_1 + 2x_2 - x_3 + xF_3 = 30$$

Primeira Tabela:

Z	X1	X2	X3	xF1	xF2	xF3	B
1	-2	-3	-1	0	0	0	0
0	1	1	1	1	0	0	40
0	2	1	-1	0	1	0	20
0	3	2	-1	0	0	1	30

1. Aqui o usuário entrará com as informações iniciais necessárias, inclusive os dados da primeira tabela como feito acima

*Importante ressaltar que os valores da tabela devem ser informados de linha em linha como visto abaixo.

```
C:\Users\leona\Documents\SimplexV2.exe
Quantas funcoes limitantes? 3
Quantos X existem? 3
Informe os valores da tabela inicial formada linha a linha:
1 -2 -3 -1 0 0 0 0
0 1 1 1 1 0 0 40
0 2 1 -1 0 1 0 20
0 3 2 -1 0 0 1 30

-----

Formato da primeira tabela:

1 -2 -3 -1 0 0 0 0
0 1 1 1 1 0 0 40
0 2 1 -1 0 1 0 20
0 3 2 -1 0 0 1 30
#####
```

2. Inicialmente o programa seguirá as ordens seguintes:
 - a. Calcular Variável que Entra
 - b. Obter a Linha Pivô e o Elemento Pivô
 - c. Calcular a Nova Linha Pivô
 - d. Calcular as Novas Linhas da tabela
 - e. Gerar a Nova Tabela
 - f. Verificar se a solução é ótima
 - i. Pode ser visto que **a solução não foi ótima** pois a primeira linha possui um elemento negativo, logo, o cálculo tem de continuar.

```
C:\Users\leona\Documents\SimplexV2.exe

Procurando Variavel Que Entra:
Assim temos que a Variavel que Entra eh |-3.00|

Obtendo Linha Pivo:

Linha Pivo: 3

Elemento Pivo: 2.00

Calculando Nova Linha Pivo:

Nova Linha Pivo:
0.00 1.50 1.00 -0.50 0.00 0.00 0.50 15.00

Calculando novas Linhas:

Nova 1 || Linha:
1.00 2.50 0.00 -2.50 0.00 0.00 1.50 45.00

Nova 2 || Linha:
0.00 -0.50 0.00 1.50 1.00 0.00 -0.50 25.00

Nova 3 || Linha:
0.00 0.50 0.00 -0.50 0.00 1.00 -0.50 5.00

Formato da tabela POS METODO:

1 2.5 0 -2.5 0 0 1.5 45
0 -0.5 0 1.5 1 0 -0.5 25
0 0.5 0 -0.5 0 1 -0.5 5
0 1.5 1 -0.5 0 0 0.5 15

-----
Verificando se a solucao eh otimo ou nao: Solucao nao eh otima
#####
```

3. Agora **a partir da Nova Tabela** o cálculo será retomado seguindo a mesma ordem descrita anteriormente.

Ao final, é possível ver que a primeira linha não possui nenhum valor negativo, logo como verificado no fim, **a solução é ótima**.

```
C:\Users\leona\Documents\SimplexV2.exe

Procurando Variavel Que Entra:
Assim temos que a Variavel que Entra eh |-2.50|

Obtendo Linha Pivo:

Linha Pivo: 1

Elemento Pivo: 1.50

Calculando Nova Linha Pivo:

Nova Linha Pivo:
0.00 -0.33 0.00 1.00 0.67 0.00 -0.33 16.67

Calculando novas Linhas:

Nova 1 || Linha:
1.00 1.67 0.00 0.00 1.67 0.00 0.67 86.67

Nova 3 || Linha:
0.00 0.33 0.00 0.00 0.33 1.00 -0.67 13.33

Nova 4 || Linha:
0.00 1.33 1.00 0.00 0.33 0.00 0.33 23.33

Formato da tabela POS METODO:

1 1.66667 0 0 1.66667 0 0.66667 86.6667
0 -0.333333 0 1 0.66667 0 -0.333333 16.6667
0 0.333333 0 0 0.333333 1 -0.66667 13.3333
0 1.33333 1 0 0.333333 0 0.333333 23.3333

-----
Verificando se a solucao eh otimo ou nao: Solucao eh otima
```

4. Agora com a Nova Tabela e com a solução verificada como ótima, o programa verificará quais variáveis são básicas e quais não são; e em seguida irá calcular seus respectivos valores.

Ao final a resposta final Z é calculada pelo programa exibindo assim o que foi pedido pelo exercício, que foi **Maximizar Z**. Com a resposta final sendo **Z = 86.67**.

```
C:\Users\leona\Documents\SimplexV2.exe

Formato da tabela POS METODO:

1 1.66667 0 0 1.66667 0 0.666667 86.6667
0 -0.333333 0 1 0.666667 0 -0.333333 16.6667
0 0.333333 0 0 0.333333 1 -0.666667 13.3333
0 1.33333 1 0 0.333333 0 0.333333 23.3333

-----
Verificando se a solucao eh otimo ou nao: Solucao eh otima

Calculando Resposta Final e Variaveis Basicas e Nao Basicas:
Variaveis Basicas:
X2 = 23.33
X3 = 16.67
xF2 = 13.33

Variaveis Nao Basicas:
X1 = 0.00
xF1 = 0.00
xF3 = 0.00

Calculando Z:
Valor de Z (LUCRO MAXIMO OU GASTO MINIMO): 86.67

Process returned 0 (0x0)   execution time : 73.359 s
Press any key to continue.
```

OBS: Outros exemplos podem ser testados para que o programa calcule, desde que as informações iniciais sejam inseridas (Quantidade de X's, quantidade de funções limitantes e valores da primeira tabela de linha em linha).

Código: <https://github.com/morpheusblack/metodoSimplex-C>