Implementação do Algoritmo Simplex para Solução de Problemas de Programação Linear

Natan da Silveira Ferreira

Junho de 2025

1. Introdução

Diversas soluções computacionais são resultados do cômputo de um modelo matemático e são transparentes para o usuário. Por exemplo, para o Uber calcular a melhor rota, e recalcular em tempo real caso o motorista faça algum desvio, ou mesmo precificar a corrida, há com certeza bastante Álgebra Linear por trás. Os sistemas de recomendação que mantém os usuários em suas plataformas, recomendando os conteúdos mais apropriados para um determinado grupo de assinantes são resultados de soluções de sistemas de equações lineares. Alguns algoritmos envolvendo fatoração de matrizes são responsáveis por soluções de problemas desta forma.

No entanto, há outra gama de problemas cujos modelos necessitam de outros algoritmos e abordagens. Por exemplo, situações onde é necessário minimizar os custos de uma produção, respeitando algumas restrições do contexto, ou maximizar o lucro, também sob certas restrições (como quantidade mínima de determinada matéria prima a ser importada ou quantidade máxima de produção para que não haja perca).

Assim, surgem os Problemas de Programação Linear (PPL), quando é necessário otimizar (maximizar ou minimizar) uma função linear sujeita a restrições também lineares. Esses problemas permitem modelar situações reais como alocação de recursos, planejamento de produção, logística, dietas, investimentos e escalonamento de tarefas. Uma vez modelado, um Problema de Programação Linear pode ser solucionado pelo algoritmo Simplex. O método Simplex é um algoritmo iterativo para resolver problemas de Programação Linear.

2. O Algoritmo Simplex

O algoritmo inicia com uma solução básica viável (isto é, uma solução que atende todas as restrições) e melhora o valor da função objetivo a cada iteração (maximizando ou minimizando) até chegar na solução ótima viável.

Neste trabalho, consideraremos os PPLs expressos na seguinte forma padrão:

$$\begin{aligned} \textbf{Maximize ou Minimize} \quad & Z = C^T \cdot X \\ \textbf{Sujeito a} \quad & A \cdot X = B \\ & X \geq 0 \end{aligned}$$

Onde:

- C é o vetor dos coeficientes da função objetivo;
- X é o vetor das variáveis de decisão;
- A é a matriz dos coeficientes das restrições;
- B é o vetor dos termos constantes das restrições.

A forma padrão exige que todas as restrições sejam equações (igualdades). Caso o problema original contenha desigualdades, estas podem ser transformadas em igualdades por meio da introdução de variáveis auxiliares, como variáveis de folga, excesso ou artificiais, conforme o caso.

3. Implementação

4. Resultados e Casos de Testes

4.1 Maximização

Resolvendo o PPL da seção anterior:

PPL de Maximização

```
Maximize z = 70x + 50y
sujeito a: 4x + 3y <= 240
           2x + y <= 100
```

Após tratar a entrada, o programa exibe as iterações:

Tableau Inicial

```
Tableux:
     VNB0  VNB1  VNB2  VNB3      SBV
 VB0 4.00  3.00  1.00  0.00    | 240.00
 VB1 2.00  1.00  0.00  1.00    | 100.00
-----------------------------------------
 Z = -70.00 -50.00 -0.00 -0.00   | 0.00
```

Tableau após 1ª Iteração

```
Tableux:
     VNB0  VNB1  VNB2  VNB3      SBV
 VB0 0.00  1.00  1.00  -2.00   | 40.00
 VB1 1.00  0.50  0.00  0.50    | 50.00
-----------------------------------------
 Z = 0.00  -15.00 0.00  35.00   | 3500.00
```

Tableau Final (Ótimo)

```
Tableux:
     VNB0  VNB1  VNB2  VNB3      SBV
 VB0 0.00  1.00  1.00  -2.00   | 40.00
 VB1 1.00  0.00  -0.50 1.50    | 30.00
-----------------------------------------
 Z = 0.00  0.00  15.00 5.00    | 4100.00
```

Saída Final

```
Solução ótima: x* = 30.00 40.00 0.00 0.00
Número de iterações do algorimo para o PPL : 2 iterações
```

4.2 Minimização

O seguinte PPL de minimização foi testado:

PPL de Minimização

```
Minimize z = 4x - 2y
sujeito a: 2x + y <= 10
           x - y <= 8
```

Resultados do programa:

Tableau Inicial

```
Tableux:
     VNB0  VNB1  VNB2  VNB3      SBV
 VB0 2.00  1.00  1.00  0.00    | 10.00
 VB1 1.00  -1.00 0.00  1.00    | 8.00
----------------------------------------
-Z = 4.00  -2.00 0.00  0.00    | 0.00
```

Tableau Final (Ótimo)

```
Tableux:
     VNB0  VNB1  VNB2  VNB3      SBV
 VB0 2.00  1.00  1.00  0.00    | 10.00
 VB1 3.00  0.00  1.00  1.00    | 18.00
----------------------------------------
-Z = 8.00  0.00  2.00  0.00    | 20.00
```

Saída Final

```
Solução ótima: x* = 0.00 10.00 0.00 18.00
Número de iterações do algorimo para o PPL : 1 iterações
```

No repositório do projeto é possível encontrar 4 casos de testes, dos quais 2 são de maximização e 2 de minimização.

4.3 Conclusão

A implementação do algoritmo simplex foi um sucesso, tornando possível a solução de problemas de programação linear com muitas variáveis e restrições, que seria impossível de solucionar manualmente. Mesmo para PPLs sem uma solução básica viável inicial, não é necessário modificar o algoritmo (para o método das duas fases), basta que seu dual seja viável e então utilizar o dual no programa desenvolvido. Afinal, como vimos no curso, é desta forma que o método das duas fases realmente funciona e conseguimos recuperar a solução ótima do primal, se houver, no tableux e no vetor X finais. Ademais, em relação a melhorias futuras, um outro programa '.c' que lê um PPL (que pode estar ou não na forma padrão) e faz a conversão (para a forma padrão), no formato esperado do programa aqui desenvolvido, seria bastante útil a fim de minimizar o trabalho do usuário final

Referências

[1] JARDIM, Maria Helena Cautiero Horta. *Slides de Aula: Otimização*. Universidade Federal do Rio de Janeiro (UFRJ), 2025. Material da disciplina ministrada no período 2025.1.

[2] UNIVESP. *Videoaula 10: Algoritmo Simplex - Problemas de Minimização*. Curso de Pesquisa Operacional. Disponível em: https://integra.univesp.br/courses/2642/pages/videoaula-10-algoritmo-simplex-problemas-de-minimizacao?module_item_id=201765. Acesso em: 30 de junho de 2025.

Apêndice A — Código-Fonte do Programa em C

O código a seguir implementa o algoritmo Simplex para resolução de problemas de Programação Linear. A linguagem utilizada foi C, com leitura interativa ou via redirecionamento de arquivo.

```c
#include<stdio.h>
#include<stdlib.h>
#include <string.h>
#include<math.h>
int max=-1; // se o PPL for de maximização, max = 1. Caso contrário, 0;
float *vetC, *vetX, *vetB, *matA;
double* output;
int nVar, nRestricoes;
float* A;


void printTableux();
int escolherVNBEntraNaBase();
int escolheVBSaiDaBase(int colunapivo);
void atualizaVetorSolucao();

void simplex(){
    // escolhe VNB que deve entrar na base
    // escolher VB que vai deixar a base
    // selecionada pivo e sua linha, deve fazer o pivoteamento

    int colunaPivo = escolherVNBEntraNaBase();
    int linhaPivo = escolheVBSaiDaBase(colunaPivo);

    vetX[linhaPivo] = 0;

    float pivo = matA[colunaPivo + linhaPivo*nVar];
    if(pivo != 1.f){
        // quando o pivo não é 1, é necessa io normalizar a linha pivô. Ou seja, dividir a linha do tableux pelo pivô
        for(int i = 0; i<nVar; i++){
            matA[linhaPivo*nVar + i] = matA[linhaPivo*nVar + i]/pivo;
        }
        vetB[linhaPivo] = vetB[linhaPivo]/pivo;
    }

    // pivoteamento na Z-linha:
    float fator = vetC[colunaPivo]*(-1);
    for(int i = 0; i<nVar; i++){
        vetC[i] = matA[linhaPivo*nVar + i]*fator + vetC[i];
    }
    vetC[nVar] = fator*vetB[linhaPivo] + vetC[nVar]; // SBV ( decidi armazenar aqui ).

    // pivoteamento na matriz das restições:
    for(int i = 0; i<nRestricoes; i++){
        if(i!=linhaPivo){
            float fator = matA[colunaPivo + i*nVar]*(-1);
            for(int j = 0; j<nVar; j++){
                matA[i*nVar + j] = matA[linhaPivo*nVar + j]*fator + matA[i*nVar + j];
            }
            vetB[i] = vetB[linhaPivo]*fator + vetB[i];
```

```
        }
    }

    atualizaVetorSolucao();

    printTableux();
}

int verificaOtimalidade(){
    for(int i = 0; i< nVar; i++){
         if(vetC[i]<0) return 0; // se há algum coef. negativo na função objetivo, então não estamos no ótimo
    }
    return 1; // se todos são positivos, então estamos no ótimo
}

void printTableux(){
    char *str = malloc(nVar*10 + 1); memset(str, '-', nVar*10 + 6); str[nVar*10+6] = '\0';
    printf("\nTableux:\n\n     ");

    for(int i = 0; i<nVar; i++) printf("VNB%d  ", i);
    printf("   SBV\n");
    for(int i = 0; i<nRestricoes; i++){
        printf(" VB%i ", i);
        for(int j = 0; j<nVar; j++){
            printf("%.2lf  ", matA[i*nVar + j]);
        }
        printf(" | %.2lf \n", vetB[i]);
    }
    if(max) printf("%s\n Z = ", str);
    else printf("%s\n-Z = ", str);
    for(int i=0; i < nVar ; i++){
       printf("%.2lf  ", vetC[i]);
    }
    printf(" | %.2lf\n", vetC[nVar]);
    printf("\n\n");
}

int escolherVNBEntraNaBase(){
    // método que retorna o índice da coluna pivô
    int indice = 0;
    float menorValor = 0; // o pivô deve ser um valor negativo

    for(int i = 0; i < nVar; i++){
        if(vetC[i] < menorValor){
            menorValor = vetC[i];
            indice = i;
        }
    }
    return indice;
}

int escolheVBSaiDaBase(int colunaPivo){
    double r = 0.f, R_Min = pow(10,4);
    int indice = 0;
    for(int i = 0; i<nRestricoes; i++){
        if(matA[colunaPivo + i*nVar] > 0.f){
```

```
                r = vetB[i]/matA[colunaPivo + i*nVar];
                if(r<R_Min){
                    R_Min = r;
                    indice = i;
                }
            }
    }
    return indice;
}

void atualizaVetorSolucao(){
    // Função que atualiza o vetor X, que contém os coef. da solução
    // Se for uma coluna de uma variável básica, só terá um elemento 1 e os demais 0. Nesse caso, a linha onde se encontra o 1 nessa coluna básica tem como solução a linha on vetor B
    // Já se encontrar algum elemento diferente de zeros com 1 um, então aquela coluna não esta na base e tem valor zero no vetor X
    for(int j = 0; j < nVar ; j++){
        int cont = 0;
        int linha = 0;
        for(int i = 0; i < nRestricoes; i++){
            if(matA[j+i*nVar] != 0.f && matA[j+i*nVar] != 1.f) {
                cont=0;
                break;
            }
            if(matA[j+i*nVar] == 1.f){
                cont++;
                linha=i;
            }
        }
        if(cont==1.f){
            vetX[j]=vetB[linha];
        }else{
            vetX[j] = 0.f;
        }
    }
}

int main(){
    // objetivo: maximizar / minimizar uma PPL na forma: c^t*x
    //                                                   sujeito a: A*x = b
    // vetor  C: coeficientes da funçao objetivo.
    // vetor  X: variáveis x.
    // matriz A: matriz dos coeficientes das restrições
    // vetor  B: constantes das restrições
    int numIteracoes = 0;
    char tipo[12];

    printf("O problema é de maximização (max) ou minimização (min) ?:\n");
    scanf("%s", tipo);
    max = (strcmp(tipo, "max") == 0) ? 1 : 0;

    printf("Digite a quantidade de variáveis distintas:\n");
    scanf("%d", &nVar);

    printf("Digite a quantidade de restrições (exceto não-negatividade)\n");
```

```
    scanf("%d", &nRestricoes);

    output = malloc(sizeof(double)*nVar);

    vetC = (float*) malloc(sizeof(float)*(nVar+1)); // resultado será armazenado aqui
    vetX = (float*) malloc(sizeof(float)*nVar);
    vetB = (float*) malloc(sizeof(float)*nRestricoes);
    matA = (float*) malloc(sizeof(float)*nRestricoes*nVar) ;

    printf("\nDigite os coeficientes da funcao objetivo:\n");
    for(int i=0; i < nVar ; i++){
        scanf("%f", &(vetC[i]));
        if(max) vetC[i] = vetC[i]*(-1); // se for um PPL de maximização, troca os sinais dos coef. da função objetivp
    }
    vetC[nVar]=0; // SBVI

    printf("\nDigite os coeficientes da matriz de restrições:\n");
    for(int i=0; i < nRestricoes ; i++){
        for(int j=0; j< nVar ; j++){
            scanf("%f", &(matA[i*nVar+j]));
        }
    }
    printf("\nDigite as constantes associadas às restrições:\n");
    for(int i=0; i < nRestricoes ; i++){
        scanf("%f", &(vetB[i]));
    }
    int offset = nVar - nRestricoes; // O número de zeros no início
    for (int i = 0; i < nVar; i++) {
        if (i < offset)  vetX[i] = 0;
        else vetX[i] = vetB[i - offset];
    }
    printf("Tableux simplex inicial: ");
    printTableux();

    while(!verificaOtimalidade()){
        simplex();
        numIteracoes++;
    }

    printf("Solução ótima: x* = ");
    for(int i=0; i<nVar; i++){
        printf("%.2lf ", vetX[i]);
    }

    printf("\nNúmero de iterações do algorimo para o PPL : %d iterações\n", numIteracoes);

    free(vetX);
    free(vetC);
    free(vetB);
    free(matA);
    return 0;
}
```

Listing 1: Algoritmo Simplex em C