Gustavo Perez Katague - #USP 6797143
Pedro Paulo Vezzá Campos - #USP 7538743

Implementação do Método Simplex

São Paulo - SP, Brasil

15 de junho de 2012

Gustavo Perez Katague - #USP 6797143
Pedro Paulo Vezzá Campos - #USP 7538743

Implementação do Método Simplex

Trabalho apresentado para avaliação na disciplina MAC0315, do curso de Bacharelado em Ciência da Computação, turma 45, da Universidade de São Paulo, ministrada pelo professor Ernesto Julián Goldberg Birgin

DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO
INSTITUTO DE MATEMÁTICA E ESTATÍSTICA
UNIVERSIDADE DE SÃO PAULO

São Paulo - SP, Brasil

15 de junho de 2012

Introdução

Neste terceiro exercício-programa de MAC0315 - Programação Linear foi pedido que implementássemos o método Simplex, algoritmo de otimização de problemas de programação linear. O método foi escrito na linguagem Octave, o que facilitou as operações com vetores e matrizes. Neste relatório serão apresentados os detalhes de implementação do trabalho, testes aplicados para verificar o bom funcionamento do método além de uma descrição do Método Simplex implementado (Simplex *full tableau* em duas fases) por fim serão apresentados comentários a respeito do exercício-programa e uma conclusão.

Implementação

Nesta seção serão apresentadas as funções implementadas no EP, contendo uma breve descrição de suas atribuições, parâmetros, dados de retorno e comentários da implementação quando pertinentes.

```
[ind x] = simplex(A,b,c,m,n,print)
```

Descrição Responsável por dados os dados de entrada do problema executar o método simplex de duas fases a um problema dado. Implementação do algoritmo descrito na página 116 de [1].

Parâmetros **A** Matriz de coeficientes das restrições

b Vetor de termos independentes das restrições

c Vetor de custos

m Número de linhas de A

n Número de colunas de A

print true para imprimir as iterações ou false caso contrário.

Retorno Nos casos de `ind` = 1 ou -1, `x` será um vetor de zeros com n posições.

Solução ótima encontrada ind = 0 e a solução ótima x

Solução ilimitada ind = 1

Problema Inviável ind = -1

Comentários Como esta função engloba tanto a preparação do problema para a execução da fase 1 e 2 do método Simplex, ela monta o tableau (Variável `T`) já adicionando as variáveis artificiais do problema e fazendo-as as variáveis básicas iniciais do problema. Como a matriz básica B dessas variáveis é uma matriz identidade mxm, não houve em nenhum momento a necessidade de calcular a inversa de matrizes. O tableau foi assim produzido: O custo inicial é simplesmente a soma do vetor b, o vetor de custos reduzidos é dado como a soma das n colunas de A com sinais trocados para as n primeiras colunas e 0 para as n colunas seguintes, o valor de x_B é o próprio vetor b e a matriz das $2n$ direções básicas é a concatenação da matriz A com a identidade nxn.

```
[ind T basicas] = fulltableau(T, m, n, basicas, print)
```

Descrição Responsável por realizar as iterações do simplex usando tableau. Implementação do algoritmo descrito na página 100 de [1].

Parâmetros **T** Tableau precalculado (Contendo as 0-ésimas linha e coluna).

m Número de linhas do tableau

n Número de colunas do tableau

print true para imprimir as iterações ou false caso contrário.

Retorno **ind** o resultado da execução do algoritmo, tal como descrito para a função `simplex()`.

T O tableau atualizado após a execução do algoritmo.

basicas Vetor de variáveis básicas. Ex: Se $x_1, x_4 e x_2$ estão na base nesta ordem, então `basicas == [1 4 2]`.

Comentários Esta função foi implementada para permitir ser utilizada tanto nas fases 1 quanto 2. Assim, para evitar o recálculos desnecessário entre produzir uma solução para a fase 1 e iniciar fase 2, a função sempre recebe o tableau a ser processado como parâmetro e o retorna ao fim da execução. Isso aumenta a eficiência do programa enquanto evita maiores problemas de erros de arredondamento. A função utiliza a regra de Bland para a escolha do pivô a cada iteração, isso garante que o algoritmo parará em um número finito de iterações.

```
imprime(tableau, m, n, iter, pivo, basicas)
```

Descrição Responsável por imprimir na saída padrão uma iteração do método simplex segundo especificação do EP.

Parâmetros **T** O tableau atual da iteração atual (Contendo as 0-ésimas linha e coluna)

m Quantidade de linhas do tableau

n Quantidade de colunas do tableau

iter Número da iteração atual

pivo Vetor indicando quais as coordenadas no tableau do pivô escolhido

basicas Vetor de variáveis básicas. Ex: Se $x_1, x_4 e x_2$ estão na base nesta ordem, então `basicas == [1 4 2]`.

Retorno **Solução ótima encontrada** ind = 0 e a solução ótima x

Solução ilimitada ind = 1

Problema Inviável ind = -1

```
[comp] = compare(x, y)
```

Descrição Responsável por comparar números considerando erros de arredondamento. Adota `eps` como erro tolerável.

Parâmetros **x** Primeiro número a ser comparado

y Segundo número a ser comparado

Retorno **|x - y|** $\leq$ **eps** `comp`$= 0$

x < y `comp`$= 1$

x > y `comp`$= -1$

Comentários Esta função foi implementada para ser utilizada no programa de forma a evitar comparações problemáticas devido a erros de arredondamento acumulados vindos do uso de notação em ponto flutuante e repetidas operações de multiplicação e divisão.

Testes Realizados

Nesta seção descreveremos os testes realizados no programa para atestar seu funcionamento adequado nas diferentes possibilidades de respostas que o método Simplex pode retornar. Como comparação, o mesmo problema foi testado no programa Maxima [2] e seus resultados foram comparados. Os problemas estão descritos no formato das matrizes A, b, c e dimensões m e n já adequados para inserção diretamente via linha de comando do EP.

Problemas com solução finita

Problema 1

Solução obtida para comparação: Solução ótima = -136

```
A = [1 2 2 1 0 0;
 2 1 2 0 1 0;
 2 2 1 0 0 1];
b = [20 20 20]';
c = [-10 -12 -12 0 0 0]';
m = 3;
n = 6;

Simplex: Fase 1
Iteração 1
           | x1      | x2      | x3      | x4      | x5      | x6      | x7      | x8      | x9      |
   -60.000 |  -5.000 |  -5.000 |  -5.000 |  -1.000 |  -1.000 |  -1.000 |   0.000 |   0.000 |   0.000 |
-----------------------------------------------------------------------------------------------------
x7  20.000 |   1.000 |   2.000 |   2.000 |   1.000 |   0.000 |   0.000 |   1.000 |   0.000 |   0.000 |
x8  20.000 |   2.000*|   1.000 |   2.000 |   0.000 |   1.000 |   0.000 |   0.000 |   1.000 |   0.000 |
x9  20.000 |   2.000 |   2.000 |   1.000 |   0.000 |   0.000 |   1.000 |   0.000 |   0.000 |   1.000 |

Iteração 2
           | x1      | x2      | x3      | x4      | x5      | x6      | x7      | x8      | x9      |
   -10.000 |   0.000 |  -2.500 |   0.000 |  -1.000 |   1.500 |  -1.000 |   0.000 |   2.500 |   0.000 |
-----------------------------------------------------------------------------------------------------
x7  10.000 |   0.000 |   1.500 |   1.000 |   1.000 |  -0.500 |   0.000 |   1.000 |  -0.500 |   0.000 |
x1  10.000 |   1.000 |   0.500 |   1.000 |   0.000 |   0.500 |   0.000 |   0.000 |   0.500 |   0.000 |
x9   0.000 |   0.000 |   1.000*|  -1.000 |   0.000 |  -1.000 |   1.000 |   0.000 |  -1.000 |   1.000 |

Iteração 3
           | x1      | x2      | x3      | x4      | x5      | x6      | x7      | x8      | x9      |
   -10.000 |   0.000 |   0.000 |  -2.500 |  -1.000 |  -1.000 |   1.500 |   0.000 |   0.000 |   2.500 |
-----------------------------------------------------------------------------------------------------
x7  10.000 |   0.000 |   0.000 |   2.500*|   1.000 |   1.000 |  -1.500 |   1.000 |   1.000 |  -1.500 |
x1  10.000 |   1.000 |   0.000 |   1.500 |   0.000 |   1.000 |  -0.500 |   0.000 |   1.000 |  -0.500 |
x2   0.000 |   0.000 |   1.000 |  -1.000 |   0.000 |  -1.000 |   1.000 |   0.000 |  -1.000 |   1.000 |

Iteração 4
           | x1      | x2      | x3      | x4      | x5      | x6      | x7      | x8      | x9      |
     0.000 |   0.000 |   0.000 |   0.000 |   0.000 |   0.000 |   0.000 |   1.000 |   1.000 |   1.000 |
-----------------------------------------------------------------------------------------------------
x3   4.000 |   0.000 |   0.000 |   1.000 |   0.400 |   0.400 |  -0.600 |   0.400 |   0.400 |  -0.600 |
x1   4.000 |   1.000 |   0.000 |   0.000 |  -0.600 |   0.400 |   0.400 |  -0.600 |   0.400 |   0.400 |
x2   4.000 |   0.000 |   1.000 |   0.000 |   0.400 |  -0.600 |   0.400 |   0.400 |  -0.600 |   0.400 |

Simplex: Fase 2
Iteração 1
           | x1      | x2      | x3      | x4      | x5      | x6      |
   136.000 |   0.000 |   0.000 |   0.000 |   3.600 |   1.600 |   1.600 |
-------------------------------------------------------------------------
x3   4.000 |   0.000 |   0.000 |   1.000 |   0.400 |   0.400 |  -0.600 |
```

```
x1   4.000 |   1.000 |   0.000 |   0.000 |  -0.600 |   0.400 |   0.400 |
x2   4.000 |   0.000 |   1.000 |   0.000 |   0.400 |  -0.600 |   0.400 |

Solução ótima encontrada com custo -136.000:
x =

   4
   4
   4
   0
   0
   0
```

Problema 2

Solução obtida para comparação: Solução ótima = -1.750

```
c = [1 1 1 0]';
A = [1 2 3 0;
 -1 2 6 0;
 0 4 9 0;
 0 0 3 1];
b = [3 2 5 1]';
m = 4;
n = 4;

Simplex: Fase 1
Iteração 1
           | x1      | x2      | x3      | x4      | x5      | x6      | x7      | x8      |
   -11.000 |  -0.000 |  -8.000 | -21.000 |  -1.000 |   0.000 |   0.000 |   0.000 |   0.000 |
-------------------------------------------------------------------------------------------
x5   3.000 |   1.000 |   2.000 |   3.000 |   0.000 |   1.000 |   0.000 |   0.000 |   0.000 |
x6   2.000 |  -1.000 |   2.000*|   6.000 |   0.000 |   0.000 |   1.000 |   0.000 |   0.000 |
x7   5.000 |   0.000 |   4.000 |   9.000 |   0.000 |   0.000 |   0.000 |   1.000 |   0.000 |
x8   1.000 |   0.000 |   0.000 |   3.000 |   1.000 |   0.000 |   0.000 |   0.000 |   1.000 |

Iteração 2
           | x1      | x2      | x3      | x4      | x5      | x6      | x7      | x8      |
    -3.000 |  -4.000 |   0.000 |   3.000 |  -1.000 |   0.000 |   4.000 |   0.000 |   0.000 |
-------------------------------------------------------------------------------------------
x5   1.000 |   2.000*|   0.000 |  -3.000 |   0.000 |   1.000 |  -1.000 |   0.000 |   0.000 |
x2   1.000 |  -0.500 |   1.000 |   3.000 |   0.000 |   0.000 |   0.500 |   0.000 |   0.000 |
x7   1.000 |   2.000 |   0.000 |  -3.000 |   0.000 |   0.000 |  -2.000 |   1.000 |   0.000 |
x8   1.000 |   0.000 |   0.000 |   3.000 |   1.000 |   0.000 |   0.000 |   0.000 |   1.000 |

Iteração 3
           | x1      | x2      | x3      | x4      | x5      | x6      | x7      | x8      |
    -1.000 |   0.000 |   0.000 |  -3.000 |  -1.000 |   2.000 |   2.000 |   0.000 |   0.000 |
-------------------------------------------------------------------------------------------
x1   0.500 |   1.000 |   0.000 |  -1.500 |   0.000 |   0.500 |  -0.500 |   0.000 |   0.000 |
x2   1.250 |   0.000 |   1.000 |   2.250 |   0.000 |   0.250 |   0.250 |   0.000 |   0.000 |
x7   0.000 |   0.000 |   0.000 |   0.000 |   0.000 |  -1.000 |  -1.000 |   1.000 |   0.000 |
```

```
x8   1.000 |   0.000 |   0.000 |   3.000*|   1.000 |   0.000 |   0.000 |   0.000 |   1.000 |

Iteração 4
           | x1      | x2      | x3      | x4      | x5      | x6      | x7      | x8      |
     0.000 |   0.000 |   0.000 |   0.000 |   0.000 |   2.000 |   2.000 |   0.000 |   1.000 |
-------------------------------------------------------------------------------------------
x1   1.000 |   1.000 |   0.000 |   0.000 |   0.500 |   0.500 |  -0.500 |   0.000 |   0.500 |
x2   0.500 |   0.000 |   1.000 |   0.000 |  -0.750 |   0.250 |   0.250 |   0.000 |  -0.750 |
x7   0.000 |   0.000 |   0.000 |   0.000 |   0.000 |  -1.000 |  -1.000 |   1.000 |   0.000 |
x3   0.333 |   0.000 |   0.000 |   1.000 |   0.333 |   0.000 |   0.000 |   0.000 |   0.333 |

Simplex: Fase 2
Iteração 1
           | x1      | x2      | x3      | x4      |
    -1.833 |   0.000 |   0.000 |   0.000 |  -0.083 |
-----------------------------------------------------
x1   1.000 |   1.000 |   0.000 |   0.000 |   0.500 |
x2   0.500 |   0.000 |   1.000 |   0.000 |  -0.750 |
x3   0.333 |   0.000 |   0.000 |   1.000 |   0.333*|

Iteração 2
           | x1      | x2      | x3      | x4      |
    -1.750 |   0.000 |   0.000 |   0.250 |   0.000 |
-----------------------------------------------------
x1   0.500 |   1.000 |   0.000 |  -1.500 |   0.000 |
x2   1.250 |   0.000 |   1.000 |   2.250 |   0.000 |
x4   1.000 |   0.000 |   0.000 |   3.000 |   1.000 |

Solução ótima encontrada com custo 1.750:
x =

   0.50000
   1.25000
   0.00000
   1.00000
```

Problemas com solução $-\infty$

Problemas com restrições linearmente dependentes

Problema 1

Solução obtida para comparação: Solução ótima = 3

```
A = [1 1 1;
 2 2 2;
 3 3 3;
 4 4 4];
b = [3 6 9 12]';
c = [1 1 1]';
m = 4;
```

```
n = 3;


Simplex: Fase 1
Iteração 1
           | x1       | x2       | x3       | x4       | x5       | x6       | x7       |
   -30.000 | -10.000 | -10.000 | -10.000 |   0.000 |   0.000 |   0.000 |   0.000 |
-------------------------------------------------------------------------------------
x4   3.000 |   1.000*|   1.000 |   1.000 |   1.000 |   0.000 |   0.000 |   0.000 |
x5   6.000 |   2.000 |   2.000 |   2.000 |   0.000 |   1.000 |   0.000 |   0.000 |
x6   9.000 |   3.000 |   3.000 |   3.000 |   0.000 |   0.000 |   1.000 |   0.000 |
x7  12.000 |   4.000 |   4.000 |   4.000 |   0.000 |   0.000 |   0.000 |   1.000 |

Iteração 2
           | x1       | x2       | x3       | x4       | x5       | x6       | x7       |
     0.000 |   0.000 |   0.000 |   0.000 |  10.000 |   0.000 |   0.000 |   0.000 |
-------------------------------------------------------------------------------------
x1   3.000 |   1.000 |   1.000 |   1.000 |   1.000 |   0.000 |   0.000 |   0.000 |
x5   0.000 |   0.000 |   0.000 |   0.000 |  -2.000 |   1.000 |   0.000 |   0.000 |
x6   0.000 |   0.000 |   0.000 |   0.000 |  -3.000 |   0.000 |   1.000 |   0.000 |
x7   0.000 |   0.000 |   0.000 |   0.000 |  -4.000 |   0.000 |   0.000 |   1.000 |

Simplex: Fase 2
Iteração 1
           | x1       | x2       | x3       |
    -3.000 |   0.000 |   0.000 |   0.000 |
-----------------------------------------
x1   3.000 |   1.000 |   1.000 |   1.000 |

Solução ótima encontrada com custo 3.000:
x =

   3
   0
   0
```

Problemas inviáveis

Método Simplex

Das várias maneiras de implementar o Simplex, tomamos como referência a implementação através do *full tableau* em duas fases descrito em [1], onde a cada iteração guardamos uma matriz $(m+1)$ x $(n+1)$:

$-c_{B^T}B^{-1}b$	$c^T - c_{B^T}B^{-1}A$
$B^{-1}b$	$B^{-1}A$

A é a matriz de coeficientes das variáveis;

B é a matriz m x m correspondente à base da solução viável básica atual;

c é o vetor de custos;

c_b é o vetor de custos nas variáveis básicas.

Temos então que:

$-c_{B^T}B^{-1}b$ é o custo negativado da solução viável básica atual;

$c^T - c_{B^T}B^{-1}A$ é o vetor de custos reduzidos

$B^{-1}b$ é o vetor correspondente às variáveis básicas da solução viável básica atual;

$B^{-1}A, i = 1..n$ são as i-ésimas colunas do tableau.

No código, é possível observar que a responsável pelas iterações do Simplex é a função full tableau. Ela segue os 5 passos descritos no livro.

1. Começa com um tableau já pronto, associado a uma base B correspondente a uma solução viável básica x.
2. Examina os custos reduzidos da 0-ésima linha do tableau. Se forem todas não negativas, a solução atual é ótima e o programa termina. Senão, ele escolhe uma nova variável x_j, onde j é o menor índice de uma variável não básica que possua custo reduzido negativo.
3. Seja então a coluna pivô $u = B^{-1}A_j$. Se nenhuma entrada de u for positiva, o custo ótimo é $-\infty$, e o programa termina.
4. Para cada i para qual u_i é positivo, calcular o valor $x_{B(i)}/u_i$. Seja l o índice da linha cujo cálculo acima apresente o menor valor. Em caso de empate, de acordo com a regra de Bland, utilizamos o menor índice básico $B(i)$. A coluna $A_{B(l)}$ sai da base e a coluna A_j entra na base.
5. Soma-se a cada linha do tableau uma constante múltipla da l-ésima linha (linha pivô) para que u_l (elemento pivô) tenha valor 1 e todas as outras entradas de u tenham valor 0.

A função full tableau, entretanto, não resolve sozinha o problema de otimização. Ela necessita inicialmente de uma solução viável básica para começar a iterar. Para isso, inicialmente o programa executa a função simplex, que é dividida em duas fases:

Fase 1

1. Multiplica as restrições necessárias para que b ≥ 0.

2. Introduz variáveis artificiais $y_1, ..., y_m$ e executa a função montar tableau, a partir da nova matriz de restrições e do novo vetor de custo, que descreve $\sum_{i=1}^{m} y_i$. Com o tableau montado, executa-se a função full tableau para o problema auxiliar.

3. Se o custo ótimo fo problema auxiliar for 0, uma solução viável para o problema original foi encontrada. Se não existirem variáveis artificiais na base final, estas e suas reespecticas colunas serão eliminadas, e temos uma base viável para o problema original.

4. Se a l-ésima variável básica é artificial, olhar para a l-ésima entrada das colunas $B^{-1}A_j, j = 1..n$. Se todas essas entradas forem 0, a l-ésima linha corresponde a uma restrição redundante, e é eliminada. Senão, se a l-ésima entrada da j-ésima coluna é diferente de 0, executa-se a mudança de base, com esta entrada como pivô. $x_{B(l)}$ sai da base e x_j entra na base. E isto é repetido até que todas as variáveis artificiais saiam da base.

Fase 2

1. Tomamos a base e o tableau final obtidos pela fase 1

2. Calcula-se os custos reduzidos para todas as variáveis na base inicial, usando os coeficientes do problema original.

3. Executa-se a função full tableau ao problema original.

Conclusão

O trabalho ajudou a fixar os conceitos vistos em aula, além de nos fazer entrar em contato com uma linguagem de programação científica. Foi possível ver melhor algoritmicamente e de maneira prática o funcionamento do método Simplex. Porém, percebemos ao longo da implementação a dificuldade de garantir que o método foi implementado segundo o que o livro sugeriu. Houve a necessidade de produzir uma quantidade suficiente de testes para testar a robustez do código. Ainda, tivemos que lidar com problemas de comparações entre ponto flutuantes, algo que já era esperado dada a forma de representar números em ponto flutuante em computadores.

Referências

[1] BERTSIMAS, D.; TSITSIKLIS, J. *Introduction to Linear Optimization*. 1st. ed. [S.l.]: Athena Scientific, 1997. ISBN 1886529191.

[2] PROJETO MAXIMA. *Maxima, a Computer Algebra System*. 2012. [Online; acessado em 15 de junho de 2012]. Disponível em: <http://maxima.sourceforge.net/>.