Universidade Federal de São Carlos – UFSCar

Departamento de Engenharia de Produção

Tutorial básico de uso do Octave/Matlab para o método simplex

Prof. Dr. Pedro Munari (munari@dep.ufscar.br)

Octave e Matlab são *softwares* muito utilizados para a realização de cálculos matemáticos. Ambos permitem definir vetores, matrizes e funções de forma rápida, de modo a utilizá-los em cálculos de expressões e até mesmo na execução de algoritmos. Possuem muitas funcionalidades já implementadas, o que agiliza o uso e permite uma prototipagem de ideias relativamente rápida. O Matlab é um software comercial e exige uma licença para seu uso (`http://www.mathworks.com/products/matlab`). O Octave é um software gratuito e possui versão para download e instalação (`http://www.gnu.org/software/octave`), além de plataforma online (`http://octave-online.net`). Ambos possuem mesma sintaxe, portanto os comandos apresentados a seguir podem ser utilizados tanto no Matlab, quanto no Octave.

Neste tutorial, apresentamos uma rápida introdução ao uso dos softwares Matlab e Octave, com enfoque no uso desses softwares para a realização de cálculos do método simplex. O intuito não é de programar o método simplex de forma completa e automatizada, mas simplesmente de ter o Octave/Matlab como uma forma de auxiliar nos cálculos de cada iteração do método. Dessa forma, o objetivo de recorremos aos softwares é de ter um apoio didático ao aprendizado do método simplex.

1 Definindo vetores e matrizes

A interação com o Octave/Matlab é por meio da declaração de comandos de texto. Embora possa parecer pouco intuitiva a princípio, em pouco tempo o usuário começa a observar que essa forma de interação passa a ser simples e ágil conforme adquire pratica.

Vamos iniciar pela definição de vetores no Octave/Matlab. Um vetor deve ser delimitado por colchetes, sendo seus elementos separados por espaço. Por exemplo, o vetor-linha $x = (5, 2, 8, 0, 1)$ é definido como:

```
>> x = [ 5 2 8 0 1 ]
```

Após digitar o comando acima e pressionar *Enter*, a seguinte saída é observada:

```
x =
   5   2   8   0   1
```

Caso o usuário queira entrar com um vetor-coluna, os elementos devem ser separados por ponto-e-vírgula. Por exemplo, para entrarmos com o transposto do vetor definido anteriormente, fazemos:

```
>> xt = [ 5; 2; 8; 0; 1 ]
```

obtendo a saída:

```
xt =
   5
   2
   8
   0
   1
```

A entrada de matrizes é similar à de vetores. Os elementos de uma mesma linha são separados por espaços, enquanto as linhas são separadas por ponto-e-vírgula. Por exemplo, considere a matriz:

$$M = \begin{bmatrix} 2 & 0{,}3 & 9 \\ 0{,}1 & 1{,}7 & 1 \\ 10 & 0 & 5 \end{bmatrix}$$

Podemos definir essa matriz usando o comando (observe que devemos usar *ponto* como separador decimal):

```
>> M = [ 2 0.3 9; 0.1 1.7 1; 10 0 5 ]
```

Após digitarmos o comando e pressionarmos *Enter*, obtemos a saída:

```
M =
    2.00000    0.30000    9.00000
    0.10000    1.70000    1.00000
   10.00000    0.00000    5.00000
```

Não é necessário atribuirmos um nome a vetores e matrizes. Por exemplo, poderíamos entrar com um vetor digitando apenas seus elementos:

```
>> [ 5 2 8 0 1 ]
ans =
   5   2   8   0   1
```

Entretanto, não teríamos como acessar esse vetor posteriormente, já que não atribuímos nenhum nome de variável a ele (como feito anteriormente). Assim, é importante definirmos nomes para os vetores e matrizes que criamos, para podermos usá-los e modificá-los posteriormente. Assim, tudo que criamos fica armazenado no Octave/Matlab. Por exemplo, se quisermos visualizar novamente o vetor x, basta digitarmos `x` e pressionar *Enter*:

```
>> x
x =
   5   2   8   0   1
```

Para imprimir apenas um elemento de x, basta usarmos o nome do vetor e indicar a posição desejada entre parênteses:

```
>> x(3)
ans =  8
```

Para modificar um elemento de um vetor ou de uma matriz, podemos usar o mesmo comando, seguido do sinal de igual e do novo valor. Por exemplo, para redefinirmos o quarto elemento do vetor x como 3, fazemos:

```
>> x(4) = 3
x =
   5   2   8   3   1
```

De forma similar, podemos modificar os elementos de uma matriz, indicando linha e coluna da posição desejada:

```
>> M(2,3) = 8
M =
    2.00000    0.30000    9.00000
    0.10000    1.70000    8.00000
   10.00000    0.00000    5.00000
```

Nesse exemplo, modificamos o elemento na linha 2 e coluna 3 da matriz M para 8.

Por fim, vale dizer que é possível criar a matriz identidade de forma automatizada, por meio do comando `eye` seguindo do número de linhas/colunas:

```
>> eye(3)
ans =
Diagonal Matrix
```

```
   1   0   0
   0   1   0
   0   0   1
```

2 Operações com vetores e matrizes

É possível realizarmos diversas operações com vetores e matrizes. Na sequência, apresentamos aquelas mais relevantes para uso no método simplex. Vamos iniciar vendo como transpor um vetor ou uma matriz. Para isso, basta adicionarmos o sinal de aspas simples após o nome do vetor/matriz. Por exemplo:

```
>> x'
ans =
   5
   2
   8
   3
   1

>> M'
ans =
    2.00000    0.10000   10.00000
    0.30000    1.70000    0.00000
    9.00000    8.00000    5.00000
```

Podemos calcular a inversa de uma matriz não-singular usando o comando `inv` seguido do nome da matriz entre parênteses. Por exemplo:

```
>> inv(M)
ans =
  -0.075791   0.013375   0.115025
  -0.708872   0.713330   0.134641
   0.151583  -0.026750  -0.030049
```

Outro cálculo útil é dado pela multiplicação de vetores e matrizes. Podemos calcular o produto entre vetores e matrizes usando o operador `*`. O produto interno entre dois vetores pode ser calculado como:

```
>> x
x =
   5   2   8   3   1

>> z = [ 1; 0; 1; 0; -1]
z =
   1
   0
   1
   0
  -1

>> x * z
ans =  12
```

O exemplo a seguir traz o produto entre uma matriz e um vetor:

```
>> y = [1; 0; 2]
y =
   1
   0
   2

>> N = [ 1 2 1; 0 1 3; 0 1 1 ]
N =
   1   2   1
   0   1   3
   0   1   1

>> N * y
ans =
   3
   6
   2
```

A multiplicação deve respeitar a dimensão dos vetores e matrizes envolvidos no cálculo. Por

exemplo, os cálculos a seguir falham devido à incompatibilidade de tamanho dos vetores:

```
>> x * y
error: operator *: nonconformant arguments (op1 is 1x5, op2 is 3x1)

>> y * M
error: operator *: nonconformant arguments (op1 is 3x1, op2 is 3x3)

>> M * x
error: operator *: nonconformant arguments (op1 is 3x3, op2 is 1x5)
```

Observe que o cálculo envolvendo a multiplicação de y à esquerda de M exige que y seja transposto:

```
>> y' * M
ans =
   22.00000    0.30000   19.00000
```

Além do produto usual entre vetores, é possível fazermos o produto termo-a-termo entre elementos. Esse produto retorna um terceiro vetor no qual cada elemento corresponde ao produto dos elementos nas mesmas posições dos outros vetores. Para essa operações, usamos o operador de produto precedido por um ponto (`.*`). Por exemplo, o comando `x' * z` executa o produto interno usual entre os vetores `x` e `z`, retornando o valor 12, como apresentado anteriormente. Já o comando `x .* z` retorna

```
>> x .* z
ans =
   5
   0
   8
   0
  -1
```

que é o vetor no qual cada elemento é igual ao produto entre os elementos na mesma posição em `x` e `z`, ou seja, a i-ésima posição é igual a $x_i \times z_i$. De forma semelhante, temos a divisão termo-a-termo, como no exemplo a seguir:

```
>> > z ./ x
```

```
ans =
   0.20000
   0.00000
   0.12500
   0.00000
  -1.00000
```

Outra operação relevante no contexto aqui considerado, consiste em definir subvetores e submatrizes a partir de vetores e matrizes já definidas. Por exemplo, para definirmos um subvetor de x, dado por seus elementos nas posições 2 a 4, fazemos:

```
>> x(2:4)
ans =
   2   8   3
```

Podemos definir um subvetor com apenas algumas posições selecionadas. Por exemplo, se quisermos só os elementos nas posições 1 e 5, podemos fazer:

```
>> x([1 5])
ans =
   5   1
```

É possível até mesmo definirmos em vetor com essas posições e, então, usá-lo para definir o subvetor desejado:

```
>> pos = [1 5]
pos =
   1   5


>> x(pos)
ans =
   5   1
```

Para definirmos uma submatriz a sintaxe é similar. No exemplo a seguir, selecionamos a primeira coluna da matriz M:

```
>> M(:,1)
ans =
    2.00000
```

```
    0.10000
   10.00000
```

Na primeira posição dentro dos parênteses, colocamos o sinal de *dois-pontos* para indicar que todas as linhas devem ser selecionadas. A segunda posição indica que apenas a coluna 1 deve ser selecionada. Se quisermos incluir apenas as duas primeiras linhas, podemos fazer:

```
>> M(1:2,1)
ans =
   2.00000
   0.10000
```

O exemplo a seguir mostra como selecionarmos a submatriz com os elementos nas linhas e colunas 2 e 3 da matriz M:

```
>> M(2:3,2:3)
ans =
   1.70000   8.00000
   0.00000   5.00000
```

3 Implementando os cálculos do método simplex

Os cálculos em cada iteração do método simplex envolvem várias operações com vetores e matrizes. Para auxiliar nesses cálculos, podemos utilizar o Octave/Matlab juntamente com os comandos apresentados acima.

Como dito anteriormente, o intuito é utilizar o software para auxiliar nos cálculos necessários para a execução manual do método, com objetivo didático. A implementação de um algoritmo que automatize a resolução de um problema de programação está fora do escopo deste tutorial.

A princípio, os comandos a seguir serão suficientes para nos auxiliar com os cálculos:

```
B = A(:,base)
xB = inv(B) * b
pt = c(base)' * inv(B)
s = c' - pt * A
y = inv(B) * A(:,k)
xB ./ y
```

Antes de iniciar a execução dos cálculos, precisamos definir os dados do problema que queremos resolver. Para facilitar a explicação, vamos utilizar o exemplo a seguir.

Uma metalúrgica produz dois tipos de ligas metálicas. Cada liga é composta de proporções diferentes de Cobre, Zinco e Chumbo, os quais estão disponíveis em quantidades limitadas em estoque. Deseja-se determinar quanto produzir de cada liga metálica, de modo a maximizar a receita bruta, satisfazendo-se as seguintes composições das ligas e a disponibilidade de matéria-prima em estoque:

Matéria-prima	Liga 1	Liga 2	Estoque
Cobre	50%	30%	3 ton
Zinco	10%	20%	1 ton
Chumbo	40%	50%	3 ton
Preço venda	3 mil	2 mil	(R$ por ton)

Um modelo de programação linear para esse exemplo pode ser definido da seguinte forma:

$$\begin{aligned} \max \quad & 3x_1 + 2x_2 \\ \text{s.a} \quad & 0{,}5x_1 + 0{,}3x_2 \leq 3 \\ & 0{,}1x_1 + 0{,}2x_2 \leq 1 \\ & 0{,}4x_1 + 0{,}5x_2 \leq 3 \\ & x_1 \geq 0, x_2 \geq 0. \end{aligned}$$

Após colocarmos o modelo na forma padrão, é possível representá-lo na forma matricial

$$\begin{aligned} \min \quad & f(x) = c^T x \\ \text{s.a} \quad & Ax = b \\ & x \geq 0 \end{aligned}$$

usando os seguintes vetores e matrizes de coeficientes:

$$c = \begin{bmatrix} -3 \\ -2 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \quad A = \begin{bmatrix} 0{,}5 & 0{,}3 & 1 & 0 & 0 \\ 0{,}1 & 0{,}2 & 0 & 1 & 0 \\ 0{,}4 & 0{,}5 & 0 & 0 & 1 \end{bmatrix}, \quad b = \begin{bmatrix} 3 \\ 1 \\ 3 \end{bmatrix}.$$

Para definirmos o problema no Octave/Matlab, entramos com os vetores e matrizes acima, usando os comandos:

```
>> c = [ -3; -2; 0; 0; 0]
```

```
c =
  -3
  -2
   0
   0
   0

>> A = [0.5 0.3 1 0 0; 0.1 0.2 0 1 0; 0.4 0.5 0 0 1]
A =
   0.50000   0.30000   1.00000   0.00000   0.00000
   0.10000   0.20000   0.00000   1.00000   0.00000
   0.40000   0.50000   0.00000   0.00000   1.00000

>> b = [3; 1; 3]
b =
   3
   1
   3
```

Feito isso, vamos definir agora uma base inicial para o problema. Como as colunas 3 a 5 da matriz A resultam em uma matriz identidade, temos uma base trivial dada pelas variáveis x_3, x_4 e x_5. Assim, definimos:

```
base = [ 3 4 5 ]
base =
   3   4   5
```

Podemos agora iniciar a primeira iteração do método simplex. O primeiro passo é definir a matriz básica B como uma submatriz de A:

```
>> B = A(:,base)
B =
   1   0   0
   0   1   0
   0   0   1
```

Apenas as colunas de A cujo índice esteja em `base` devem entrar em B (todas as linhas são permitidas). Em seguida, para calcularmos a solução básica $x_{\mathcal{B}} = B^{-1}b$, fazemos:

```
>> xB = inv(B) * b
xB =
   3
   1
   3
```

Agora vamos calcular o vetor multiplicador $p^t = c_{\mathcal{B}}^T B^{-1}$

```
>> pt = c(base)' * inv(B)
pt =
   0   0   0
```

Note que foi necessário usarmos o sinal de aspas simples para indicar que o vetor $c_{\mathcal{B}}$ deve ser transposto.

Os custos relativos são dados pela expressão $s_j = c_j - p^T a_j$, para todo $j \in \mathcal{N}$. Para facilitar, vamos calcular os custos relativos de *todas* as variáveis, i.e., para todo $j = 1, \ldots, n$. Assim, a expressão do custo relativo é dada por $s = c - p^T A$, que pode ser calculada no Octave/Matlab da seguinte forma:

```
>> s = c' - pt * A
s =
  -3  -2   0   0   0
```

Pelo resultado, podemos observar que a solução atual não é ótima, pois existem custos relativos negativos. O menor deles é dado pelo s_1 e, portanto, x_1 deve entrar na base. Para sinalizar isso, vamos armazenar o índice 1 na variável k, fazendo:

```
>> k = 1
```

Precisamos determinar agora qual variável deve sair da base atual. Para isso, devemos recorrer ao teste da razão, o qual utiliza o vetor $y = B^{-1} a_k$ que pode ser calculado como:

```
>> y = inv(B) * A(:,k)
y =
   0.50000
   0.10000
   0.40000
```

O teste da razão é então realizado usando os elementos positivos do vetor resultante do seguinte cálculo:

```
>> xB ./ y
ans =
    6.0000
   10.0000
    7.5000
```

O menor valor do resultado acima é 6, dado pela divisão $x_{\mathcal{B}_1}/y_1$, indicando que $_{\mathcal{B}_1} = x_3$ deve sair da base. Assim, para sinalizar a saída do primeiro elemento da base, fazemos

```
>> l = 1
```

Observação: Cuidado para não confundir a letra l com o número 1 ao digitar a expressão acima.

Por fim, tendo definido as variáveis que entram e saem da base, podemos realizar a troca de base usando o comando:

```
>> base(l) = k
base =
   1   4   5
```

Após esse comando, podemos iniciar uma nova iteração do método simplex com a base $\mathcal{B} = \{1, 4, 5\}$. Dessa vez, podemos entrar com os comandos de forma mais rápida, pois todos já estão salvos no Octave/Matlab. Basta pressionar a *seta-para-cima* no teclado, para visualizar os comandos digitados anteriormente. Você pode apertar a tecla várias vezes até encontrar o comando que deseja e então pressionar *Enter* para executá-lo. A tecla *seta-para-baixo* também funciona nesse caso. É sempre possível modificar um comando antes de executá-lo.

Os comandos usados nos cálculos da segunda iteração são os seguintes:

```
>> B = A(:,base)
B =
   0.50000   0.00000   0.00000
   0.10000   1.00000   0.00000
   0.40000   0.00000   1.00000

>> xB = inv(B) * b
xB =
   6.00000
   0.40000
   0.60000
```

```
>> pt = c(base)' * inv(B)
pt =
  -6   0   0

>> s = c' - pt * A
s =
   0.00000  -0.20000   6.00000   0.00000   0.00000
```

Temos ainda um custo relativo negativo e assim a base atual não pode ser ótima. Como temos $s_2 < 0$, então a variável x_2 deve entrar na base. Continuando com os cálculos da iteração, temos:

```
>> k = 2
k =  2

>> y = inv(B)*A(:,k)
y =
   0.60000
   0.14000
   0.26000

>> xB ./ y
ans =
   10.0000
    2.8571
    2.3077

>> l = 3
l =  3

>> base(l) = k
base =
   1   4   2
```

Com a base $\mathcal{B} = \{1, 4, 2\}$, vamos para a iteração 3 do método. Os comandos são os seguintes:

```
>> B = A(:,base)
B =
   0.50000   0.00000   0.30000
   0.10000   1.00000   0.20000
   0.40000   0.00000   0.50000

>> xB = inv(B) * b
xB =
   4.615385
   0.076923
   2.307692

>> pt = c(base)' * inv(B)
pt =
  -5.38462   0.00000  -0.76923

>> s = c' - pt * A
s =
   0.00000   0.00000   5.38462   0.00000   0.76923
```

Após o cálculo dos custos relativos, observamos que todos são maiores ou iguais a 0, indicando que a solução ótima foi encontrada. A solução ótima do problema deve ser obtida a partir do vetor `xB`. Como a base ótima é $\mathcal{B} = \{1, 4, 2\}$, temos que a solução ótima obtida é dada por $x_1 \approx 4.615$ e $x_2 \approx 2.308$. O valor ótimo pode ser calculado ao observamos que $f(x) = c^T x = c_{\mathcal{B}}^T c_{\mathcal{B}}$ e, assim:

```
>> c(base)'*xB
ans = -18.462
```

Portanto, considerando que ao colocarmos o problema na forma padrão tivemos de modificar o sinal da função objetivo, o valor ótimo do problema é 18,462.