MS428: Projeto Computacional

Implementação do método Simplex Primal

Nome: Akemi Hayashi RA: 163283
Nome: Marco Antônio RA: 221487
Nome: Jorge Reis RA: 237966
Nome: Nicolas Toledo de Camargo RA: 242524

Sumário

1 Introdução

O presente relatório tem por finalidade descrever o desenvolvimento e funcionamento do algoritmo do método primal simplex para resolução de um problema de programação linear na forma padrão, para isso descrevendo as entradas, funcionamento do algoritmo e saídas.

O projeto foi desenvolvido utilizando a linguagem *Python*.

2 Método Primal Simplex

2.1 Entradas

Como o escopo do projeto é resolver o problema linear na forma padrão, o programa já assume igualdade nas restrições, não negatividade, minimização da função objetivo. Assim, admitindo como entrada apenas o número de restrições m, número de variáveis n, vetor de custos c, vetor dos recursos b e matriz dos coeficientes das restrições A. Na nossa implementação, a função principal $'simplex()'$, aceitará entradas na forma: $'simplex(m, n, c, b, A)'$, em que os vetores c e b e a matriz A são listas. Ou seja, para:

$$m = 3;\;\; n = 4;\;\; A = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{pmatrix}\;;\;\; b = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}\;;\;\; c = \begin{pmatrix} 1 \\ 2 \\ 3 \\ 4 \end{pmatrix};$$

A função $'simplex()'$ deve ser chamada com:

$$simplex(3,\;\; 4,\;\; [1, 2, 3, 4],\;\; [1, 2, 3],\;\; [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12])$$

2.2 Algoritmo

Biblioteca usada: python

A função $'check\_fun()'$ é a mais simples do programa. Apenas calcula o valor da função objetivo da solução básica. Chamada por $'particao()'$, para checar, no caso de problema artificial, se o problema linear original é factível. E chamada por $'simplex()'$ para calcular o valor da função para a solução ótima.

A função $'custos\_rel()'$ é a função que realiza a iteração simplex propriamente dita. Ela realiza os cálculos para: solução básica, vetor multiplicador simplex, custos relativos, escolha da variável para entrar, teste de otimalidade, direção simplex, cálculo do passo e variável a sair, e a atualização da base. Esta função é chamada pela função $'particao()'$, no caso em que o problema linear precisa do problema artificial. E também é chamado pela função principal $'simplex()'$ para resolver o problema linear original dada a partição factível. Note que ela também indica se o problema não possui solução limitada ao calcular o vetor de direção simplex se todas as componentes forem menores ou iguais a zero.

A função $'prob\_artf()'$ configura o problema artificial da primeira fase. Ela é chamada por $'particao()'$ e recebe informações sobre o problema linear original e configura variáveis artificiais necessárias e a função objetivo artificial.

A função $'particao()'$ procura uma partição identidade para base na matriz A, vendo se cada coluna é eletiva (se possui apenas um valor 1 e o resto de zeros) e , se for, guarda a posição dessa coluna e da linha do 1 – iterando por toda a matriz. Com essas colunas eletivas, é feita uma matriz. O processo continua até que seja possível formar uma identidade de tamanho m ou até acabar as colunas de A. As colunas que não forem selecionadas são guardadas em uma matriz como a partição não básica, com seus índices referentes a A inicial.

- Se o número de colunas selecionadas for m, então essa pode ter forma de uma identidade e é uma base factível para iniciar o problema. A função então retorna o PL com as partições configuradas para o início da segunda fase.

- Se o número de colunas selecionadas não for m, recorremos ao problema de duas fases. A função $'prob\_artf()'$ é chamada, que retorna a configuração do problema artificial.

  O problema artificial fica na forma:

$$\text{minimizar} \quad f(x, y) = \sum_{i=1}^{m} y_i$$

$$\text{sujeito} \quad \text{a:} \quad Ax + y = b \quad x \geq 0, y \geq 0$$

- Agora $'particao()'$ itera sobre o problema artificial, chamando $'custos\_rel()'$ até que uma solução ótima seja encontrada.
- Uma vez encontrada, é chamada $'check\_fun()'$ para o cálculo do valor da função objetivo na solução ótima.
  - Se este valor for diferente de zero, o PL original é infactível.
  - Se for igual a zero, o PL original é factível e, especificamente, a partição básica ótima do problema artificial é uma partição básica factível pro PL original.

No primeiro caso a função retorna para $'simplex()'$ uma variável indicadora de que o PL é infactível. No segundo caso, achamos uma partição básica factível para o problema original e, então, os índices artificiais e suas colunas da matriz não básica são removidos. O custo do problema inicial é então atualizado e particionado referente às colunas que ficaram na base e não base do problema artificial. A função retorna o PL com a partição básica factível encontrada pela função $'simplex()'$, que então prossegue com a segunda fase.

A função $'simplex()'$ é a principal. Primeiro, ela modifica as entradas na matriz dos coeficientes das restrições, vetor de custo e vetor de recursos (usando os parâmetros m e n), de forma que os dados possam ser manipulados por funções do *Numpy*. A função $'particao()'$ então é chamada para iniciar a primeira fase e definir se o problema é factível; se não for, retorna "*Não há solução factível*", mas, se for, é introduzido um indicador que enquanto não se alterar, chama a função $'custos\_rel()'$ até se chegar em uma solução ótima finita ou ilimitada. Se indicar solução ilimitada, retorna "*Problema não tem solução ótima finita*"; se houver solução ótima limitada, é utilizado o $'check\_fun()'$ para ver o valor da função objetivo e retorna esse valor com os valor das variáveis na solução ótima.

2.3 Saídas

O algoritmo vai dizer se o problema é factível e/ou limitado. Se o problema factível e limitado, o algoritmo encontrará a solução ótima e retornará as informações chave: o vetor das variáveis de decisão ótima e o valor ótimo da função objetivo. No fim do código há alguns exemplos para teste retirados das listas de exercícios.

2.4 Exemplos

Exemplos:

```
#%%problema degenerado sem vr artificial
c=[-1,-1,0,0,0]
m,n=3,5
A=[1,1,1,0,0,2,1,0,1,0,1,2,0,0,1]
b=[10,15,15]
#%%problema sem vr artificial

c=[-3,-5,0,0,0]
A=[1,0,1,0,0,0,1,0,1,0,3,2,0,0,1]
b=[6,6,18]
m,n=3,5

#%%problema com vr artificial
c=[-1,2,0,0]
A=[1,1,-1,0,-1,1,0,-1]
b=[2,1]
m,n=2,4
#%%problema infactivel
c=[3,4,0,0]
A=[1,1,1,0,2,1,0,-1]
b=[1,4]
m,n=2,4
#%%problema ilimitado
c=[-1,-1,0,0]
A=[1,-1,1,0,-1,1,0,1]
b=[4,4]
m,n=2,4
#%%
c=[1,1,0,0,0,0,0]
A=[3,2,-1,0,0,0,0,1,2,0,-1,0,0,0,1,0,0,0,-1,0,0,1,0,0,0,0,1,0,0,1,0,0,0,0,1]
b=[24,12,2,15,15]
m,n=5,7
```

Saída do exemplo 1:

```
Solução ótima:
 [[ 5.00000000e+00]
 [ 5.00000000e+00]
 [ 0.00000000e+00]
 [ 0.00000000e+00]
 [-4.16333634e-16]]
Com função objetivo valendo: -10.0
```

Saída do exemplo 2:

```
Solução ótima:
 [[2.]
 [6.]
 [4.]
 [0.]
 [0.]]
Com função objetivo valendo: -36.0
```

Saída do exemplo 3:

```
Solução ótima:
 [[0.5]
 [1.5]
 [0. ]
 [0. ]]
Com função objetivo valendo: 2.5
```

Saída do exemplo 4:

```
Não há solução factível
```

Saída do exemplo 5:

```
Problema não tem solucao ótima finita.
```

Saída do exemplo 6:

```
Solução ótima:
 [[ 6.]
 [ 3.]
 [ 0.]
 [ 0.]
 [ 4.]
 [ 9.]
 [12.]]
Com função objetivo valendo: 9.0
```