Otimização Lista 4

Pedro Thiago [1]

11 de Junho de 2020

[1] Matrícula: 1612702

Declaração:

Declaro para devidos fins que Eu, Pedro Thiago de Souza M. Marmello fiz todos os exercícios da lista sem auxílio de outros alunos. Apenas consultando os livros, os slides, notas de aula e as aulas gravadas.

Questão 7

a-)

Dado o problema da produção visto em aula, podemos criar as matrizes A,C e b. Assim como definir o β e η para que haja uma condição inicial. Desta forma, o que segue no programa é a difinição da matriz identidade para A e C, tais que condizem com o número de linhas de A e o número de colunas de C, respectivamente, respeitando seus tamanhos. Ou seja, a matriz identidade de C é apenas uma linha de zeros, enquanto para A é uma matriz Identidade quadrada e nxn, onde n é o número de linhas de A. Desta forma, podemos estabelecer novos A e C que terão suas dimensões ajustadas para os passos realizados no simplex.

Para esta questão, a resposta fora separada em 2 funções. A primeira, "linear programming"recebe β, η, A,C,b, chama a segunda função e espera seu retorno. Com o retorno da segunda função, são criadas e estabelecidas as devidas respostas (d,X, custo reduzido, Z, $\beta*$, $\eta*$) para retornar a resposta ao usuário.

Já na segunda função, assim chamada de Simplex FaseII, é a função principal a qual desenvolve o papel de realizar o simplex revisado. Recebendo o novo A, novo C, b, β e η.

Os primeiros passos são de inicialização do que será utilizado - já que a mesma função será utilizada no desenvolvimento da Questão 8; seguindo para a construção do vetor Vzero, o qual será utilizado como vetor de comparação para o vetor Custo Reduzido. Dessa forma, podemos criar um "while"que nos permitirá realizar algumas manipulações de B, β e η sabendo que quando o custo reduzido for maior que zero, teremos nosso ótimo.

Dentro do While, o primeiro passo é encontrar o menor i que será utilizado para a troca de linhas e colunas. Além de definirmos o vetor y. Seguindo para o segundo passo, temos o j máximo a ser utilizado na troca de linhas e colunas segundo o método Simplex Revisado.

Após algumas manipulações de β e η, podemos enfim manipular as Matrízes B e y visando as alterações necessárias para o desenvolvimento do Simplex. E por fim, verificamos novamente o custo reduzido; caso não entre mais no loop, podemos seguir para o retorno elaborado da função objetivo, β e η.

b-)

Questão 8

a-)

Para esta questão 3 funções separadas foram criadas. 1 é a função principal, Simplex Programming, a qual tem como principal objetivo definir e verificar as matrízes afim de criterizar se deverá prosseguir para o método Simplex Fase I ou FaseII. Retorna caso haja erros de Matrízes, ou Retornará o resultado final (seja ótimo ou não).

Já a 2ª função, é caracterizada pelo seu objetivo no desempenho da FaseI do Simplex revisado; a qual deve verificar se o problema é viável ou inviável, além de encontrar e definir as condições iniciais que serão colocadas na chamada da função que desempenha o papel da FaseII. Retorna "Inviável"caso não seja possível solucionar, ou retorna a resposta do problema.

A 3ª função como fora explicada na Questão 7, é a função que desempenha o desenvolvimento do Simplex Fase II, retornando assim o ótimo que fora encontrado.

b-)

c-)

Imagens

```
using JuMP,Plots, GLPK, LinearAlgebra

A = [2 1;1 2]
C = [4 3]
b = [4;4]

β=[3;4]
η=[1; 2]    > Vector{Int64} with 2 elements

## simplex_FaseII e todo seu desenvolvimento pelo algoritmo do Simplex Revisado

function simplex_FaseII(β_init , η_init , A , C , b)

    ## Inicializa o que será utilizado
    B = A[:,β_init]
    N = A[:,η_init]
    cB = C[:,β_init]
    cN = C[:,η_init]
    XB = B\b
    b_barra = B\b

    w = cB*inv(B)
    C_reduzido = w*N - cN

    V_zero = zeros(size(C_reduzido,2))

    β=β_init
    η=η_init
    ## Loop que verifica o Custo reduzido.
    ## Parte principal do Programa
    new_count =0
    while all(C_reduzido .> V_zero)
        ## encontra o menor valor para i (que neste caso foi nomeado k)
        for i in 1:size(N,1)
            a = N[:,i]
            p = w*a-cN[:,i]
            if i == 1
                global save = p
                global k = i
            elseif p < save
                global save = p
```

Figura 1: Simplex FaseII Pt1

```
            global save = p
            global k = i
        end
    end
    global save = 0
    ## N[:,k] é o que entra na base
    y =inv(B)*N[:,k]

    ## encontra o maior valor para j
    for i in 1:size(y,1)
    end

    ## realiza as devidas mudanças nos índices

    aux = η[k]
    η[k] = size(A,1) + l
    β[l] = aux

    MatrizB = [inv(B) y]
    sizeMB = size(MatrizB,1)

    ultima_col = MatrizB[:,size(MatrizB,2)]
    escolhido = ultima_col[l]

    ## realiza as mudanças na matriz [B^-1 y]
    if escolhido != 0
        escolhido = MatrizB[l,:]/escolhido
    end

    for d in 1:sizeMB
        ## sim aqui dava pra ter melhorado o código
        if d != l
            if (ultima_col[d] != 0)
                norma = (- ultima_col[d])
                MatrizB[d,:] = MatrizB[d,:] + norma*MatrizB[l,:]
                b_barra[d] = b_barra[d] + norma*b_barra[l]
            end
        end
    end

    ## Reestabelece as matrizes que foram utilizadas no passo anterior
    v = MatrizB[:,size(MatrizB,2)]
```

Figura 2: Simplex FaseII Pt2

```julia
        ## Reestabelece as matrizes que foram utilizadas no passo anterior
        y = MatrizB[:,size(MatrizB,2)]
        new_β= zeros(Int8,size(MatrizB,2)-1)
        for i in 1:size(MatrizB,2)-1
            new_β[i] = i
        end
        B = MatrizB[:,new_β]
        cB = C[:,β]
        cN = C[:,η]

        ## contador para o número de vezes que o loop acontece
        ## Etapa final:
        w = cB*inv(B)
        C_reduzido = w*N - cN

        if C_reduzido .> V_zero
            new_count+=1
        end

        # Devido a troca do auxiliar no passo de troca de índices, teremos essa mudança aqui também.
        β_init = η
        η_init = β
    end # while

    println("Ótimo Encontrado!")
    Xβ= B\b_barra
    Xη=zeros(size(A,2)-size(Xβ,1))
    X=[Xβ;Xη]
    Z = C*X

    return Z, η_init, β_init, new_count
end    > simplex_FaseII
```

Figura 3: Simplex FaseII Pt3

```julia
function linear_programming(β, η, A, C, b)


    ## Inicializa as matrízes que serão enviadas ao simplex_FaseII

    Id = diagm(0=>fill(1., size(A,1)))
    A = [A Id]
    Id_C = transpose(fill(0,size(Id,1)))
    C = [C Id_C]

    Z, β, η, count = simplex_FaseII(β, η, A, C, b)

    # Constroi as matrízes utilizando os dados retornados do simplex
    B = A[:,β]
    BT = transpose(B)
    N = A[:,η]
    NT = transpose(N)
    cB = C[:,β]
    cN = C[:,η]
    cTB = transpose(cB)
    cTN = transpose(cN)
    CT = transpose(C)

    X = [1:size(A,1)]
    w = cB*inv(B)
    C_reduzido = w*N - cN

    ## encontra o j máximo para construir dB e constroi dN
    j=findmax(C_reduzido)[2][2]
    dB= -inv(B)*N[:,j]
    dN= -inv(N)*B*dB

    ## Cria as entregas d, X, Z
    d = [dB;dN]

    Xβ=B\b
    Xη=zeros(size(A,2)-size(Xβ,1))
    X=[Xβ;Xη]

    return d, X, β, η, Z, C_reduzido, new_count
end    > linear_programming
```

Figura 4: Q7 -Inicio e Simplex FaseII

```julia
using CSV,LinearAlgebra,GLPK,Plots



A = [2 1;1 2]
C = [4 3]
b = [4;4]

## Matrizes das Restrições
R = [-1 -1]
bR = [-1]    > Vector{Int64} with 1 element
##

    ## Função principal que organiza a ida para as fases I e II
function simplex_Programming(A , C, b, R, bR)
end


## Início Fase I
function simplex_FaseI(A,C,b)
end

## Início Fase II
function simplex_FaseII(β_init , η_init , A , C , b)

end
##

        #Teste
Z, β, η, count = simplex_Programming(A, C, b, R, bR)    (> Vector{Float64} with 1 element, > Vector{Int8} with 2 elements, > Vector{Int8} with 2 elements, 0)
```

Figura 5: Q8 - Resumo

```
    ## Função principal que organiza a ida para as fases I e II
function simplex_Programming(A , C, b, R, bR)

    ## Análise::
    V_zero = zeros(size(b))
    n = size(A,2)
    m = size(A,1)


    ## Verifica se b >=0, dessa forma, sendo direcionado para fase II
    if (b >= V_zero)
        η_init = zeros(Int8,n)
        β_init = zeros(Int8,m)

        for i in 1:n
            η_init[i] = Int(i)
        end
        for i in 1:m
            β_init[i] = Int(n+i)
        end
        ## União das Matrízes e Preparação para as Fases::

        V_zero = zeros(size(R,2))

        if R != transpose(V_zero)
            ## Tamanho das Linhas e colunas de A
            i = size(A,1)
            j = size(A,2)
            ##

            ## Número de Colunas de b (=1)
            jb = size(b,2)

            ## Tamanho das colunas de R tem que ser igual a de A
            jR = size(R,2)
            ## Tamanho das colunas do b das restrições(bR) tem que ser igual a de b
            jbR = size(bR,2)
            ##

            ## Verifica se há algum erro de tamanho de matrízes
            if( j != jR || jb != jbR)
```

Figura 6: Q8 - SimplexProgramming Pt1

```
        for i in 1:n
            η_init[i] = Int(i)
        end
        for i in 1:m
            β_init[i] = Int(n+i)
        end
        ## União das Matrízes e Preparação para as Fases::

        V_zero = zeros(size(R,2))

        if R != transpose(V_zero)
            ## Tamanho das Linhas e colunas de A
            i = size(A,1)
            j = size(A,2)
            ##

            ## Número de Colunas de b (=1)
            jb = size(b,2)

            ## Tamanho das colunas de R tem que ser igual a de A
            jR = size(R,2)
            ## Tamanho das colunas do b das restrições(bR) tem que ser igual a de b
            jbR = size(bR,2)
            ##

            ## Verifica se há algum erro de tamanho de matrízes
            if( j != jR || jb != jbR)
                A = [A ; R]
                b = [b ; bR]
                return "ERROR! Matrix Mismatch"
            end
        end

        return simplex_FaseII(β_init, η_init, A, C, b)
    end

    ## Caso vá para a FaseI
    return simplex_FaseI(A,C,b)
end
```

Figura 7: Q8 - SimplexProgramming P2

```julia
## Início Fase I
function simplex_FaseI(A,C,b)

    ## n é o número de colunas, m é o número de linhas de A_barra
    n=size(A,2)
    m=size(A,1)

    # Inicializa a matriz e (que multiplica w) e a nova matriz A
    ϵ = fill(-1.,m)
    Ae = [A ϵ]

    #Inicializa o eta e beta que marcarão as posições
    η_init = zeros(n+1)
    β_init = zeros(m)

    for i in 1:n
        η_init[i] = i
    end

            ## Essa parte não precisa, já que vai substituir de qualquer forma
            η_init[n+1] = n+m+1

    for i in 1:m
        β_init[i] = n+i
    end

    # θ=zeros(length(b))
    for i in 2:length(b)
        if b[i]< b[i-1]
            global θ=i
        else
            global θ=i-1
        end
    end

    aux = β_init[θ]
    β_init[θ] = η_init[n+1]
    η_init[n+1] = aux


    w = zeros(n+m+1)
```

Figura 8: Q8 - SimplexFaseI Pt1

```
    β_init = zeros(m)

    for i in 1:n
        η_init[i] = i
    end

            ## Essa parte não precisa, já que vai substituir de qualquer forma
            η_init[n+1] = n+m+1

    for i in 1:m
        β_init[i] = n+i
    end

    # θ=zeros(length(b))
    for i in 2:length(b)
        if b[i]< b[i-1]
            global θ=i
        else
            global θ=i-1
        end
    end

    aux = β_init[θ]
    β_init[θ] = η_init[n+1]
    η_init[n+1] = aux


    w = zeros(n+m+1)
    w[n+m+1]= -1.

    ### Call fase II para w

    Z_faseI, β_faseI, η_faseI = simplex_FaseII( β_init, η_init, Ae, -w, b )
    tol = 10^-5
    # Em algum momento isso aqui deve ser feito:
    if (Z_faseI == 0 || Z_faseI > -tol)
        return simplex_FaseII( β_faseI, η_faseI, A, C, b)
    else
        return "Inviável"
    end
end
```

Figura 9: Q8 - SimplexFaseI Pt2