ENG1467 - OTIMIZAÇÃO - PUC-Rio, 2022.1

Lista 4: Simplex Parte 1

13/05/2022

Professor: Alexandre Street *Aluno: Thiago Novaes*

O código completo desenvolvido durante essa atividade pode ser encontrado em `https://github.com/Thiago-NovaesB/MestradoPuc.jl/tree/main/Programa%C3%A7%C3%A3o%20Linear/Lista%204/Simplex`

1 Motivação

O algoritmo Simplex fase 2 desenvolvido na ultima lista, requer um vértice viável inicial. Em alguns casos, como no problema da produção apresentando, era trivial achar um vértice, bastava usar as slacks como base. Porém isso é apenas um caso particular da classe dos LPs, sendo assim, o simplex fase 1, visa encontrar um vértice viável (quando ele existe). Para isso, o poliedro será inchado até conter a origem.

2 Implementação

Para a implementação do algoritmo, foi criado o pacote *Simplex.jl*:

```
module Simplex

include("types.jl")
include("utils.jl")
include("log.jl")
include("solver.jl")

export create, solve

end # module
```

O arquivo *types.jl* contém estruturas em julia para guardar as entradas e saídas:

```
mutable struct Input
    A::Matrix{}
    b::Vector{}
    c::Vector{}
    n::Int
    m::Int
    tol::Float64
    max_iter::Int
    verbose::Bool
end
```

```
11
12 Base.@kwdef mutable struct MidTerm
13     termination_status::Int = 0
14     iter::Int = 0
15     d::Vector{} = []
16     base::Vector{Int} = []
17     nbase::Vector{Int} = []
18     i::Int = 0
19     j::Int = 0
20     z::Float64 = 0.0
21     x::Vector{Float64} = []
22     red_cost::Vector{Float64} = []
23 end
24
25 struct Output
26     x::Vector{Float64}
27     z::Float64
28     termination_status::Int
29     base::Vector{Int}
30     nbase::Vector{Int}
31 end
```

O arquivo *utils.jl* contém funções para tratar entradas e saídas:

```
1  function create(A::Matrix{}, b::Vector{}, c::Vector{}
2                  ; tol::Float64 = 1E-6, max_iter::Int = 1000,
3                  verbose::Bool = true)
4          n = length(c)
5          m = length(b)
6          input = Simplex.Input(A,b,c,n,m,tol,max_iter,verbose)
7      return input
8  end
9
10 function update_midterm!(input::Input, midterm::MidTerm)
11     A = input.A
12     b = input.b
13     c = input.c
14     base = midterm.base
15     B = view(A,:,base)
16     x = B \ b
17     midterm.z = c[base]'x
18
19     x_opt = zeros(input.n)
20     x_opt[base] = x
21
22     midterm.x = x_opt
23     nothing
24 end
25
26 function write_output(input::Input, midterm::MidTerm)
27
28     if midterm.termination_status == 2
```

```
        output = Simplex.Output(midterm.d, Inf, midterm.termination_status, midterm.base, midterm.nbase)
        last_log(input, output)
    elseif midterm.termination_status == 1
        output = Simplex.Output(midterm.x, midterm.z, midterm.termination_status, midterm.base, midterm.nbase)
        last_log(input, output)
    elseif midterm.termination_status == 3
        output = Simplex.Output(midterm.x, midterm.z, midterm.termination_status, midterm.base, midterm.nbase)
        last_log(input, output)
    end
    return output
end
```

O arquivo *log.jl* contém as funções para escrita de log:

```
function init_log1(input::Simplex.Input)
    if input.verbose
        var = length(input.c)
        con = length(input.b)
        println("---------------Inicio do algoritmo Simplex 1----------------")
        println("O problema possui $var variaveis e $con restrições")
    end
end

function init_log2(input::Simplex.Input)
    if input.verbose
        var = length(input.c)
        con = length(input.b)
        println("---------------Inicio do algoritmo Simplex 2----------------")
        println("O problema possui $var variaveis e $con restrições")
    end
end

function iteration_log(input::Simplex.Input,midterm::Simplex.MidTerm)
    if input.verbose

        println("------------------------Iteração $(midterm.iter) ------------------------")
        println("Base: $(midterm.base)")
        println("Não-Base: $(midterm.nbase)")
        println("Deixa a base: $(midterm.base[midterm.i])")
        println("Entra na base: $(midterm.nbase[midterm.j])")
        println("Função objetivo: $(midterm.z)")
        println("Variaveis: $(midterm.x)")
        println("Custo reduzido: $(midterm.red_cost)")
    end
end

```

```
function last_log(input::Simplex.Input, output::Simplex.Output)
    if input.verbose
        println("----------------------Fim do algoritmo----------------------")
    )
        if output.termination_status == 1
            println("Status: Optimal")
            println("Base: $(output.base)")
            println("Não-Base: $(output.nbase)")
            println("Função objetivo: $(output.z)")
            println("Variaveis: $(output.x)")
        elseif output.termination_status == 2
            println("Status: Unbound")
            println("Base: $(output.base)")
            println("Não-Base: $(output.nbase)")
            println("Função objetivo: Inf")
            println("Direção extrema: $(output.x)")
        elseif output.termination_status == 3
            println("Status: Infeasible")
        end
    end
end
```

O arquivo *solver.jl* contém o algortimo simplex:

```
function solve(input::Simplex.Input)

    midterm = MidTerm()
    if check_phase_1(input)
        midterm.base =  collect((input.n - input.m + 1):input.n)
        midterm.nbase = collect(1:(input.n-input.m))
        val, i = findmin(input.b)

        aux = midterm.base[i]
        push!(midterm.nbase, aux)
        midterm.base[i] = input.n + 1

        c_mem = copy(input.c)
        input.A = hcat(input.A,-ones(input.m))
        input.c = zeros(input.n + 1)
        input.c[end] = -1
        input.n = input.n + 1

        init_log1(input)
        while midterm.termination_status == 0 && midterm.iter < input.max_iter
            midterm = Simplex.iterate(input, midterm)
        end
        update_midterm!(input, midterm)
        if midterm.z < -input.tol
            midterm.termination_status = 3 #infeasible
            output = write_output(input, midterm)
            return output
        end
```

```
        midterm.iter = 0
        input.n = input.n - 1
        deleteat!(midterm.nbase, findall(x->x==input.n + 1,midterm.nbase))
        input.c = c_mem
        input.A = input.A[:,1:input.n]
        midterm.termination_status = 0
    else
        midterm.base =  collect((input.n - input.m + 1):input.n)
        midterm.nbase = collect(1:(input.n-input.m))
    end

    init_log2(input)
    while midterm.termination_status == 0 && midterm.iter < input.max_iter
        midterm = Simplex.iterate(input, midterm)
    end
    update_midterm!(input, midterm)
    output = write_output(input, midterm)
    return output
end

function check_phase_1(input::Simplex.Input)
    b_min = minimum(input.b)
    return b_min < 0
end

function iterate(input::Simplex.Input, midterm::Simplex.MidTerm)

    midterm.iter += 1
    A = input.A
    b = input.b
    c = input.c
    base = midterm.base
    nbase = midterm.nbase
    tol = input.tol
    B = view(A,:,base)
    N = view(A,:,nbase)
    xB = B \ b
    y = B' \ c[base]
    midterm.red_cost = c[nbase] - N'*y
    val, midterm.j = findmax(midterm.red_cost)
    if val <= tol
        midterm.termination_status = 1
        return midterm #optimal
    end
    d = zeros(length(c))
    d_base = B \ N[:,midterm.j]
    d[base] = - d_base
    d[nbase[midterm.j]] = 1
    midterm.d = d
    d_base = max.(d_base, 0)
    r = xB ./ d_base
```

```
    val, midterm.i = findmin(r)
    if val == Inf
        midterm.termination_status = 2
        return midterm #unbounded
    end
    midterm.z = c[base]'xB
    midterm.x = zeros(input.n)
    midterm.x[base] = xB
    iteration_log(input, midterm)
    base[midterm.i], nbase[midterm.j] = nbase[midterm.j], base[midterm.i]
    return midterm #max iteration
end
```

3 Testes

3.1 Caso com solução ótima sem fase 1

Este caso foi mostrado em aula como sendo um exemplo pequeno de um problema de maximização com solução.

```
A = [2 1 1 0; 1 2 0 1]
b = [4, 4]
c = [4, 3, 0, 0]
input = Simplex.create(A, b, c)
output = Simplex.solve(input)
```

3.2 Caso com solução ótima com fase 1

Este caso foi mostrado em aula como sendo um exemplo pequeno de um problema de maximização sem um vértice trivial.

```
A = [2 1 1 0 0; 1 2 0 1 0; -1 -1 0 0 1]
b = [4, 4, -1]
c = [4, 3, 0, 0, 0]
input = Simplex.create(A, b, c)
output = Simplex.solve(input)
```

3.3 Caso inviável

Este caso foi mostrado em aula como sendo um exemplo pequeno de um problema de maximização inviável.

```
A = [2 1 1 0 0; 1 2 0 1 0; -1 -1 0 0 1]
b = [4, 4, -5]
c = [4, 3, 0, 0, 0]
input = Simplex.create(A, b, c)
output = Simplex.solve(input)
```

4 Resultados

4.1 Caso com solução ótima sem fase 1

```
---------------Inicio do algoritmo Simplex 2---------------
O problema possui 4 variaveis e 2 restrições
------------------------Iteração 1------------------------
Base: [3, 4]
Não-Base: [1, 2]
Deixa a base: 3
Entra na base: 1
Função objetivo: 0.0
Variaveis: [0.0, 0.0, 4.0, 4.0]
Custo reduzido: [4.0, 3.0]
------------------------Iteração 2------------------------
Base: [1, 4]
Não-Base: [3, 2]
Deixa a base: 4
Entra na base: 2
Função objetivo: 8.0
Variaveis: [2.0, 0.0, 0.0, 2.0]
Custo reduzido: [-2.0, 1.0]
---------------------Fim do algoritmo---------------------
Status: Optimal
Base: [1, 2]
Não-Base: [3, 4]
Função objetivo: 9.333333333333334
Variaveis: [1.3333333333333335, 1.3333333333333333, 0.0, 0.0]
```

4.2 Caso com solução ótima com fase 1

```
---------------Inicio do algoritmo Simplex 1---------------
O problema possui 6 variaveis e 3 restrições
------------------------Iteração 1------------------------
Base: [3, 4, 6]
Não-Base: [1, 2, 5]
Deixa a base: 6
Entra na base: 1
Função objetivo: -1.0
Variaveis: [0.0, 0.0, 5.0, 5.0, 0.0, 1.0]
Custo reduzido: [1.0, 1.0, -1.0]
---------------Inicio do algoritmo Simplex 2---------------
O problema possui 5 variaveis e 3 restrições
------------------------Iteração 1------------------------
Base: [3, 4, 1]
Não-Base: [2, 5]
Deixa a base: 3
Entra na base: 5
Função objetivo: 4.0
```

```
Variaveis: [1.0, 0.0, 2.0, 3.0, 0.0]
Custo reduzido: [-1.0, 4.0]
----------------------Iteração 2----------------------
Base: [5, 4, 1]
Não-Base: [2, 3]
Deixa a base: 4
Entra na base: 2
Função objetivo: 8.0
Variaveis: [2.0, 0.0, 0.0, 2.0, 1.0]
Custo reduzido: [1.0, -2.0]
---------------------Fim do algoritmo---------------------
Status: Optimal
Base: [5, 2, 1]
Não-Base: [4, 3]
Função objetivo: 9.333333333333332
Variaveis: [1.3333333333333333, 1.3333333333333335, 0.0, 0.0,
    1.6666666666666667]
```

4.3 Caso inviável

```
--------------Inicio do algoritmo Simplex 1--------------
O problema possui 6 variaveis e 3 restrições
----------------------Iteração 1----------------------
Base: [3, 4, 6]
Não-Base: [1, 2, 5]
Deixa a base: 3
Entra na base: 1
Função objetivo: -5.0
Variaveis: [0.0, 0.0, 9.0, 9.0, 0.0, 5.0]
Custo reduzido: [1.0, 1.0, -1.0]
----------------------Iteração 2----------------------
Base: [1, 4, 6]
Não-Base: [3, 2, 5]
Deixa a base: 4
Entra na base: 2
Função objetivo: -2.0
Variaveis: [3.0, 0.0, 0.0, 3.0, 0.0, 2.0]
Custo reduzido: [-0.3333333333333333, 0.3333333333333333, -0.6666666666666666]
---------------------Fim do algoritmo---------------------
Status: Infeasible
```