HOMEWORK 6

Mathieu Gaillard

February 24, 2019

Problem 0: Homework checklist

I mainly used the class material, Google, Wikipedia and my previous knowledge.

Problem 1

Using the codes from class (or your own implementations in another language) illustrate the behavior of the simplex method on the LP from problem 13.9 in Nocedal and Wright:

$$\begin{aligned} \text{minimize} \quad & -5x_1 - x_2 \\ \text{subject to} \quad & x_1 + x_2 \leq 5 \\ & 2x_1 + (1/2)x_2 \leq 8 \\ & \mathbf{x} \geq 0 \end{aligned}$$

starting at $[0, 0]^T$ after converting the problem to standard form.

Use your judgement in reporting the behavior of the method.

The standard form is:

$$\begin{aligned} \text{minimize} \quad & -5x_1 - x_2 \\ \text{subject to} \quad & x_1 + x_2 + s_1 = 5 \\ & 2x_1 + (1/2)x_2 + s_2 = 8 \\ & \mathbf{x}, \mathbf{s} \geq 0 \end{aligned}$$

This problem is easily solved by the Simplex algorithm. See the Julia code in the following listing. Starting at $[0, 0]^T$ the algorithm moves to the vertex on the right, which is the optimum. A region plot is shown in Figure 1. The BFP contains columns 1 and 3 and the value of $\mathbf{x}$ at the optimum is $[4.0, 0.0, 1.0, 0.0]^T$.

```
using LinearAlgebra
using Plots
plotly(size = (800, 600));

include("plotregion.jl");

struct SimplexState
    c::Vector
    A::Matrix
    b::Vector
    bset::Vector{Int} # columns of the BFP
end

struct SimplexPoint
    x::Vector
    binds::Vector{Int}
    ninds::Vector{Int}
    lam::Vector # equality Lagrange mults
    sn::Vector  # non-basis Lagrange mults
    B::Matrix   # the set of basic cols
    N::Matrix   # the set of non-basic cols
end

# These are constructors for SimplexPoint
function SimplexPoint(T::Type)
```

```
    return SimplexPoint(zeros(T,0),zeros(Int,0),zeros(Int,0),
        zeros(T,0), zeros(T,0), zeros(T,0), zeros(T,0))
end

function SimplexPoint(T::Type, B::Matrix, N::Matrix)
    return SimplexPoint(zeros(T,0),zeros(Int,0),zeros(Int,0),
        zeros(T,0), zeros(T,0), B, N)
end

function simplex_point(s::SimplexState)
    m,n = size(state.A)
    @assert length(state.bset) == m "need more indices to define a BFP"
    binds = state.bset # basic variable indices
    ninds = setdiff(1:size(state.A,2),binds) # non-basic
    B = state.A[:,binds]
    N = state.A[:,ninds]
    cb = state.c[binds]
    cn = state.c[ninds]
    c = state.c

    # @show cn

    if rank(B) != m
        return (:Infeasible, SimplexPoint(eltype(c), B, N))
    end

    xb = B\state.b
    x = zeros(eltype(xb),n)
    x[binds] = xb
    x[ninds] = zeros(eltype(xb),length(ninds))

    lam = B'\cb
    sn = cn - N'*lam

    # @show sn

    if any(xb .< 0)
        return (:Infeasible, SimplexPoint(x, binds, ninds, lam, sn, B, N))
    else
        if all(sn .>= 0)
            return (:Solution, SimplexPoint(x, binds, ninds, lam, sn, B, N))
        else
            return (:Feasible, SimplexPoint(x, binds, ninds, lam, sn, B, N))
        end
    end
end

function simplex_step!(state::SimplexState)
    # get the current point from the new basis
    stat,p::SimplexPoint = simplex_point(state)

    if stat == :Solution
        return (stat, p)
    elseif state == :Infeasible
        return (:Breakdown, p)
    else # we have a BFP
        #= This is the Simplex Step! =#

        # take the Dantzig index to add to basic
        qn = findmin(p.sn)[2]
        q = p.ninds[qn] # translate index
        # check that nothing went wrong
        @assert all(state.A[:,q] == p.N[:,qn])

        d = p.B \ state.A[:,q]
        #@show d

        # TODO, implement an anti-cycling method /
        # check for stagnation and lack of progress
        # this checks for unbounded solutions
        if all(d .<= eps(eltype(d)))
            return (:Degenerate, p)
        end

        # determine the index to remove
        xq = p.x[p.binds]./d
        ninds = d .< eps(eltype(xq))
        xq[d .< eps(eltype(xq))] .= Inf
        pb = findmin(xq)[2]
        pind = p.binds[pb] # translate index

        #@show p.binds, pb, pind, state.bset, q

        # remove p and add q
```

```
        @assert state.bset[pb] == pind
        state.bset[pb] = q

        return (stat, p)
    end
end

function solve!(state::SimplexState)
    PlotRegion.plotregion(state.A, state.b);

    @show state.bset;
    status, p = simplex_step!(state);
    iter = 1;

    while status != :Solution
        @show state.bset;
        @show p.x;

        scatter!([p.x[1]],[p.x[2]],
                series_annotations=["$(iter)"],marker=(15,0.2,:orange),label="");
        status, p = simplex_step!(state);
        iter += 1;
    end

    @show state.bset;
    @show p.x;

    scatter!([p.x[1]],[p.x[2]],
        series_annotations=["$(iter)"],marker=(15,0.2,:red),label="")
end

# Problem 1: everything goes well
c1 = [-5.0; -1.0];
A1 = [1.0 1.0;
      2.0 0.5];
b1 = [5.0; 8.0];
A1_slacks = [A1 Matrix{Float64}(I,2,2)]
c1_slacks = [c1; 0.0; 0.0];

# Problem 2: unbounded, the solution is minus infinite
c2 = [-1.0; -3.0];
A2 = [-2.0 1.0;
      -1.0 2.0];
b2 = [2.0; 7.0];
A2_slacks = [A2 Matrix{Float64}(I,2,2)]
c2_slacks = [c2; 0.0; 0.0];

# Problem 3: degenerate, it is slower (we are already at the optimal point)
# The first BFP is optimal but the algorithm is not aware of that.
c3 =[-(3.0/4.0); 150.0; -(1.0/50.0); 6.0];
A3 = [0.25 -60.0 -(1.0/25.0) 9.0;
      0.50 -90.0  (1.0/50.0) 3.0;
       0.0   0.0         1.0 0.0];
b3 = [0.0; 0.0; 1.0];
A3_slacks = [A3 Matrix{Float64}(I,3,3)]
c3_slacks = [c3; 0.0; 0.0; 0.0];

# Start off with the point (0,0)
state = SimplexState(c1_slacks, A1_slacks, b1, [3; 4]);
solve!(state);
```

Problem 2

Using the codes from class (or your own implementations in another language) illustrate the behavior of the simplex method on the LP.

$$\begin{aligned} \text{minimize} \quad & -x_1 - 3x_2 \\ \text{subject to} \quad & -2x_1 + x_2 \leq 2 \\ & -x_1 + 2x_2 \leq 7 \\ & \mathbf{x} \geq 0 \end{aligned}$$

starting at $[0, 0]^T$ after converting the problem to standard form.

Use your judgement in reporting the behavior of the method.

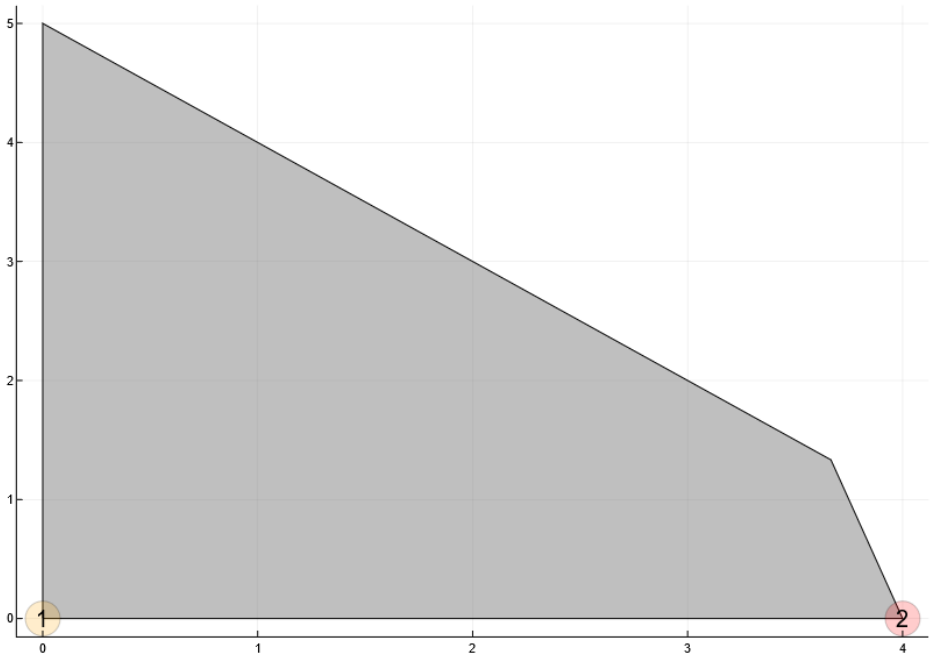

Figure 1: Problem 1: Region plot with visited vertices.

The standard form is:

$$\begin{array}{ll} \text{minimize} & -x_1 - 3x_2 \\ \text{subject to} & -2x_1 + x_2 + s_1 = 2 \\ & -x_1 + 2x_2 + s_2 = 7 \\ & \mathbf{x}, \mathbf{s} \geq 0 \end{array}$$

This problem is unbounded. Starting at $[0, 0]^T$ the algorithm moves to the vertex on the top (2) and then on the right (3). Unfortunately because the problem is unbounded and the objective function decreasing in the unbounded direction, the solution is not on a vertex. The algorithm is cycling and stays at the vertex 3 indefinitely. A region plot is shown in Figure 2.

Problem 3

Using the codes from class (or your own implementations in another language) illustrate the behavior of the simplex method on the LP.

$$\begin{array}{ll} \text{minimize} & -3/4x_1 + 150x_2 - 1/50x_3 + 6x_4 \\ \text{subject to} & 1/4x_1 - 60x_2 - 1/25x_3 + 9x_4 \leq 0 \\ & 1/2x_1 - 90x_2 + 1/50x_3 + 3x_4 \leq 0 \\ & x_3 \leq 1 \\ & \mathbf{x} \geq 0 \end{array}$$

starting at $[0, 0, 0, 0]^T$ after converting the problem to standard form.

Use your judgement in reporting the behavior of the method.

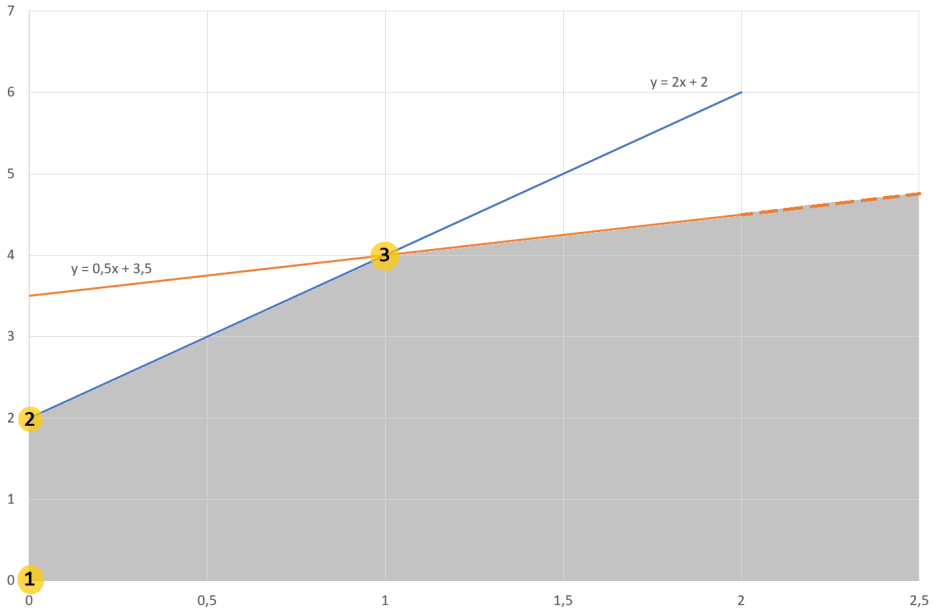

Figure 2: Problem 2: Region plot with visited vertices.

The standard form is:

$$\begin{array}{ll} \text{minimize} & -3/4x_1 + 150x_2 - 1/50x_3 + 6x_4 \\ \text{subject to} & 1/4x_1 - 60x_2 - 1/25x_3 + 9x_4 + s_1 = 0 \\ & 1/2x_1 - 90x_2 + 1/50x_3 + 3x_4 + s_2 = 0 \\ & x_3 + s_3 = 1 \\ & \mathbf{x} \geq 0 \\ & \mathbf{s} \geq 0 \end{array}$$

The first BFP is optimal but the algorithm is not aware of that because the problem is degenerate. It is slower because even though we are already at the optimum, the algorithm moves to other BFPs until one that satisfies the conditions is found.

Here is the list of traversed vertices:

- state.bset = [5, 6, 7] p.x = [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0]
- state.bset = [1, 6, 7] p.x = [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0]
- state.bset = [1, 2, 7] p.x = [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0]
- state.bset = [1, 3, 7] p.x = [0.0, -0.0, 0.0, 0.0, 0.0, 0.0, 1.0]

Problem 4

Show that if we have:

$$\begin{array}{ll} \text{minimize} & \mathbf{c}^T\mathbf{x} \\ \text{subject to} & \boldsymbol{A}\mathbf{x} \leq \mathbf{b} \\ & \mathbf{x} \geq 0 \end{array}$$

and $\mathbf{b} \geq 0$, then $\mathbf{x} = 0$ is always a vertex after converting to standard form.

First we convert the problem into standard form.

$$\begin{aligned} \underset{\hat{\mathbf{x}}}{\text{minimize}} \quad & \hat{\mathbf{c}}^T \hat{\mathbf{x}} \\ \text{subject to} \quad & \hat{\boldsymbol{A}} \hat{\mathbf{x}} = \mathbf{b} \\ & \hat{\mathbf{x}} \geq 0 \end{aligned}$$

With

$$\hat{\mathbf{c}} = \begin{bmatrix} \mathbf{c} \\ 0 \end{bmatrix}; \hat{\boldsymbol{A}} = \begin{bmatrix} \boldsymbol{A} & I \end{bmatrix}; \hat{\mathbf{x}} = \begin{bmatrix} \mathbf{x} \\ \mathbf{s} \end{bmatrix}$$

We prove that the point with $\mathbf{x} = 0$ is a Basic Feasible Point. First, it is feasible because $\boldsymbol{A}\mathbf{x} = 0$ and $\mathbf{b} \geq 0$. Then, the set β of columns we select for the BFP are the last columns of $\hat{\boldsymbol{A}}$ so that $\hat{\boldsymbol{A}}\boldsymbol{P} = \begin{bmatrix} I & \boldsymbol{A} \end{bmatrix}$. Thus $\boldsymbol{B} = I$. Of course, $\boldsymbol{B}$ is non-singular. Now we just need to prove that $\mathbf{x}_B$ is positive. According to the definition, $\mathbf{x}_B = \boldsymbol{B}^{-1}\mathbf{b} = I^{-1}\mathbf{b} = \mathbf{b}$. Because $\mathbf{b} \geq 0$, $\mathbf{x}_B \geq 0$. Therefore, the point with $\mathbf{x} = 0$ is a Basic Feasible Point. We also know that BFPs are vertices of the polytopes. So, the point with $\mathbf{x} = 0$ is always a vertex after converting to standard form.