PURDUE UNIVERSITY · CS 52000
COMPUTATIONAL OPTIMIZATION

HOMEWORK
David F. Gleich
February 27, 2023

Please answer the following questions in complete sentences in a clearly prepared manuscript and submit the solution by the due date on Gradescope.

Remember that this is a graduate class. There may be elements of the problem statements that require you to fill in appropriate assumptions. You are also responsible for determining what evidence to include. An answer alone is rarely sufficient, but neither is an overly verbose description required. Use your judgement to focus your discussion on the most interesting pieces. The answer to "should I include 'something' in my solution?" will almost always be: Yes, if you think it helps support your answer.

Problem 0: Homework checklist

- Please identify anyone, whether or not they are in the class, with whom you discussed your homework. This problem is worth 1 point, but on a multiplicative scale.
- Make sure you have included your source-code and prepared your solution according to the most recent Piazza note on homework submissions.

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.

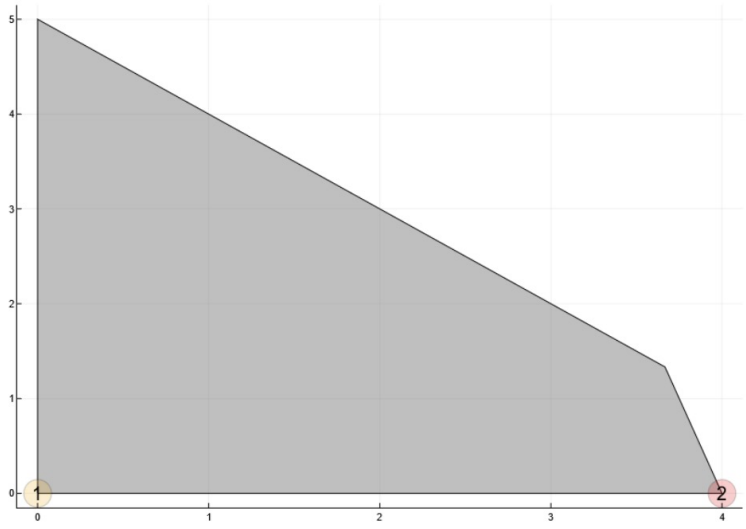
Use your judgement in reporting the behavior of the method.

The standard form for this problem is:

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

The Simplex algorithm can be used to solve this problem, and you can find the Julia code for it in the following listing. The algorithm starts at the point $[0, 0]^T$ and moves towards the right vertex, which is the best solution. The plot of the feasible region for the first problem can be seen in the figure below. The BFP (Basis Feasible Point) contains columns 1 and 3, and the optimal values for x are $[4.0, 0.0, 1.0, 0.0]^T$. .

```
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

function runsimplex(state::SimplexState)
    @show state
    status, p = simplex_step!(state)
    iter = 1
    while status != :Solution
    @show state.bset
    @show p.x
    status, p = simplex_step!(state)
    iter += 1
    end
    return p, state
    end
    function simplex_init(c,A,b)
    # need to get an initial BFP
    # setup the LP for phase 1
    m,n = size(A)
    e = sign.(b)
    cc = [zeros(n); ones(m)]
    CA = [A Diagonal(e)]
    cb = b
    bset = collect(n+1:n+m) # the last columns of cc
    phase1 = SimplexState(cc,CA,cb,bset)
    p, state = runsimplex(phase1)
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 5

teams = ["duke","miami","unc","uva","vt"]
data = [ # team 1 team 2, team 1 pts, team 2 pts
1 2 7 52 # duke played Miami and lost 7 to 52
1 3 21 24 # duke played unc and lost 21 to 24
1 4 7 38
1 5 0 45
2 3 34 16
2 4 25 17
```

```
2 5 27 7
3 4 7 5
3 5 3 30
4 5 14 52
]
ngames = size(data,1)
nteams = length(teams)
G = zeros(ngames, nteams)
p = zeros(ngames, 1)
for g=1:ngames
i = data[g,1]
j = data[g,2]
Pi = data[g,3]
Pj = data[g,4]
G[g,i] = 1
G[g,j] = -1
p[g] = Pi - Pj
end
A1 = [G -G Matrix{Float64}(-I,10,10);
-G G Matrix{Float64}(-I,10,10)]
A = [A1 Matrix{Float64}(I,20,20)]
b = [vec(p);
-vec(p)]

# Find startstate
ptstart, startstate = simplex_init([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
                                    1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0,
                                   0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
                                   , 0, 0, 0, 0, 0, 0],A,b)

state = startstate
@show state.bset
status, p = simplex_step!(state)
iter = 1
while status != :Solution
@show state.bset
status, p = simplex_step!(state)
iter += 1
end

p.x[1:10]
```

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.

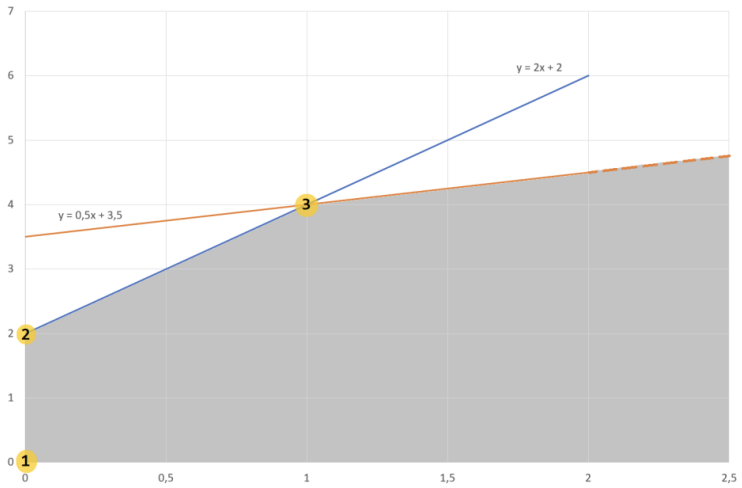
Use your judgement in reporting the behavior of the method.

Similarly, the standard form for this problem is:

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

The problem has no upper bound. The Simplex algorithm begins at the point $[0, 0]^T$ and moves first to vertex 2, then to vertex 3. However, due to the problem

being unbounded and the objective function decreasing in the unbounded direction, the solution is not located at any vertex. As a result, the algorithm is cycling and staying indefinitely at vertex 3. Below the figure shows a plot of the feasible region for this problem.



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.

Similarly, the problem can be standardized as:

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

The initial Basis Feasible Point (BFP) is the best solution, but due to the problem being degenerate, the algorithm is not aware of that fact. As a result, it runs slower because even though the algorithm has already reached the optimal point, it continues searching for other BFPs until it finds one that meets the necessary conditions.

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 let's convert the problem into its standard form.

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

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

Let us demonstrate that the point with $x = 0$ is a Basis Feasible Point. Firstly, it is feasible because $Ax = 0$ and $b \geq 0$. Next, we choose the last columns of $\hat{\boldsymbol{A}}$ as the set β of columns to be selected for the BFP so that $\hat{\boldsymbol{A}}\mathbf{P} = \begin{bmatrix} I & \boldsymbol{A} \end{bmatrix}$. Therefore, $B = I$, which is non-singular. All we need to do now is demonstrate that x_B is positive. By definition, $x_B = B^{-1}b = I^{-1}b = b$. Since $b \geq 0$, $x_B \geq 0$. As a result, the point with $x = 0$ is a Basis Feasible Point. We also know that Basis Feasible Points are vertices of the polytopes. Hence, after converting to standard form, the point with $x = 0$ is always a vertex.

Problem 5

The goal here is to develop an LP-based solver for a 1-norm regression problem. Consider the problem:

$$\underset{\mathbf{x}}{\text{minimize}} \quad \|\boldsymbol{A}\mathbf{x} - \mathbf{b}\|_1 = \textstyle\sum_i |\mathbf{a}_i^T \mathbf{x} - b_i| \ ,$$

where $\mathbf{a}_i^T$ is the ith row of $\boldsymbol{A}$. This can be converted into a linear program and solved via the simplex method. Compute the solution for the sports ranking problem.

Similarly, we can first derive the standardized problem:

$$min \sum_{i=1}^{n} t_i \quad s.t. a_i^T x - t_i \leq -b, t_i \geq 0 \tag{1}$$

where $t_i = [a_i^T - b]$.

Then we can also split $x = x^+ - x^-$ and introduce the slack variable:

$$min \sum_{i=1}^{n} t_i \ s.t. a_i^T x^+ - a_i^T x^- - t_i + s_i = b_i, -a_i^T x^+ + a_i^T x^- - t_i + s_i = -b_i t_i \geq 0, x^+ \geq 0, x^- \geq 0, s_i \geq 0 \tag{2}$$

The result of scores are displayed in the following figure. Please refer to the code listings for detailed codes of this problem.

```
10-element Vector{Float64}:
  0.0
 20.0
  2.0
  0.0
 38.0
 31.0
  0.0
  0.0
  0.0
  0.0
```

Thus the final scores are -31, 20, 2, 0, 38 and the corresponding ranking is VT > Miami > UNC > UVA > Duke.