HOMEWORK 6
Elena Gómez
February 22, 2023

PROBLEM 0: Homework checklist
I worked alone.

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:

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

starting at $[0,0]^T$ **after converting the problem to standard form.**
Use your judgement in reporting the behavior of the method

Firstly, we have to convert the problem into the standard form
The standard form for a linear program is

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & c^t x \\ \text{subject to} \quad & Ax = b \\ & x \geq 0 \end{aligned}$$

If we convert the inequalities into equalities and we add slack variables, our problem becomes

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

Let's define $\hat{A}$, $\hat{b}$, $\hat{c}$,$\hat{x}$ in order to define the problem in a standard form

$\hat{A} = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 2 & \frac{1}{2} & 0 & 1 \end{bmatrix}$

$\hat{b} = (5, 8)^T$

$\hat{c} = (-5, -1, 0, 0)^T$

$\hat{x} = (x_1, x_2, s_1, s_2)^T$

Let's define $\hat{A}$, $\hat{b}$, $\hat{c}$, as code in order to execute the simplex code and find the solution

In [1]:

```
using LinearAlgebra
A1 = [1 1;
      2 0.5]
b = [5; 8]
c=[-5;-1;0;0]
A = [A1 Matrix{Float64}(I,2,2)] # form the problem with slacks.

```

Out[1]:

```
2×4 Matrix{Float64}:
 1.0  1.0  1.0  0.0
 2.0  0.5  0.0  1.0
```

Now, let's execute the simplex method code from class in order to get the solution

In [2]:

```
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
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(state::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(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\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)
```

```julia
        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
        @show xq
        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
```

Out[2]:

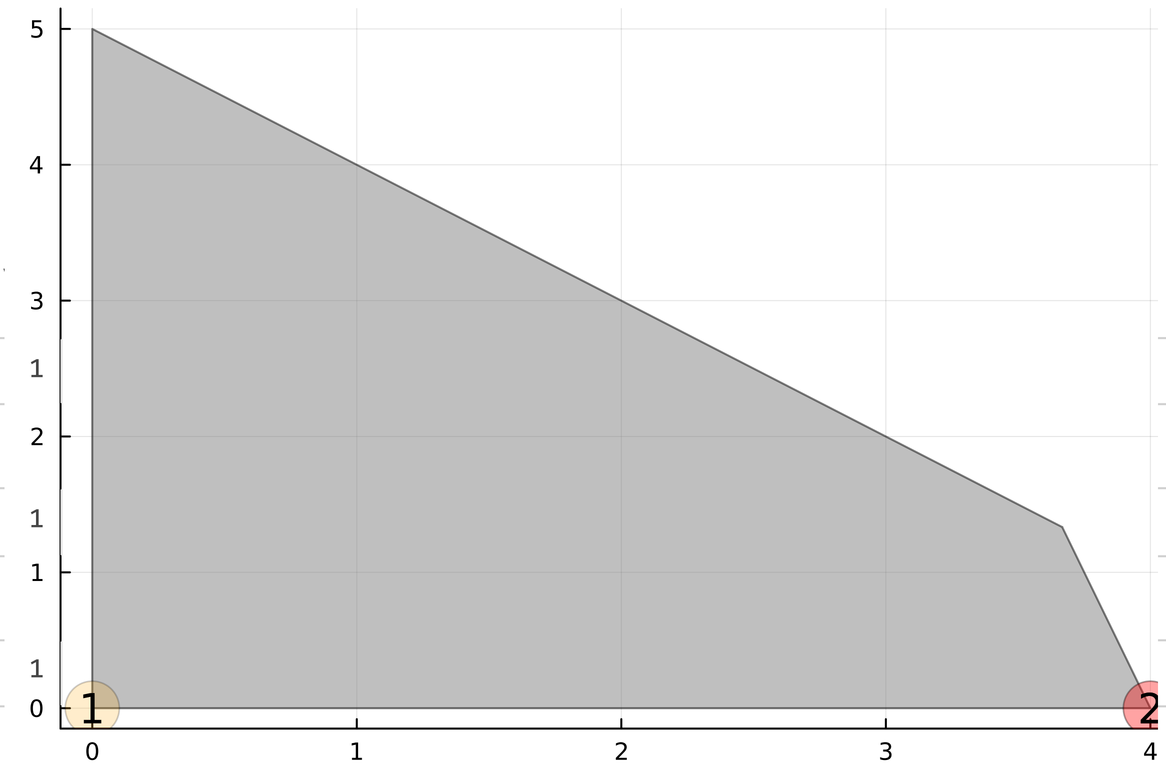
```
simplex_step! (generic function with 1 method)
```

In [3]:

```
using Plots




include("C:/Users/elepu/OneDrive/Escritorio/UNIVERSIDAD/3º DATA SCIENCE/2ºCUATRI/CS-
PlotRegion.plotregion(A,b)

# start off with the point (0,0)
state = SimplexState(c,A,b,
    [3,4])
@show state.bset
status, p = simplex_step!(state)
iter = 1
while status != :Solution
    @show state.bset
    scatter!([p.x[1]],[p.x[2]],
        series_annotations=["$(iter)"],marker=(15,0.2,:orange),label="")
    status, p = simplex_step!(state)
    iter += 1
end
scatter!([p.x[1]],[p.x[2]],
    series_annotations=["$(iter)"],marker=(15,0.2,:red),label="")
@show p.x
scatter!([p.x[1]],[p.x[2]],
    series_annotations=["$(iter)"],marker=(15,0.2,:red),label="")
```

```
state.bset = [3, 4]
cn = [-5, -1]
sn = [-5.0, -1.0]
xq = [5.0, 4.0]
(p.binds, pb, pind, state.bset, q) = ([3, 4], 2, 4, [3, 4], 1)
state.bset = [3, 1]
cn = [-1, 0]
sn = [0.25, 2.5]
p.x = [4.0, 0.0, 1.0, 0.0]
```

Out[3]:



In [ ]:

```
1
```

After executing the simplex method starting at $[0,0]^T$ with BFP columns 1 and 3, we can conclude that the algorithm moves to the right, to the optimum vertex, which is vertex 2.

PROBLEM 2:

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:

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

starting at $[0,0]^T$ **after converting the problem to standard form.**
Use your judgement in reporting the behavior of the method

Firstly, we have to convert the problem into the standard form.
If we convert the inequalities into equalities and we add slack variables, our problem becomes

$$\begin{aligned} \text{minimize} \quad & -x_1 - 3x_2 \\ \text{subject to} \quad & -2x_1 + x_2 + s1 = 2 \\ & -x_1 + x_2 + s-2 = 7 \\ & x, s \ge 0 \end{aligned}$$

Let's define $\hat{A}$, $\hat{b}$, $\hat{c}$,$\hat{x}$ in order to define the problem in a standard form

$$\hat{A} = \begin{bmatrix} -2 & 1 & 1 & 0 \\ -1 & 1 & 0 & 1 \end{bmatrix}$$
$$\hat{b} = (2, 7)^T$$
$$\hat{c} = (-1, -3, 0, 0)^T$$
$$\hat{x} = (x_1, x_2, s_1, s_2)^T$$

Let's define $\hat{A}$, $\hat{b}$, $\hat{c}$ as code in order to execute the simplex code and find the solution

In [4]:

```
using LinearAlgebra
A1 = [-2.0 1.0;
      -1.0 1.0]
b = [2.0; 7.0]
c=[-1;-3;0;0]
A = [A1 Matrix{Float64}(I,2,2)] # form the problem with slacks.

```

Out[4]:

```
2×4 Matrix{Float64}:
 -2.0  1.0  1.0  0.0
 -1.0  1.0  0.0  1.0
```

Now, let's execute the simplex method code from class in order to get the solution

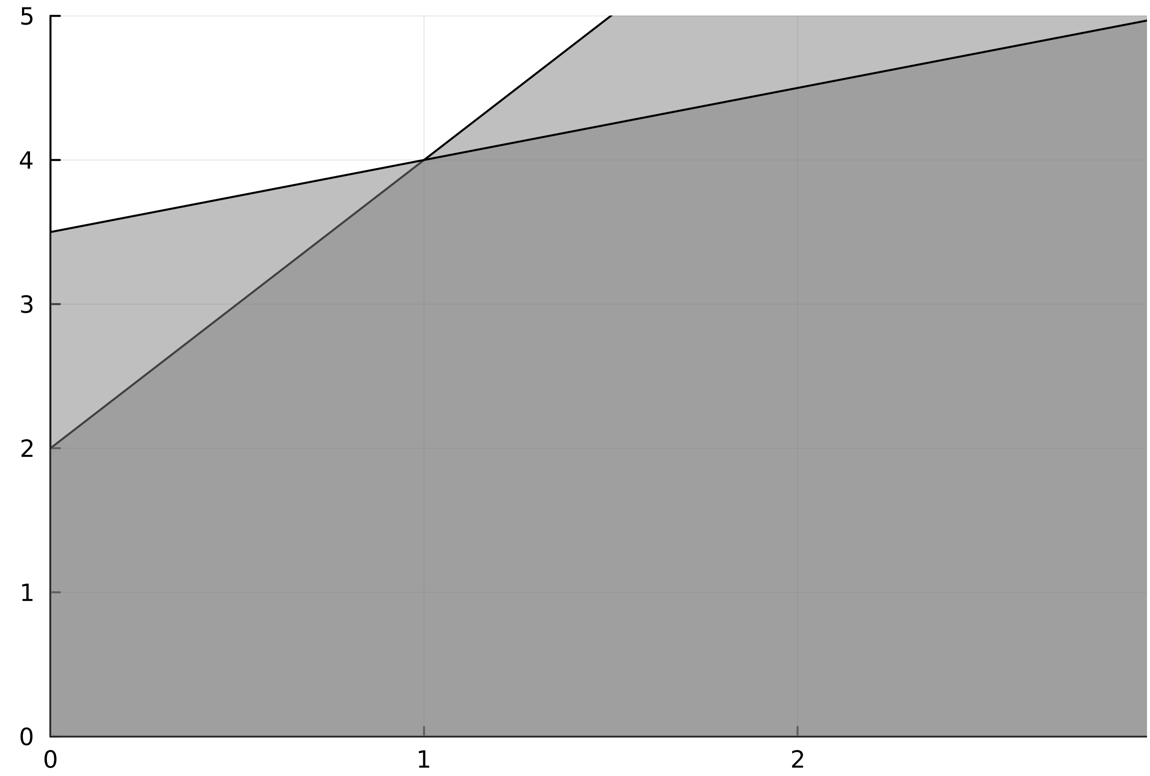
In [5]:

```
using Plots
include("C:/Users/elepu/OneDrive/Escritorio/UNIVERSIDAD/3º DATA SCIENCE/2ºCUATRI/CS-
PlotRegion.plotregion(A,b)

# start off with the point (0,0)
state = SimplexState(c,A,b,
    [3,4])
@show state.bset
status, p = simplex_step!(state)
iter = 1
while status != :Solution
    @show state.bset
    scatter!([p.x[1]],[p.x[2]],
        series_annotations=["$(iter)"],marker=(15,0.2,:orange),label="")
    status, p = simplex_step!(state)
    iter += 1
end
scatter!([p.x[1]],[p.x[2]],
    series_annotations=["$(iter)"],marker=(15,0.2,:red),label="")
@show p.x
scatter!([p.x[1]],[p.x[2]],
    series_annotations=["$(iter)"],marker=(15,0.2,:red),label="")
```

```
state.bset = [3, 4]
cn = [-1, -3]
sn = [-1.0, -3.0]
xq = [2.0, 7.0]
(p.binds, pb, pind, state.bset, q) = ([3, 4], 1, 3, [3, 4], 2)
state.bset = [2, 4]
cn = [-1, 0]
sn = [-7.0, 3.0]
xq = [-1.0, 5.0]
(p.binds, pb, pind, state.bset, q) = ([2, 4], 2, 4, [2, 4], 1)
state.bset = [2, 1]
cn = [0, 0]
sn = [-4.0, 7.0]
state.bset = [2, 1]
cn = [0, 0]
sn = [-4.0, 7.0]
state.bset = [2, 1]
cn = [0, 0]
sn = [-4.0, 7.0]
```

In [6]:

```julia
using Plots

fs1(x1) = 2*x1 +2
fs2(x1) = 0.5*x1 +3.5

x1 = 0:0.01:3
p = plot(x1, [fs1.(x1), fs2.(x1)], fill=(0, 0.5, :grey),
        fillalpha=0.5,
        xlim=(0, 3),
        ylim=(0, 5),
        linecolor=:black,
        legend=false)

display(p)

```



The region plot is above, the region is the one in dark grey. After executing the code, we notice that starting at $[0, 0]^T$, the algorithm moves to the vertex $(0, 2)$ and then to $(1, 4)$. But the problem is unbounded. For this reason, the objective function decrease in the unbounded direction, so the solution is noton a vertex.And The algoritmh cycles and stays at vertex $(1, 4)$ indefinitely.

PROBLEM 3

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

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

starting at $[0,0,0,0]^T$ after converting the problem to standard form. Use your judgement in reporting the behavior of the method.

Firstly, we have to convert the problem into the standard form
If we convert the inequalities into equalities and we add slack variables, our problem becomes

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

Let's define $\hat{A}$, $\hat{b}$, $\hat{c}$,$\hat{x}$ in order to define the problem in a standard form

$$\hat{A} = \begin{bmatrix} \frac{1}{4} & -60 & -\frac{1}{25} & 9 & 1 & 0 & 0 \\ \frac{1}{2} & -90 & \frac{1}{50} & 3 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 \end{bmatrix}$$

$\hat{b} = (0,0,1)^T$
$\hat{c} = (-\frac{3}{4}, 150, -\frac{1}{50}, 6, 0, 0, 0)^T$
$\hat{x} = (x_1, x_2, x_3, x_4, s_1, s_2, s_3, s_4)^T$

Let's define $\hat{A}$, $\hat{b}$, $\hat{c}$, as code in order to execute the simplex code and find the solution

In [7]:

```
using LinearAlgebra
A1 = [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];

b = [0.0; 0.0; 1.0];
c=[-(3.0/4.0); 150.0; -(1.0/50.0); 6.0;0.0;0.0;0.0]
A = [A1 Matrix{Float64}(I,3,3)] # form the problem with slacks.

```

Out[7]:

```
3×7 Matrix{Float64}:
 0.25  -60.0  -0.04  9.0  1.0  0.0  0.0
 0.5   -90.0   0.02  3.0  0.0  1.0  0.0
 0.0     0.0   1.0   0.0  0.0  0.0  1.0
```

Now, let's execute the simplex method code from class in order to get the solution

In [8]:

```
# start off with the point (0,0)
state = SimplexState(c,A,b,
    [5,6,7])
@show state.bset
status, p = simplex_step!(state)
iter = 1
while status != :Solution
    @show state.bset
    scatter!([p.x[1]],[p.x[2]],
        series_annotations=["$(iter)"],marker=(15,0.2,:orange),label="")
    status, p = simplex_step!(state)
    iter += 1
end
scatter!([p.x[1]],[p.x[2]],
    series_annotations=["$(iter)"],marker=(15,0.2,:red),label="")
@show p.x
```

```
state.bset = [5, 6, 7]
cn = [-0.75, 150.0, -0.02, 6.0]
sn = [-0.75, 150.0, -0.02, 6.0]
xq = [0.0, 0.0, Inf]
(p.binds, pb, pind, state.bset, q) = ([5, 6, 7], 1, 5, [5, 6, 7], 1)
state.bset = [1, 6, 7]
cn = [150.0, -0.02, 6.0, 0.0]
sn = [-30.0, -0.13999999999999999, 33.0, 3.0]
xq = [-0.0, 0.0, Inf]
(p.binds, pb, pind, state.bset, q) = ([1, 6, 7], 2, 6, [1, 6, 7], 2)
state.bset = [1, 2, 7]
cn = [-0.02, 6.0, 0.0, 0.0]
sn = [-0.04, 18.0, 1.0, 1.0]
xq = [0.0, -0.0, 1.0]
(p.binds, pb, pind, state.bset, q) = ([1, 2, 7], 2, 2, [1, 2, 7], 3)
state.bset = [1, 3, 7]
cn = [150.0, 6.0, 0.0, 0.0]
sn = [12.0, 11.999999999999998, 0.19999999999999973, 1.4000000000000001]
p.x = [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0]
```

Out[8]:

```
7-element Vector{Float64}:
 0.0
 0.0
 0.0
 0.0
 0.0
 0.0
 1.0
```

The transverse vertices are $state.bset = [5, 6, 7]$, $state.bset = [1, 6, 7]$, $state.bset = [1, 2, 7]$, $state.bset = [1, 3, 7]$ in all of these $p.x = [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0]$

After executing the code, we know that starting at $[0, 0, 0, 0]^T$, the value of x at the optimum is $[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0]^T$. The first BFP is optimal, but the algorithms move to other BFP, until it finds a BFP that satisfies the condition.

PROBLEM 4
Show that if we have:

$$\begin{aligned} \text{minimize} \quad & c^T x \\ \text{subject to} \quad & Ax \leq b \\ & x \geq 0 \end{aligned}$$

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

Firstly, we have to convert the problem into the standard form
If we convert the inequalities into equalities and we add slack variables, our problem becomes

$$\begin{aligned} \text{minimize} \quad & c^T x \\ \text{subject to} \quad & Ax + s = b \\ & x, s \geq 0 \end{aligned}$$

Let's define $\hat{A}$, $\hat{b}$, $\hat{c}$,$\hat{x}$ in order to define the problem in a standard form
$\hat{A} = \begin{bmatrix} A & I \end{bmatrix}$
$\hat{b} = b$
$\hat{c} = (c, 0)^T$
$\hat{x} = (x, s)^T$

So, our problem is

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

Firstly, let's show that, the point with $x = 0$ is a **BFP**.
A **BASIC FEASIBLE POINT (BFP)** is a feasible point ($Ax = b, x \geq 0$) where $AP = [B \;\; N]$, B is non-singular and $\hat{p}^T = \begin{bmatrix} x_B \\ 0 \end{bmatrix} = \begin{bmatrix} B^{-1}b \\ 0 \end{bmatrix}$

It is feasible because $Ax = b$ and $x \geq 0$. The columns of B we choose for the BFP are the last colums of $\hat{A}$, so that $\hat{A}P = [I \;\; A]$. Thus, $B = I$,so B is non-singular.
As explained above, $x_B = B^{-1}b$. As $B = I$, $x_B = I^{-1}b = b$, As $b \geq 0$, $x_B \geq 0$, x_B is positive.
We proved that the point with $x = 0$ is a BFP.

As, it is A BFP, it is a vertex of the polytope. So, we have shown that the point with $x = 0$ is always a vertex after converting to standard form.

PROBLEM 5
The goal here is to develop an LP-based solver for a 1-norm regression problem.
Consider the problem:

$$\underset{x}{\text{minimize}} \qquad ||Ax - b||_1 = \sum_i |a_i^T x - b_i|$$

where a_i^T is the ith row of A. This can be converted into a linear program and solved via the simplex method. Compute the solution for the sports ranking problem

Firstly, we have to convert the problem into the LP standard form in order to solve it later via the simplex method

Let's introduce an auxiliary veriable $t_i = a_i x - b_i$. So now, we have:

$$\begin{aligned} \text{minimize} \quad & \sum_i t_i \\ \text{subject to} \quad & t_i \geq a_i^T x - b_i \leq t_i \end{aligned}$$

If we separate the inequalities, we have:

$$\begin{aligned} \text{minimize} \quad & \sum_i t_i \\ \text{subject to} \quad & a_i^T x - b_i \leq t_i \\ & a_i^T x - b_i \geq -t_i \end{aligned}$$

This can be rewritten as

$$\begin{aligned} \text{minimize} \quad & \sum_i t_i \\ \text{subject to} \quad & t_i - a_i^T x + b_i \geq 0 \\ & t_i + a_i^T x - b_i \geq 0 \end{aligned}$$

Let's introduce slack variables in order to convert the inequalities into equalities

$$\begin{aligned} \text{minimize} \quad & \sum_i t_i \\ \text{subject to} \quad & t_i - a_i^T x + b_i + s_1 = 0 \\ & t_i + a_i^T x - b_i + s_2 = 0 \\ & s_1, s_2, t \geq 0 \end{aligned}$$

As x is unconstrained in sign, we convert it into a pair of positive variables.
$x = x^+ - x^-$
Now, our problem becomes

$$\begin{aligned} \text{minimize} \quad & \sum_i t_i \\ \text{subject to} \quad & t_i - a_i^T x^+ + a_i^T x^- + b_i + s_1 = 0 \\ & t_i + a_i^T x^+ - a_i^T x^- - b_i + s_2 = 0 \\ & s_1, s_2, x^+, x^-, t \geq 0 \end{aligned}$$

This problem can be rewrittwn as follows:

$$\begin{aligned} \text{minimize} \quad & te \\ \text{subject to} \quad & -Ax^+ + Ax^- + te + s_1 = -b \\ & Ax^+ - Ax^- + te + s_2 = b \\ & s_1, s_2, x^+, x^-, t \geq 0 \end{aligned}$$

Let's define $\hat{A}$, $\hat{b}$, $\hat{c}$,$\hat{x}$ in order to define the problem in a standard form

$$\hat{A} = \begin{bmatrix} -A & A & e & I & 0 \\ A & -A & e & 0 & I \end{bmatrix}$$
$$\hat{b} = (-b, b)^T$$
$$\hat{c} = (0, 0, e, 0, 0)^T$$
$$\hat{x} = (x^+, x^-, t, s_1, s_2)^T$$

We have to use the sport data that is the following one:

In [9]:

```
using LinearAlgebra, Plots, Random, SparseArrays
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
```

Let's define $\hat{A}$, $\hat{b}$, $\hat{c}$,$\hat{x}$ as code in order to compute the simplex method. IN the sport data $G = A$ and $p = b$

Let's define $\hat{A}$, $\hat{b}$, $\hat{c}$,$\hat{x}$ in order to define the problem in a standard form

$$\hat{A} = \begin{bmatrix} -A & A & e & I & 0 \\ A & -A & e & 0 & I \end{bmatrix}$$
$$\hat{b} = (-b, b)^T$$
$$\hat{c} = (0, 0, e, 0, 0)^T$$
$$\hat{x} = (x^+, x^-, t, s_1, s_2)^T$$

In [10]:

```
using LinearAlgebra
e=[1,1,1,1,1,1,1,1,1,1]
A1 = [-G G e;
G -G e;];

b = [-p;p];
c=[0.0,0.0,e,0.0,0.0]
A = [A1 Matrix{Float64}(I,20,20)] # form the problem with slacks.
```

Out[10]:

```
20×31 Matrix{Float64}:
 -1.0   1.0  -0.0  -0.0  -0.0   1.0  …  0.0  0.0  0.0  0.0  0.0  0.0  0.0
 -1.0  -0.0   1.0  -0.0  -0.0   1.0     0.0  0.0  0.0  0.0  0.0  0.0  0.0
 -1.0  -0.0  -0.0   1.0  -0.0   1.0     0.0  0.0  0.0  0.0  0.0  0.0  0.0
 -1.0  -0.0  -0.0  -0.0   1.0   1.0     0.0  0.0  0.0  0.0  0.0  0.0  0.0
 -0.0  -1.0   1.0  -0.0  -0.0   0.0     0.0  0.0  0.0  0.0  0.0  0.0  0.0
 -0.0  -1.0  -0.0   1.0  -0.0   0.0  …  0.0  0.0  0.0  0.0  0.0  0.0  0.0
 -0.0  -1.0  -0.0  -0.0   1.0   0.0     0.0  0.0  0.0  0.0  0.0  0.0  0.0
 -0.0  -0.0  -1.0   1.0  -0.0   0.0     0.0  0.0  0.0  0.0  0.0  0.0  0.0
 -0.0  -0.0  -1.0  -0.0   1.0   0.0     0.0  0.0  0.0  0.0  0.0  0.0  0.0
 -0.0  -0.0  -0.0  -1.0   1.0   0.0     0.0  0.0  0.0  0.0  0.0  0.0  0.0
  1.0  -1.0   0.0   0.0   0.0  -1.0  …  0.0  0.0  0.0  0.0  0.0  0.0  0.0
  1.0   0.0  -1.0   0.0   0.0  -1.0     0.0  0.0  0.0  0.0  0.0  0.0  0.0
  1.0   0.0   0.0  -1.0   0.0  -1.0     0.0  0.0  0.0  0.0  0.0  0.0  0.0
  1.0   0.0   0.0   0.0  -1.0  -1.0     1.0  0.0  0.0  0.0  0.0  0.0  0.0
  0.0   1.0  -1.0   0.0   0.0  -0.0     0.0  1.0  0.0  0.0  0.0  0.0  0.0
  0.0   1.0   0.0  -1.0   0.0  -0.0  …  0.0  0.0  1.0  0.0  0.0  0.0  0.0
  0.0   1.0   0.0   0.0  -1.0  -0.0     0.0  0.0  0.0  1.0  0.0  0.0  0.0
  0.0   0.0   1.0  -1.0   0.0  -0.0     0.0  0.0  0.0  0.0  1.0  0.0  0.0
  0.0   0.0   1.0   0.0  -1.0  -0.0     0.0  0.0  0.0  0.0  0.0  1.0  0.0
  0.0   0.0   0.0   1.0  -1.0  -0.0     0.0  0.0  0.0  0.0  0.0  0.0  1.0
```

Once that everything is defined, let's compute the Simplex method

In [11]:

```
# start off with the point (0,0)
state = SimplexState(c,A,b,
    [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20])
@show state.bset
status, p = simplex_step!(state)
iter = 1
while status != :Solution
    @show state.bset
    scatter!([p.x[1]],[p.x[2]],
        series_annotations=["$(iter)"],marker=(15,0.2,:orange),label="")
    status, p = simplex_step!(state)
    iter += 1
end
scatter!([p.x[1]],[p.x[2]],
    series_annotations=["$(iter)"],marker=(15,0.2,:red),label="")
@show p.x

```

```
MethodError: no method matching (Vector)(::Matrix{Float64})
Closest candidates are:
  (Array{T, N} where T)(::AbstractArray{S, N}) where {S, N} at boot.jl:481
  (Vector)() at baseext.jl:38
  (Vector)(::AbstractSparseVector{Tv}) where Tv at C:\Users\elepu\AppData
\Local\Programs\Julia-1.8.5\share\julia\stdlib\v1.8\SparseArrays\src\spars
evector.jl:950
  ...

Stacktrace:
 [1] convert(#unused#::Type{Vector}, a::Matrix{Float64})
   @ Base .\array.jl:617
 [2] SimplexState(c::Vector{Any}, A::Matrix{Float64}, b::Matrix{Float64},
bset::Vector{Int64})
   @ Main .\In[2]:4
 [3] top-level scope
   @ In[11]:2
```

I can't get a numeric solution because I don't know how I have to define the indices. I think that my error is in the vector e, I have been a long time working on this exercise, but I am not ablo to fin my error and solve it