IE 535 Project

February 22, 2019

1 Brian Baller IE 535 Project; ID 0011872161

2 INTRODUCTION

I used the Julia language for this Simplex coding project. If you're unfamiliar, it's similar to Python. The main advantage is a simple syntax for matrices. Like Python, it works with Jupyter Notebooks, which I used for this project report.

Because this project was done in Jupyter Notebooks, all functions used are in the succeeding cell. I used the two-phase method and a tableau implementation. Bland's rule was used for anti-cycling.

3 SIMPLEX CODE

In [270]:
```
#### Cell with all functions for Simplex Method  ####

### Key Variable Definitions
# A, b, c, m, n -- per convention
# J -- set of non-basic variables
# h -- set of basic variables
# a -- set of artificial variables
# c_ph1 -- cost vector for Phase 1
# Y -- matrix of y vectors (essentially the Tableau without row zero or the RHS)
# invB -- the matrix which stores the inverse of B for Tableau initialization
# zc -- reduced cost coefficients (row zero minus the RHS)
# RHS -- the right-hand side row zero value (current Obj function value)
# Tab -- matrix for the Tableau
# a_in_B -- the set of indices that are artificial variables and in the basis
# del_rows -- set of rows which have art variables = 0 at end of Phase 1
# x -- solution vector

### Gets the initial basis and parameters for Phase 1
function get_phase1_setup(A)
    m,n = size(A)
    c_ph1 = vcat(zeros(n),ones(m))                # cost vector for Phase 1
    h = find(c_ph1 .> 0)                          # intial Basis for Phase 1
    a = find(c_ph1 .> 0)                          # cataloguing which indices are a
```

```julia
    Y = [A eye(m)]                                    # this Y matrix is the A matrix p
    J = setdiff(1:n, h)                               # set of non-basic variables is d
    return J, h, a, Y, c_ph1
end

### Populates Row Zero for both Phase 1 and Phase 2
function get_rowzero(A, b, c, J, h, invB)

    zc = Array{Float64}(length(c))                    # initializes reduced cost row fo

    for i in 1:length(c)                              # Loop calculates each reduced co
        if i in J
            zc[i] = c[h]'*invB*A[:,i] - c[i]
        else
            zc[i] = 0
        end
    end

    RHS = c[h]'*invB*b                                # Right-hand side calculation
    rowzero = [zc' RHS]                               # concatenation for row zero
#    println("rowzero is $rowzero")

    return rowzero
end

### Returns the Phase 1 Tableau
function get_tab_phase1(rowzero, Y, b)
    return [rowzero; Y b]
end

### Returns the Phase 1 Tableau
function get_tab_phase2(rowzero, Tab, a)
    return [rowzero; Tab[2:end,setdiff(1:length(Tab[1,:]),a)]]
end


### Finds and returns pivot column and row
function pivot_location(Tab, h)
                                                      # *****REQUIREMENT #5 -- Bland's 
    enter = findfirst(Tab[1,1:end-1] .> 0)            # Bland's rule -- first positive 
#    println("Incoming index is ", enter)
    y = Tab[2:length(Tab[:,1]), enter]                # y is defined as in BJS10
#    println("y is: ", y)

    if(maximum(y)<=0)                                 # check for boundedness -- if all
        error("LP is unbounded")                      # *****REQUIREMENT #6 -- Unbounde
    end
```

```
    posrows = find(y .> 0)                          # finds set of row numbers with p
    ratiotest = Tab[posrows+1,end] ./ y[posrows]    # standard ratio test for all row

#    println("Ratios are: $ratiotest")
                                                    # *****REQUIREMENT #5 Bland's Rul
    exitrow = posrows[indmin(ratiotest)]            # row that exits is the min ratio
    ex_ind = h[exitrow]                             # this just gives the index of th

#    println("Outgoing index is ", ex_ind, " in position ", exitrow)
    return enter, exitrow
end

### Updates B and N after each pivot
function get_new_basis(J, h, exitrow, enter, Tab)

    for i in 1:length(h)                            # loop finds exiting row and repl
        if h[i] == h[exitrow]
            h[i] = enter
        end
    end

    J = setdiff(1:length(Tab[1,:])-1, h)            # determines updated J

#    println("J is: ", J, "  B is: ", h)
    return J, h
end

### Performs row operations on Tableau to generate new Tableau
function row_operations(Tab, exitrow, enter)

    pivrowold = Array{Float64}(length(Tab[1,:]))    # array to temporarily store old
    mult = Array{Float64}(length(Tab[:,1]))              # multipliers for all row op

    for i in 1:length(Tab[:,1])                     # loop calculates multiplier for
        if i == exitrow + 1
            mult[i] = 1/Tab[exitrow+1,enter]
            pivrowold = copy(Tab[i,:])
#            println("old pivot row is: ", pivrowold')
        else
            mult[i] = -Tab[i,enter]/Tab[exitrow+1,enter]
        end
#    println("row",i-1," is: ",mult[i])
    end

    for i in 1:length(Tab[:,1])                     # loop uses multipliers to calcul
        if i != exitrow + 1
            tempROW = mult[i]*pivrowold' + Tab[i,:]'
            Tab[i,:] = tempROW'
```

```
        else
            Tab[i,:] = mult[i]*Tab[i,:]
        end
#        println("new row",i-1," is: ", Tab[i,:]')
    end

    return Tab
end

### Checks for optimal solution and performs one Tableau update iteration
function update_tab(J, h, Tab)
    if(maximum(Tab[1,1:end-1]) <= 0)           # checks reduced costs
        xb = Tab[2:length(Tab[:,1]),end]
#        println("h is: ", h)
        return J, h, Tab
    else                                       # calls the tableau manipulation func
        enter, exitrow = pivot_location(Tab, h)
        J, h = get_new_basis(J, h, exitrow, enter, Tab)
        Tab = row_operations(Tab, exitrow, enter)
        return update_tab(J,h, Tab)             # after Tableau updated, sends back
    end
end

### Finds and returns an initial Identity basis if available   *****REQUIREMENT #4**
function find_initial_basis(A)
    m,n = size(A)
    h = ones(m)
    h = find(h .> 0)
    I = eye(m)
    for i in 1:n
        for j in 1:m
            if I[:,j] == A[:,i]
                h[j] = i
            end
        end
    end
#    println(h)
    return h
end

### Main function for Simplex
function simplex(A, b, c)

    m,n = size(A)
    invB = eye(m)                                    # I start all problems by add

    ### Gets Phase 1 Tableau      *****REQUIREMENT #4******
    J, h, a, Y, c_ph1 = get_phase1_setup(A)          # Gets the initial basis and
```

```
    rowzero = get_rowzero(A, b, c_ph1, J, h, invB)     # Populates Row Zero for Phas
    Tab = get_tab_phase1(rowzero, Y, b)                 # Returns the Phase 1 Tableau

    ### Updates Tableau
    J, h, Tab = update_tab(J, h, Tab)

    ### Prints solution from Phase 1
    x = zeros(n+m)
    x[h] = Tab[2:end,end]
#    println("x is $x and Obj is ", Tab[1,end])
#    println("art vars are:",x[a])

    ### Checks for feasibility  *****REQUIREMENT #2******
    a_in_B = intersect(a,h)
    if length(a_in_B) != 0 && maximum(x[a_in_B]) > 0.0000001   # If artificial varia
        return("Infeasible")                                    # Using 0.00000001 b/
    end

    ### Check for redundancy and deletes redundant rows     *****REQUIREMENT #3*****
    if length(a_in_B) != 0                              # If art. variables in basis
        del_rows = findin(h, a_in_B)                    # Finds any art variables sti
        ## Deletes redundant rows from Tab and A matrices
        Tab = Tab[setdiff(1:length(Tab[:,1]), del_rows+1),setdiff(1:length(Tab[1,:])
        A = A[setdiff(1:length(A[:,1]), del_rows),setdiff(1:length(A[1,:]),a_in_B)]
        h = setdiff(h, a_in_B)                          # Updates h
        a = collect(minimum(a):length(Tab[1,:])-1)      # Updates a (to ensure proper
        b = b[setdiff(1:length(b[:,1]), del_rows),:]    # Deletes redundant row from
#        println(Tab[:,1])
#        println("h is $h and ainB is $a_in_B and del is $del_rows and a is $a")
    end


    ### Gets Phase 2 Tableau and Updates
    invB = Tab[2:end,a]
#    println(invB)
    rowzero = get_rowzero(A, b, c, J, h, invB)          # Populates Row Zero for Phas
#    println(rowzero, a)
    Tab = get_tab_phase2(rowzero, Tab, a)               # Returns the Phase 2 Tableau
    J, h, Tab = update_tab(J, h, Tab)


    ### Prints final solution
    x = zeros(n)
    x[h] = Tab[2:end,end]
    println("Obj is ", Tab[1,end])
    println("Opt x*:")
    for j=1:length(x)
        println("x_$j = ", x[j])
```

```
    end

#    return h
end
```

```
Out[270]: simplex (generic function with 1 method)
```

4 COMMERCIAL SOLVER

All the project problems and "test" problems were solved with both my algorithm and Julia's built-in solver. Julia has a modeling language called "JuMP" which is similar to AMPL. Gurobi and Cplex can be used with JuMP, but I used the open-source Cbc solver because Gurobi was causing an error on my Mac due to some setting on my Mac. The nice thing about using JuMP was that I could use both my solver and the commercial solver in the same Julia Notebook. Information on JuMP can be accessed at http://www.juliaopt.org/JuMP.jl/0.16/quickstart.html.

I tested my algorithm with about 15 example problems from the BJS book, the MIT book and your lecture notes. All of these "test" problems are not included, except for three problems to demonstrate that my algorithm met the requirements. All "test" problems and project problems are solved first with my algorithm then with the commercial solver.

5 Commercial Solver Code

```
In [271]: # Commercial Solver Function

          using JuMP
          using Cbc

          function comm_simplex(A, b, c)
              mod = Model(solver=CbcSolver())

              m, n = size(A)

              @variable(mod, x[1:n] >= 0)

              @objective(mod, Min, sum( c[j]*x[j] for j=1:n) )

              @constraint(mod, constraint[i=1:m], sum( A[i,j]*x[j] for j=1:n ) == b[i] )

              status = solve(mod)

              println("JuMP Model:")
              print(mod)

              println("Objective value: ", getobjectivevalue(mod))
              println("Opt x*:")
              for j=1:n
                println("x_$j = ", getvalue(x[j]))
```

```
        end
    end
```

Out[271]: comm_simplex (generic function with 1 method)

6 CODING REQUIREMENTS

Below I outline the six coding requirement from the posted guidance.

1. All problems were converted manually to standard form.
2. Feasibility is checked at the completion of Phase 1. If any artificial variables remain in the basis at the end of Phase 1, the problem is either infeasible or has redundant rows. If any $x_a! = 0$ and is still in the basis at Phase 1 end, the original LP is infeasible. Because I used phase 1 to check for feasibility and redundancy, I ran all problems through Phase 1 and Phase 2 (maybe not the most efficient, but worked well for my smaller problems)

6.0.1 Infeasible Example

My Algorithm

In [272]:
```
# This is an example from the notes on Oct. 30th with my algorithm

A = [1 1
     2 2]
c = [1,2]
b = [1,3]

simplex(A,b,c)
```

Out[272]: "Infeasible"

Commercial Solver

In [273]:
```
# Ex from notes on Oct. 30th with Commercial Solver

comm_simplex(A, b, c)
```

```
JuMP Model:
Min x[1] + 2 x[2]
Subject to
 x[1] + x[2] = 1
 2 x[1] + 2 x[2] = 3
 x[i]   0   i   {1,2}
Objective value: NaN
Opt x*:
x_1 = NaN
x_2 = NaN
```

```
Warning: Not solved to optimality, status: Infeasible
Warning: Infeasibility ray (Farkas proof) not available
Warning: Variable value not defined for x[1]. Check that the model was properly solved.
Warning: Variable value not defined for x[2]. Check that the model was properly solved.
```

3. If any $x_a = 0$ and is still in the basis at Phase 1 end, then that row is redundant. It can be deleted from the Tableau (as well as its corresponding column from B^{-1})

6.0.2 Redundancy Example

My Algorithm

```
In [274]: # MIT Ex. 3.8 with my algorithm -- Redundant Rows

          A = [1 2 3 0
              -1 2 6 0
               0 4 9 0
               0 0 3 1]
          c = [1,1,1,0]
          b = [3,2,5,1]

          simplex(A,b,c)
```

```
Obj is 1.75
Opt x*:
x_1 = 0.5
x_2 = 1.25
x_3 = 0.0
x_4 = 1.0
```

Commercial Solver

```
In [275]: # MIT Ex. 3.8 with Commercial Solver

          comm_simplex(A, b, c)
```

```
JuMP Model:
Min x[1] + x[2] + x[3]
Subject to
 x[1] + 2 x[2] + 3 x[3] = 3
 -x[1] + 2 x[2] + 6 x[3] = 2
 4 x[2] + 9 x[3] = 5
 3 x[3] + x[4] = 1
 x[i]  0  i  {1,2,3,4}
Objective value: 1.75
Opt x*:
x_1 = 0.4999999999999999
```

```
x_2 = 1.25
x_3 = 0.0
x_4 = 0.9999999999999998
```

4. I made a function to find an initial identity matrix, but as mentioned earlier, I used the two-phase method for all problems to check for feasibility and redundancy. Thus, I started all problems with an identity matrix by including m artificial variables and executing the two-phase method.
5. Bland's rule was used.
6. Both finite optimal solution and unbounded solutions are handled.

6.0.3 Unbounded Example

My Algorithm

In [276]:
```
#BJS Problem 3.28 with my algorithm -- Unbounded

A = [2 -3 -1 1 1 0 0
     -1 2 2 -3 0 1 0
     -1 1 -4 1 0 0 1]
c = [3, -2, 1, -1,0,0,0]
b = [0,1,8]

simplex(A, b, c)
```

```
LP is unbounded



Stacktrace:

 [1] pivot_location(::Array{Float64,2}, ::Array{Int64,1}) at ./In[270]:69

 [2] update_tab(::Array{Int64,1}, ::Array{Int64,1}, ::Array{Float64,2}) at ./In[270]:13

 [3] simplex(::Array{Int64,2}, ::Array{Int64,1}, ::Array{Int64,1}) at ./In[270]:206

 [4] include_string(::String, ::String) at ./loading.jl:522

 [5] execute_request(::ZMQ.Socket, ::IJulia.Msg) at /Users/barovoljko/.julia/v0.6/IJuli

 [6] (::Compat.#inner#6{Array{Any,1},IJulia.#execute_request,Tuple{ZMQ.Socket,IJulia.Ms

 [7] eventloop(::ZMQ.Socket) at /Users/barovoljko/.julia/v0.6/IJulia/src/eventloop.jl:8

 [8] (::IJulia.##13#16)() at ./task.jl:335
```

Commercial Solver

```
In [277]: # BJS Problem 3.28 with Commercial Solver

          comm_simplex(A, b, c)

JuMP Model:
Min 3 x[1] - 2 x[2] + x[3] - x[4]
Subject to
 2 x[1] - 3 x[2] - x[3] + x[4] + x[5] = 0
 -x[1] + 2 x[2] + 2 x[3] - 3 x[4] + x[6] = 1
 -x[1] + x[2] - 4 x[3] + x[4] + x[7] = 8
 x[i]  0  i  {1,2,,6,7}
Objective value: NaN
Opt x*:
x_1 = NaN
x_2 = NaN
x_3 = NaN
x_4 = NaN
x_5 = NaN
x_6 = NaN
x_7 = NaN

Warning: Not solved to optimality, status: Unbounded
Warning: Unbounded ray not available
Warning: Variable value not defined for x[1]. Check that the model was properly solved.
Warning: Variable value not defined for x[2]. Check that the model was properly solved.
Warning: Variable value not defined for x[3]. Check that the model was properly solved.
Warning: Variable value not defined for x[4]. Check that the model was properly solved.
Warning: Variable value not defined for x[5]. Check that the model was properly solved.
Warning: Variable value not defined for x[6]. Check that the model was properly solved.
Warning: Variable value not defined for x[7]. Check that the model was properly solved.
```

7 Assigned Problems

My assigned problems were 11 and 12 with 7* for extra credit. All are solved below and the solutions match the commercial solver!

7.1 Model 11

x_{ij} is the weight of ith type cargo in the jth compartment; i = 1,2,3,4 and j = f, c, b (front, center, back)

$\max 280x_{1f} + 280x_{1c} + 280x_{1b} + 360x_{2f} + 360x_{2c} + 360x_{2b} + 320x_{3f} + 320x_{3c} + 320x_{3b} + 250x_{4f} + 250x_{4c} + 250x_{4b}$

s.t. $x_{1f} + x_{2f} + x_{3f} + x_{4f} \leq 12$ Front weight constraint
$x_{1c} + x_{2c} + x_{3c} + x_{4c} \leq 18$ Center weight constraint
$x_{1b} + x_{2b} + x_{3b} + x_{4b} \leq 10$ Back weight constraint
$500x_{1f} + 700x_{2f} + 600x_{3f} + 400x_{4f} \leq 7000$ Front space constraint
$500x_{1c} + 700x_{2c} + 600x_{3c} + 400x_{4c} \leq 9000$ Center space constraint
$500x_{1b} + 700x_{2b} + 600x_{3b} + 400x_{4b} \leq 5000$ Back space constraint
$18(x_{1f} + x_{2f} + x_{3f} + x_{4f}) - 12(x_{1c} + x_{2c} + x_{3c} + x_{4c}) = 0$ Front/center proportional constraint
$10(x_{1f} + x_{2f} + x_{3f} + x_{4f}) - 12(x_{1b} + x_{2b} + x_{3b} + x_{4b}) = 0$ Front/back proportional constraint
$10(x_{1c} + x_{2c} + x_{3c} + x_{4c}) - 18(x_{1b} + x_{2b} + x_{3b} + x_{4b}) = 0$ Center/back proportional constraint
$x_{1f} + x_{1c} + x_{1b} \leq 20$ Total type 1 cargo available
$x_{2f} + x_{2c} + x_{2b} \leq 16$ Total type 2 cargo available
$x_{3f} + x_{3c} + x_{3b} \leq 25$ Total type 3 cargo available
$x_{4f} + x_{4c} + x_{4b} \leq 13$ Total type 4 cargo available

My Algorithm

In [278]:
```
# Model 11 with My Algorithm

#      1f    1c    1b    2f    2c    2b    3f    3c    3b    4f    4c    4b   s1  s2  s3  s4  s
A =    [1    0     0     1     0     0     1     0     0     1     0     0    1   0   0   0   0
        0    1     0     0     1     0     0     1     0     0     1     0    0   1   0   0   0
        0    0     1     0     0     1     0     0     1     0     0     1    0   0   1   0   0
        500  0     0     700   0     0     600   0     0     400   0     0    0   0   0   1   0
        0    500   0     0     700   0     0     600   0     0     400   0    0   0   0   0   1
        0    0     500   0     0     700   0     0     600   0     0     400  0   0   0   0   0
        18   -12   0     18    -12   0     18    -12   0     18    -12   0    0   0   0   0   0
        10   0     -12   10    0     -12   10    0     -12   10    0     -12  0   0   0   0   0
        0    10    -18   0     10    -18   0     10    -18   0     10    -18  0   0   0   0   0
        1    1     1     0     0     0     0     0     0     0     0     0    0   0   0   0   0
        0    0     0     1     1     1     0     0     0     0     0     0    0   0   0   0   0
        0    0     0     0     0     0     1     1     1     0     0     0    0   0   0   0   0
        0    0     0     0     0     0     0     0     0     1     1     1    0   0   0   0   0
b = [12, 18, 10, 7000, 9000, 5000, 0, 0, 0, 20, 16, 25, 13]
c = [-280, -280, -280, -360, -360, -360, -320, -320, -320, -250, -250, -250, 0,0,0,0

simplex(A, b, c)      #Calls main simplex function
```

```
Obj is -11730.000000000011
Opt x*:
x_1 = 6.999999999999993
x_2 = 0.0
x_3 = 3.99999999999999
x_4 = 5.000000000000003
x_5 = 0.0
x_6 = 2.000000000000006
x_7 = 0.0
x_8 = 8.999999999999998
x_9 = 0.0
```

```
x_10 = 0.0
x_11 = 9.000000000000002
x_12 = 4.000000000000007
x_13 = 1.3877787807815862e-17
x_14 = 3.6221026178395724e-15
x_15 = 0.0
x_16 = 0.0
x_17 = 0.0
x_18 = 0.0
x_19 = 9.000000000000009
x_20 = 8.999999999999993
x_21 = 15.999999999999996
x_22 = 0.0
```

Commercial Solver

```
In [279]: # Model 11 with Commercial Solver

          comm_simplex(A, b, c)

JuMP Model:
Min -280 x[1] - 280 x[2] - 280 x[3] - 360 x[4] - 360 x[5] - 360 x[6] - 320 x[7] - 320 x[8] - 32
Subject to
 x[1] + x[4] + x[7] + x[10] + x[13] = 12
 x[2] + x[5] + x[8] + x[11] + x[14] = 18
 x[3] + x[6] + x[9] + x[12] + x[15] = 10
 500 x[1] + 700 x[4] + 600 x[7] + 400 x[10] + x[16] = 7000
 500 x[2] + 700 x[5] + 600 x[8] + 400 x[11] + x[17] = 9000
 500 x[3] + 700 x[6] + 600 x[9] + 400 x[12] + x[18] = 5000
 18 x[1] - 12 x[2] + 18 x[4] - 12 x[5] + 18 x[7] - 12 x[8] + 18 x[10] - 12 x[11] = 0
 10 x[1] - 12 x[3] + 10 x[4] - 12 x[6] + 10 x[7] - 12 x[9] + 10 x[10] - 12 x[12] = 0
 10 x[2] - 18 x[3] + 10 x[5] - 18 x[6] + 10 x[8] - 18 x[9] + 10 x[11] - 18 x[12] = 0
 x[1] + x[2] + x[3] + x[19] = 20
 x[4] + x[5] + x[6] + x[20] = 16
 x[7] + x[8] + x[9] + x[21] = 25
 x[10] + x[11] + x[12] + x[22] = 13
 x[i]  0  i  {1,2,,21,22}
Objective value: -11730.000000000002
Opt x*:
x_1 = 7.0
x_2 = 0.0
x_3 = 3.9999999999999862
x_4 = 5.0
x_5 = 0.0
x_6 = 2.000000000000067
x_7 = 0.0
x_8 = 9.000000000000007
```

```
x_9 = 0.0
x_10 = 0.0
x_11 = 8.999999999999993
x_12 = 4.000000000000007
x_13 = 0.0
x_14 = 0.0
x_15 = 0.0
x_16 = 0.0
x_17 = 0.0
x_18 = 0.0
x_19 = 9.000000000000014
x_20 = 8.999999999999993
x_21 = 15.999999999999993
x_22 = 0.0
```

7.2 Model 12

x_i is the lbs of the ith alloy used per pound of total product
min $19x_1 + 17x_2 + 23x_3 + 21x_4 + 25x_5$ cost per pound of total product
st $0.6x_1 + 0.25x_2 + 0.45x_3 + 0.2x_4 + 0.5x_5 = 0.4$ tin constraint
$0.1x_1 + 0.15x_2 + 0.45x_3 + 0.5x_4 + 0.4x_5 = 0.35$ zinc constraint
$0.3x_1 + 0.6x_2 + 0.1x_3 + 0.3x_4 + 0.1x_5 = 0.25$ lead constraint
$x_1 + x_2 + x_3 + x_4 + x_5 = 1$ ensures total of all alloys used equals one lb

My Algorithm

```
In [280]: # Model 12 with My Algorithm

          #  x_1 x_2 x_3 x_4 x_5
          A = [.6 .25 .45 .2 .5
               .1 .15 .45 .5 .4
               .3 .60 .10 .3 .1
                1   1    1   1  1]
          b = [.4, .35, .25, 1]
          c = [19, 17, 23, 21, 25]

          simplex(A, b, c)
```

```
Obj is 20.8125
Opt x*:
x_1 = 0.34375000000000006
x_2 = 0.0
x_3 = 0.25000000000000006
x_4 = 0.4062499999999999
x_5 = 0.0
```

Commercial Solver

```
In [281]: # Model 12 with Commercial Solver

          comm_simplex(A, b, c)

JuMP Model:
Min 19 x[1] + 17 x[2] + 23 x[3] + 21 x[4] + 25 x[5]
Subject to
 0.6 x[1] + 0.25 x[2] + 0.45 x[3] + 0.2 x[4] + 0.5 x[5] = 0.4
 0.1 x[1] + 0.15 x[2] + 0.45 x[3] + 0.5 x[4] + 0.4 x[5] = 0.35
 0.3 x[1] + 0.6 x[2] + 0.1 x[3] + 0.3 x[4] + 0.1 x[5] = 0.25
 x[1] + x[2] + x[3] + x[4] + x[5] = 1
 x[i]  0  i  {1,2,3,4,5}
Objective value: 20.8125
Opt x*:
x_1 = 0.3437500000000001
x_2 = 0.0
x_3 = 0.24999999999999994
x_4 = 0.40624999999999994
x_5 = 0.0
```

7.3 Model 7* d_k is the total direct workers in the kth week w/ k = 1,2,3,4 (direct workers are actually producing product)

i_k is the total indirect workers in the kth week w/ k = 1,2,3,4 (indirect workers are idle)

t_k is the total trainers in the kth week w/ k = 1,2,3,4 (trainers are training 3 new workers each)

max $50(15d_1 + 13d_2 + 11d_3 + 9d_4) - 200(d_1 + i_1 + t_1 + d_2 + i_2 + t_2 + d_3 + i_3 + t_3 + d_4 + i_4 + t_4) - 100(3)(t_1 + t_2 + t_3 + t_4)$

max $550d_1 + 450d_2 + 350d_3 + 250d_4 - 200i_1 - 200i_2 - 200i_3 - 200i_4 - 500t_1 - 500t_2 - 500t_3 - 500t_4$

s.t. $d_1 + i_1 + t_1 = 40$

$d_2 + i_2 + t_2 = d_1 + i_1 + t_1 + 3t_1$

$d_3 + i_3 + t_3 = d_2 + i_2 + t_2 + 3t_2$

$d_4 + i_4 + t_4 = d_3 + i_3 + t_3 + 3t_3$

$50d_1 + 50d_2 + 50d_3 + 50d_4 \geq 21475$

s.t. $d_1 + i_1 + t_1 = 40$

$-d_1 - i_1 - 4t_1 + d_2 + i_2 + t_2 = 0$

$0d_1 + 0i_1 + 0t_1 - d_2 - i_2 - 4t_2 + d_3 + i_3 + t_3 = 0$

$0d_1 + 0i_1 + 0t_1 + 0d_2 + 0i_2 + 0t_2 - d_3 - i_3 - 4t_3 + d_4 + i_4 + t_4 = 0$

$50d_1 + 50d_2 + 50d_3 + 50d_4 - s_1 = 21475$

My Algorithm

```
In [282]: # Model 7* with My Algorithm

          #    d_1 d_2 d_3 d_4  i_1   i_2   i_3   i_4   t_1   t_2   t_3   t_4 s_1
          A =    [1   0   0   0   1     0     0     0     1     0     0     0  0
```

```
          -1 1    0    0    -1    1    0    0    -4    1    0    0   0
          0 -1    1    0    0    -1    1    0    0    -4    1    0   0
          0  0   -1    1    0    0    -1    1    0    0    -4    1   0
          50 50 50   50    0    0    0    0    0    0    0    0   -1
          0  0    0    0    0    0    0    0    0    0    0    1   0]
     b = [40, 0, 0, 0, 21475, 0]
     c = [-550, -450, -350, -250, +200, +200, +200, +200, +500, +500, +500, +500, 0]

     simplex(A, b, c)
```

```
Obj is -284000.0
Opt x*:
x_1 = 0.0
x_2 = 0.0
x_3 = 640.0
x_4 = 640.0
x_5 = 0.0
x_6 = 0.0
x_7 = 0.0
x_8 = 0.0
x_9 = 40.0
x_10 = 160.0
x_11 = 0.0
x_12 = 0.0
x_13 = 42525.0
```

Commercial Solver

```
In [283]: # Model 7* with Commercial Solver

          comm_simplex(A, b, c)
```

```
JuMP Model:
Min -550 x[1] - 450 x[2] - 350 x[3] - 250 x[4] + 200 x[5] + 200 x[6] + 200 x[7] + 200 x[8] + 50
Subject to
 x[1] + x[5] + x[9] = 40
 -x[1] + x[2] - x[5] + x[6] - 4 x[9] + x[10] = 0
 -x[2] + x[3] - x[6] + x[7] - 4 x[10] + x[11] = 0
 -x[3] + x[4] - x[7] + x[8] - 4 x[11] + x[12] = 0
 50 x[1] + 50 x[2] + 50 x[3] + 50 x[4] - x[13] = 21475
 x[12] = 0
 x[i]  0  i  {1,2,,12,13}
Objective value: -284000.0
Opt x*:
x_1 = 0.0
x_2 = 0.0
x_3 = 640.0
```

```
x_4 = 640.0
x_5 = 0.0
x_6 = 0.0
x_7 = 0.0
x_8 = 0.0
x_9 = 40.0
x_10 = 160.0
x_11 = 0.0
x_12 = 0.0
x_13 = 42525.0
```