Open in Colab

Math 210

Aaron Graybill

Midterm 1

3/18/21

Problem 1.

First let me bring in all of the predefined scripts and packages:

In [2]:
```
import numpy as np
```

In [3]:
```
def print_tableau(a,indep_names,dep_names):
#
# Given matrix "a" and lists of variables names "indep_names" and "dep_names",
# this function prints the matrix and labels in standard tableau format
# (including adding the -1, the minus signs in the last column, and labeling the lower-right as obj)
#
# First, check the inputs: indep_names should be one shorter than the number of columns of A
#                          dep_names should be one shorter than the number of rows of A
#
    nrows = a.shape[0]    # use the shape function to determine number of rows and cols in A
    ncols = a.shape[1]
    nindep = len(indep_names)
    ndep = len(dep_names)
    if nindep != ncols-1:
        print("WARNING: # of indep vbles should be one fewer than # columns of matrix")
    if ndep != nrows-1:
        print("WARNING: # of dep vbles should be one fewer than # rows of matrix")
# Now do the printing (uses a variety of formatting techniques in Python)
    for j in range(ncols-1):                    # Print the independent variables in the first row
        print(indep_names[j].rjust(10),end="")  # rjust(10) makes fields 10 wide and right-justifies;
                                                #    the end command prevents newline)
    print("        -1")                         # Tack on the -1 at the end of the first row
    for i in range(nrows-1):
        for j in range(ncols):                  # Print all but the last row of the matrix
            print("%10.3f" % a[i][j],end="") # The syntax prints in a field 10 wide, showing 3 decimal points
        lab = "= -" + dep_names[i]
        print(lab.rjust(10))
    for j in range(ncols):
        print("%10.3f" % a[nrows-1][j],end="")  # Print the last row of the matrix, with label "obj" at end
    lab = "= obj"
    print(lab.rjust(10))
    print(" ")    # Put blank line at bottom
```

In [4]:
```
def pivot(a,pivrow,pivcol,indep_names,dep_names) :
#
# Given matrix "a", a row number "pivrow" and column number "pivcol",
#  and lists of variable names "indep_names" and "dep_names", this
#  function does three things:
#    (1) outputs the new version of the matrix after a pivot,
#    (2) updates the lists of variable names post-pivot
#    (3) prints the new matrix, including labels showing the variable names
#
# First, check the inputs: indep_names should be one shorter than the number of columns of A
#                          dep_names should be one shorter than the number of rows of A
#                          you should not be pivoting on the last row or last column
#
    a = a.astype(float)   # make sure entries are treated as floating point numbers
    nrows = a.shape[0]    # use the shape function to determine number of rows and cols in A
    ncols = a.shape[1]
    nindep = len(indep_names)
    ndep = len(dep_names)
    if nindep != ncols-1:
        print("WARNING: # of indep vbles should be one fewer than # columns of matrix")
    if ndep != nrows-1:
        print("WARNING: # of dep vbles should be one fewer than # rows of matrix")
    if pivrow > nrows-1 or pivcol > ncols-1:
        print("WARNING: should not pivot on last row or column")
```

```
    newa = a.copy()       # make a copy of A, to be filled in below with result of pivot
    p = a[pivrow-1][pivcol-1]   # identify pivot element
    newa[pivrow-1][pivcol-1] = 1/p   # set new value of pivot element
    # Set entries in p's row
    for j in range(ncols):
        if j != pivcol-1:
            newa[pivrow-1][j]=a[pivrow-1][j]/p;
    # Set entries in p's column
    for i in range(nrows):
        if i != pivrow-1:
            newa[i][pivcol-1]=-a[i][pivcol-1]/p;
    # Set all other entries
    for i in range(nrows):
        for j in range(ncols):
            if i != pivrow-1 and j != pivcol-1:
                r = a[i][pivcol-1]
                q = a[pivrow-1][j]
                s = a[i][j]
                newa[i][j]=(p*s-q*r)/p
    # Now swap the variable names
    temp = indep_names[pivcol-1]
    indep_names[pivcol-1]=dep_names[pivrow-1]
    dep_names[pivrow-1]=temp
    print_tableau(newa,indep_names,dep_names) # Print the matrix with updated labels
    return newa;
```

In [5]:

```
def target(a) :
    nrows = a.shape[0]    # use the shape function to determine number of rows and cols in "a"
    ncols = a.shape[1]
    import numpy as np
    v = np.empty(ncols-1)
    for i in range(ncols-1):
        v[i]=a[nrows-1,i]
    biggest_c = np.max(v)
    where_is_biggest_c = np.argmax(v)+1
    if biggest_c > 0 :
        return where_is_biggest_c
    else :
        return -1
```

In [36]:

```
def select(a,pivcolnum) :
    nrows = a.shape[0]    # use the shape function to determine number of rows and cols in A
    ncols = a.shape[1]
# First task: work down the column and record the b/a ratios in a vector v
#     except record -1 if a is negative or zero
    import numpy as np
    v = np.zeros(nrows-1)
    for i in range(nrows-1):
        if a[i,pivcolnum-1]>0 :
            v[i] = a[i,ncols-1]/a[i,pivcolnum-1]
        else :
            v[i] = -1
# Second task: if max b/a > -1, find min b/a by hand (ignoring zero entries in v)
    if np.max(v) > -1 :
        min_so_far = np.max(v)+1  # Initialize variable to be for-sure bigger than the min
        for i in range(nrows-1):
            if v[i] > -1 and v[i] < min_so_far :
                min_so_far = v[i]
                where_is_min = i+1   # Add 1 to use human numbering
        return where_is_min      # Once we've scanned v for min, we can return result
    else :
        return -1
```

In [7]:

```
# Create Simplex BF
def SimplexBF(a,indep_names,dep_names):
  nrows, ncols = a.shape
  a_new = a
  print_tableau(a_new,indep_names,dep_names)
  while np.max(a_new[nrows-1,:-1])>0:
    pivcol=target(a_new)
    pivrow=select(a_new,pivcol)
    if pivrow == -1:
      return("Unbounded")
    else:
      a_new=pivot(a_new,pivrow,pivcol,indep_names,dep_names)
      print_tableau(a_new,indep_names,dep_names)
```

a.

Okay now I can start on SimplexNBF:

In [8]:
```
def targetnbf(tab):
  nrows, ncols = tab.shape
  new_i = -1
  for i in range(nrows-1) :# don't check obj fn row
    if tab[i,ncols-1] < 0:
      new_i = i+1
  return(new_i)
```

b.

In [9]:
```
def candidateone(tab,targetedrow):
  #don't specify a row that's the obj fn row, I have no protocol against it
  nrows, ncols = tab.shape
  for i in range(ncols-1): #don't check last col bc it's the b's
    if tab[targetedrow-1,i]<0 :
      return(i+1)#bailout if found one
  return(-1) #none found in loop, return -1
```

c.

In [10]:
```
def  selectnbf(tab,targetedrow,pivcolumn) :
  nrows, ncols = tab.shape

  #subset of two columns in question exclude obj fn row
  candidate_column =tab[targetedrow-1:nrows-1,pivcolumn-1]
  b_column=tab[targetedrow-1:nrows-1,ncols-1]

  # compute ratios
  ratios=b_column/candidate_column

  # Find row in subset and convert back to index in full table:
  #don't need to start at first row, let those be the starting values
  cur_ratio= ratios[0]
  best_row=0
  # the seemingly misplaced -1s and +1s are because we start at second row
  for i in range(len(candidate_column)-1):
    if candidate_column[i+1]>0 and ratios[i+1]<cur_ratio :
      cur_ratio=ratios[i+1]
      best_row=i+1

  #the targetedrow already has the +1, so this converts to start at 1
    #and converts table subset index to full tableau index
  best_row=best_row+targetedrow

  return([best_row,pivcolumn]) #output
```

d.

In [21]:
```
def simplexnbf(tab,indep_names,dep_names):
  nrows, ncols = tab.shape #get dims
  current_target_row=np.Inf #set current target row to nonsense
  tab_new=tab # get primer value for tableau iterator

  NotReadyForBF = True #changed when problem is BF
  SolutionPossible = True #false when no solution is discovered

  print_tableau(tab,indep_names,dep_names) #print initial tableau

  # while NBF and solution still possible apply NBF rules
  while NotReadyForBF and SolutionPossible:
    current_target_row=targetnbf(tab_new) # find target
    if current_target_row ==-1 : #if target is -1, Ready for BF
      NotReadyForBF = False
    else : #find candidate from targeted row
      current_candidate_column=candidateone(tab_new,current_target_row)
      if current_candidate_column ==-1 : #if no candidates, no solution
        SolutionPossible = False
      else : #pivot based off of the computed selection
        pivot_row,pivot_col=selectnbf(tab_new,current_target_row,current_candidate_column)
        tab_new=pivot(tab_new,pivot_row,pivot_col,indep_names,dep_names)
  if not NotReadyForBF : #if we bailed because ready for BF apply SimplexBF
    SimplexBF(tab_new,indep_names,dep_names)
  if not SolutionPossible: # if we bailed bc constraint set emtpy, say so
    return("-1, Constraint set empty, sorry 😕")

  #we can't exit the while loop for anything but the reasons above,
```

```
  # so don't need any else statements
```

That's actually not what the question is asking for, so let me tighten it up to give the requested output (though you can gather the desired result from the above).

In [12]:
```
def simplexnbf_final(tab,indep_names,dep_names):
  output = simplexnbf(tab,indep_names,dep_names)
  # my sad error message is a string, so when we see that, return -1
  if output=="-1, Constraint set empty, sorry 😕":
    return -1
  #else we must have a tableau for BF, return 0
  else :
    return 0
```

Here is proof that all of my functions work as desired.

In [32]:
```
a=np.array([
            [-2,-10,3,-20,8],
            [-1,1,0,-3,-6],
            [1,4,-1,8,-1],
            [1,1,1,1,0]])
indep_names=["x","y","z","w"]
dep_names=["t1","t2","t3"]

print(targetnbf(a))
print(candidateone(a,3))
print(selectnbf(a,3,1))


simplexnbf(a,indep_names,dep_names)
```

```
3
3
[3, 1]
       x         y         z         w        -1
  -2.000   -10.000     3.000   -20.000     8.000     = -t1
  -1.000     1.000     0.000    -3.000    -6.000     = -t2
   1.000     4.000    -1.000     8.000    -1.000     = -t3
   1.000     1.000     1.000     1.000     0.000     = obj

       x         y        t3         w        -1
   1.000     2.000     3.000     4.000     5.000     = -t1
  -1.000     1.000     0.000    -3.000    -6.000     = -t2
  -1.000    -4.000    -1.000    -8.000     1.000      = -z
   2.000     5.000     1.000     9.000    -1.000     = obj

      t2         y        t3         w        -1
   1.000     3.000     3.000     1.000    -1.000     = -t1
  -1.000    -1.000    -0.000     3.000     6.000      = -x
  -1.000    -5.000    -1.000    -5.000     7.000      = -z
   2.000     7.000     1.000     3.000   -13.000     = obj
```

Out[32]:
```
'-1, Constraint set empty, sorry 😕'
```

In [34]:
```
indep_names=["x","y","z","w"]
dep_names=["t1","t2","t3"]

simplexnbf_final(a,indep_names,dep_names)
```

```
       x         y         z         w        -1
  -2.000   -10.000     3.000   -20.000     8.000     = -t1
  -1.000     1.000     0.000    -3.000    -6.000     = -t2
   1.000     4.000    -1.000     8.000    -1.000     = -t3
   1.000     1.000     1.000     1.000     0.000     = obj

       x         y        t3         w        -1
   1.000     2.000     3.000     4.000     5.000     = -t1
  -1.000     1.000     0.000    -3.000    -6.000     = -t2
  -1.000    -4.000    -1.000    -8.000     1.000      = -z
   2.000     5.000     1.000     9.000    -1.000     = obj

      t2         y        t3         w        -1
   1.000     3.000     3.000     1.000    -1.000     = -t1
  -1.000    -1.000    -0.000     3.000     6.000      = -x
  -1.000    -5.000    -1.000    -5.000     7.000      = -z
   2.000     7.000     1.000     3.000   -13.000     = obj
```

Out[34]:
```
-1
```

Problem 2

a.

Since we pivot on a_{62} using the pivot formula we have the following values for the new b_is:

$$b_i^{new} = \begin{cases} \frac{b_i}{a_{62}} & i = 6 \\ \frac{a_{62}b_i - a_{i2}b_6}{a_{62}} & i \neq 6 \end{cases}$$

Plugging in the zeros and simplyfing gives:

$$b_i^{new} = \begin{cases} 0 & i = 6 \\ b_i & i \neq 6 \end{cases}$$

But since $b_6 = 0$, we are back where we started with all of the bs reamining the same.

b.

Let a_{ij}^n be the value a_{ij} in tableau T_n.

Proof by contradiction, suppose that a_{62}^2 was indeed the pivot for T^2. Since all of the b_i^2s remain the same in T_2 as the b_i^1s in T_1. we must again target row four.

To pivot on a_{62}^2, by the candidate rules, $a_{42}^2 < 0$.

Since we pivoted on a_{62}^1, $a_{62}^1 > 0$ (bc $b_6 > 0$ or it would be target). For T_1, we targeted row 4, so $a_{42}^1 < 0$ again by the candidate rules.

Using the pivot rules, I compute a_{42}^2 as follows:

$$a_{42}^2 = -\frac{a_{42}^1}{a_{62}^1}$$

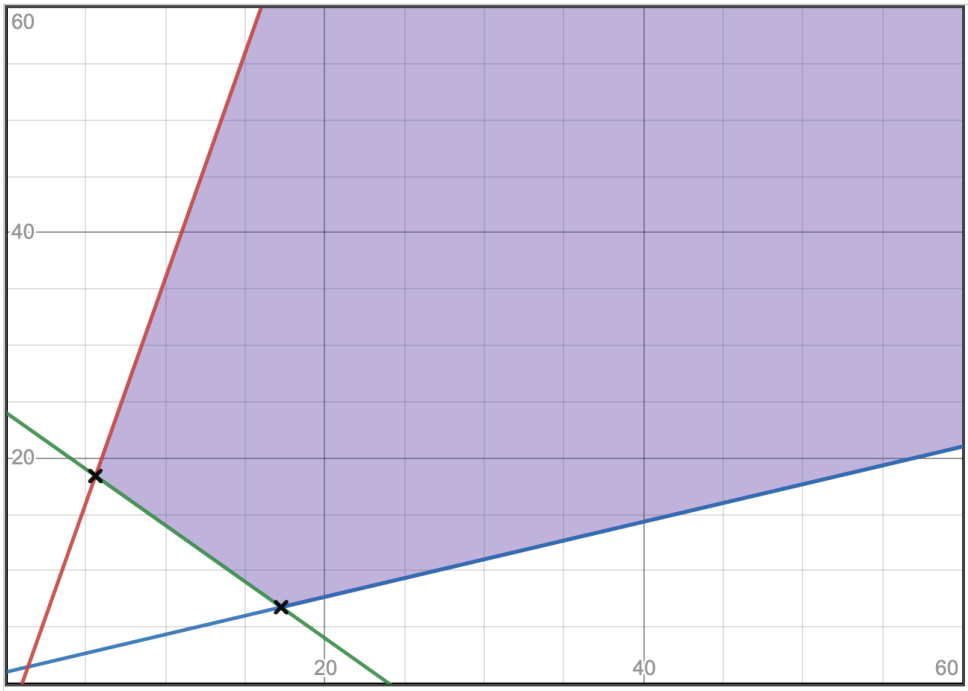
From the results above the numerator must be negative while the denominator must be positive, so the whole fraction when multiplied by -1 is positive. Therefore $a_{42}^2 > 0$. Which contradicts the earlier result. Therefore the supposition that a_{62}^2 is the pivot must be incorrect. We pivot elsewhere. ■

Problem 3.

There are two ways to have no minimum. First if the constraint set is empty which in this case it is not (see plot below). Or if the objective function is unconstrained in a direction that decreases the objective function to decrease ad infiinitum. For the function objective function $y - Ax$ to be decreasing while $x, y \geq 0$, at least one of those terms has to be negative. Therefore $A > 0$.

Now if the max exists, it must occur at a finite point. By the geometric method, we know that this point must be at the intersection of wto constraints (because there are two indep vars). The plot below outlines which intersections form the corners of the region.

In [47]:
```
%%HTML
<iframe src="https://www.desmos.com/calculator/6y1stgwg3x" width="1000px" height="500px" style="border: 1px soli
```



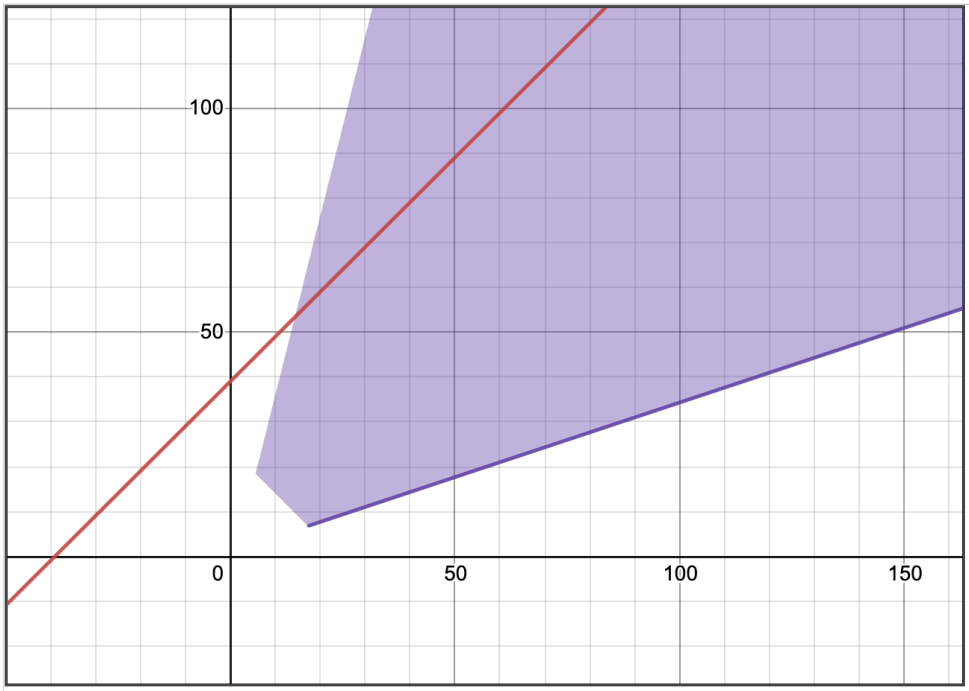
Reading off of the graph above, we have that the corners at which a maximum could occur are at the intersection of the edges of the 1st and 3rd constraints and the 2nd and 3rd constraints.

Solving these systems gives that the maximum must occcur at either $\left(\frac{28}{5}, \frac{92}{5}\right)$ or $\left(\frac{69}{4}, \frac{27}{4}\right)$

I will now show the extant maximum unbounded min property occurs whenever $A > 4$. I spent a long, fruitless, time trying to come up with a concise algebraic proof, but alas I will have to appeal to graphical inuition.

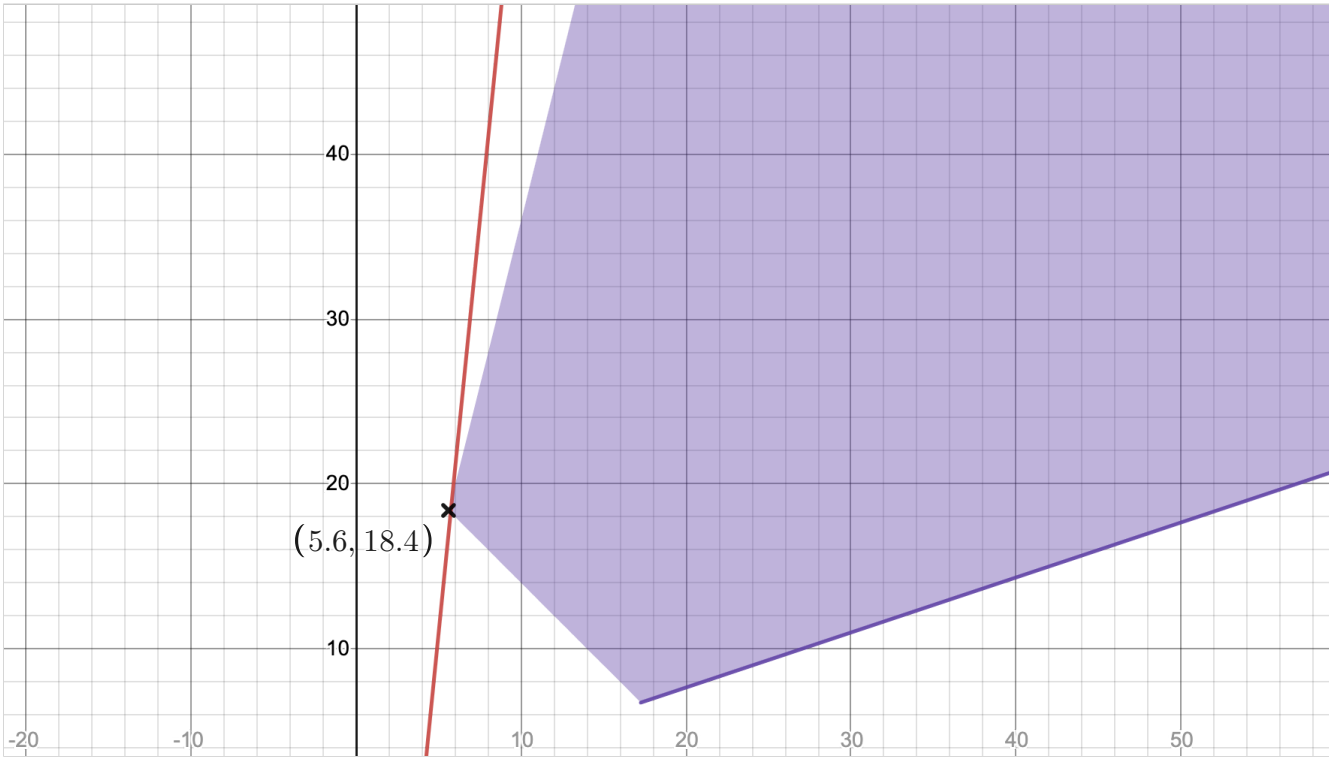
First suppose that $A \leq 4$, I will show that an arbitrarily high value C can be attained. When $A \leq 4$ as plotted below, note that the slope of the contour is less than that of the upper part of the constraint set. Note that along this contour $C = y - Ax \implies y = Ax + C$, so the contour value is only affect the y-intercept of the contour, increasing the y-intercept might push the intersection to a different place, but since the slope of the contour is less than that of the top of the constraint set, there must be an intersection, we can attain the desired C.

In [70]:
```
%%html
<iframe src="https://www.desmos.com/calculator/v6bdqqvagy" width="1000px" height="500px" style="border: 1px soli
```



This shows that $A \leq 4$ cannot have the desired property, so we must see if we can find the desired property when $A > 4$. First to show it has a maximum. Consider the plot below, whenever $A \geq 4$, the contours move in the northwest direction as the value of the contour increases and the slope of this line is greater than the top part of the constraint, so eventually we will be left with just one point in the constraint set and by the geometric method, this must be the unique maximizer at $(5.6, 18.4)$.

In [71]:
```
%%html
<iframe src="https://www.desmos.com/calculator/2916qk5amf?embed" width="1000px" height="500px" style="border: 1p
```



Now to show that the minimum is unbounded. The contours moving down go in the southeast direction, and since the slope of the contour is steeper than that of the bottom of the constraint set there always must be an intersection, we can find arbitrarily small values of the objective function. This completes the skeleton of the proof.

Problem 4.

Thankfully, the problem is already in the proper form to be added to the tableau, though we will have to run NBF. Note that the objective function is asking us to find the largest value of x_4 which means all of the other x_is add nothing and each $x-4$ is worth one. The objective function must then be $f(X)=x_4$ I setup the tableau by reading off of the constraints and applying SimplexNBF:

In [51]:
```
a=np.array([[1,2,1,1,16],[-1,-1,2,3,5],[-1,2,-4,5,-14],[1,-4,-3,6,24],[0,0,0,1,0]])
indep_names=["x1","x2","x3","x4"]
dep_names=["t1","t2","t3","t4"]
simplexnbf(a,indep_names,dep_names)
```

```
      x1        x2        x3        x4        -1
   1.000     2.000     1.000     1.000    16.000     = -t1
  -1.000    -1.000     2.000     3.000     5.000     = -t2
  -1.000     2.000    -4.000     5.000   -14.000     = -t3
   1.000    -4.000    -3.000     6.000    24.000     = -t4
   0.000     0.000     0.000     1.000     0.000     = obj

      t3        x2        x3        x4        -1
   1.000     4.000    -3.000     6.000     2.000     = -t1
  -1.000    -3.000     6.000    -2.000    19.000     = -t2
  -1.000    -2.000     4.000    -5.000    14.000     = -x1
   1.000    -2.000    -7.000    11.000    10.000     = -t4
   0.000     0.000    -0.000     1.000    -0.000     = obj

      t3        x2        x3        x4        -1
   1.000     4.000    -3.000     6.000     2.000     = -t1
  -1.000    -3.000     6.000    -2.000    19.000     = -t2
  -1.000    -2.000     4.000    -5.000    14.000     = -x1
   1.000    -2.000    -7.000    11.000    10.000     = -t4
   0.000     0.000    -0.000     1.000    -0.000     = obj

      t3        x2        x3        t1        -1
   0.167     0.667    -0.500     0.167     0.333     = -x4
  -0.667    -1.667     5.000     0.333    19.667     = -t2
  -0.167     1.333     1.500     0.833    15.667     = -x1
  -0.833    -9.333    -1.500    -1.833     6.333     = -t4
  -0.167    -0.667     0.500    -0.167    -0.333     = obj

      t3        x2        x3        t1        -1
   0.167     0.667    -0.500     0.167     0.333     = -x4
  -0.667    -1.667     5.000     0.333    19.667     = -t2
  -0.167     1.333     1.500     0.833    15.667     = -x1
  -0.833    -9.333    -1.500    -1.833     6.333     = -t4
  -0.167    -0.667     0.500    -0.167    -0.333     = obj

      t3        x2        t2        t1        -1
   0.100     0.500     0.100     0.200     2.300     = -x4
  -0.133    -0.333     0.200     0.067     3.933     = -x3
   0.033     1.833    -0.300     0.733     9.767     = -x1
  -1.033    -9.833     0.300    -1.733    12.233     = -t4
  -0.100    -0.500    -0.100    -0.200    -2.300     = obj

      t3        x2        t2        t1        -1
   0.100     0.500     0.100     0.200     2.300     = -x4
  -0.133    -0.333     0.200     0.067     3.933     = -x3
   0.033     1.833    -0.300     0.733     9.767     = -x1
  -1.033    -9.833     0.300    -1.733    12.233     = -t4
  -0.100    -0.500    -0.100    -0.200    -2.300     = obj
```

I will briefly talk through those pivots because the question seems to be asking us to figure out the pivots by hand. First we have to run NBF because one of the b_is is negative. As such we target row 3, which makes us take row one as the candidates and then we compute smallest ratio which happens to be in a_{31}. Pivoting there gives the next tableau in the sequence, which happens to be BF.

We target the column with the highest value, that happens to be column 4. I compute the ratios that have positive denominators, and we select the smallest which happens to be a_{14}. Continue in a similar way until we get the following result:

$$x_1^* = 9.767 \tag{1}$$
$$x_2^* = 0 \tag{2}$$
$$x_3^* = 3.933 \tag{3}$$
$$x_4^* = 2.3 \tag{4}$$

And the value of the objective function is 2.3, implying that the maximum value that x_4 can attain in the set is 2.3

Problem 5.

We have five choice variables, $\mathcal{A}_1$ and $\mathcal{A}_2$ the number of apartments built in periods one and two, $\mathcal{O}_1$ and $\mathcal{O}_2$ for the number of of offices built in period one and two and $\mathcal{T}$ for the number of electricians sent to training in period one The master electricians in period one is a static number and they can be allocated to either $\mathcal{A}$ or $\mathcal{O}$. Their constraint would be:

$$\mathcal{A}_1 + \mathcal{O}_1 \leq 20$$

In period two, the constraint is eased by the number of electricians sent to training. Therefore the $t = 2$ master electrician constraint is

$$\mathcal{A}_2 + \mathcal{O}_2 \leq 20 + \mathcal{T}$$

.

We have a similar constraint but going in the opposite direction for regular electricians. We send $\mathcal{T}$ away in the first period meaning:

$$2\mathcal{A}_1 + 5\mathcal{O}_1 \leq 120 - \mathcal{T}$$

Those electricians become masters in period two, so are also removed from that constraint giving the same constraint for $t = 2$

$$2\mathcal{A}_2 + 5\mathcal{O}_2 \leq 120 - \mathcal{T}$$

We then have the carptenter and painter constraints which are more straightforward and give:

$$\begin{aligned} 4\mathcal{A}_1 + 2\mathcal{O}_1 &\leq 100 \\ 4\mathcal{A}_2 + 2\mathcal{O}_2 &\leq 100 \\ 5\mathcal{A}_1 + 2\mathcal{O}_1 &\leq 90 \\ 5\mathcal{A}_2 + 2\mathcal{O}_2 &\leq 90 \end{aligned}$$

The final thing to mention is the objective function, each apartment in either period brings in 25k and an office brings in 30k in either period. Each trainee subtracts 5k from profit, so our objective function in thousands is:

$$25\mathcal{A}_1 + 25\mathcal{A}_2 + 30\mathcal{O}_1 + 30\mathcal{O}_2 - 5\mathcal{T}$$

Let's tableau-ify and solve:

In [58]:
```
a=np.array([[1,1,0,0,0,20],
            [0,0,1,1,-1,20],
            [2,5,0,0,1,120],
            [0,0,2,5,1,120],
            [4,2,0,0,0,100],
            [0,0,4,2,0,100],
            [5,2,0,0,0,90],
            [0,0,5,2,0,90],
            [25,30,25,30,-5,0]])
indep_names=["A1","O1","A2","O2","T"]
dep_names=["mElec1","mElec2","Elec1","Elec2","Carp1","Carp2","Paint1","Paint2"]

simplexnbf(a,indep_names,dep_names)
```

```
      A1       O1       A2       O2        T       -1
   1.000    1.000    0.000    0.000    0.000   20.000 = -mElec1
   0.000    0.000    1.000    1.000   -1.000   20.000 = -mElec2
   2.000    5.000    0.000    0.000    1.000  120.000  = -Elec1
   0.000    0.000    2.000    5.000    1.000  120.000  = -Elec2
   4.000    2.000    0.000    0.000    0.000  100.000  = -Carp1
   0.000    0.000    4.000    2.000    0.000  100.000  = -Carp2
   5.000    2.000    0.000    0.000    0.000   90.000 = -Paint1
   0.000    0.000    5.000    2.000    0.000   90.000 = -Paint2
  25.000   30.000   25.000   30.000   -5.000    0.000     = obj

      A1       O1       A2       O2        T       -1
   1.000    1.000    0.000    0.000    0.000   20.000 = -mElec1
   0.000    0.000    1.000    1.000   -1.000   20.000 = -mElec2
   2.000    5.000    0.000    0.000    1.000  120.000  = -Elec1
   0.000    0.000    2.000    5.000    1.000  120.000  = -Elec2
   4.000    2.000    0.000    0.000    0.000  100.000  = -Carp1
   0.000    0.000    4.000    2.000    0.000  100.000  = -Carp2
   5.000    2.000    0.000    0.000    0.000   90.000 = -Paint1
   0.000    0.000    5.000    2.000    0.000   90.000 = -Paint2
  25.000   30.000   25.000   30.000   -5.000    0.000     = obj

      A1   mElec1       A2       O2        T       -1
   1.000    1.000    0.000    0.000    0.000   20.000     = -O1
   0.000   -0.000    1.000    1.000   -1.000   20.000 = -mElec2
  -3.000   -5.000    0.000    0.000    1.000   20.000  = -Elec1
   0.000   -0.000    2.000    5.000    1.000  120.000  = -Elec2
   2.000   -2.000    0.000    0.000    0.000   60.000  = -Carp1
   0.000   -0.000    4.000    2.000    0.000  100.000  = -Carp2
   3.000   -2.000    0.000    0.000    0.000   50.000 = -Paint1
   0.000   -0.000    5.000    2.000    0.000   90.000 = -Paint2
  -5.000  -30.000   25.000   30.000   -5.000 -600.000     = obj
```

A1	mElec1	A2	O2	T	-1	
1.000	1.000	0.000	0.000	0.000	20.000	= -O1
0.000	-0.000	1.000	1.000	-1.000	20.000	= -mElec2
-3.000	-5.000	0.000	0.000	1.000	20.000	= -Elec1
0.000	-0.000	2.000	5.000	1.000	120.000	= -Elec2
2.000	-2.000	0.000	0.000	0.000	60.000	= -Carp1
0.000	-0.000	4.000	2.000	0.000	100.000	= -Carp2
3.000	-2.000	0.000	0.000	0.000	50.000	= -Paint1
0.000	-0.000	5.000	2.000	0.000	90.000	= -Paint2
-5.000	-30.000	25.000	30.000	-5.000	-600.000	= obj

A1	mElec1	A2	mElec2	T	-1	
1.000	1.000	0.000	-0.000	0.000	20.000	= -O1
0.000	-0.000	1.000	1.000	-1.000	20.000	= -O2
-3.000	-5.000	0.000	-0.000	1.000	20.000	= -Elec1
0.000	0.000	-3.000	-5.000	6.000	20.000	= -Elec2
2.000	-2.000	0.000	-0.000	0.000	60.000	= -Carp1
0.000	0.000	2.000	-2.000	2.000	60.000	= -Carp2
3.000	-2.000	0.000	-0.000	0.000	50.000	= -Paint1
0.000	0.000	3.000	-2.000	2.000	50.000	= -Paint2
-5.000	-30.000	-5.000	-30.000	25.000	-1200.000	= obj

A1	mElec1	A2	mElec2	T	-1	
1.000	1.000	0.000	-0.000	0.000	20.000	= -O1
0.000	-0.000	1.000	1.000	-1.000	20.000	= -O2
-3.000	-5.000	0.000	-0.000	1.000	20.000	= -Elec1
0.000	0.000	-3.000	-5.000	6.000	20.000	= -Elec2
2.000	-2.000	0.000	-0.000	0.000	60.000	= -Carp1
0.000	0.000	2.000	-2.000	2.000	60.000	= -Carp2
3.000	-2.000	0.000	-0.000	0.000	50.000	= -Paint1
0.000	0.000	3.000	-2.000	2.000	50.000	= -Paint2
-5.000	-30.000	-5.000	-30.000	25.000	-1200.000	= obj

A1	mElec1	A2	mElec2	Elec2	-1	
1.000	1.000	0.000	0.000	-0.000	20.000	= -O1
0.000	0.000	0.500	0.167	0.167	23.333	= -O2
-3.000	-5.000	0.500	0.833	-0.167	16.667	= -Elec1
0.000	0.000	-0.500	-0.833	0.167	3.333	= -T
2.000	-2.000	0.000	0.000	-0.000	60.000	= -Carp1
0.000	0.000	3.000	-0.333	-0.333	53.333	= -Carp2
3.000	-2.000	0.000	0.000	-0.000	50.000	= -Paint1
0.000	0.000	4.000	-0.333	-0.333	43.333	= -Paint2
-5.000	-30.000	7.500	-9.167	-4.167	-1283.333	= obj

A1	mElec1	A2	mElec2	Elec2	-1	
1.000	1.000	0.000	0.000	-0.000	20.000	= -O1
0.000	0.000	0.500	0.167	0.167	23.333	= -O2
-3.000	-5.000	0.500	0.833	-0.167	16.667	= -Elec1
0.000	0.000	-0.500	-0.833	0.167	3.333	= -T
2.000	-2.000	0.000	0.000	-0.000	60.000	= -Carp1
0.000	0.000	3.000	-0.333	-0.333	53.333	= -Carp2
3.000	-2.000	0.000	0.000	-0.000	50.000	= -Paint1
0.000	0.000	4.000	-0.333	-0.333	43.333	= -Paint2
-5.000	-30.000	7.500	-9.167	-4.167	-1283.333	= obj

A1	mElec1	Paint2	mElec2	Elec2	-1	
1.000	1.000	-0.000	0.000	0.000	20.000	= -O1
0.000	0.000	-0.125	0.208	0.208	17.917	= -O2
-3.000	-5.000	-0.125	0.875	-0.125	11.250	= -Elec1
0.000	0.000	0.125	-0.875	0.125	8.750	= -T
2.000	-2.000	-0.000	0.000	0.000	60.000	= -Carp1
0.000	0.000	-0.750	-0.083	-0.083	20.833	= -Carp2
3.000	-2.000	-0.000	0.000	0.000	50.000	= -Paint1
0.000	0.000	0.250	-0.083	-0.083	10.833	= -A2
-5.000	-30.000	-1.875	-8.542	-3.542	-1364.583	= obj

A1	mElec1	Paint2	mElec2	Elec2	-1	
1.000	1.000	-0.000	0.000	0.000	20.000	= -O1
0.000	0.000	-0.125	0.208	0.208	17.917	= -O2
-3.000	-5.000	-0.125	0.875	-0.125	11.250	= -Elec1
0.000	0.000	0.125	-0.875	0.125	8.750	= -T
2.000	-2.000	-0.000	0.000	0.000	60.000	= -Carp1
0.000	0.000	-0.750	-0.083	-0.083	20.833	= -Carp2
3.000	-2.000	-0.000	0.000	0.000	50.000	= -Paint1
0.000	0.000	0.250	-0.083	-0.083	10.833	= -A2
-5.000	-30.000	-1.875	-8.542	-3.542	-1364.583	= obj

Let's interpret the above results. We have first that the company attains a maximal profit of $\$1364.583$k dollars or $\$1,364,583$ in profit. To attain that profit, the company sends 8.75 electricians to training. They produce 20 offices and no apartments in period one, but in period two, they produce 17.917 offices and 10.833 apartments. Note that we send just enough electricians to training such that neither master electricians or electrians are left not working in period two.

Problem 6.

Define the following: B number of protein bars used, A number of apples used, P number of poptarts, T number of twinkies and H for hot pockets.

The constraints are fairly straightforward:

$$\begin{gathered}
8B + A + 2.3P + 2T + 19H \geq 300 \\
130B + 95A + 209P + 260T + 600H \geq 10000 \\
130B + 95A + 209P + 260T + 600H \leq 20000 \\
120B + 0A + 172P + 350T + 1050H \leq 15000 \\
4B + 0A + 7P + 8T + 26H \leq 500 \\
3B + 3A + .7P + 0T + 2H \geq 100
\end{gathered}$$

To CanonicalMax Form:

$$\begin{gathered}
-8B - A - 2.3P - 2T - 19H \leq -300 \\
-130B - 95A - 209P - 260T - 600H \leq -10000 \\
130B + 95A + 209P + 260T + 600H \leq 20000 \\
120B + 0A + 172P + 350T + 1050H \leq 15000 \\
4B + 0A + 7P + 8T + 26H \leq 500 \\
-3B - 3A - .7P - 0T - 2H \leq -100
\end{gathered}$$

But we are trying to find the minimum of the function so, we must also multiply the objective function by -1 giving:

$$-1B - .8A - .75P - .6T - 1.75H$$

tableauing and simplexing nbf gives:

In [67]:
```
a=np.array([[-8,-1,-2.3,-2,-19,-300],
            [-130,-95,-209,-260,-600,-10000],
            [130,95,209,260,600,10000],
            [120,0,172,350,1050,15000],
            [4,0,7,8,26,500],
            [-3,-3,-.7,0,-2,-100],
            [-1,-.8,-.75,-.6,-1.75,0]
            ])
indep_names=["ProtBar","Apple","PopTart","Twinkie","HotPock"]
dep_names=["Protein","Calories_low","Calories_high","Sodium","Fat","Fiber"]
simplexnbf(a,indep_names,dep_names)
```

```
   ProtBar     Apple   PopTart   Twinkie   HotPock        -1
    -8.000    -1.000    -2.300    -2.000   -19.000  -300.000= -Protein
  -130.000   -95.000  -209.000  -260.000  -600.000-10000.000= -Calories_low
   130.000    95.000   209.000   260.000   600.000 10000.000= -Calories_high
   120.000     0.000   172.000   350.000  1050.000 15000.000 = -Sodium
     4.000     0.000     7.000     8.000    26.000   500.000    = -Fat
    -3.000    -3.000    -0.700     0.000    -2.000  -100.000  = -Fiber
    -1.000    -0.800    -0.750    -0.600    -1.750     0.000     = obj

     Fiber     Apple   PopTart   Twinkie   HotPock        -1
    -2.667     7.000    -0.433    -2.000   -13.667   -33.333= -Protein
   -43.333    35.000  -178.667  -260.000  -513.333 -5666.667= -Calories_low
    43.333   -35.000   178.667   260.000   513.333  5666.667= -Calories_high
    40.000  -120.000   144.000   350.000   970.000 11000.000 = -Sodium
     1.333    -4.000     6.067     8.000    23.333   366.667    = -Fat
    -0.333     1.000     0.233    -0.000     0.667    33.333= -ProtBar
    -0.333     0.200    -0.517    -0.600    -1.083    33.333     = obj

Calories_low     Apple   PopTart   Twinkie   HotPock        -1
    -0.062     4.846    10.562    14.000    17.923   315.385= -Protein
    -0.023    -0.808     4.123     6.000    11.846   130.769  = -Fiber
     1.000    -0.000    -0.000    -0.000    -0.000    -0.000= -Calories_high
     0.923   -87.692   -20.923   110.000   496.154  5769.231 = -Sodium
     0.031    -2.923     0.569     0.000     7.538   192.308    = -Fat
    -0.008     0.731     1.608     2.000     4.615    76.923= -ProtBar
    -0.008    -0.069     0.858     1.400     2.865    76.923     = obj

Calories_low     Apple   PopTart   Twinkie   HotPock        -1
    -0.062     4.846    10.562    14.000    17.923   315.385= -Protein
    -0.023    -0.808     4.123     6.000    11.846   130.769  = -Fiber
     1.000    -0.000    -0.000    -0.000    -0.000    -0.000= -Calories_high
     0.923   -87.692   -20.923   110.000   496.154  5769.231 = -Sodium
     0.031    -2.923     0.569     0.000     7.538   192.308    = -Fat
    -0.008     0.731     1.608     2.000     4.615    76.923= -ProtBar
    -0.008    -0.069     0.858     1.400     2.865    76.923     = obj

Calories_low     Apple   PopTart   Twinkie     Fiber        -1
```

```
    -0.027     6.068     4.323     4.922    -1.513   117.532= -Protein
    -0.002    -0.068     0.348     0.506     0.084    11.039= -HotPock
     1.000    -0.000     0.000     0.000     0.000     0.000= -Calories_high
     1.890   -53.864  -193.610  -141.299   -41.883   292.208 = -Sodium
     0.045    -2.409    -2.055    -3.818    -0.636   109.091    = -Fat
     0.001     1.045     0.001    -0.338    -0.390    25.974= -ProtBar
    -0.002     0.126    -0.140    -0.051    -0.242    45.292     = obj

Calories_low     Apple   PopTart   Twinkie     Fiber        -1
    -0.027     6.068     4.323     4.922    -1.513   117.532= -Protein
    -0.002    -0.068     0.348     0.506     0.084    11.039= -HotPock
     1.000    -0.000     0.000     0.000     0.000     0.000= -Calories_high
     1.890   -53.864  -193.610  -141.299   -41.883   292.208 = -Sodium
     0.045    -2.409    -2.055    -3.818    -0.636   109.091    = -Fat
     0.001     1.045     0.001    -0.338    -0.390    25.974= -ProtBar
    -0.002     0.126    -0.140    -0.051    -0.242    45.292     = obj

Calories_low   Protein   PopTart   Twinkie     Fiber        -1
    -0.004     0.165     0.712     0.811    -0.249    19.369  = -Apple
    -0.002     0.011     0.397     0.562     0.067    12.360= -HotPock
     1.000     0.000     0.000     0.000     0.000     0.000= -Calories_high
     1.653     8.876  -155.234   -97.608   -55.313  1335.474 = -Sodium
     0.035     0.397    -0.338    -1.864    -1.237   155.752    = -Fat
     0.006    -0.172    -0.744    -1.186    -0.129     5.725= -ProtBar
    -0.002    -0.021    -0.229    -0.154    -0.210    42.849     = obj

Calories_low   Protein   PopTart   Twinkie     Fiber        -1
    -0.004     0.165     0.712     0.811    -0.249    19.369  = -Apple
    -0.002     0.011     0.397     0.562     0.067    12.360= -HotPock
     1.000     0.000     0.000     0.000     0.000     0.000= -Calories_high
     1.653     8.876  -155.234   -97.608   -55.313  1335.474 = -Sodium
     0.035     0.397    -0.338    -1.864    -1.237   155.752    = -Fat
     0.006    -0.172    -0.744    -1.186    -0.129     5.725= -ProtBar
    -0.002    -0.021    -0.229    -0.154    -0.210    42.849     = obj
```

The least expensive way to feed the camper is using 19.369 Apples, 12.360 hot pockets, 5.725 protein bars and using none of the other items.