CS 3430: SciComp with Py
Assignment 04
Simplex Algorithm

Vladimir Kulyukin
Department of Computer Science
Utah State University

February 01, 2020

Learning Objectives

1. Pivots, Entering and Departing Variables
2. Tableaus
3. Simplex Algorithm

Introduction

In this assignment, we'll implement the Simplex Algorithm for standard maximum problems, a fully automatic method of solving LP problems in any number of dimensions with pure algebra. You'll save your coding solutions in `cs3430_s20_hw04.py` included in the zip and submit it in Canvas.

Problem 1: (2 points)

Let's consider the following SMP.

```
A fertilizer company makes 4 types of fertilizer: 20-8-8
for lawns, 4-8-4 for gardens, and 4-4-2 for general purposes.
The numbers in each fertilizer brand refer to the weight
percentages of nitrate, phosphate, and potash, respectively,
in a sack of fertilizer. The company has 6,000 pounds of nitrate,
10,000 of phosphate, and 4,000 pounds of potash. The profit is
$3 per 100 pounds of lawn fertilizer, $8 per 100 pounds of
garden fertilizer, and $6 per 100 pounds of general purpose
fertilizer. How many pounds of each fertilizer should the
company produce to maximize the profit and satisfy the constraints?
```

Let's define the following decision variables: x - the number of 100's of pounds of 20-8-8 fertilizer; y - the number of 100's of pounds of 4-8-4 fertilizer; and z - the number of 100's of pounds of 4-4-2 fertilizer. Using these decision variables, we can express the constraints on the amounts of nitrate, phosphate, and potash as

1. $20x + 4y + 4z \leq 6,000$ (nitrate);
2. $8x + 8y + 4z \leq 10,000$ (phosphate);

3. $8x + 4y + 2z \leq 4,000$ (potash);
4. $x \geq 0$;
5. $y \geq 0$;
6. $z \geq 0$.

The objective function is $p = 3x+8y+6z$. Since we want to form a tableau for the Simplex Algorithm, let's add three non-negative slacks u, v, and w for the unused numbers of 100's of pounds of nitrate, phosphate, and potash, respectively, to the first three constraints. With the slacks added, these constraints turn into equations:

1. $20x + 4y + 4z + u = 6,000$ (nitrate);
2. $8x + 8y + 4z + v = 10,000$ (phosphate);
3. $8x + 4y + 2z + w = 4,000$ (potash).

Now we can set up the tableau.

	x	y	z	u	v	w	B.S.
u	20	4	4	1	0	0	6,000
v	8	8	4	0	1	0	10,000
w	8	4	2	0	0	1	4,000
p	-3	-8	-6	0	0	0	0

So far so good! We can do better though in that we can further simplify the tableau setup by using only the variable x with subscripts. Let $x = x_0$, $y = x_1$, $z = x_2$, $u = x_3$, $v = x_4$, and $w = x_5$. Then the tableau looks as follows.

	x_0	x_1	x_2	x_3	x_4	x_5	B.S.
x_3	20	4	4	1	0	0	6,000
x_4	8	8	4	0	1	0	10,000
x_5	8	4	2	0	0	1	4,000
p	-3	-8	-6	0	0	0	0

We can simplify the tableau some more by droping the x's and using only the subscripts.

	0	1	2	3	4	5	B.S.
3	20	4	4	1	0	0	6,000
4	8	8	4	0	1	0	10,000
5	8	4	2	0	0	1	4,000
p	-3	-8	-6	0	0	0	0

Observe that 0 stands for x_0, 1 for x_1, etc. Note also that the first (unlabeled) column in the tableau contains the entered variables or *in-variables*. The entered variables are initially the slack variables. The last column of the tableau contains the basic solution values and the bottom row of the tableau is the p-row. This tableau offers the following basic solution: $x_3 = 6,000$, $x_4 = 10,000$, $x_5 = 4,000$, $x_1 = x_2 = 0$. Recall that the convention is that if the variable is not an in-variable, its value is 0. Since x_1 and x_2 are both departed variables, they are equal to 0, by convention.

Let's translate this notation into Python. We can use a dictionary to represent the in-variables (i.e., entered variables).

```
in_vars = {0:3, 1:4, 2:5}
>>> in_vars[0]
3
>>> in_vars[1]
4
>>> in_vars[2]
5
```

This representation means that the 0-th in-variable is x_3, the 1st – x_4, and the 2-nd – x_5. The tableau's matrix on which the simplex algorithm will operate can be represented as a numpy array.

```
m = np.array([[20,  4,  4, 1, 0, 0, 6000],
              [8,   8,  4, 0, 1, 0, 10000],
              [8,   4,  2, 0, 0, 1, 4000],
              [-3, -8, -6, 0, 0, 0, 0]],
             dtype=float)
```

The tableau is a 2-tuple that consists of the dictionary with the in-variables and the tableau's matrix.

```
>>> tab = (in_vars, m)
```

Implement the function `simplex(tab)` that takes a tableau represented as a 2-tuple, runs the simplex algorithm on the tableau, and returns two values: the modified tableau and the boolean variable that is `True` when the returned tableau contains a solution and `False` when the returned tableau does not contain a solution (i.e., the problem formulated by the initial tableau has no solution).

Below is the Py console's output of my implementation of `simplex()` applied to `tab`. In the trace, `evc` stands for "entering variable column" and `dvr` stands for "departing variable row." These two numbers identify the location of the pivot on which the pivoting is done. When no entering variable can be found, `evc` is equal to -1. When no departing variable can be found, `dvc` is equal to -1. The new tableau (i.e., the dictionary and the matrix) is displayed after each pivoting operation.

```
>>> tab, solved = simplex(tab)

in vars: {0: 3, 1: 4, 2: 5}
mat:
[[ 2.e+01  4.e+00  4.e+00  1.e+00  0.e+00  0.e+00  6.e+03]
 [ 8.e+00  8.e+00  4.e+00  0.e+00  1.e+00  0.e+00  1.e+04]
 [ 8.e+00  4.e+00  2.e+00  0.e+00  0.e+00  1.e+00  4.e+03]
 [-3.e+00 -8.e+00 -6.e+00  0.e+00  0.e+00  0.e+00  0.e+00]]
evc = 1
dvr = 2
pivoting dvr=2, evc=1

in vars: {0: 3, 1: 4, 2: 1}
mat:
[[ 1.2e+01  0.0e+00  2.0e+00  1.0e+00  0.0e+00 -1.0e+00  2.0e+03]
 [-8.0e+00  0.0e+00  0.0e+00  0.0e+00  1.0e+00 -2.0e+00  2.0e+03]
 [ 2.0e+00  1.0e+00  5.0e-01  0.0e+00  0.0e+00  2.5e-01  1.0e+03]
 [ 1.3e+01  0.0e+00 -2.0e+00  0.0e+00  0.0e+00  2.0e+00  8.0e+03]]
evc = 2
dvr = 0
pivoting dvr=0, evc=2
```

```
in vars: {0: 2, 1: 4, 2: 1}
mat:
[[ 6.0e+00  0.0e+00  1.0e+00  5.0e-01  0.0e+00 -5.0e-01  1.0e+03]
 [-8.0e+00  0.0e+00  0.0e+00  0.0e+00  1.0e+00 -2.0e+00  2.0e+03]
 [-1.0e+00  1.0e+00  0.0e+00 -2.5e-01  0.0e+00  5.0e-01  5.0e+02]
 [ 2.5e+01  0.0e+00  0.0e+00  1.0e+00  0.0e+00  1.0e+00  1.0e+04]]

evc = -1
in_vars = {0: 2, 1: 4, 2: 1}
tab = [[ 6.0e+00  0.0e+00  1.0e+00  5.0e-01  0.0e+00 -5.0e-01  1.0e+03]
 [-8.0e+00  0.0e+00  0.0e+00  0.0e+00  1.0e+00 -2.0e+00  2.0e+03]
 [-1.0e+00  1.0e+00  0.0e+00 -2.5e-01  0.0e+00  5.0e-01  5.0e+02]
 [ 2.5e+01  0.0e+00  0.0e+00  1.0e+00  0.0e+00  1.0e+00  1.0e+04]]

>>> print(tab[0])
{0: 2, 1: 4, 2: 1}
>>> print(tab[1])
[[ 6.0e+00  0.0e+00  1.0e+00  5.0e-01  0.0e+00 -5.0e-01  1.0e+03]
 [-8.0e+00  0.0e+00  0.0e+00  0.0e+00  1.0e+00 -2.0e+00  2.0e+03]
 [-1.0e+00  1.0e+00  0.0e+00 -2.5e-01  0.0e+00  5.0e-01  5.0e+02]
 [ 2.5e+01  0.0e+00  0.0e+00  1.0e+00  0.0e+00  1.0e+00  1.0e+04]]
>>> solved
True
```

The scientific notation of floats is helpful in debugging but it's no fun to read (in my humble opinion). So, let's write a couple of functions to display the results better.

```
def get_solution_from_tab(tab):
    in_vars, mat = tab[0], tab[1]
    nr, nc = mat.shape
    sol = {}
    for k, v in in_vars.items():
        sol[v] = mat[k,nc-1]
    sol['p'] = mat[nr-1,nc-1]
    return sol

def display_solution_from_tab(tab):
    sol = get_solution_from_tab(tab)
    for var, val in sol.items():
        if var == 'p':
            print('p\t=\t{}'.format(val))
        else:
            print('x{}\t=\t{}'.format(var, val))
```

The function `get_solution_from_tab()` puts the tableau's solution into a dictionary mapping each in-variable to its value in the B.S. column. The value of the objective function is mapped to by the key `'p'`. The function `display_solution_from_tab()` prints the dictionary returned by `get_solution_from_tab()` in a more palatable manner.

```
>>> display_solution_from_tab(tab)
x2 = 1000.0
x4 = 2000.0
```

```
x1 = 500.0
p = 10000.0
```

This is better, because we can see right away that the maximum value of the objective function found by the simplex algorithm is 10,000. We also know that the basic solution that corresponds to this value is $x_1 = 500.0$, $x_2 = 1000.0$, and $x_4 = 2000.0$. Recall that our mapping convention is: $x = x_0$, $y = x_1$, $z = x_2$, $u = x_3$, $v = x_4$, and $w = x_5$. Thus, the optimal production schedule for the company is to produce no 20-8-8 fertilizer, 500x100 = 50,000 pounds of the 4-8-4 fertilizer, and 1000x100 = 100,000 pounds of the 4-4-2 fertilizer. Note that in the optimal solution found by the algorithm the slack variable x_4 is equal 2000, which means that there will be 2,000 pounds of phosphate left over.

Let's solve another SMP. Maximize $p = 10x + 6y + 2z$ subject to

1. $x \geq 0$;
2. $y \geq 0$;
3. $z \geq 0$;
4. $2x + 2y + 3z \leq 160$;
5. $5x + y + 10z \leq 100$.

We'll map x to x_0, y to x_1, z to x_2 and introduce two slacks – x_3 and x_4 for the last two constraints. Let's go straight to Python and form the tableau.

```
>>> in_vars = {0:3, 1:4}
>>> m = np.array([[2,    2,   3, 1, 0, 160],
                  [5,    1,  10, 0, 1, 100],
                  [-10, -6,  -2, 0, 0, 0]],
                 dtype=float)
>>> tab = (in_vars, m)
>>> tab, solved = simplex(tab)

in vars: {0: 3, 1: 4}
mat:
[[  2.   2.   3.   1.   0. 160.]
 [  5.   1.  10.   0.   1. 100.]
 [-10.  -6.  -2.   0.   0.   0.]]
evc = 0
dvr = 1
pivoting dvr=1, evc=0

in vars: {0: 3, 1: 0}
mat:
[[  0.    1.6  -1.    1.   -0.4 120. ]
 [  1.    0.2   2.    0.    0.2  20. ]
 [  0.   -4.   18.    0.    2.  200. ]]
evc = 1
dvr = 0
pivoting dvr=0, evc=1

in vars: {0: 1, 1: 0}
mat:
```

```
[[ 0.000e+00  1.000e+00 -6.250e-01  6.250e-01 -2.500e-01  7.500e+01]
 [ 1.000e+00  0.000e+00  2.125e+00 -1.250e-01  2.500e-01  5.000e+00]
 [ 0.000e+00  0.000e+00  1.550e+01  2.500e+00  1.000e+00  5.000e+02]]
evc = -1

in_vars = {0: 1, 1: 0}
tab = [[ 0.000e+00  1.000e+00 -6.250e-01  6.250e-01 -2.500e-01  7.500e+01]
 [ 1.000e+00  0.000e+00  2.125e+00 -1.250e-01  2.500e-01  5.000e+00]
 [ 0.000e+00  0.000e+00  1.550e+01  2.500e+00  1.000e+00  5.000e+02]]

>>> solved
True
>>> display_solution_from_tab(tab)
x1 = 75.0
x0 = 5.0
p = 500.0
```

Let's test `simplex()` on a tableau that has no solution, where there are 3 decision variables (i.e., x_0, x_1, x_2) and two slacks (i.e., x_3 and x_4).

```
>>> in_vars = {0:3, 1:4}
>>> m = np.array([[1,  -1,  1, 1, 0, 5],
                  [2,   0, -1, 0, 1, 10],
                  [-1, -2, -1, 0, 0, 0]],
                 dtype=float)
>>> tab = (in_vars, m)
>>> tab, solved = simplex(tab)
in vars: {0: 3, 1: 4}
mat:
[[ 1. -1.  1.  1.  0.  5.]
 [ 2.  0. -1.  0.  1. 10.]
 [-1. -2. -1.  0.  0.  0.]]
evc = 1
dvr = -1

in_vars = {0: 3, 1: 4}
tab = [[ 1. -1.  1.  1.  0.  5.]
 [ 2.  0. -1.  0.  1. 10.]
 [-1. -2. -1.  0.  0.  0.]]
>>> solved
False
```

Why is `solved False`? Because in the initial tableau the algorithm failed to find the departing variable in the column of the most negative entrhy (-2) in the p-row. In other words, the problem has no solution.

Let's solve the Ted's Toys problem with `simplex()`. Read Assignment 03 if you don't remember the details.

```
>>> in_vars = {0:2, 1:3}
>>> m = np.array([[ 4,  3,  1, 0, 480],
                  [ 3,  6,  0, 1, 720],
                  [-5, -4,  0, 0, 0]],
```

```
                  dtype=float)
>>> tab = (in_vars, m)
>>> tab, solved = simplex(tab)

in vars: {0: 2, 1: 3}
mat:
[[  4.   3.   1.   0. 480.]
 [  3.   6.   0.   1. 720.]
 [ -5.  -4.   0.   0.   0.]]
evc = 0
dvr = 0
pivoting dvr=0, evc=0

in vars: {0: 0, 1: 3}
mat:
[[ 1.00e+00  7.50e-01  2.50e-01  0.00e+00  1.20e+02]
 [ 0.00e+00  3.75e+00 -7.50e-01  1.00e+00  3.60e+02]
 [ 0.00e+00 -2.50e-01  1.25e+00  0.00e+00  6.00e+02]]
evc = 1
dvr = 1
pivoting dvr=1, evc=1

in vars: {0: 0, 1: 1}
mat:
[[ 1.00000000e+00  0.00000000e+00  4.00000000e-01 -2.00000000e-01
   4.80000000e+01]
 [ 0.00000000e+00  1.00000000e+00 -2.00000000e-01  2.66666667e-01
   9.60000000e+01]
 [ 0.00000000e+00  0.00000000e+00  1.20000000e+00  6.66666667e-02
   6.24000000e+02]]
evc = -1

in_vars = {0: 0, 1: 1}
tab = [[ 1.00000000e+00  0.00000000e+00  4.00000000e-01 -2.00000000e-01
   4.80000000e+01]
 [ 0.00000000e+00  1.00000000e+00 -2.00000000e-01  2.66666667e-01
   9.60000000e+01]
 [ 0.00000000e+00  0.00000000e+00  1.20000000e+00  6.66666667e-02
   6.24000000e+02]]

>>> solved
True
>>> display_solution_from_tab(tab)
x0 = 48.0
x1 = 96.0
p = 624.0
```

With pure algebra the simplex algorithm found automatically the same solution that we found with 2D geometry in a semi-automatic fashion in the previous assignment. Amazing!

Problem 2: (2 points)

Use your implementation of `simplex()` to solve the following problems and save your solutions in the appropriate stubs in `cs3430_s20_hw04.py`.

Problem 2.1 ($\frac{1}{4}$ point)

Maximize $p = 2x + 3y$ subject to

1. $x \geq 0$;
2. $y \geq 0$;
3. $3x + 8y \leq 24$;
4. $6x + 4y \leq 30$.

Problem 2.2 ($\frac{1}{4}$ point)

Maximize $p = x$ subject to

1. $x \geq 0$;
2. $y \geq 0$;
3. $x - y \leq 4$;
4. $-x + 3y \leq 4$.

Problem 2.3 ($\frac{3}{4}$ point)

The Brown Brothers Box Company is bidding on a new contract to manufacture boxes for computers, printers, and paint. A computer box requires 12 square feet of heavy-duty cardboard, 18 square feet of regular cardboard, and 15 square feet of white facing paper. A printer box requires 6 square feet of heavy-duty cardboard, 12 square feet of regular cardboard, and 8 square feet of white facing paper. A paint box requires 10 square feet of regular cardboard. Current arrangements with the suppliers allow the company the following weekly allocations: 1,500 square feet of heavy-duty cardboard, 2,500 square feet of regular cardboard, and 2,000 square feet of facing paper. The profit is \$1.50 for a computer box, \$0.80 for a printer box, and \$0.25 for a paint box. Solve for the weekly allocation of resources which yields maximum profit.

Problem 2.4 ($\frac{3}{4}$ point)

A farmer has 1,000 acres and water rights to 600 acre-feet of water for next season. An acre-foot of water is the amount of water which covers 1 acre at a depth of 1 foot. Crop A yeilds 120 bushels per acre and requires 6 inches of water per acre, crop B yields 80 bushels per acre and requires 4 inches of water per acre, and crop C yields 50 bushels per acre and requires 4 inches of water per acre. The farmer expects crop A to yield a profit of \$1.00 per bushel, crop B a profit of \$1.20 per bushel, and crop C a profit of \$2.00 per bushel. Solve for the land allocation to grow crops for maximum profit.

Happy Hacking!