

SIMPLEX IMPLEMENTATION
SA405, SPRING 2012
INSTRUCTORS: FORAKER AND PHILLIPS

For this assignment, you will finish an implementation of Simplex based on the code from the Rader textbook. The implementation will solve the problem:

$$\begin{aligned} \max \quad & c^\top x \\ \text{s.t.} \quad & Ax = b \\ & x \geq 0. \end{aligned}$$

Post your completed code to dropbox.

All code files should have your lastname followed by an underscore and then the requested function name. If you do not follow this convention, your code will not be graded. Make sure your function name and file name are identical or they may not work correctly.

E.g., Commander Foraker would turn in:
foraker_plex.m, foraker_FindImpDir.m, foraker_FindStepSize.m, and foraker_UpdateSol.m

Before beginning the assignment, download the files, USNAplex.m and PhaseOne.m, from dropbox. Change the file and function name to **your lastname_plex.m** and **your lastname_plex**. (put your actual lastname in for **your lastname**). Here is a description of the variables used:

- **A, b, c**: These are the constraint matrix, right-hand side vector, and objective function coefficients, respectively. The constraint matrix should be full-row rank.
- **initsol, initB**: These are optional arguments to be used if an initial solution and corresponding basis is known. Otherwise, a phase 1 LP will be formed and called.
- **R**: This is the return data structure, a matlab struct with several fields. See the comments in the code for the various fields it contains.
- **x**: This is the current solution vector.
- **B**: This is a boolean vector indicating whether a variable is basic or not.
- **done, unbdd**: Boolean variables indicating whether the main loop is complete and the linear program has an unbounded optimal solution.
- **iters**: A variable that keeps track of the number of main iteration loops required.
- **d**: The vector with a simplex direction.
- **p**: The variable index corresponding to **d**.
- **stepsize**: The maximum value that **d** can be scaled by and added to **x** and still have a feasible solution.
- **leavingind**: The index of **B** that corresponds to the leaving variable. Note that this is **not** (in general) the same as the variable index. See the example below for clarification of this.

The basis data structure The basis data structure allows the basis matrix and basic components of the current solution to be referenced. The boolean vector is set so that each component indicates whether the corresponding variable is basic or not. E.g., if

$$A = \begin{bmatrix} 1 & 2 & 5 & 0 \\ 3 & 2 & 1 & 5 \end{bmatrix} \quad b = \begin{bmatrix} 5 \\ 7 \end{bmatrix}$$

then setting

$$B = \begin{bmatrix} false \\ false \\ true \\ true \end{bmatrix}$$

would mean a basic matrix

$$Bmat = A(:,B) = \begin{bmatrix} 5 & 0 \\ 1 & 5 \end{bmatrix}$$

and a basic solution found via the commands $\mathbf{x} = \mathbf{zeros}(4, 1)$, $\mathbf{x}(\mathbf{B}) = \mathbf{Bmat} \backslash \mathbf{b} = \mathbf{Bmat}^{-1} * \mathbf{b}$ so that

$$\mathbf{x} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 6/5 \end{bmatrix}.$$

Note that \mathbf{Bmat} is the submatrix of \mathbf{A} corresponding to the columns indicated by the basis vector \mathbf{B} . Suppose x_2 is a desired entering variable. We form the associated simplex direction by first setting $\mathbf{d} = \mathbf{zeros}(4, 1)$ and then $\mathbf{d}(2) = 1$. Then, we must solve the equation

$$\mathbf{A} \begin{bmatrix} 0 \\ 1 \\ d_3 \\ d_4 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

which can be done via the matlab command $\mathbf{d}(\mathbf{B}) = -\mathbf{Bmat} \backslash \mathbf{A}(:, 2)$ which results in

$$\mathbf{d} = \begin{bmatrix} 0 \\ 1 \\ -2/5 \\ -8/25 \end{bmatrix}.$$

To find the leaving variable and step size, we need to determine the maximum λ such that

$$\mathbf{x} + \lambda \mathbf{d} \geq 0 \Leftrightarrow \lambda = \min\{-\mathbf{x}(\mathbf{i})/\mathbf{d}(\mathbf{i}) \mid \mathbf{d}(\mathbf{i}) < 0, \mathbf{B}(\mathbf{i}) = \mathbf{true}\}.$$

In matlab, we will use the more descriptive `stepsize` instead of λ . To calculate λ , we first form the boolean vector indicating the “good indices”: find the limiting value via `goodinds = boolean(B .* (d < -.001))`. Note that this command uses a few matlab tricks:

1. the `boolean` “casting” function which assures that `goodinds` is a boolean vector;
2. the expression, `d < -.001`, which creates a vector with boolean variables set to true when `d(i) < -.001`. Note that `-0.001` is used so as to avoid precision errors; and
3. component-wise multiplication, `.*`, which produces a vector where each component is the product of the boolean variables from the vectors `B` and `d < -.001`.

Then, we set `stepsize = min(-x(goodinds) ./ d(goodinds))`. We can, additionally, determine the leaving variable by setting `leavingvar = find(-x ./ d == stepsize)`.

Finally, to update the basis, we set `B(2) = true` and `B(leavingvar) = false`.

Complete the following:

1. **Implement the FindImprovingDirection function.** This takes, as input, the constraint matrix, `A`; the current basis, `B`; the objective coefficient vector, `c`; and the current solution, `x`. As output, the function should return an improving direction vector, `d`; the entering variable index associated with the simplex direction vector, `enteringvar`; and a boolean variable set to true if there are no improving simplex directions, `done`. Note that the outputs corresponding to the direction vector and entering variable index should be set to some dummy values if there are no improving simplex directions.
2. **Implement the FindStepSize function.** This takes, as input a simplex direction, `d`; the current solution, `x`; and the current basis, `B`. As output, the function should return the maximum amount the entering variable can be increased and still be feasible, `stepsize`; the indice in the leaving variable, `leavingvar`; and a boolean variable set to true if the direction is unbounded (i.e., no maximum value exists), `unbounded`.
3. **Implement the UpdateSol function.** The inputs are the simplex direction vector, `d`, the stepsize, `stepsize`, the nonbasic variable to be added to the basis, `enteringvar`, the leaving basic variable index, `leavingvar`, the current basic solution, `x`, and the current basic array, `B`. The outputs are the updated basic solution, `newx`, and the updated basic array, `newB`.

4. Download the 20 test problems in the file, testprobs.mat, and **write a script to test your code with the word testscript plus your lastname in the filename.** The test problem input data is in the 20 by 25 by 100 array, `testA`; the 20 by 25 matrix `testb`; and the 20 by 100 matrix, `testc`; The `testubdd` and `testinf` vectors contain boolean variables indicating whether the problems are unbounded or infeasible, respectively. The optimal solutions and objectives are in the 20 by 100 matrix `testsol` and the 20 dimensional vector `testobj`. Use this data to test that your code is correct. You first check whether your problem is infeasible or unbounded and that this agrees with your results. If so, and if feasible and bounded, then check whether your optimal objective and solution agree with the data. Note that it may be the case your solution and/or objective differs by a small amount so you should be checking whether the difference is less than some small value such as .001 or .0001.