CO 353 Winter 2018: Project 3

Due: March 19 at 8pm

1 Objective

The project consists in implementing the dual simplex method for linear programming with general lower and upper bounds. The implementation must be *correct* and *robust* to the numerical inaccuracies of floating-point arithmetic. In particular, given $c \in \mathbb{R}^n$, $\ell \in \mathbb{R}^n$, $u \in \mathbb{R}^n$, $b \in \mathbb{R}^m$ and $A \in \mathbb{R}^{m \times n}$, a linear programming problem is formulated as

$$\begin{aligned} \min \quad & c^T x \\ \text{s.t.} \quad & Ax = b \\ & \ell \le x \le u \\ & x \in \mathbb{R}^n. \end{aligned}$$

If the problem is feasible, the implementation must find a basic solution that is (a) *feasible* within the tolerance ε, i.e., $\ell_B - \varepsilon \le x^*_B \le u_B + \varepsilon$ where x^*_B are the values of the basic variables, and (b) *optimal* within the same tolerance ε. Letting $\bar{c}$ be the reduced costs corresponding to the basic solution, (b) means that $\bar{c}_j \ge -\varepsilon$ if x_j is at lower bound, and $\bar{c}_j \le \varepsilon$ if x_j is at upper bound. We will use $\varepsilon = 10^{-6}$. Note that your code must implement the dual simplex method, so, for example, enumerating all the possible bases is not considered a correct implementation (it would be exceedingly slow).

This project can be completed in groups of 1 or 2. You can also form groups of 3, but *only* if you first get my approval by email (lpoirrier (at) uwaterloo) by March 5th. If you already had approval for a group of 3 for a previous assignment, then you are automatically allowed (but not required) to form *the same* group of 3 for this assignment (no need to contact me in this case).

2 Input and output

The input consists in $m \in \mathbb{Z}_+$, $n \in \mathbb{Z}_+$, $c \in \mathbb{R}^n$, $\ell \in \mathbb{R}^n$, $u \in \mathbb{R}^n$, $b \in \mathbb{R}^m$ and $A \in \mathbb{R}^{m \times n}$. They are given in a file of the form:

$$\begin{array}{llllll} m & n & & & & \\ c_0 & c_1 & c_2 & \dots & c_{n-1} & \\ \ell_0 & \ell_1 & \ell_2 & \dots & \ell_{n-1} & \\ u_0 & u_1 & u_2 & \dots & u_{n-1} & \\ a_{0,0} & a_{0,1} & a_{0,2} & \dots & a_{0,n-1} & \\ a_{1,0} & a_{1,1} & a_{1,2} & \dots & a_{1,n-1} & \\ \vdots & & & & & \\ a_{m-1,0} & a_{m-1,1} & a_{m-1,2} & \dots & a_{m-1,n-1} & \\ b_0 & b_1 & \dots & b_{m-1} & & \end{array}$$

All the numbers given in input will have an absolute value smaller than 10^9, and $\ell_j \le u_j$ for all $j = 0, 1, \dots, n-1$.

The output file contains two lines (aside from optional comment and empty lines that are ignored, see below). The first line is either the word `infeasible` if the problem is infeasible, or the word `optimal` if the problem is feasible. The second line contains $n+m$ numbers that can be -1, 0, or 1. Each of these numbers describes the status of the corresponding column in the final tableau: -1 means at lower bound, 1 means at upper bound, and 0 means basic. If the problem is feasible, then this tableau must be optimal. Otherwise, it is the tableau that is in use when the ratio test detects infeasibility.

Note that there are $n+m$ columns and not just n: indices $n, \ldots, n+m-1$ correspond to artificial variables fixed to zero that are used to create a starting basis. They are the z variables in the equivalent problem

$$\begin{aligned} \min \quad & c^T x + 0^T z \\ \text{s.t.} \quad & Ax + Iz = b \\ & \ell \le x \le u \\ & 0 \le z \le 0 \\ & x \in \mathbb{R}^n, z \in \mathbb{R}^m. \end{aligned}$$

In order to (optionally) let you include more information in the output file, everything between a # sign and the end of a line will be ignored. Empty lines are also allowed and ignored.

Example input and output files are given at `https://www.math.uwaterloo.ca/~lpoirrie/co353.html`.

3 Running the code

Your code must take exactly two arguments. The first argument is the name of a file containing the input linear programming instance. The second argument is the name of a file where the output is to be written.

You have a choice between the following languages for the implementation: C, C++ or Python.

3.1 In C or C++

Your code must be portable to any environment with a standards-conforming C or C++ compiler. It must be possible to compile and run it by using the following commands:

in C:

```
gcc -O3 -o dsm dsm.c
./dsm input.txt output.txt
```

in C++:

```
g++ -O3 -o dsm dsm.cpp
./dsm input.txt output.txt
```

In other words, you must implement your code in a file called `dsm.c` (in C) or `dsm.cpp` (in C++). DSM stands for Dual Simplex Method.

Note: Optionally, you may provide a `Makefile`, in which case your code will be compiled by running `make`. The resulting executable must be named `dsm`. If you choose to do this, it is your responsibility to ensure that the `Makefile` is correct, so do it only if you are familiar with `Makefile`s already.

3.2 In Python

It must be possible to run your code by executing the following command:

```
python dsm.py input.txt output.txt
```

In other words, you must provide a source file called `dsm.py` (if you want, `dsm.py` can import code from other files). DSM stands for Dual Simplex Method. If you need a specific version of the Python language, include the string "python2" or "python3" somewhere in `dsm.py`, for example in a comment.

4 Submitting the project

You submit your implementation by sending a single email to both `lpoirrier (at uwaterloo)` and `wjtoth (at uwaterloo)`, by 8pm on Monday, March 19th, 2018. Late submissions will not be accepted. The subject line of your email must contain the string `CO353`. Your email must contain **a single attachment file**: an archive in the format `.zip` or `.tgz`. The name of the archive is formed by the UWaterloo IDs of your group members, in any order, separated by underscores. For example: `jwtoth_lpoirrie.tgz`. The archive contains (at least) two files:

- a source file for your implementation, and
- a file called `notes.txt` (or `notes.md` if you prefer).

The file `notes.txt` only specifies the names of the (1, 2 or 3) members of your group. No further explanations are required for this assignment (although you are allowed to put explanations there if you wish to highlight something you did particularly well in your code).

5 Contribution

You can use the standard library in your language of choice. **In addition, you are allowed (and strongly advised) to use an external library that handles linear algebra**, in particular matrix operations. For C++ you can use Eigen, for Python you can use NumPy. Do not include these libraries in the archive that you send by email: you can assume that they are installed and available (for example, `#include "Eigen/Dense"`, `#include <Eigen/Dense>`, `import numpy` and `from numpy import *` will all work in their respective languages). If you prefer, you can choose another library for linear algebra (i.e. not Eigen or Numpy), but **only if** you first get my approval by email by March 5th (in particular, libraries that feature an implementation of the simplex method will not be allowed).

Aside from that, no other library is allowed. In case of doubt ask me. You can consult any source of information you want (books, internet, scientific papers, etc.), but copying code is not allowed, regardless of the source. Every single line of your code must be written by a member of the group. Each member of the group must understand (and must be able to justify) every line of the code, including the ones they did not write. After the due date, I may

call an individual group member to come to my office hours; if that member does not show sufficient understanding of the code, *all* group members will get zero.

6 Grading criteria

The project will be graded on a total of 15 marks, divided in 5 categories. Pay attention to the first one: if your code does not follow the specification, it will not be tested, and you will get zero for all the rest.

- Specification (1 mark). Your submission must follow the specifications given in this document (your email must have the correct recipients/subject/attachment, your source files must follow the template given above, your code must compile and take the right arguments). The rules are rigid because testing will be automated.
- Code quality (2 marks). It should be reasonably easy to understand your code. Comments can be used to help in that regard, but with moderation. The code itself must be clear and readable.
- Correctness (4 marks). Your code must be an implementation of the dual simplex method with general lower and upper bounds. When the LP instance given as input is feasible, the output must describe an optimal solution. When the LP instance is infeasible, this must be correctly identified.
- Robustness (4 marks). Your code must be able to deal with numerical issues arising in larger instances (i.e., floating-point inaccuracies). All floating-point operations must take those issues into account, and handle them properly.
- Efficiency (4 marks). Your code must not be more than 100× slower than a reference implementation in the same language.