OA - Homework 1

Gabriel BATHIE

I) Detail of the project

A) Source files

In the `src/` folder :

- `main.py`: Contains the main function, entry point of the program
- `parsing.py` : Contains the functions that parse command line arguments or files containing the LP
- `simplex.py` : Contains the functions of the simplex algorithm, used to solve the LP
- `rules.py` : Contains the implementation of the different rules
- `display.py` : Contains the functions to print nicely a LP/dictionary
- `random_generator.py` : Generator of random LPs

B) Other files

Example files are distributed between the `mandatory-tests/`, `interesting/` and `random/` folder. Their content is discussed in the `bathie-discussion` file.

C) Data Structure for LPs

The LPs are not stored in a class, they are stored in `numpy` arrays of `Fraction`s.
One array for `A`, the constraint matrix, one array for `b`, one for `c` (usually named `obj` throughout the code, for "objective function"), and one value `objv` which correspond to the current value of the objective function. These are equivalent to the usual "tableau", but split into four parts.

Additional Remark : In my program, the variables and constraints are indexed starting at 0. Therefore, the output will begin with `x_0 = ...` and not with `x_1 = ...`.

II) Features

A) Rules available :

- **Maximum coefficient rule**
- **Bland's rule**
- **Random positive pivot rule** :
  This rule first checks whether the objective function is unbounded.
  If it is not, it selects uniformly at random a variable with a strictly positive coefficient in the objective function.
  It breaks ties for the leaving variable randomly.
- **Custom rule : Accelerated Bland's rule**
  This rule makes the greatest increase possible in the objective function.
  If we can only increase by zero (we are at a degenerate point), we use Blands's rule.
  This ensures that we do not cycle : strict increase cannot cycle, and Blands's rule does not cycle, and yields better results (see discussion for more details).

B) Execution and Command Line Options

Running the program :

Run the program as follows :
`python3 main.py input_file [-v] [-d] [-m/b/r/c] [-o filename]`.
The arguments are :

- `input_file` : name of the file where we should read the input linear program. **It has to be the first argument.**
- `-v` : set verbose mode
- `-m` : use Maximum Coefficient rule
- `-b` : use Bland's rule
- `-r` : use Random rule
- `-c` : use Custom rule (Accelerated Bland's Rule)
- `-o filename` : if specified, the output is written to the file `filename`, overwriting its content. If not specified, the output is written on standard output
- `-d` : Also solve the dual, using the same parameters (as a "negative maximization" problem, therefore the result (if finite) is the opposite of that of the primal).

If no rule is specified, the Random rule is used. If more than one rule is specified, the choice is made arbitrarily.

Generating a random LP :

Generate a random LP as follows : `python3 random_generator.py N M filename [b0 [b1 [d]]]`

- `N` : number of variables in the LP
- `M` : number of constraints in the LP
- `filename` : number of constraints in the LP
- `b0` : optional if the next are not given. Lower bound for the coefficients, default to 0
- `b1` : optional if the next are not given. Upper bound for the coefficients, default to 10
- `d` : optional. Upper bound for the denominator : the denominator is in the interval `[1, d]`. Default value is 1 (and therefore the output is only integers).

The order of the parameters is very important.

Example : `python3 random_generator.py 30 10 test.dat -10 10 1000`

III) Dependencies

The program was tested with `python 3.6.8`.
The program uses the following Python3 modules : `numpy` (tested with `numpy 1.17.2`), `fractions`, `random` and `sys`.