Coursework

This coursework is linked to the Simplex Method. In theory the worst case complexity of the Simplex Method is $\mathcal{O}(n!)$. That is, the computational cost grows factorially with the number of decision variables. However, in practical cases the complexity is often much less. Here we will study how much less.

You will

1. code, in Python, alternative pivoting rules for the Simplex Method;
2. run the different rules on multiple different linear programs with different numbers of decision variables, outputting the run-time data as an Excel spreadsheet;
3. within Excel, compute the average and worst case run-time as a function of the instance size;
4. within Excel, produce plots showing (quantitatively) the complexity;
5. within LaTeX, write a report analysing the pivoting rules, including your analysis using tables and plots.

Algorithms

The Simplex Method performs pivoting operations swapping an index between the basic set B and its complement N. The pivoting rule chooses a (row, column) pair (r, s). Given that pivot entry, row operations are performed so that the tableau entry $\bar{a}_{rs} \to 1$, and all other entries in that column, $\bar{a}_{is}, i \neq r \to 0$.

In the theory part of the course the pivoting rule shown was *Bland's rule*, also called the *smallest subscript* rule. This first chooses s to be the smallest (column) index with negative reduced costs $\bar{c}_s < 0$. The rule next chooses r to be the smallest (row) index that satisfies the minimum ratio test.

Largest change rule

In the *largest change rule*, we look at every column j with negative reduced costs $\bar{c}_j < 0$. Within each column, choose r_j to be the smallest (row) index that satisfies the minimum ratio test (as in Bland's rule). Then choose s to be the smallest (column) index such that the change in the objective function value $|\bar{c}_j \bar{b}_j / \bar{a}_{r_j,j}|$ is maximised.

Python

On `repl.it` you will find code implementing the (one phase) Simplex Method and Bland's rule.

The function `simplex` takes as input the tableau (in basic form), the initial basis (as a list of integer indexes), and a function that implements the pivoting rule. It returns the solution as a dictionary containing its status (did it succeed), the optimal solution, and the final basis.

The function `find_pivot_bland` implements Bland's rule. It takes as input the tableau, and returns the `(r, s)` pair giving the row and column indexes on which to pivot.

Task 1

At the top of the file there is an integer variable

```
student_id = 12345678
```

You **must** replace this number with your own eight digit ID. This is used to generate the problems the Simplex Method will solve.

Task 2

Using `find_pivot_bland` as a template, you should implement `find_pivot_largest_change`. A stub is given. The function should implement the largest change rule given above.

There are tests on `repl.it` that perform very basic checks of the code: they do not guarantee complete correctness.

Task 3

It is **essential** that you complete task 1 before doing this step.

At the top of the script there is a boolean variable set, as

```
run_timings = False
```

Once you are happy with your code, change this to

```
run_timings = True
```

Run your code using the `Run` button (not the `Run tests` button). This should take around one minute (on `repl.it`: times will vary on other machines). At the end it will have produced an Excel spreadsheet `timings.xlsx`.

Note: if you cannot complete task 2, or want to continue on with the analysis before completing it, at the top of the script there is a boolean variable

```
only_time_Bland = False
```

By setting this to

```
only_time_Bland = True
```

a spreadsheet creating the timing data just for Bland's rule will be produced.

Excel

The timing data in the Excel spreadsheet was created by looping over a range of problem sizes (quantified by the number of decision variables `n`; the number of constraints was set to `n+5`). For each problem size, 50 instances were created. For each instance the simplex method was run using both Bland's rule and the largest change rule.

The spreadsheet contains two sheets associated with the two pivoting rules. On each sheet, each row contains the times to solve each individual instance. The first column gives the instance size; the next 50 columns gives the times.

Task 4

Download the spreadsheet from `repl.it`. Load it into Excel. Create a new sheet in the workbook; call it `Analysis`. Using appropriate Excel functions, add five columns to this sheet. The first should contain the instance sizes `n`. The next four should contain the average time and worst-case time for each pivoting rule and instance size (`Smallest subscript, Average`, `Largest change, Average`, `Smallest subscript, Worst`, `Largest change, Worst`). From these, create five more columns containing the logarithms (base 10) of `n` and each time respectively.

Task 5

Create two plots. One should plot the (logarithm of the) average run time for each pivoting rule as a function of the (logarithm of the) instance size. The other should plot the (logarithm of the) worst case run time for each pivoting rule as a function of the (logarithm of the) instance size. Both plots should be appropriately labelled and include trend lines for each pivoting rule.

Save your figures in `pdf` format.

Task 6

Using appropriate functions (e.g., `INTERCEPT`, `SLOPE`), compute the best fit coefficients for

$$\log(t) = p_0 \log(n) + p_1,$$

where t is the timing data and n is the instance sizes. There should be different coefficients for each pivoting rule and for the average and worst cases (so four pairs of coefficients in total).

LaTeX

On Overleaf there is a report template. You need to complete the report to communicate your results.

Task 7

In the first section of the LaTeX document, include your Python code for the function `find_pivot_largest_change`.

Task 8

In the second section of the LaTeX document, using big $\mathcal{O}$ notation, analyse your Python code.

Task 9

In the third section of your LaTeX document, include the figures created in Excel. Add captions explaining what each is.

Also in the third section, add a table reporting the $p_{0,1}$ coefficients computed in Task 6.

Task 10

In the fourth and final section of the LaTeX document, write one paragraph to explain which of the pivoting rules is more efficient in which cases. You should use the results from the complexity analysis in task 8, and your figures and table in task 9, to back up your statements.

Submission

All work should be submitted **through Blackboard**.

Four "things" need submitting.

1. All the Python code used, as a `zip` file. If `repl.it` was used, go to `Files` (first item in the left vertical bar - looks like the "New Document" icon in Word). Click on the three dots menu. The third item should say `Download as zip`. This zip file should contain all Python scripts needed.
2. The Excel spreadsheet with the completed analysis.
3. All LaTeX and figure files needed to create the report, as a `zip` file. If Overleaf was used, go to the `Menu` (top left), and from `Download` select `Source`. This zip file should contain everything needed.

4. The report as a `pdf` file. If Overleaf was used, go the the `Menu` (top left), and from `Download` select `PDF`.

Marking scheme (rough)

- **[8 marks]** Python code runs correctly
- **[2 marks]** Python code documented correctly
- **[4 marks]** Excel data analysis organised correctly
- **[4 marks]** Excel figures created, labelled, with trend lines, correctly
- **[2 marks]** Intercept and slope used correctly
- **[4 marks]** LaTeX document compiles without errors
- **[2 marks]** Python code reported in document
- **[5 marks]** Complexity analysis correct
- **[2 marks]** Figures included correctly, with captions
- **[2 marks]** Table created correctly, with caption
- **[5 marks]** Efficiency argument reasonable, supported by evidence