# TP2 – How data structures affect models

The goal of this practical is to learn how to implement various python data structures (such as lists,sets and dictionary) in PULP models.

## 1 An illustration with the admissible cells problem

Given a rectangular grid $T^{m\times n}$ of boxes called *cells*. Some of these cells are *admissible*, the others are *non-admissible*. Also are given $m+n$ non-negative integers $l_0 \dots l_{m-1}$, $c_0 \dots c_{n-1}$. The goal is to fill in the admissible cells with positive integers such as:

— The sum of the numbers allocated on the admissible cells on row $i$ should be less or equal than $l_i$;

— The sum of the numbers allocated on the admissible cells on column $j$ should be less or equal than $c_j$;

— The total sum of all these numbers should be maximum.

**Example:** Let:

| | | | | | |
|---|---|---|---|---|---|
| $(m=4)$ | $l_0=9$ | $l_1=10$ | $l_2=15$ | $l_3=2$ | |
| $(n=5)$ | $c_0=7$ | $c_1=5$ | $c_2=9$ | $c_3=4$ | $c_4=8$ |

and the set of admissible cells is:

$$A=\{(0,0),\,(0,1),\,(1,0),\,(1,2),\,(1,3),\,(2,1),\,(2,4),\,(3,2),\,(3,4)\}$$

where the first index corresponds to the row, second index to the column number.

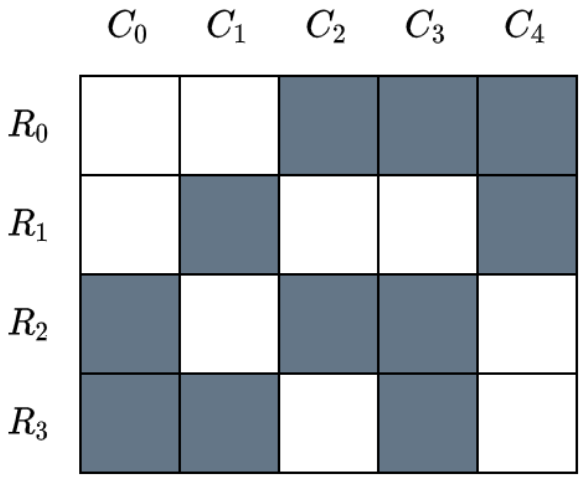


**Figure 1:** Only the blank cells can be filled with a value. $l_i$ is the limit value of the sum of cells in row $i$, $c_j$ is the limit value of the sum of cells in column $j$

# 2 Understanding and extending models using different data structures

The `TP2` folder contains four python files :

1. `tp2_admissible_cells_array_partial_correction.py`,
2. `tp2_admissible_cells_set_partial_correction.py`,
3. `tp2_setting_input_data.py`
4. `tp2_read_data_files.py`.

The first two programs propose two different approaches for implementing the admissible cells problem. The first code uses data structure `list`, while the second one uses data structures `set` and `dictionary`. These programs are incomplete, some constraints are missing.

**Question 1.**

Your first task is to read and understand them. Run them with:

```
(.venv_3) python3 tp2_admissible_cells_array_partial_correction.py
```

```
(.venv_3) python3 tp2_admissible_cells_set_partial_correction.py
```

What is the solution?

**Question 2.**

What are the missing constraints? Add them and run the programs again. Explain the changes made to the calculated solutions as a result of adding new constraints.

**Question 3.**

Figures 2 and 3 illustrate the two types of diagonal of a matrix (the positive diagonal 2 and the negative diagonal 3).

1. Give the linear constraints that model any of the above requirements.
2. Choose one the previously described data structure and add these constraints in your PULP program.

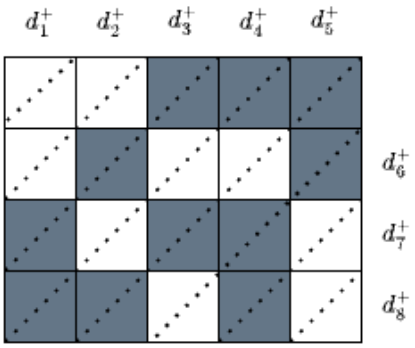


**Figure 2:** The sum of the numbers allocated on the admissible cells on the positive diagonal $i$ should be less or equal than the given number $d_i^+$;

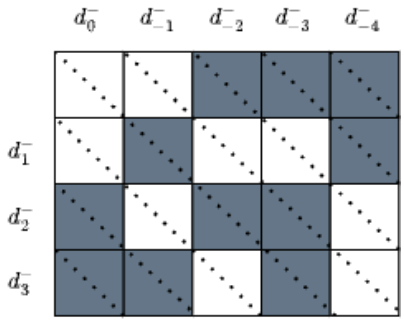


**Figure 3:** The sum of the numbers allocated on the admissible cells on the negative diagonal $i$ should be less or equal than the given number $d_i^-$

The last two codes contain functions for entering input data for eligible cells. The file `tp2_setting_input_data.py` describes the function `get_adm_cells_data_bis` which is used to enter input data in the same way as in TP1. This is the function that is used by default in the above programs.

The file `tp2_read_data_files.py` is a FileReader API. It codes functions to extract admissible cells from files in `data` directory, and a function to parse command line's argument. It is not necessary to understand how they work, you need only to use them.

Especially, the function `get_adm_cells_data` lets you iterating over it to get admissible cells data from each file. You can use it as following:

```
for adm_cells, row_limits, col_limits in get_adm_cells_data():
    # adm_cells: list of tuple (row_i, col_j)
    #   corresponding to admissible cells
    # row_limits: list of row limits (int)
    # col_limits: list of column limits (int)
    solve_admissible_cells(adm_cells, row_limits, col_limits)
```

In fact, `get_adm_cells_data` *yield* the three lists instead of returning them. It allows to iterate over the function in a stream way.

**Question 4.**

Find how and where these functions can be used in the above codes. Make the necessary in order to use them. Run them with:

```
(.venv_3) python3  tp2_admissible_cells_array_partial_correction.py
```

and

```
(.venv_3) python3  tp2_admissible_cells_set_partial_correction.py
```

What is the solution?

**Question 5.**

In order to test more complex instances, you can run your program like this:

```
(.venv_3) python3 your_prog.py --all-data > tp2_all_results.log
```

Run the program for all the data and plot the CPU time according to the number of variables, and according to the number of constraints, with the method of your choice.

**Tip:** you can use `CTRL+F` on `tp2_all_results.log` and search for `FILE` word in order to find very quickly statistics.