3.6 Documentation of optimization tool

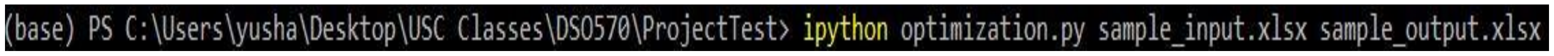
This document describes how to use our team's optimization tool and reproduce the sample output file using our sample input. In addition, it documents the appropriate format of the user-specified input file and the view of the returned output file.

Please note that for the reproduction of the demonstration shown below, all respective files ('optimization.py', 'sample_input.xlsx' and 'sample_output.xlsx') should be contained in the same local repository like a folder on your personal computer.

Using the optimization tool

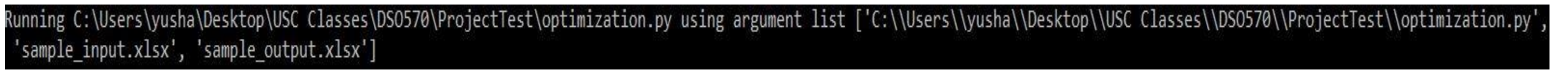
This document outlines two possible ways to run our team's optimization file, both of which require Python 3.6+ with base packages and Gurobi installed or in an operational state. The process for each, using sample input and output files, is illustrated as follows:

1) *Using command line*
    a. Open Anaconda Prompt/PowerShell Prompt (Windows) or Terminal (Mac/Linux)
    b. Navigate to your local repository that contains the respective files
    c. Type the following command **'ipython optimization.py sample_input.xlsx sample_output.xlsx'**. For a general form, replace the previous command with **'ipython optimization.py inputFile outputFile'**, where *inputFile* and *outputFile* indicate input and output filenames, respectively. Before running this command, make sure that your interface looks similar to this – with your local repository located on the left side:



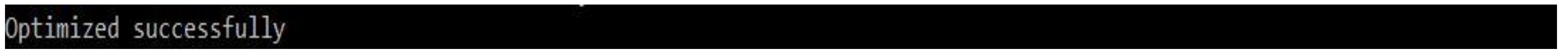
```
(base) PS C:\Users\yusha\Desktop\USC Classes\DSO570\ProjectTest> ipython optimization.py sample_input.xlsx sample_output.xlsx
```

    d. Run the command, seconds after which you should see the following message - with your local repository identified in the first argument in the list of arguments:



```
Running C:\Users\yusha\Desktop\USC Classes\DSO570\ProjectTest\optimization.py using argument list ['C:\\Users\\yusha\\Desktop\\USC Classes\\DSO570\\ProjectTest\\optimization.py',
 'sample_input.xlsx', 'sample_output.xlsx']
```

    e. Thereafter, the command will be executed for quite some time. After the execution is completed and if everything was input correctly and in accordance with the previous steps, you will see the following message – indicating that the optimization has been done successfully and the output file has been created:

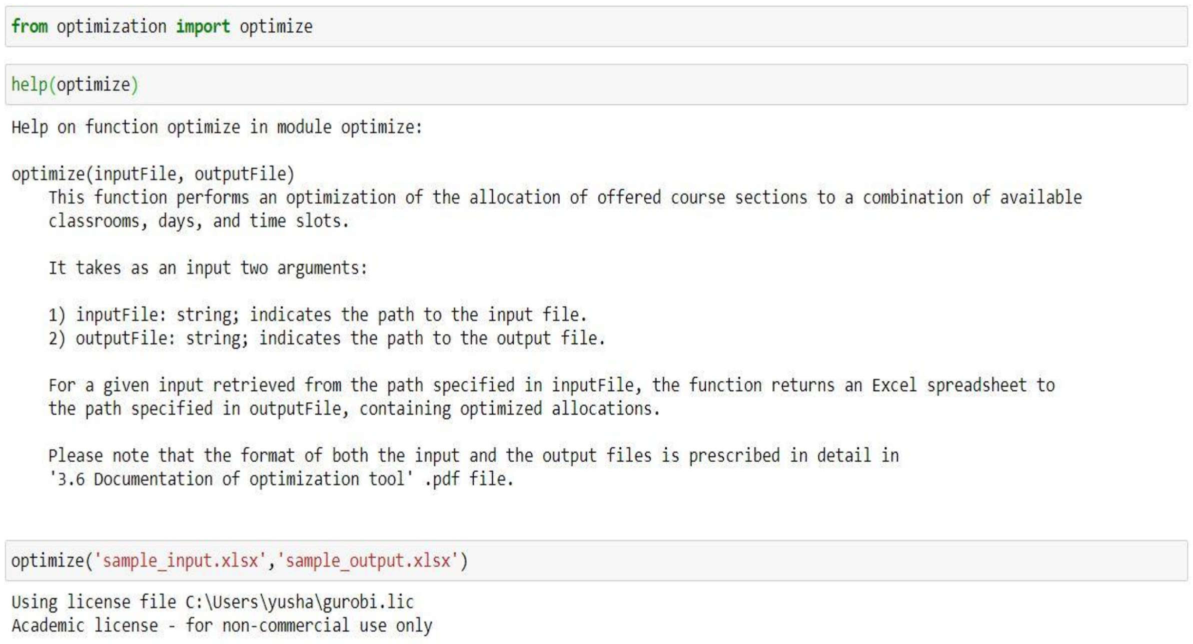


```
Optimized successfully
```

2) *Using Jupyter notebook*

   a. Open your Jupyter dashboard and navigate to your local repository that contains the respective files
   b. Open a new Jupyter notebook
   c. In a new cell, type and run **‘from optimization import optimize’**. This will allow the contents of the optimization tool, namely the optimization function, to be accessed by Python in its global environment.
   d. (Optional) To refer to the in-built documentation of the optimization function, you can run the following line of code: **‘help(optimize)’**.
   e. After you successfully perform the importing, type and run the following command **‘optimize(‘sample_input.xlsx’,’sample_output.xlsx’)’**. For a general form, replace the previous command with **‘optimize(inputFile, outputFile)’**, where *inputFile* and *outputFile* indicate paths to input and output files, respectively. After the code is done running, you will be able to locate the output file in your local repository.

You are welcome to refer to the following illustration of implementation of steps c-e:



```
from optimization import optimize
```

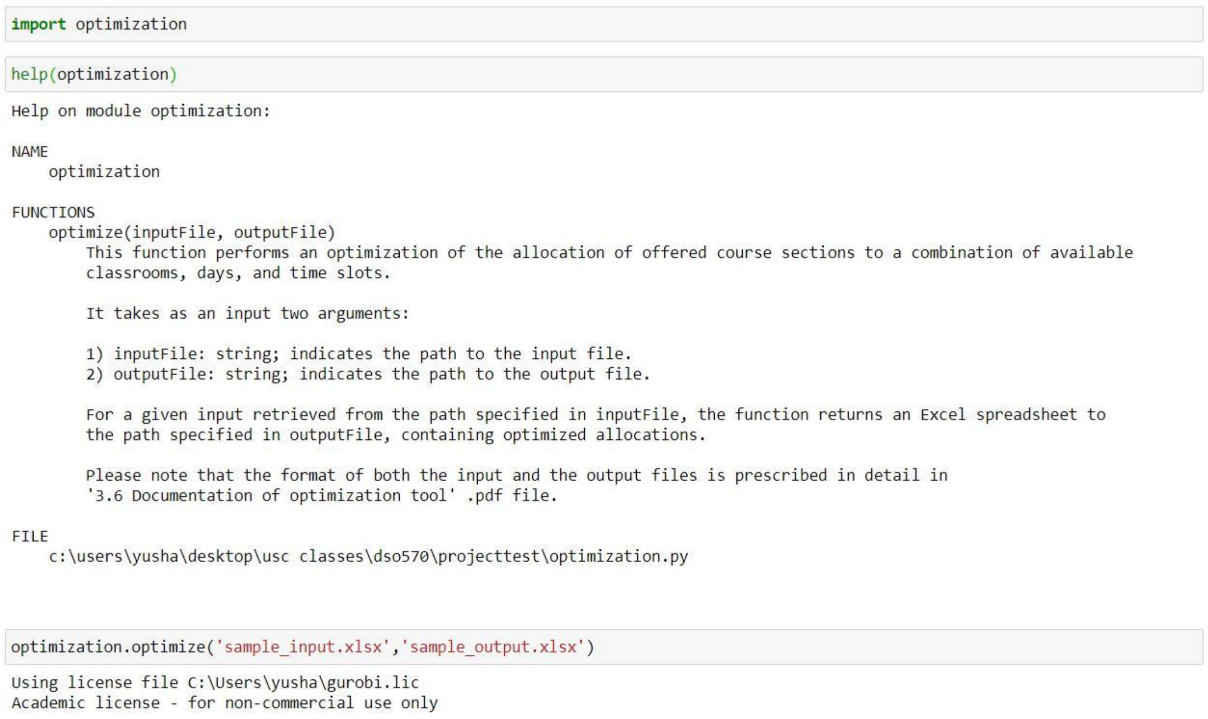
```
help(optimize)
```

```
Help on function optimize in module optimize:

optimize(inputFile, outputFile)
    This function performs an optimization of the allocation of offered course sections to a combination of available
    classrooms, days, and time slots.

    It takes as an input two arguments:

    1) inputFile: string; indicates the path to the input file.
    2) outputFile: string; indicates the path to the output file.

    For a given input retrieved from the path specified in inputFile, the function returns an Excel spreadsheet to
    the path specified in outputFile, containing optimized allocations.

    Please note that the format of both the input and the output files is prescribed in detail in
    '3.6 Documentation of optimization tool' .pdf file.
```

```
optimize('sample_input.xlsx','sample_output.xlsx')
```

```
Using license file C:\Users\yusha\gurobi.lic
Academic license - for non-commercial use only
```

Note that you can also import the whole module instead and redo steps c-e using the following, slightly modified commands:



```
import optimization
```

```
help(optimization)
```

```
Help on module optimization:

NAME
    optimization

FUNCTIONS
    optimize(inputFile, outputFile)
        This function performs an optimization of the allocation of offered course sections to a combination of available
        classrooms, days, and time slots.

        It takes as an input two arguments:

        1) inputFile: string; indicates the path to the input file.
        2) outputFile: string; indicates the path to the output file.

        For a given input retrieved from the path specified in inputFile, the function returns an Excel spreadsheet to
        the path specified in outputFile, containing optimized allocations.

        Please note that the format of both the input and the output files is prescribed in detail in
        '3.6 Documentation of optimization tool' .pdf file.

FILE
    c:\users\yusha\desktop\usc classes\dso570\projecttest\optimization.py
```

```
optimization.optimize('sample_input.xlsx','sample_output.xlsx')
```

```
Using license file C:\Users\yusha\gurobi.lic
Academic license - for non-commercial use only
```

Description of input and output file formats

Input

The input file (‘sample_input.xlsx’ in the illustrations above) should be an Excel workbook of .xlsx file type, containing three sheets: **‘course’**, **‘classroom’**, and **‘timeslot’**.

The first sheet, **‘course’**, should list the information on offered course sections for a given semester. Structure-wise, the first field should be an empty-labeled list of section indices ranging from 0 to the (number of course sections - 1). Among the necessary fields that the sheet should contain, without having which the optimization tool will produce an error message, are:

- **‘section’** - indicating the section of the offered course
- **‘length’** - indicating the class length of the course in minutes
- **‘frequency’** - indicating how many times the course meets during a week
- **‘semester’** - indicating whether the course is offered for first-half, second-half or full duration of the semester
- **‘first_instructor’** - indicating the instructor for the course
- **‘0**’, **‘1’**, **‘2’**, … (**‘number of available time slots - 1’**) - indicating the total preference for the course during the specified 0th, 1st, 2nd, etc. time slot (where **‘0’** indicates the earliest time slot – as implied from the **‘timeslot’** sheet). Preferences should be ranked ascendingly, where the smallest value would

denote least desirability and the greatest value would denote greatest desirability, and should be expressed in numbers.

The second sheet, **‘classroom’**, should list the information on all available-for-use classrooms. Structure-wise, the first field should contain the set of all available-for-use classrooms (e.g., JKP 102). Second field, named **‘capacity’**, should indicate the maximum capacities for the respective classrooms. Although the capacity constraint is not defined and integrated in the latest version of our optimization tool, it is highly recommended that the capacity data is provided, so that future implementations can account for the new constraint.

The third and final sheet, **‘timeslot’**, should list the information on all 90-minute-long, available-for-use during all working days (between Monday and Friday) time slots. Structure-wise, the first field should contain list of time slot indices ranging from 0 to the (number of time slots - 1). Second field, named **‘time’**, should indicate the timeslots themselves, sorted from the earliest to the latest time slot.

Output

The output file (‘sample_output.xlsx’ in the illustrations above) is an Excel workbook of .xlsx file type, containing only one sheet: **‘Schedule’**. The **‘Schedule**’ sheet serves as a timetable representation of course section assignments to the respective classroom during the respective time slot on the respective day of week.

Rows of this timetable contain the set of time slots that can be found in **‘time’** field of **‘timeslot’** sheet of the input file. On the other hand, columns are condensed in a multi-index format. Upper layer of the multi-index object contains the set of available classrooms that can be found in the first field of **‘classroom’** sheet of the input file. Lower layer contains the set of days of week for each of the classrooms in the upper layer.

If present, the value for a given combination of row and columns simply denotes the course section assigned to the time slot in that row, and the classroom and the day of week in these columns. If the cell is empty, however, there were no allocations deemed optimal for that cell’s classroom, day of week and time slot.