Developing interfaces for the TRANUS system

Julien Armand

Report for M1 TER internship

Developing interfaces for the TRANUS system

Julien Armand
·
Supervised by: Peter Sturm

June 2016

Abstract TRANUS is a software used to simulate the impact of plans of urban development. It uses an extensive array of parameters to simulate the environment in a mathematical model. It is however difficult to use, especially when wanting to study the impact of one parameter on the rest of the model. Doing so currently requires to enter a new set of data through the graphical user interface, even if only one variable is modified. In order to use the software to study the impact of one variable on the model, a new tool making this easier is required.

Keywords TRANUS · Interfaces · INRIA · Python

J. Armand
4, rue Prosper Mérimée 26100 Romans-sur-Isère
Tel.: 06 19 31 81 49
E-mail: julien.armand26@gmail.com

1 Introduction

1.1 INRIA and STEEP

INRIA (National Institute for Research in Computer Science and Automation) was founded in 1967. With over 2700 employees and eight centers of research, it is dedicated to both theoretical and applied research in computer science.
INRIA has established partnerships with many other centers of research, both national and international. These include the University of Illinois, Microsoft Research, and the Chinese Academy of Sciences.
INRIA is also, with Pierre and Marie Curie University and Paris Diderot University, one of the founders of IRILL (*Initiative pour la Recherche et l'Innovation sur le Logiciel Libre*), a research center created in September 2010.
I spent my internship with the STEEP (Sustainability Transition, Environment, Economy and local Policy) research team, located at the Grenoble - Rhne-Alpes INRIA research center.

1.2 TRANUS

Developed by Modelistica since the 80s, TRANUS is described as an "Integrated Land Use and Transport Model". It links all the parameters (transport, land use, activities ...) together through a specifically designed theory, which allows the user to simulate the effects of urban policies.
According to its developers, TRANUS can be applied to detailed urban areas; metropolitan areas; metropolitan regions; regions, states or provinces; national level and regions made of several countries. The area of studies is divided in socioeconomic sectors of different types(residence, industry, services ...). During my internship, the models I used for testing were of the Grenoble agglomeration.
TRANUS is composed of two parts. The first is the TRANUS User Shell (TUS), which is a graphical interface and database for the configuration of the model. The second is an ensemble of binary programs that perform the calculations from the data generated by TUS and can be run either through it or through the console/command window.

1.3 My contribution

The goal of my internship was to simplify the use of TRANUS by creating an interface capable of executing TRANUS multiple times with minute variations to a single variable of the model, and to enable the user to study the impact of this variation. In order to do that, I had to understand how the various elements of TRANUS interact with each other, as well as the format of the files it uses. I also often spoke with the rest of the team, who use TRANUS in their own work, in order to learn what would be most useful to them. In the end, I developed two separate interfaces in Python. One simplifies the execution of the TRANUS binaries, and the other allows to execute multiple iterations of TRANUS with variations and study the impact of the modification.

2 First interface: automating of TRANUS' execution

2.1 Objectives

The first part of my internship was dedicated to the design and programming of an interface that would allow the execution of the TRANUS binary programs without the need to use the console or TUS. It also had to allow the simultaneous execution of several of these binaries, with several options and for different scenarios of the same project. This was for me to learn how to handle TRANUS and control it through a Python script, as well as how to code interfaces with it.

2.2 Location of necessary files: the configuration file

The first part of my work was to identify what any program interacting with TRANUS would need to know in order to function. This was the location of the TRANUS binaries, the location of the TRANUS project, and its name. For both interfaces I developed, this information is contained within a configuration file of the following format:

Path to the directory containing the TRANUS binaries
Path to the directory containing the TRANUS project
Identifier of the project
Identifier of the scenario

During development, I encountered difficulties when testing my code on machines with different OS. This was because of the method employed to read the configuration file, which resulted in OS-specific characters being added to the paths upon extraction. This problem was solved by using the `readLines` method to extract the data, which works the same with all OS.

2.3 Extraction of the project's scenarios

TRANUS functions with projects, each corresponding to a particular environment (i.e, Grenoble and its surroundings) and with a list of scenarios corresponding to specific situations. The list of the scenarios of a project is found within its `W_TRANUS.CTL` file, in the project's main directory.

The first interface needed to be able to execute the binaries for each scenario in the project. For that, the program needed to know what scenarios existed.
To that end, I wrote a script in that extracts the list of scenarios of the project indicated by the configuration file. Named **extractionsScenarios.py**, it creates an object `Scenarios`, made up of a list of names and IDs of the project's scenarios.

Example of use of **extractionScenarios.py**:

```
>>> filepath = ("ExampleC/W_TRANUS.CTL")
>>> r = extractionScenarios(filepath)
>>> r.listCodes
>>> ['03A', '08A','08B','13A','13B']
```

2.4 Design of the Graphical User Interface and selection of useful options

There exists a total of 13 binaries for TRANUS, each having several options. After discussing with the rest of the team, it was decided to limit the interface to the following: **imploc**, **imptra**, and **mats**.

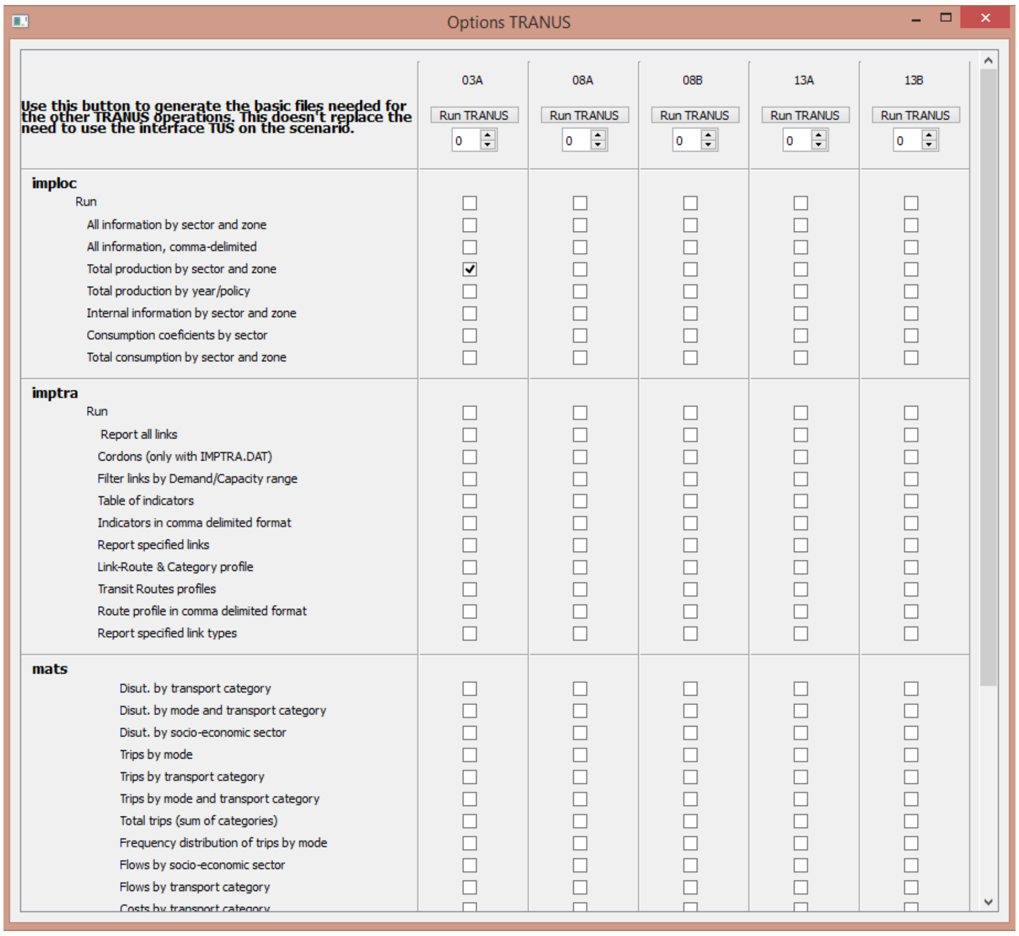
My interfaces were developed using the Python library PyQt4, though I first designed the skeleton of both interfaces using Qt Designer, a software tool allowing to create interfaces more easily than through simple coding. With it, I created the python class `OptionsTRANUSUI`, which was then inherited by the class actually implementing the interaction: `OptionsTRANUS`. This class uses `extractionScenarios` to know how many scenarios there are in the project, and creates the interface with the corresponding number of columns through a **while** loop in its constructor. You can see the interface itself in the figures 1 and 2, on the next page.

2.5 Execution of TRANUS programs with options selected

For each selected binary, there is a list of options that is proposed, as well as the ”Run” option, which is the basic execution. The user must check the boxes for the options he wants to use (the buttons ”Check All” at the bottom of the interface fill the boxes for their column).

Clicking on the button labeled ”Generate” executes the selected options. This is done through the `LcalInterface` class, which executes the TRANUS programs with the **subprocess** module. The button labeled ”Run TRANUS” at the top of each column activates the basic execution of TRANUS, with the value of the box below being used for one of the programs it executes.

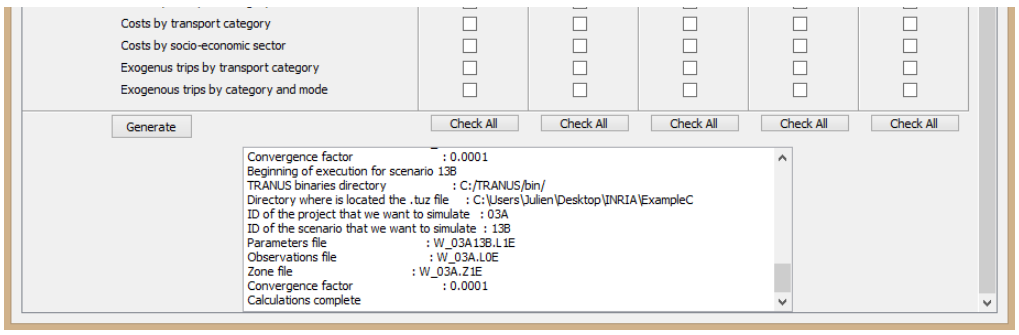
The files created by the interface are matrices, named after the program and option that created them and with the .MTX suffix. They are put in the corresponding scenario’s directory. The exact location is displayed in the interface’s log, alongside with various indicators of the process’ progress after launch.



Options TRANUS

Use this button to generate the basic files needed for the other TRANUS operations. This doesn't replace the need to use the interface TUS on the scenario.	03A	08A	08B	13A	13B
	Run TRANUS	Run TRANUS	Run TRANUS	Run TRANUS	Run TRANUS
	0	0	0	0	0
imploc					
Run	☐	☐	☐	☐	☐
All information by sector and zone	☐	☐	☐	☐	☐
All information, comma-delimited	☐	☐	☐	☐	☐
Total production by sector and zone	☑	☐	☐	☐	☐
Total production by year/policy	☐	☐	☐	☐	☐
Internal information by sector and zone	☐	☐	☐	☐	☐
Consumption coeficients by sector	☐	☐	☐	☐	☐
Total consumption by sector and zone	☐	☐	☐	☐	☐
imptra					
Run	☐	☐	☐	☐	☐
Report all links	☐	☐	☐	☐	☐
Cordons (only with IMPTRA.DAT)	☐	☐	☐	☐	☐
Filter links by Demand/Capacity range	☐	☐	☐	☐	☐
Table of indicators	☐	☐	☐	☐	☐
Indicators in comma delimited format	☐	☐	☐	☐	☐
Report specified links	☐	☐	☐	☐	☐
Link-Route & Category profile	☐	☐	☐	☐	☐
Transit Routes profiles	☐	☐	☐	☐	☐
Route profile in comma delimited format	☐	☐	☐	☐	☐
Report specified link types	☐	☐	☐	☐	☐
mats					
Disut. by transport category	☐	☐	☐	☐	☐
Disut. by mode and transport category	☐	☐	☐	☐	☐
Disut. by socio-economic sector	☐	☐	☐	☐	☐
Trips by mode	☐	☐	☐	☐	☐
Trips by transport category	☐	☐	☐	☐	☐
Trips by mode and transport category	☐	☐	☐	☐	☐
Total trips (sum of categories)	☐	☐	☐	☐	☐
Frequency distribution of trips by mode	☐	☐	☐	☐	☐
Flows by socio-economic sector	☐	☐	☐	☐	☐
Flows by transport category	☐	☐	☐	☐	☐

Fig. 1 Upper portion of the `OptionsTRANUS` interface



Costs by transport category	☐	☐	☐	☐	☐
Costs by socio-economic sector	☐	☐	☐	☐	☐
Exogenus trips by transport category	☐	☐	☐	☐	☐
Exogenous trips by category and mode	☐	☐	☐	☐	☐
Generate	Check All	Check All	Check All	Check All	Check All

```
Convergence factor                          : 0.0001
Beginning of execution for scenario 13B
TRANUS binaries directory                   : C:/TRANUS/bin/
Directory where is located the .tuz file    : C:\Users\Julien\Desktop\INRIA\ExampleC
ID of the project that we want to simulate   : 03A
ID of the scenario that we want to simulate  : 13B
Parameters file                          : W_03A13B.L1E
Observations file                        : W_03A.L0E
Zone file                              : W_03A.Z1E
Convergence factor                          : 0.0001
Calculations complete
```

Fig. 2 Lower portion of the `OptionsTRANUS` interface

3 Second interface: execution of TRANUS for multiple iterations with variation on a single variable

3.1 Objectives

The goal of the second part of my internship was to program an interface that would allow to make successive iterations of TRANUS, with variations on a single variable of the model, in order to study the impact of that variable. In order to achieve this, I needed to be able to modify the L1E files used by TRANUS, and interpret the data produced by the **imploc** program.

3.2 Edit of the L1E file

Each scenario of a TRANUS project has a L1E file, which contains the values used by the module **imploc** for its calculations. The TRANUS model divides its environment in socioeconomic sectors, each of which has a series of values, either independently or in combination with other sectors. In order for my program to function, I had to find a way to modify this file. I based my work on the class `LCALparam`, developed by Thomas Capelle. This class had the methods to read the various files created by TRANUS and storing the data contained within.

I created the method write_L1E, which replaces the contents of the L1E file targeted by the `LCALparam` object with that contained by the object itself. In this way, in order to modify a value of the L1E file, one only needs to read it, modify the corresponding attribute in the `LCALparam` object, then use the write_L1E method.

Because the TRANUS programs are quite vulnerable to the slightest modification in format, I had to find a way to rewrite the L1E file while respecting the alignment it initially had. This required a study of several L1E files, and a lot of trial and error to get the numbers to align properly. In the end, apart from the values being converted from integer to float, a L1E file before and after its rewriting is identical as long as the values aren't modified.

I also had to modify the class `LCALparam` so that it could read and store more values of the L1E file that before, in order to extend the range of variables that it could modify.

Example of reading, and re-writing a L1E file:
```
>>> t = TranusConfig(path,directory,IDproject,IDscenario)
>>> param = LCALparam(t)
>>> param.beta[0] = 1
>>> param.write_L1E(t)
```

The `TranusConfig` class, written by T. Capelle and unmodified by me, is a class that uses the information of the configuration file in order for the other methods to access it.

3.3 Design of the interface

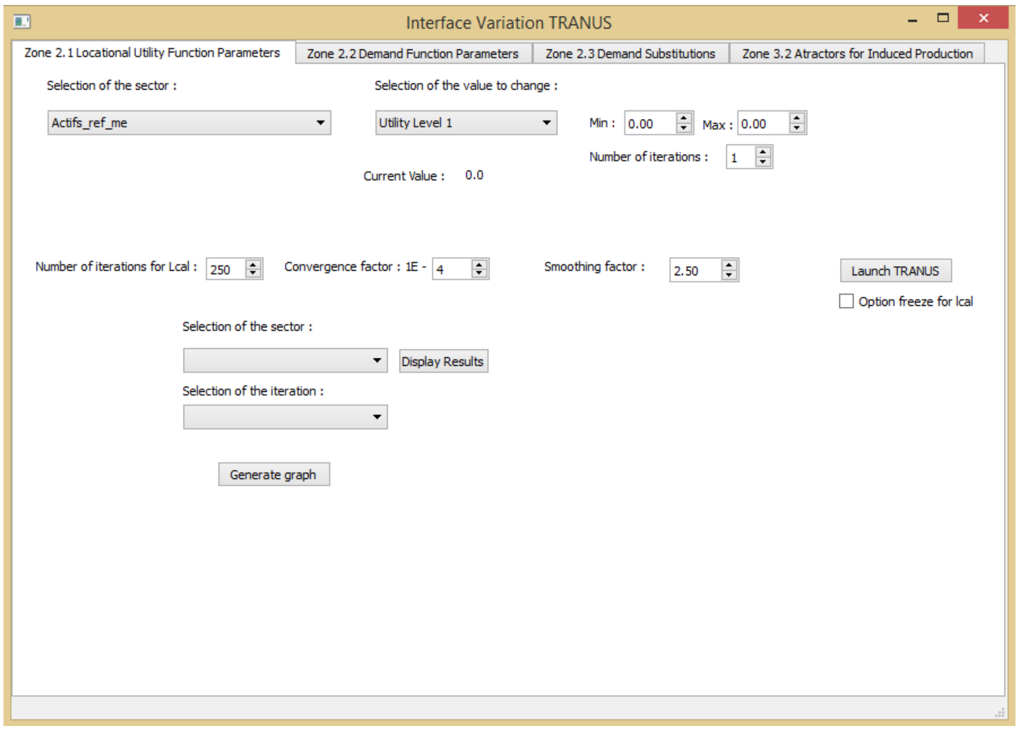
The interface is based on the structure of the L1E file. In the L1E file, the data is divided in sections, with different data formats in each. After discussion with the rest of the team, it was decided that only four of these sections needed to be modifiable (the names are those used by TRANUS, misspellings included):

- 2.1 Locational Utility Function Parameters
  (a series of attributes for each sector)
- 2.2 Demand Function Parameters
  (a trio of attributes for combinations of two different sectors)
- 2.3 Demand Substitutions
  (a trio of attributes for combinations of two different sectors)
- 3.2 Atractors for Induced Production
  (a duo of attributes for combinations of two different sectors)

Each of these sections is given a tab in the `InterfaceVariationTRANUS` (see figure 3), with a specific interface depending on the format of the data (whether it is a variable for a single sector, or a variable for a combination of sectors). In each tab, the user must first select one or two sectors, then which variable he wants to modify. The interface then displays the current value of the variable.

Once a variable has been selected, the interface displays its current value, and the user can insert the minimal and maximal values as well as the number of iterations. There are safeguards in place that will prevent the user from selecting an impossible combination of sectors (i.e. twice the same) by notifying the user of his mistake and correcting it.

Each tab also allows to modify three global values: the number of iterations for the **Lcal** binary program, the convergence factor, and the smoothing factor. These values are located in the L1E file. The user can also select whether or not to execute **Lcal** with the freeze option active, which speeds up the process.
Because each tab uses different elements of the interface, each method needs to be written four times so that there is one for each tab.


Interface Variation TRANUS
Zone 2.1 Locational Utility Function Parameters
Zone 2.2 Demand Function Parameters
Zone 2.3 Demand Substitutions
Zone 3.2 Atractors for Induced Production
Launch TRANUS
Display Results
Generate graph

Selection of the sector : Actifs_ref_me
Selection of the value to change : Utility Level 1
Min : 0.00 Max : 0.00
Number of iterations : 1
Current Value : 0.0
Number of iterations for Lcal : 250
Convergence factor : 1E - 4
Smoothing factor : 2.50
☐ Option freeze for lcal
Selection of the sector :
Selection of the iteration :

Fig. 3 Capture of the `InterfaceVariationTRANUS` interface

3.4 Execution and management of resulting files

Once the user clicks on the "Launch TRANUS" button, the method **launch** for the tab will be executed. The program will first identify which variable is going to be modified by extracting the values of the drop-down lists. Since there is no **switch** statement in Python, this works through a succession of **if**.
Then, there is a loop with the selected number of iterations. For each iteration, the value of the selected variable is calculated based on the minimal and maximal values selected, the L1E file is appropriately updated, and the **Lcal** and **imploc** programs are run.
At the end of the execution, the variable of the L1E file on which the iterations were performed is restored to its initial value.
Initially, the files generated by this process were stored in directories contained within the TRANUS project's main directory, named after the variable being modified and the number of the iteration. In order to allow the user to come back to his work later and know what the iterations are about, I had to modify this system. Now, all the output of the interface is contained within an **INTERFACE_OUTPUT** directory in the project directory. Every time the interface is launched, a sub-directory is created inside that directory, named after the date and hour of execution. It is within this directory that the output files are stored, in sub-directories named from the variable being modified and the number of the iteration. At the demand of the team, I added a copy of the L1E file used for the iteration in the directory.
When I was trying to change the destination of the output, I encountered a problem: the **imploc** program cannot take as parameter a long string of characters for the output file. In order to solve this, I instead created the output file in the project's main directory, before copying it in the correct directory.

3.5 Management of graphs

The interface generates graphs when the user clicks on the "Generate graph" button on a tab where there has already been a launch of TRANUS. Through the use of a boolean, the button does nothing otherwise.
My program uses the file generated by **imploc** for each iteration in order to obtain the needed values. The **pandas** library allows to extract the data directly from the file and into a python object, and provides various methods for the manipulation of the data.
After extraction, the program adjusts the data in order to eliminate noise. When the production of a zone is equal to zero, the value of the adjust is also put to zero. Furthermore, if the price for a zone is equal to zero, it is ignored when the graph is traced by putting it to **None**. This creates graphs with holes in them, but prevents the graphs from being difficult to read due to wild jumps from the axis to the curb and back.
Once the data has been extracted and prepared, the graph itself is traced using the **matplotlib** library. By keeping track of whether or not a graph has already been generated, the interface allows the user to display the curves of several iterations on the same graph.
For example, the interface can plot graphs showing the prices of housing unites within the spatial zones of the TRANUS model that result from the tested values of model variables (see figure 4).

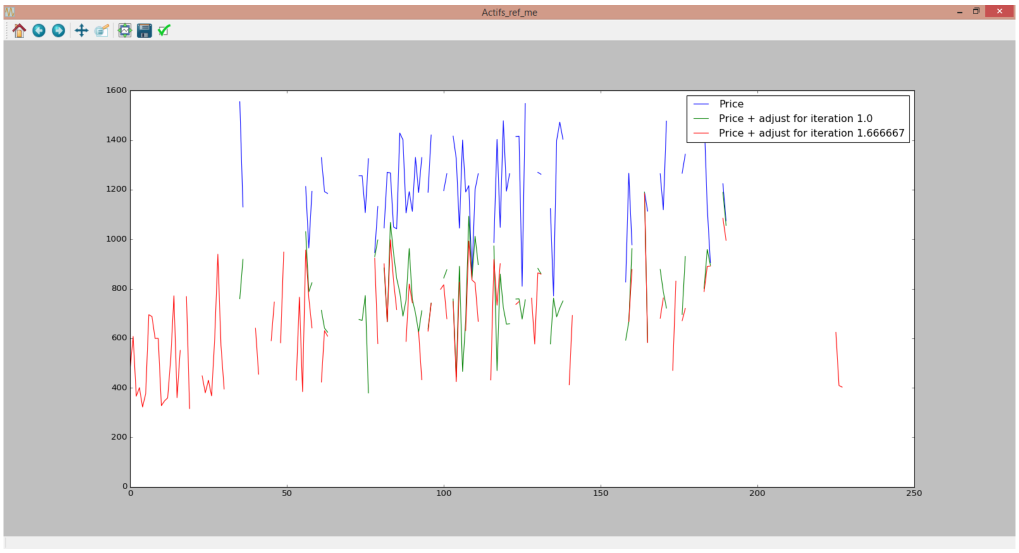
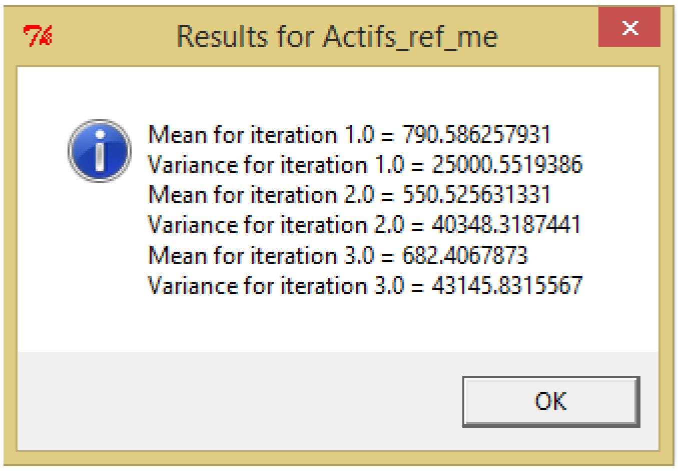


Fig. 4 Example of graph generated by the `InterfaceVariationTRANUS` interface with the values for two iterations shown alongside the basic price

3.6 Access to the mean and variance for each iteration

After a launch of TRANUS has been realized, the button labeled "Display Results" becomes functional. It displays in a message box the mean and variance of the Price for the sector selected in the list to the left, for each iteration. Since the results are only interesting if there is a variation in the variance, this allows the user to check the importance of his results without needing to look at the graphs. The figure 5 shows an example of this message box.


Results for Actifs_ref_me

```
Mean for iteration 1.0 = 790.586257931
Variance for iteration 1.0 = 25000.5519386
Mean for iteration 2.0 = 550.525631331
Variance for iteration 2.0 = 40348.3187441
Mean for iteration 3.0 = 682.4067873
Variance for iteration 3.0 = 43145.8315567
```

Fig. 5 Example of message box generated by the interface

4 Conclusion

At the time of writing, I was able to complete my assignment. The second interface simplifies the use of TRANUS, allowing the users to launch a great number of iterations at once, without needing to supervise the computer in order to insert a new set of data each time the current iteration is over.
There are several things I would have liked to add - and will try to, in the time before the end of my internship. The specs of the configuration file could be simplified, and the scripts converted into binary programs that do not require the installation of several libraries and can be easily employed even by non computer-savvy users.
The code I realized during my internship is located on Github, at the following web address:
https://github.com/JulienArmand/InterfacesTRANUS2016
It will be at the disposition of STEEP for future modifications, so that they can best adapt my work to their own needs.

Acknowledgements I would like to thank the entire STEEP team for their welcome and help during my internship. I am especially grateful for the aid of Thomas Capelle, who provided me with some of the code upon which I built my own programs, and helped me figure out which features needed to be added to the interfaces.

References

1. Description of the INRIA: http://www.inria.fr/en/institute/inria-in-brief/inria-in-a-few-words
2. Page of the STEEP team: https://team.inria.fr/steep/
3. Presentation of TRANUS: http://www.tranus.com
4. User Manual: Models and report-generating programs of TRANUS, by Modelistica, 2/8/2008
5. Presentation and guide to the PyQt library: https://wiki.python.org/moin/PyQt
6. Presentation of the Pandas library for Python: http://pandas.pydata.org/
7. Loading A CSV Into Pandas: http://chrisalbon.com/python/pandas_dataframe_importing_csv.html