Documentation of the TTT project

NNN

15 January 2017

CHAPTER

ONE

INTRODUCTION

Documentation on the rationale, Waf, and more background is at http://hmgaudecker.github.io/econ-project-templates/

The Python version of the template uses a modified version of Stachurski's and Sargent's code accompanying their Online Course *[2]* for Schelling's (1969, *[1]*) segregation model as the running exmaple.

1.1 Getting started

This assumes you have completed the steps in the README.md file **and everything worked.**

The logic of the project template works by step of the analysis:

1. Data management
2. The actual estimations / simulations / ?
3. Visualisation and results formatting (e.g. exporting of LaTeX tables)
4. Research paper and presentations.

It can be useful to have code and model parameters available to more than one of these steps, in that case see sections *Model specifications*, *Model code*, and *Code library*.

First of all, think about whether this structure fits your needs – if it does not, you need to adjust (delete/add/rename) directories and files in the following locations:

- Directories in **src/**;
- The list of included wscript files in **src/wscript**;
- The documentation source files in **src/documentation/** (Note: These should follow the directories in **src** exactly);
- The list of included documentation source files in **src/documentation/index.rst**

Later adjustments should be painlessly possible, so things won't be set in stone.

Once you have done that, move your source data to **src/original_data/** and start filling up the actual steps of the project workflow (data management, analysis, final steps, paper). All you should need to worry about is to call the correct task generators in the wscript files. Always specify the actions in the wscript that lives in the same directory as your main source file. Make sure you understand how the paths work in Waf and how to use the auto-generated files in the language you are using particular language (see the section *Project paths* below).

1.2 Project paths

A variety of project paths are defined in the top-level wscript file. These are exported to header files in other languages. So in case you require different paths (e.g. if you have many different datasets, you may want to have one path to each of them), adjust them in the top-level wscript file.

The following is taken from the top-level wscript file. Modify any project-wide path settings there.

```
def set_project_paths(ctx):
    """Return a dictionary with project paths represented by Waf nodes."""

    pp = {}
    pp['PROJECT_ROOT'] = '.'
    pp['IN_DATA'] = 'src/original_data'
    pp['IN_MODEL_CODE'] = 'src/model_code'
    pp['IN_MODEL_SPECS'] = 'src/model_specs'
    pp['OUT_DATA'] = '{}/out/data'.format(out)
    pp['OUT_ANALYSIS'] = '{}/out/analysis'.format(out)
    pp['OUT_FINAL'] = '{}/out/final'.format(out)
    pp['OUT_FIGURES'] = '{}/out/figures'.format(out)
    pp['OUT_TABLES'] = '{}/out/tables'.format(out)
```

As should be evident from the similarity of the names, the paths follow the steps of the analysis in the `src` directory:

1. **data_management** → **OUT_DATA**
2. **analysis** → **OUT_ANALYSIS**
3. **final** → **OUT_FINAL**, **OUT_FIGURES**, **OUT_TABLES**

These will re-appear in automatically generated header files by calling the `write_project_paths` task generator (just use an output file with the correct extension for the language you need – `.py`, `.r`, `.m`, `.do`)

By default, these header files are generated in the top-level build directory, i.e. `bld`. The Python version defines a dictionary `project_paths` and a couple of convencience functions documented below. You can access these by adding a line:

```
from bld.project_paths import XXX
```

at the top of you Python-scripts. Here is the documentation of the module:

> **bld.project_paths** Define a dictionary *project_paths* with path definitions for the entire project.
>
> This module is automatically generated by Waf, never change it!
>
> If paths need adjustment, change them in the root wscript file.
>
> **project_paths_join**(*key*, **args*)
>
> Given input of a *key* in the *project_paths* dictionary and a number of path arguments *args*, return the joined path constructed by:

```
os.path.join(project_paths[key], *args)
```

project_paths_join_latex(*key*, **args*)

Given input of a *key* in the *project_paths* dictionary and a number of path arguments *args*, return the joined path constructed by:

```
os.path.join(project_paths[key], *args)
```

and backslashes replaced by forward slashes.

CHAPTER

TWO

ORIGINAL DATA

Documentation of the different datasets in *original_data*.

In the original data section you would store the raw data, which you should not manipulate to ensure reproducibility.

If you want to include multiple data sets, you can also create subfolders for the sake of a clear structure.

CHAPTER

THREE

DATA MANAGEMENT

Documentation of the code in *src.data_management*. Draw simulated samples from two uncorrelated uniform variables (locations in two dimensions) for two types of agents and store them in a 3-dimensional NumPy array.

Note: In principle, one would read the number of dimensions etc. from the "IN_MODEL_SPECS" file, this is to demonstrate the most basic use of *run_py_script* only.

CHAPTER

FOUR

MAIN MODEL ESTIMATIONS / SIMULATIONS

Documentation of the code in *src.analysis*. This is the core of the project.

4.1 Schelling example

Run a Schelling (1969, *[1]*) segregation model and store a list with locations by type at each cycle.

The scripts expects that a model name is passed as an argument. The model name must correspond to a file called `[model_name].json` in the "IN_MODEL_SPECS" directory.

`run_analysis`(*agents*, *model*)
Given an initial set of *agents* and the *model*'s parameters, return a list of dictionaries with *type: N x 2* items.

`setup_agents`(*model*)
Load the simulated initial locations and return a list that holds all agents.

CHAPTER

FIVE

VISUALISATION AND RESULTS FORMATTING

Documentation of the code in *src.final*.

5.1 Schelling example

plot_locations(*locations_by_round*, *model_name*)

Plot the distribution of agents after cycle_num rounds of the loop.

CHAPTER

SIX

RESEARCH PAPER / PRESENTATIONS

Purpose of the different files (rename them to your liking):

- `research_paper.tex` contains the actual paper.
- `research_pres_30min.tex` contains a typical conference presentation.
- `research_pres_90min.tex` contains a full-length seminar presentation (add by yourself).
- `formulas` contains short files with the LaTeX formulas – put these into a library for re-use in paper and presentations.

CHAPTER

SEVEN

MODEL CODE

The directory *src.model_code* contains source files that might differ by model and that are potentially used at various steps of the analysis.

For example, you may have a class that is used both in the *Main model estimations / simulations* and the *Visualisation and results formatting* steps. Additionally, maybe you have different utility functions in the baseline version and for your robustness check. You can just inherit from the baseline class and override the utility function then.

7.1 The `Agent` class of the Schelling example

class **`Agent`**(*typ*, *initial_location*, *n_neighbours*, *require_same_type*, *max_moves*)

An Agent as in the Schelling (1969, *[1]*) segregation model. Move each period until enough neighbours of the same type are found or the maximum number of moves is reached.

Code is based on the example in the Stachurski and Sargent Online Course *[2]*.

`move_until_happy`(*agents*)

If not happy, then randomly choose new locations until happy.

CHAPTER

EIGHT

MODEL SPECIFICATIONS

The directory *src.model_specs* contains JSON files with model specifications. The choice of JSON is motivated by the attempt to be language-agnostic: JSON is quite expressive and there are parsers for nearly all languages. [1]

The best way to use this is to save a model as `[model_name].json` and then pass `[model_name]` to your code using the `append` keyword of the `run_py_script` task generator.

[1] Stata is the only execption I know of. You find a converter in the wscript file of the Stata branch. Note that there is insheetjson, but that will read a JSON file into the data set rather than into macros, which is what we need here.

CHAPTER

NINE

CODE LIBRARY

The directory *src.library* provides code that may be used by different steps of the analysis. Little code snippets for input / output or stuff that is not directly related to the model would go here.

The distinction from the *Model code* directory is a bit arbitrary, but I have found it useful in the past.

CHAPTER

TEN

REFERENCES

[1] Thomas C. Schelling. Models of segregation. *The American Economic Review*, 59(2):488–493, 1969.

[2] John Stachurski and Thomas J. Sargent. Quantitative economics. Available at http://quant-econ.net/index.html, 2013.

a

b

d

f

m