Firm Calibration Python Overview

This document gives an outline of the python modules/scripts as well as the data in the “Firm Calibration python” folder. This is not meant to be comprehensive, but rather to be a bridge between what is going on in the firm calibration parameter calibration guides/outlines and the python code its sphinx documentation.

In this main directory, there is one python script and six folders:

* Python Script (firm\_calibration.py)—this is an example script that uses all of the final functionalities in the program. It primarily just calls functions in python modules in the “processing” files to do all of the work.
* “Processing”—this has modules with helper functions for doing all of nitty-gritty coding as well as calling other modules and helper functions.
* “Data”—this has raw data and may contain modules for processing the data.
* “Constants”—this has modules that define constants that are used throughout the program. These constants are usually names or keys corresponding to data frames or columns. This way names can be changed by one place instead of manually going through all of the programs to make a change. While it is true that variable names a still fixed, those aren’t outputted in any way, so there is less reason to change them. Further, this allows the names to be written with greater consistency.
* “Output”—this is where all the final output gets written to.
* “Data Structures”—this is where all the custom data structures and related functions are defined.
* “Parameters”—contains folders partitioning the parameter calibration tasks. Each folder has data and modules for processing the corresponding parameters. (Everything in this folder could and probably should be moved to the data folder.)
* Other files and folders are all part of Sphinx, which is used to create documentation for python modules and functions. This documentation consists of html files in the “\_build/html/” directory.

Most subdirectories will have a subset of this layout. For instance, a folder might have a couple python modules along with a “data” and “processing” folder fulfilling the same tasks for those modules. The directories that don’t have this layout are ones that contain just input or output files.

Problems and Solutions

Many problems arose when writing the python scripts to calibrate the various firm parameters. The goal of this folder is to setup a systematic way of compartmentalizing these issues. This creates a framework for dealing with the generic problems, and allows the same solution, i.e. module, object, or function, to be useful for calibrating many firm parameters.

Problems:

1. The first and most glaring issue is the level of industry specificity of the data sources. Some data sources break their data down into very specific industry categories while others have somewhat generic categories. The guides and outlines go into great detail in every specific case about how to combine two specific datasets with different levels of specificity. However, the coding would be to arduous and clumsy if large chunks of code were copied/pasted and then modified for each specific case.

A NAICS tree data structure as well as helper functions were created to handle the data in consistent ways. There are numerous helper functions that do very specific tasks. A wide variety of situations can be dealt with by using different combinations of the functions. Some helper functions have inputs that customizes details of what the function to allow further flexibility.

2. It is not always clear what the name of an industry category corresponds to. The solution is to make a crosswalk for each data source mapping the industries to NAICS codes. Helper functions have been made for processing a crosswalk and codes into a tree.

3. Reading in data from files from raw data files is really rather difficult without hard coding rows and columns to read from. It was extremely time-intensive and somewhat clumsy to do so, but the programs were made fairly smart so that even if the input files change significantly in the future, it should still read the input files correctly. This was done by searching for keywords in the file as well as checking that row/column names matched.

In retrospect this was probably inadvisable because of the time-intensity and clumsiness of the code. Further, it is difficult to test if it is working for changes in the code. The alternative, is that every time the data in the files gets updated, someone manually creates a copy of dataset in a nice standard format that the program can read relatively simply.

4. One of the biggest issues was being organized and minimizing duplicate code. That is what the file structure helps deal with. Throughout programming, the code was written to do very specific tasks. The vast majority would later get deleted and replaced with a function. The function would then often get rewritten, and replace even more code. This processes repeated itself over and over again. That’s what made the programming process really tedious. It’s an important lesson to plan everything out in detail before starting programming.

Most likely, the biggest contribution of this attempt at calibrating the firm parameters will be the structure: the file structure and the data structures.

5. Testing the functions. Only spot checks at each intermediate steps and holistic checks were done to check that the functions worked. There was no standard procedure. This should be addressed at some point. There are standard coding testing practices and programs (kind of like sphinx) that help with testing. Rigorous testing can take as long as the original coding.