Background Information for “Transition to Clean Technology”

Daron Acemoglu, Ufuk Akcigit, Douglas Hanley, and William Kerr

Data

The data consists of historical CO2 emissions and concentration data, as well as information on firms and industries broken down by clean and dirty technologies. The construction of the firm and industry level data is descibed in detail in section A-3 on the Online Appendix. The relevant files are:

1. `us_emissions.csv` - Data on US emission of CO2 since 1800 comes from Oak Ridge National Labs. For raw data and source info see: http://cdiac.ornl.gov/ftp/trends/co2_emis/usa.dat
2. `world_emissions.csv` - Data on world CO2 emissions also comes from Oak Ridge National Labs. See: http://cdiac.ornl.gov/ftp/ndp030/global.1751_2008.ems
3. `siple_co2.csv` - Data on historical CO2 concentrations comes from the Siple Station ice core records. See: http://cdiac.ornl.gov/ftp/trends/co2/siple2.013
4. `maunaloa_co2.csv` - More detailed data on recent CO2 concentrations comes from the Mauna Loa Observatory. See: http://cdiac.ornl.gov/ftp/trends/co2/maunaloa.co2
5. `product_counts.csv` - Histogram of firm patent counts split by clean and dirty types. From US Census and patent data.
6. `gap_sample_sic3.csv` - Industry level (SIC3) totals for clean and dirty patents. From US Census and patent data.
7. `gap_sample_sic4.csv` - Same as above but for SIC4.

All data loading and transformation (which is fairly minimal) is done automatically in the code using the above files as input.

Code

The model solving algorithm and simulation/estimation routines are described in some detail in section A-2 of the Online Appendix. Everything is implemented in Python with some inner loops coded in C++ and integrated with the `weave` module (part of `scipy`). For non-graphical usage, only `numpy` and `scipy` are required. For graphics, one also needs to install `matplotlib` and `seaborn`. The relevant files are:

1. `infinite_weave.py` - Model solver and policy evaluator for infinite horizon problem. Employs C++ code from `dynamcis_class.cpp` for inner loop. Main policy interface is through function `simulate_type`.
2. `estimation_weave.py` - Moment simulator and estimation optimizer. Main interface is through `smm_obj`. Uses C++ code from `simulation.cpp`.
3. `generate_policy.py` - Given optimal policy parameters defined in solver, generates solutions and figures for paper content.
4. `load_data.py` - Loads input data into module. Used by higher level routines.
5. `weave_class.py` - Wrapper for the `weave` module to allow a class based interface.

To find optimal policies, one can use `sim_robust_obj` as an objective in the annealing optimizer `anneal0_pol` (or the optimizer of your choice), both of which are found in `infinite_weave.py`.

To find best fit parameters for the estimation, one can use `estimation_obj` with the `anneal0_est` optimizer (or any other), both of which are found in `estimation_weave.py`.