Replication Read Me

Jonathan Elliott, Georges V. Houngbonon, Marc Ivaldi, and Paul Scott

This document describes the data and code used to reproduce *Market Structure, Investment, and Technical Efficiencies in Mobile Telecommunications* (Elliott, Houngbonon, Ivaldi, and Scott, *Journal of Political Economy*). This document is organized as follows:

1. Description of data files
2. Description of code to run model
3. Description of code to clean data and produce paper results
4. **Example of how to run equilibrium simulation code (can skip to here)**, which can be adapted for other settings

Readers interested in computing an equilibrium given a specified market structure should consult section 4 and the notebook `example_counterfactual_simulation.ipynb`, which demonstrates with our code how to compute an equilibrium.

1. Data

Listed below are each of the datasets we use to produce the paper. Below each dataset is a description. Datasets in italics are proprietary and are therefore not included in the replication package.

- antennas
    - data on antenna characteristics
- *GSMA data*
    - data on operator market shares
- INSEE data
    - data on commune populations, areas, and incomes
- *Ookla data*
    - data on Ookla download speed tests
- *Orange customer data*
    - data from Orange on customer contracts and usage
- *quality data*
    - data from Orange on antenna usage
- spatial
    - data on detailed populations and shapefiles for France
- tariffs data
    - data on phone plans offered by each operator

2. Code to Run Model

The following Python files implement the model of supply and demand presented in the paper. There are three packages, `demand`, `supply`, and `welfare`, and within them several modules.

- `demand/`
    - `demandfunctions.py`

        - supplies functions that summarize demand, including market shares, elasticities, and diversion ratios
    - `coefficients.py`
        - supplies functions that return demand parameters for each type of consumer.
    - `dataexpressions.py`
        - supplies functions that return expected data consumption and expected utility derived from data consumption
    - `demandsystem.py`
        - defines a class named `DemandSystem`, which contains data that describes markets and includes functions of the data
    - `iteration.py`
        - supplies functions used to solve for "ξ"s quickly, copied from PyBLP (Conlon and Gortmaker, 2020), adjusted to allow use of `autograd` package
- `supply/`
    - `costs.py`
        - supplies functions that can back out marginal costs based on observed investment and pricing decisions
    - `infrastructureequilibrium.py`
        - supplies functions that compute the market equilibrium given firms' costs, demand, bandwidth allocations, and spectral efficiencies
    - `infrastructurefunctions.py`
        - supplies functions that describe signal quality based on investment levels
    - `priceequilibrium.py`
        - supplies functions that describe pricing first order conditions and elasticities
    - `transmissionequilibrium.py`
        - supplies functions to calculate average download speeds
- `welfare/`
    - `welfare.py`
        - supplies functions summarizing welfare based on prices and investment levels

3. Code to Produce Paper

Running the following files in order reproduces the paper:

Process market sizes

- `msize_alt.do`
    - computes market sizes
- `agg_shares.do`
    - constructs aggregate market shares

Process choice set

- `chset.do`
    - constructs the choice set

Process download speeds

- `import_ookla.do`
  - combines speed test datasets together
- `get_insee_code.py`
  - determines from latitude-longitude coordinates the municipality of speed tests
- `ookla_quality.do`
  - determines average download speeds for each operator-market

Process data consumption

- `dbar.do`
  - constructs average data usage from Orange customer records for each phone plan in each market

Process incomes

- `income_alt.do`
  - constructs income distributions

Process infrastructure

- `pop_dist.R`
  - computes population densities that we use
- `base_stations.do`
  - imports coordinates of base stations
- `base_stations.R`
  - combines population density and base station data
- `infrastructure_calibration.do`
  - constructs data describing infrastructure in each market by operator

Process data for demand estimation

- `demand_data.do`
  - constructs dataset used for estimating demand

Create descriptions of data

- `contracts_table.do`
  - processes a summary of data on phone contracts
- `data_consumption_patterns.do`
  - summarizes data consumption over the day
- `contracts_table.py`
  - creates table summarizing phone plans
- `descriptive_graphs.py`
  - creates graphs in paper describing data
- `summary_stats.py`
  - creates tables in paper describing data
- `data_consumption_patterns.py`
  - determines statistics about data consumption over the day

Estimate demand

- `demand.py`
  - estimates demand parameters

Process demand estimations

- `process_demand.py`

    - creates table and graphs of demand estimates

Run counterfactual simulations

- `preprocess_counterfactuals.py`
    - uses demand estimates to construct inputs to counterfactual simulations
- `counterfactuals.py`
    - runs counterfactual simulations
- `process_counterfactual_arrays.py`
    - constructs standard errors on counterfactual outcomes

Process counterfactual simulations

- `process_counterfactuals.py`
    - creates tables, graphs, and statistics summarizing counterfactual results

In addition to these files, there are several auxiliary files called by the above ones:

- `paths.py`
    - provides the location in which data files are located and output is to be saved
- `moments.py`
    - provides a list of all moments used for demand estimation
- `variancematrix.py`
    - provides functions for calculating the variance matrix
- `weightingmatrix.py`
    - provides functions for calculating the GMM weighting matrix
- `estimation.py`
    - provides functions to estimate demand via GMM
- `gmm.py`
    - provides GMM objective function

There is a Singularity image and several overlay images that contain the programs and packages required to run these scripts:

- `cuda11.0-cudnn8-devel-ubuntu18.04.sif`
- `ubuntu-20.04.1.sif`
- `telecom-cuda11.0.ext3`
- `texlive-ubuntu20.04.1.sqf`
- `r4.1.2-ubuntu20.04.1-20211129.sqf`

Due to the size of these Singularity images, they are saved in parts. To reconstruct the compressed folder of Singularity files, in the directory in which the files are saved, run
`cat singularity_part_* > singularity.tar.gz`
and then decompress `singularity.tar.gz`.

We run all of these files using SLURM's batch processing system on NYU's High Performance Computing cluster. The file `run_all.sh` submits batch jobs described in detail in the scripts with the form `run_[INSERT NAME HERE].sh`, which each provide SLURM with the information needed to run each of the files. The file `run_all.sh` submits these jobs in sequence and is a good reference for determining which scripts can be run simultaneously and

which must be run sequentially based on job dependencies. Each job requests different resources, described in the batch script (i.e., `run_[INSERT NAME HERE].sh`). In total, running our code on a Lenovo SD650 (for small memory tasks) or a Lenovo SR850 (for large memory tasks) with at most 50 GB of memory and a maximum of 18 cores, our code takes approximately 1.5 days to run. The most memory- and time-intensive task is the estimation of demand. Note that a single counterfactual equilibrium only takes a matter of minutes to run; see section 4 for more details.

Changing paths on local machine In order to reproduce the paper, you must have a "home" folder and a "scratch" folder. These can be located anywhere on your machine. Note that the scratch folder will need to store very large files. Within these folders must be the following structure:

- home folder
    - The replication package must exist in here (e.g., the subfolder `code/` will live under the home folder).
- scratch folder
    - `Databases/`
        - Within this folder must be subfolders with the same names as the datasets listed in section 1.

With this structure in place you must then change the locations in `paths.py` for the home folder and the scratch folder. There are two other paths corresponding to the Singularity image and the Singularity overlay image. These may be located anywhere on your machine as long as the paths are updated in `paths.py`.

Because some of our data is proprietary (see section 1 for a list of which datasets are proprietary and therefore cannot be publicly disseminated), it is not possible to run our code from beginning to end without obtaining access to the proprietary datasets. However, our counterfactuals are reproducible. We provide intermediate data files in the folder `intermediate_data/`, which, if added to the folder specified by `arrays_path` in `paths.py`, allows the user to run `counterfactuals.py` and the following scripts. To run (without using SLURM),[1] perform the following:

1. Add contents of `intermediate_data/` to the folder specified by `arrays_path`
2. In `counterfactuals.py`, change `task_id` to 0 and `num_cpus` to the number of CPUs available on your machine (lines 28 and 29), then run `counterfactuals.py`
3. In process_`counterfactual_arrays.py`, change `task_id` to 0 (line 14), then run process_`counterfactual_arrays.py`
4. Perform steps 2 and 3 but this time changing `task_id` to 1 for both files
5. Run `process_counterfactuals.py` (no changes needed)

4. Equilibrium Simulation Example

[1] If using SLURM, you can edit `run_all.sh` so that it begins with running counterfactuals. Delete lines 46–82 and remove the dependency in line 86. Then, in `run_counterfactuals.sh`, remove line 19 (the preprocessing for counterfactuals). Note that this will require setting up the Singularity images.

This package includes a notebook, `example_counterfactual_simulation.ipynb`, which demonstrates to the interested reader how to use our code to compute an equilibrium in prices and infrastructure given demand, products, bandwidth allocations, costs, and market characteristics. It uses the same values for these variables as used in the representative market in our counterfactuals to compute the four-firm symmetric equilibrium. It also demonstrates how to make changes to the market structure or offered products.

To run this notebook, certain versions of packages are required. There are a few ways to ensure that your machine has a correct version of the packages. We list these below, ordered from what we believe is simplest to what is most complicated:

1. Ensure that your environment has the following versions of the imported packages (others may work, but the code has been tested with these versions of the packages):
    - Python version: 3.9.19
    - NumPy version: 1.22.4
    - SciPy version: 1.13.1
    - Pandas version: 1.1.5
    - Autograd version: 1.3
    - Matplotlib: 3.8.4
2. Use the file `environment.yml` included in the replication package to create an environment with the same versions of packages as we use. If using conda, you can do so by running the following line:
   ```
   conda env create -n telecom_test -f environment.yml
   ```
3. Use the Singularity overlay image with the package environment already configured (requires Linux).

References

Conlon, Christopher, and Jeff Gortmaker. 2020. “Best practices for differentiated products demand estimation with pyblp.” *The RAND Journal of Economics*, 51(4): 1108–1161.