Replication Package for "How the Wealth Was Won"

This replication package contains the code required to reproduce the tables, figures, and estimation for "How the Wealth Was Won" by Greenwald, Lettau, and Ludvigson in the *Journal of Political Economy*. Following the instructions in this README will generate all the results in the main text and online appendix of the paper.

While the code is completely self-contained, and can create all results from scratch, this is computationally intensive. To address this, the replication package also provides a full set of intermediate inputs so that later steps (e.g., the creation of tables and figures) can be run without necessarily going through the computationally intensive earlier steps (e.g., the Random Walk Metropolis Hastings estimation).

We provide additional details below, as well as an "all-in-one" shell script that can run all steps in the correct order. Please contact Dan Greenwald at dlg340@stern.nyu.edu with additional questions.

Unzipping the compressed directories

The replication package directory begins as a set of shell scripts and a set of zip files. The zip files need to be unzipped for the replication package to run. These can be done using standard zip file extraction tools, or using the convenience script `unzip_all.sh` on a UNIX-like system. The original zipped files can then be removed to save memory, for example using `rm *.zip` inside the replication directory.

Directory map

Once the directories are unzipped, the directory contains the following subdirectories:

- `code` : contains the Python code used to create the results.
- `data` : contains the raw data used for the estimation and motivating plots.
- `paper_tables_and_figures` : contains the tables and figures used in the paper, labeled according to their index in the paper (e.g., fig_1.pdf). Running the full replication code as described below will generate these results as the final output.
- `plots` : a full set of results, including those not used in the paper.
- `save` : contains the intermediate estimation outputs, including draws of the estimated parameters and the latent states.
- `tex` : contains the tex files used to compile the paper. These read directly from the results in the `plots` directory. Running the replication code and then compiling the tex files will therefore fully replicate the compiled paper.

The directory also contains the following shell (bash) scripts:

- `collect_tables_and_figures.sh` : once the replication code has been run to reproduce the results, this collects the figures and tables used in the figure, renames them to match the paper's numbering scheme, and places them in the `paper_tables_and_figures` directory.
- `run_all.sh` : runs the complete replication code in one pass (see details below).
- `run_chain.sh` : runs the RWMC estimation for a single MCMC chain (see details below).
- `unzip_all.sh` : unzip all zipped directories.

Data sources

The `data` directory contains all data sources used in the paper, described in detail in Internet Appendix A of the paper. The mapping from these data sources to our `data` directory is as follows:

- `data/compustat` : Compustat data. Used in the internet appendix only, the replication package includes only aggregate means of several accounting variables to avoid revealing proprietary firm-level data. The full data set can be most easily accessed via Wharton Research Data Services (WRDS).
- `data/crsp` : Center for Research in Securities Prices, or "CRSP". These data use only the value-weighted returns with and without distributions to compute the aggregate price-dividend and equity returns. These data can be most easily accessed via Wharton Research Data Services (WRDS).
- `data/fof` : data from the Flow of Funds, also known as the Financial Accounts of the United States. This data source is primarily used for the value of corporate equity, corporate payouts, and total corporate equity returns. The data are created by the Federal Reserve Board of Governors and are publicly available.
- `data/nipa` : data from the National Income and Product Accounts, or "NIPA". This is our main data source for firm net value added and profits, used to compute factor shares. The data are created by the Bureau of Economic Analysis and are publicly available.
- `data/fred` : data from Federal Reserve Economic Data, or "FRED". This contains supplementary data, mainly on price indices and inflation. These data are publicly available from the Federal Reserve's FRED portal.
- `data/misc` : miscellaneous data. This aggregates two smaller data sources:
    - `martin_epbound` : the SVIX lower bound on the risk premium from "What is the Expected Return on the Market?" by Ian Martin in the 2017 *Quarterly Journal of Economics*, available from Ian Martin's website.
    - `direct_investment_income_sa` : data on net direct income from foreign subsidiaries from Table 4.2 of the US International Transactions in Primary Income on Direct Investment, publicly available from the BEA's International Data section.
- `data/pkl` : this is not a primary data source but instead collects data that have been processed by the code and saved in a friendly pickle format.
- `data/shiller` : contains Shiller's *Irrational Exuberance* data, including S&P prices, dividends, earnings, and the CAPE, from the website of Robert Shiller.
- `data/spf` : contains data from the Survey of Professional Forecasters, used for our long-term forecasts of short-term real interest rates and real economic growth. The data are publicly available and provided by the Federal Reserve Bank of Philadelphia.

Python setup

The code runs in Python 3, using the following packages:

- matplotlib
- mpi4py
- numpy
- openpyxl
- pandas
- pandas-datareader
- patsy
- scipy
- statsmodels
- tabulate
- xlrd

These can be installed using e.g.,

```
pip install matplotlib mpi4py numpy openpyxl pandas pandas-datareader patsy scipy statsmodels tabulate xlrd
```

or for Anaconda

```
conda install matplotlib mpi4py numpy openpyxl pandas pandas-datareader patsy scipy statsmodels tabulate xlrd
```

All-in-one shell script

A shell script that can run all the relevant code in the correct sequence is included as `run_all.sh`. Please note, however, that this script if run in serial may take a number of days to run. Large speedups can be attained by parallelizing the MCMC step across multiple chains, compared to the script which runs ten chains one at a time. We provide the companion script `run_chain.sh`, which can run the Estimation Step 4 for a single MCMC chain. Running multiple versions of `run_chain.sh` in parallel with different values of `GLL_MCMC_CHAIN` will deliver large speedups compared to the serial code in `run_all.sh`, and should be able to run overnight on a cluster.

Instructions: motivating plots

To create the motivating plots (Figures 1 to 4 in the paper), run `python data_only_plots.py` in the `code` directory.

Instructions: main estimation

Running the main estimation code and producing the results is performed from the script `main.py` in the `code` directory. Before providing step-by-step instructions on running the estimation, we can first provide some basic information about this script.

- Choosing which model to run is done via the `spec_no` option at the top of the script. The main specification in the paper is 49, while the alternative specification with Estimated Risk is 50.
- Choosing which tasks to run is done by uncommenting items in `run_list`, in the second section of the script. See instructions below for details on this.
- The code can be computationally intensive, particularly for the estimation, making it potentially desirable to run the code on a cluster. For this purpose, both of the options above (and many others) can be set using environment variables. For example, to set the spec_no to 50, you could first pass the command `export GLL_SPEC_NO=50` in a UNIX or Linux terminal before running the code. This is particularly convenient for setting the MCMC chain, which can be done using e.g., `export GLL_MCMC_CHAIN=2`, as explained further below.

We now provide step-by-step instructions on running the code.

Estimation Step 1: Construct the data set

The replication package includes a saved copy of the main data set that will automatically be loaded by the routine. However, to create the dataset from its original sources, uncomment the `'recompute_data'` line in `run_list` in `main.py` and then run `python main.py`.

Estimation Step 2: (Optional) Estimate the model using Sequential Monte Carlo

To initialize the RWMC estimation, we begin at the parameter mode. However, for the paper's results we first ran an estimation using the Sequential Monte Carlo method of Herbst and Schorfheide ("Sequential Monte Carlo Sampling for DSGE Models", 2014 *Journal of Applied Econometrics*). The benefit of this estimation is that it samples the likelihood at a large number of promising parameter draws. By choosing the best of these as a starting point, it makes it much easier to find the parameter mode. The only downside of the approach is that it requires specification of a prior, for which we supply a relatively loose prior distribution that does directly affect the final paper results.

In addition, while it is standard to use the inverse Hessian of the log likelihood as the initial covariance matrix for the jump density for Random Walk Metropolis Hastings, we instead use the covariance matrix of the parameters obtained from the Sequential Monte Carlo. While both approaches are equally valid from an asymptotic perspective, we have found this matrix to perform slightly better in simulations.

To run this estimation, run the file `main.py` while uncommenting the `'estimate_smc'` line in `run_list`.

This estimation can optionally run in parallel using MPI. To do this, run the script using e.g.,

```
mpiexec -np 10 python main_atsm.py
```

to break the job into 10 parallel MPI jobs. The code is completely parallelizable, and the optimal number of MPI jobs is typically the number of cores available.

Estimation Step 3: Compute the maximum likelihood estimate and Hessian

We now maximize the log likelihood, which will serve as the starting point for the RWMH estimation. To run this, uncomment the `'find_mode'` line in `run_list` in `main.py`.

If you have run a SMC estimation in Estimation Step 2, you can set `default_init_type` to `'smc'` to initialize the parameter search at the highest likelihood point found in the SMC sample.

Once the parameter mode has been found, you can compute the Hessian of the log likelihood function at the mode, whose inverse when scaled can be used for the covariance of the jump distribution. To run this, uncomment the `'compute_hessian'` line in `run_list` in `main.py` and then run `python main.py`. It is safe to run `find_mode` and `compute_hessian` in the same run.

Note that the Hessian computation substep can be skipped if you are going to use the covariance matrix from the SMC step in place of the inverse Hessian matrix.

Estimation Step 4: Estimate parameters using Random Walk Metropolis Hastings

Next, we can perform the main estimation step: estimating the parameters using Random Walk Metropolis Hastings (referred to as `rwmc` for "Random Walk Monte Carlo" in the code).

To perform this estimation, simply uncomment the `'sample_rwmc'` entry in `run_list` in `main.py` and then run `python main.py`.

An important feature of this estimation is that while the routine itself cannot be parallelized, multiple chains can be run separately, which can effectively parallelize this process. To do this, set `chain_no` to your preferred number, or alternatively pass the environment variable using e.g.,

```
GLL_MCMC_CHAIN = 5
```

to set the chain number to be 5. In the paper, we used 10 separate chains numbered 0 through 9.

Last, the RWMH routine requires an initial covariance matrix for the jump distribution. If you have SMC estimates available, you can use the covariance matrix from this distribution by setting `init_cov` to `'smc'`. Otherwise, you should set `init_cov` to `'hessian'` on line 1569 to set the initial covariance matrix to the inverse Hessian of the log likelihood at the parameter mode.

Once the parameters are estimated, there are a number of other routines that are added to `run_list` in "Estimation Step 5" that perform computations using the estimated parameters:

- `'draw_states_rwmc'`: draw from the posterior of the model's latent states using the disturbance smoother of Durbin and Koopman
- `'simulate_rwmc'`: run random simulations of the model on the set of estimated parameters
- `'fitted_values_rwmc'`: use the estimated latent states to compute returns and other asset pricing variables of interest
- `'misc_rwmc'`: compute various statistics used in the paper

These post-estimation routines are also run at the chain level, making it appropriate to run them alongside the estimation itself (i.e., to uncomment them alongside `'sample_rwmc'` in `run_list`.

Estimation Step 5: Additional computations

Once Estimation Step 4 is complete, some additional computations can be run in Estimation Step 5. The only reason that Estimation Steps 4 and 5 are separated is because these computations aggregate over all chains, making it inappropriate to run them chain by chain. These tasks include:

- `'rwmc_counterfactuals'`: compute our main counterfactual experiments holding a single latent state fixed or shutting down a single stochastic process innovation
- `'rwmc_bond_pricing'`: compute the implied path long-term real bond prices in our model

As before these can be run by uncommenting the relevant entries in `run_list` in `main.py` and then running `main.py`.

Estimation Step 6: Results

For the last step, run `python main.py` with `'results'` uncommented in `run_list`. This will create the main figures and tables used in the paper.

Additional plots

Once the main model results have been run, there are a few additional plots that use the model output to create additional results. These can be created by running `python other_plots.py`.

Recovering the tables and figures

When the steps above have been run, as well as the code for the motivating figures, the figures and tables will be saved in various subdirectories within the replication directory. However, it may not be obvious which files correspond to which model tables and figures.

To address this, the shell script `collect_table_and_figures.sh` copies the relevant tables and figures used into the paper into the directory `paper_tables_and_figures` using names corresponding to the paper (e.g., `fig_1.pdf` corresponds to Figure 1 of the paper).